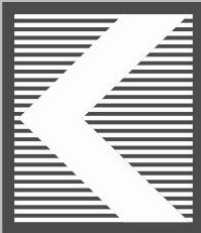


आर. एस. सलारिया

समस्या सोडवण्यासाठी प्रोग्रामिंग

प्रयोगशाळा नियमपुस्तिकेसह



KHANNA BOOK PUBLISHING CO. (P) LTD.

PUBLISHER OF ENGINEERING AND COMPUTER BOOKS

4C/4344, Ansari Road, Darya Ganj, New Delhi-110002

Phone: 011-23244447-48

Mobile: +91-99109 09320

E-mail: contact@khannabooks.com

Website: www.khannabooks.com

Dear Readers,

To prevent the piracy, this book is secured with HIGH SECURITY HOLOGRAM on the front title cover. In case you don't find the hologram on the front cover title, please write us to at contact@khannabooks.com or whatsapp us at +91-99109 09320 and avail special gift voucher for yourself.

Specimen of Hologram on front Cover title:



Moreover, there is a SPECIAL DISCOUNT COUPON for you with EVERY HOLOGRAM.

How to avail this SPECIAL DISCOUNT:

Step 1: Scratch the hologram

Step 2: Under the scratch area, your "coupon code" is available

Step 3: Logon to www.khannabooks.com

Step 4: Use your "coupon code" in the shopping cart and get your copy at a special discount

Step 5: Enjoy your reading!

ISBN: 978-93-5538-021-0

Book Code: UG055MA

Programming for Problem Solving

by R. S. Salaria

[Marathi Edition]

First Edition: 2021

Published by:

Khanna Book Publishing Co. (P) Ltd.

Visit us at: www.khannabooks.com

Write us at: contact@khannabooks.com

CIN: U22110DL1998PTC095547

To view complete list of books,
Please scan the QR Code:



Printed in India.

Copyright © Reserved

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior permission of the publisher.

This book is sold subject to the condition that it shall not, by way of trade, be lent, re-sold, hired out or otherwise disposed of without the publisher's consent, in any form of binding or cover other than that in which it is published.

Disclaimer: The website links provided by the author in this book are placed for informational, educational & reference purpose only. The Publisher do not endorse these website links or the views of the speaker/ content of the said weblinks. In case of any dispute, all legal matters to be settled under Delhi Jurisdiction only.



प्रो. अनिल डी. सहस्रबुद्धे
अध्यक्ष
Prof. Anil D. Sahasrabudhe
Chairman



सत्यमेव जयते

अखिल भारतीय तकनीकी शिक्षा परिषद्

(भारत सरकार का एक सांविधिक निकाय)

(शिक्षा मंत्रालय, भारत सरकार)

नेल्सन मंडेला मार्ग, वसंत कुंज, नई दिल्ली-110070

दूरभाष : 011-26131498

ई-मेल : chairman@aicte-india.org

ALL INDIA COUNCIL FOR TECHNICAL EDUCATION

(A STATUTORY BODY OF THE GOVT. OF INDIA)

(Ministry of Education, Govt. of India)

Nelson Mandela Marg, Vasant Kunj, New Delhi-110070

Phone : 011-26131498

E-mail : chairman@aicte-india.org

प्रास्ताविक

शतकानुशतके भारतीय समाजाच्या प्रगती आणि विस्तारामध्ये अभियांत्रिकीने अत्यंत महत्त्वपूर्ण भूमिका बजावली आहे. भारतीय उपखंडात उगम पावलेल्या अभियांत्रिकी संकल्पनांचा जगावर प्रभाव पडला आहे.

ऑल इंडिया कौन्सिल फॉर टेक्निकल एज्युकेशन (एआयसीटीई) 1987 मध्ये स्थापनेपासून तंत्रशास्त्राच्या विद्यार्थ्यांना शक्य त्या सर्व प्रकारे मदत करण्यात नेहमीच आघाडीवर असते. एआयसीटीईचे ध्येय तांत्रिक शिक्षणाला प्रोत्साहन देणे आणि त्याद्वारे उद्योगाला अधिक उंचीवर नेणे आणि शेवटी आपल्या प्रिय मातृभूमी भारताला आधुनिक विकसित राष्ट्र बनण्याचे आहे. येथे हे नमूद करणे योग्य ठरेल की अभियंते आधुनिक समाजाचा कणा आहेत – चांगले अभियंते, म्हणजे चांगले उद्योग आणि चांगले उद्योग म्हणजे चांगला देश.

NEP 2020 मध्ये प्रादेशिक भाषांमध्ये सर्वांना शिक्षणाची कल्पना मांडण्यात आली आहे, ज्यामुळे प्रत्येक विद्यार्थी पुरेसा सक्षम होईल आणि राष्ट्रीय विकासासाठी योगदान देण्याच्या स्थितीत येईल याची खात्री होईल.

एआयसीटीई गेल्या काही वर्षांपासून अविरतपणे काम करत असलेल्या क्षेत्रांपैकी एक म्हणजे सर्व अभियांत्रिकी विद्यार्थ्यांना विविध प्रादेशिक भाषांमध्ये तयार केलेल्या आंतरराष्ट्रीय दर्जाची पुस्तके माफक किमतीमध्ये उपलब्ध करून देणे. ही पुस्तके सोप्या भाषेत, वास्तविक जीवनातील उदाहरणे, समृद्ध सामग्री आणि बदलत्या जगाच्या उद्योगाच्या गरजा लक्षात घेऊनच तयार केलेली आहेत. ही पुस्तके अभियांत्रिकी आणि तंत्रज्ञानासाठी एआयसीटीई मॉडेल अभ्यासक्रम – 2018 नुसार आहेत.

संपूर्ण भारतातील प्रख्यात, उत्तम ज्ञान आणि अनुभव संपन्न प्राध्यापकांनी शैक्षणिक क्षेत्राच्या सोईसाठी ही पुस्तके लिहिली आहेत. एआयसीटीईला विश्वास आहे की ही पुस्तके त्यांच्या समृद्ध सामग्रीसह तांत्रिक विद्यार्थ्यांना अधिक सहजतेने आणि गुणवत्तेसह विषयांवर प्रभुत्व मिळविण्यात मदत करतील.

या अभियांत्रिकी विषयांना अधिक सुबक बनविण्याच्या प्रयत्नांसाठी एआयसीटीई मूळ लेखक, समन्वयक आणि अनुवादकांच्या मेहनतीचे कौतुक करते.

(Anil D. Sahasrabudhe)

ऋणनिर्देश

अभियांत्रिकी आणि तंत्रज्ञानाच्या विद्यार्थ्यांसाठी तांत्रिक पुस्तक प्रकाशित करण्यासाठी एआयसीटीईचे सूक्ष्म नियोजन आणि अंमलबजावणी केल्याबद्दल लेखक त्यांचे आभारी आहेत.

आज आम्हाला सांगायला अभिमान वाटतो की प्रादेशिक भाषांमध्ये शिक्षणाचा प्रसार करण्यासाठी हे पुस्तक प्रादेशिक भाषांमध्ये अनुवादित केलेले आहे. त्याचप्रमाणे हे पुस्तक एआयसीटीईच्या आदर्श अभ्यासक्रमाशी जुळणारे व नवीन शैक्षणिक धोरण (NEP) -2020 मार्गदर्शन तत्त्वानुसार तयार केले आहे.

मराठी भाषेतील अनुवाद कार्यासाठी योगदान दिल्याबद्दल प्रा. आसावरी शिपोस्कर आणि मराठी भाषेत समीक्षा केल्याबद्दल डॉ. पल्लवी पी. इंगळे यांचेही आम्ही आभार मानू इच्छितो.

श्री. बुद्धा चंद्रशेखर, CCO NEAT AICTE यांना आम्ही विनम्र अभिवादन करू इच्छितो. ज्यांचे AI आधारित अनुवादक साधन भाषांतराच्या उद्देशाने वापरले गेले.

शेवटी, मे .खन्नाबुक पब्लिकेशन कंपनी प्रायव्हेट लिमिटेड, नवी दिल्ली. या प्रकाशन संस्थेच्या संपूर्ण टीमने जे अमूल्य असे सहकार्य व अद्भुत पूर्ण अनुभव दिला. यासाठी त्यांचे आभार व्यक्त करायला मनापासून आनंद होत आहे.

आर. एस. सलारिया

प्रस्तावना

समस्या सोडवणे हे निःसंशयपणे सर्वात महत्वाचे कौशल्य आहे; कार्यक्षम कोड लिहिणे, प्रभावी संभाषण, कार्यसंघासह काम करणे आणि इतर अनेक कौशल्ये देखील खूप महत्वाची आहेत. पण कोणालाही कोणते कौशल्य सर्वात महत्वाचे आहे असे ठामपणे म्हणणे अशक्य आहे.

समस्या सोडवण्यासाठी प्रोग्रामिंग (प्रोग्रामिंग फॉर प्रॉब्लेम सॉल्व्हिंग) या विषयाचा उद्देश समस्या सोडवण्याची आणि त्यांच्या अंमलबजावणीसाठी C भाषेत प्रोग्राम तयार करण्याचे कौशल्य विकसित करणे आहे.

एआयसीटीईच्या आदर्श अभ्यासक्रमानुसार हे पाठ्यपुस्तक अभियांत्रिकी आणि तंत्रज्ञान (बीई/बीटेक) मधील पदवीपूर्व कार्यक्रमाच्या सर्व शाखांच्या पहिल्या वर्षाच्या विद्यार्थ्यांसाठी तयार केले गेले आहे.

प्रोग्रामिंग समस्या कशी सोडवावी हे शिकवतात. पण हे कौशल्य बहुतेक सरावाने सक्षम होते, जरी दुर्दैवाने प्रत्यक्षात काय शिकवले जाऊ शकते यावर मर्यादा आहेत.

मला महत्वाचा मुद्दा मांडायचा आहे तो म्हणजे विद्यार्थ्यांनी समस्या सोडवण्याची प्रक्रिया कृतीत करून पाह्यावी. उदाहरणार्थ, वर्गीकरण अल्गोरिदम डिझाइन करणे हे “समस्या” चे मूलभूत उदाहरण आहे ज्यास “सोडवणे” आवश्यक आहे. विविध अल्गोरिदम कसे अंमलात आणावेत आणि वर्गीकरणासाठी सर्वोत्तम धोरण कसे निवडावे हे समजून घेणे आपल्याला समस्यांचे निराकरण कसे करावे हे शिकण्यास मदत करते, अगदी प्राथमिक पद्धतीने. या प्रकारच्या विश्लेषणामध्ये काय समाविष्ट आहे याची काळजीपूर्वक तपासणी केल्यास तुमची समस्या सोडवण्याची प्रक्रिया सुधारेल. दुर्दैवाने, बहुतेक विद्यार्थी फक्त अल्गोरिदम शिकतात आणि अभ्यास पूर्ण करतात ,त्यापेक्षा जास्त खोल जात नाही.

जर तुम्ही फक्त वर्गात घेतलेले पुस्तक किंवा नोट्स वाचली आणि उपाय अंमलात आणला, तर तुम्ही समस्या सोडवायला शिकत नाही. अधिक प्रभावी पद्धत म्हणजे समस्या वाचणे, नंतर पुस्तक/नोट्स बंद करणे आणि त्यावर उपाय शोधण्याचा प्रयत्न करणे. स्वतःच एक उपाय तयार केल्यानंतर, मागे जा आणि पुस्तक/नोट्समध्ये जे लिहिले आहे त्याच्याशी तुमच्या परिणामांची तुलना करा. मग आपण समस्यांचे निराकरण कसे करावे हे शिकता.

समस्या सोडवणे ही खरोखर एक स्वयं-निर्देशित प्रक्रिया आहे. त्यासाठी कुतूहल, परिवर्तनशीलता, काळजीपूर्वक निरीक्षण आणि विश्लेषण आणि कालांतराने हळूहळू वाढणारी वैचारिक चौकट आवश्यक आहे. त्या गोष्टींपैकी, फक्त एक शिकवली जाऊ शकते ती म्हणजे वैचारिक चौकट, म्हणजे पुस्तके आणि वर्ग हेच आपल्याला देतात. बाकी तुमच्यावर अवलंबून आहे.

तुमच्या हातात असलेले हेपुस्तक तुम्हाला एक चांगले समस्या सोडवणारा आणि चांगला प्रोग्रामर होण्यासाठी आवश्यक असलेले सर्व काही प्रदान करेल.

हे पुस्तक वाचण्याचा आनंद घ्या आणि आपल्या स्पष्ट टिप्पण्या, सूचना आणि सकारात्मक टीका पाठवा. तुमच्या अपेक्षांची पूर्तता करण्यासाठी तुमची मौल्यवान माहिती मला पुस्तक सुधारण्यास आणि तुमच्या योग्य करण्यास मदत करेल.

शेवटचे , पुस्तक लिहिताना लुटी (चुकीची छपाई/चुका) टाळण्यासाठी योग्य काळजी घेण्यात आली आहे, तरीही परिपूर्णतेचा दावा करणे कठीण आहे. वाचकांनी काही लुटी आढळल्या आणि त्या आमच्या पर्यंत पोहोचवल्या तर मी त्यांची खूप आभारी असेल.

आर. एस. सलारिया

परिणाम आधारित शिक्षण

परिणाम-आधारित शिक्षण (OBE) एक विद्यार्थी-केंद्रित शिक्षण पद्धत आहे ज्यात अभ्यासक्रम वितरण, मूल्यांकन हे उद्दिष्टे आणि परिणाम साध्य करण्यासाठी नियोजित आहेत. हे विद्यार्थ्यांच्या कामगिरीचे विविध स्तरांवर परिणाम मोजमाप करण्यावर लक्ष केंद्रित करते.

निकालावर आधारित शिक्षणाचे काही महत्वाचे पैलू:

1. अभ्यासक्रमाची व्याख्या एक सेमेस्टरमध्ये अभ्यासलेला सिद्धांत, व्यावहारिक किंवा सिद्धांत आणि व्यावहारिक विषय म्हणून केली जाते.
2. अभ्यासक्रम परिणाम (COs) ही अशी विधाने आहेत जी विद्यार्थ्यांनी साध्य केलेल्या महत्त्वपूर्ण आणि आवश्यक शिक्षणाचे वर्णन करतात आणि कोर्सच्या शेवटी विश्वासाहर्पणे प्रदर्शित करू शकतात. साधारणपणे तीन किंवा अधिक कोर्सचे निकाल प्रत्येक कोर्ससाठी त्याच्या वेळेवर आधारित निर्दिष्ट केले जाऊ शकतात.
3. कार्यक्रमाची व्याख्या पदवीचे विशेषीकरण किंवा शिस्त म्हणून केली जाते. पदवी देण्याकडे नेणारी पूर्वनियोजित उद्दिष्टे साध्य करण्यासाठी अभ्यासक्रम, सह-अभ्यासक्रम आणि अतिरिक्त अभ्यासक्रमांची ही परस्पर जोडलेली व्यवस्था आहे. उदाहरणार्थ: बीई/बीटेक - संगणक विज्ञान आणि अभियांत्रिकी
4. कार्यक्रम परिणाम (पीओ) ही संकुचित विधाने आहेत जी पदवीच्या वेळेपर्यंत विद्यार्थ्यांनी काय करू शकतील अशी अपेक्षा करतात. POs पदवीधर गुणधर्मांशी जवळून जुळले जाणे अपेक्षित आहे.
5. एखाद्या कार्यक्रमाचे कार्यक्रम शैक्षणिक उद्दिष्टे (PEOs) ही अशी विधाने आहेत जी पदवीधरांच्या त्यांच्या कारकीर्दीतील अपेक्षित कामगिरीचे वर्णन करतात आणि विशेषतः पदवीधरांनी पदवीनंतर पहिल्या काही वर्षांमध्ये काय करावे आणि काय साध्य करावे अशी अपेक्षा आहे.
6. प्रोग्राम विशिष्ट परिणाम (पीएसओ) विद्यार्थ्यांना विशिष्ट शिस्तीच्या संदर्भात पदवीच्या वेळी काय करता आले पाहिजे. सहसा एका कार्यक्रमासाठी दोन ते चार PSO असतात.

ब्लूम वर्गीकरण आधारित परिणामांवर आधारित शिक्षणाच्या मूल्यांकनासाठी ज्ञानाची पातळी:

Level	Parameter	Description
K1	Remember	It is the ability to remember the previously learned material/information
K2	Understand	It is the ability to grasp the meaning of material.
K3	Apply	It is the ability to use learned material in new and concrete situations
K4	Analyze	It is the ability to break down material/concept into its component parts/subsections so that its organizational structure may be understood.
K5	Evaluate	It is the ability to judge the value of material/concept/statement/creative material /research report) for a given purpose
K6	Create	It is the ability to put parts/subsections together to form a new whole material/idea/concept/information

कार्यक्रमाचे परिणाम (POs)

- पीओ - 1. **अभियांत्रिकी ज्ञान:** जटिल अभियांत्रिकी समस्यांचे निराकरण करण्यासाठी गणित, विज्ञान, अभियांत्रिकी मूलभूत आणि अभियांत्रिकी विशेषज्ञतेचे ज्ञान लागू करा.
- पीओ - 2. **समस्येचे विश्लेषण:** गणित, नैसर्गिक विज्ञान आणि अभियांत्रिकी विज्ञानाच्या पहिल्या तत्वांचा वापर करून ठोस निष्कर्षपर्यंत पोहोचणाऱ्या जटिल अभियांत्रिकी समस्यांची ओळख करा, तयार करा, त्यांचे पुनरावलोकन करा आणि त्यांचे विश्लेषण करा.
- पीओ - 3. **समाधानाची रचना/विकास:** जटिल अभियांत्रिकी समस्या आणि डिझाइन प्रणाली घटक किंवा प्रक्रियेसाठी डिझाइन सोल्यूशन्स जे सार्वजनिक आरोग्य आणि सुरक्षितता, सांस्कृतिक, सामाजिक आणि पर्यावरणीय विचारांसाठी योग्य विचाराने निर्दिष्ट गरजा पूर्ण करतात.
- पीओ - 4. **गुंतागुंतीच्या समस्यांचे अन्वेषण करा:** वैध निष्कर्ष देण्यासाठी संशोधन आधारित ज्ञान आणि प्रयोग पद्धती, विश्लेषण आणि डेटाचे स्पष्टीकरण आणि माहितीचे संश्लेषण यासह संशोधन पद्धतीचा वापर.
- पीओ - 5. **आधुनिक साधनांचा वापर:** मर्यादा समजून घेऊन जटिल अभियांत्रिकी क्रियाकलापांसाठी अंदाज आणि मॉडेलिंगसह योग्य तंत्र, संसाधने आणि आधुनिक अभियांत्रिकी आणि आयटी साधने तयार करा, निवडा आणि लागू करा.
- पीओ - 6. **अभियंता आणि समाज:** सामाजिक, आरोग्य, सुरक्षा, कायदेशीर आणि सांस्कृतिक समस्या आणि व्यावसायिक अभियांत्रिकी सरावाशी संबंधित परिणामी जबाबदाऱ्यांचे मूल्यांकन करण्यासाठी संदर्भित ज्ञानाने सूचित केलेले तर्क लागू करा.
- पीओ - 7. **पर्यावरण आणि शाश्वतता:** सामाजिक आणि पर्यावरणीय संदर्भात व्यावसायिक अभियांत्रिकी समाधानाचा प्रभाव समजून घ्या आणि शाश्वत विकासाचे ज्ञान आणि गरज प्रदर्शित करा.
- पीओ - 8. **नैतिकता:** नैतिक तत्त्वे लागू करा आणि व्यावसायिक नैतिकता आणि अभियांत्रिकी अभ्यासाच्या जबाबदाऱ्या आणि निकषांशी बांधिलकी बाळगा.
- पीओ - 9. **वैयक्तिक आणि सांघिक कार्य:** एक व्यक्ती म्हणून, आणि विविध संघांमध्ये सदस्य किंवा नेता म्हणून आणि बहु-विषयक सेटिंग्जमध्ये प्रभावीपणे कार्य करा.
- पीओ - 10. **संप्रेषण:** अभियांत्रिकी समुदायासह आणि मोठ्या प्रमाणात समाजासह जटिल अभियांत्रिकी क्रियाकलापांवर प्रभावीपणे संवाद साधा, जसे की, प्रभावी अहवाल आणि डिझाइन दस्तऐवजीकरण समजून घेणे आणि लिहिणे, प्रभावी सादरीकरणे करणे आणि स्पष्ट सूचना देणे आणि प्राप्त करणे.
- पीओ - 11. **प्रकल्प व्यवस्थापन आणि वित्त:** अभियांत्रिकी आणि व्यवस्थापन तत्वांचे ज्ञान आणि समज प्रदर्शित करा आणि एखाद्या व्यक्तीचे कार्य, एक संघाचे सदस्य आणि नेते म्हणून, प्रकल्पांचे व्यवस्थापन करण्यासाठी आणि बहु-विषयक वातावरणात हे लागू करा.
- पीओ - 12. **आयुष्यभर शिकणे:** तांत्रिक बदलांच्या व्यापक संदर्भात स्वतंत्र आणि आयुष्यभर शिक्षण घेण्याची तयारी आणि क्षमता असणे आवश्यक आहे हे ओळखा, आणि तयार करा.

कोर्स आउटकम

हा अभ्यासक्रम विद्यार्थ्यांना खाली दिलेल्या मध्ये सक्षम करेल:

CO-1: अरीथमेटिक आणि लॉजिकल समस्यांसाठी साधे अल्गोरिदम तयार करणे

CO-2: अल्गोरिदमचे प्रोग्राममध्ये भाषांतर करण्यासाठी (C भाषेत)

CO-3: प्रोग्राम्सची चाचणी आणि एग्झीक्यूट करण्यासाठी त्याचप्रमाणे सिन्टाक्स आणि लॉजिकल त्रुटी दूर करण्यासाठी.

CO-4: कंडिशनल ब्रॅचिंग, ईट्रेशन आणि रिकरशन लागू करण्यासाठी

CO-5: एखाद्या समस्येचे फंक्शन्समध्ये विघटन करणे आणि डिव्हिड अँड कॉन्कर अॅप्रोच वापरून संपूर्ण प्रोग्रामचे संश्लेषण करणे

CO-6: अल्गोरिदम आणि प्रोग्राम तयार करण्यासाठी अॅरे, पॉइंटर्स आणि स्ट्रक्चर्स वापरणे

CO-7: मॅट्रिक्सची बेरीज आणि गुणाकार समस्या सोडवण्यासाठी, सर्चिंग आणि सॉर्टिंग साठी प्रोग्रामिंग लागू करणे .

CO-8: साध्या संख्यात्मक पद्धतीच्या समस्यांचे निराकरण करण्यासाठी म्हणजे मूळ समीकरणाचा शोध, कार्याचे वेगळेपण आणि साधे एकत्रीकरण प्रोग्रामिंग लागू करणे.

खाली दिलेल्या मॅट्रिक्सनुसार प्रोग्राम आउटकम्स आणि कोर्स आउटकम्सचे मॅपिंग करावे :

कोर्स आउट कम्स	प्रोग्राम आउटकम्स बरोबर अपेक्षित मॅपिंग (1- किमान परस्पर संबंध; 2- मध्यम परस्पर संबंध; 3- घनिष्ट परस्पर संबंध)											
	PO-1	PO-2	PO-3	PO-4	PO-5	PO-6	PO-7	PO-8	PO-9	PO-10	PO-11	PO-12
CO-1	3	3	3	-	-	-	-	-	-	-	-	-
CO-2	3	-	1	-	-	-	-	-	-	-	-	-
CO-3	3	-	1	-	3	-	-	-	-	-	-	-
CO-4	3	-	-	-	-	-	-	-	-	-	-	-
CO-5	3	3	1	2	-	-	-	-	-	-	-	-
CO-6	3	-	1	-	-	-	-	-	-	-	-	-
CO-7	3	2	1	1	-	-	-	-	-	-	-	-
CO-8	3	2	1	1	-	-	-	-	-	-	-	-

संक्षिप्तरूपे आणि चिन्हे

Abbreviations	Full form
ALU	Arithmetic and Logic Unit
ANSI	American National Standards Institute
ASCII	American Standard Code For Information Interchange
BIOS	Basic Input Output System
BIOS	Basic Input Output System
BMI	Body Mass Index
BODMAS	Bracket, Of, Division, Multiplication, Addition, And Subtraction
BOSS	Bharat Operating System Solutions
CPU	Central Processing Unit
EOF	End-Of-File
GCD	Greatest Common Divisor
GUI	Graphical User Interfaces
HCF	Highest Common Factor
HLL	High-Level Language
IDE	Integrated Developments Environment
IPO	Input-Process-Output
OS	Operating System
PDL	Program Design Language
POST	Power On Self Test
RAM	Random Access Memory
ROM	Read Only Memory
VDU	Visual Display Unit
VLSI	Very Large Scale Integration

आकृत्यांची सूची

युनिट 1: प्रोग्रामिंगची ओळख

चित्र 1.1:	संगणक प्रणालीचे कार्यात्मक घटक	3
चित्र 1.2:	सामान्य इनपुट साधने	3
चित्र 1.3:	सामान्य आउटपुट साधने	4
चित्र 1.4:	मेमरी चिप्स	4
चित्र 1.5:	सामान्य स्टोरेज साधने	5
चित्र 1.6:	वैयक्तिक संगणकाचे कार्यात्मक आकृती	6
चित्र 1.7:	संगणक प्रणालीचे स्तरित आर्किटेक्चर	7
चित्र 1.8:	संगणक प्रणालीच्या विविध संसाधनांचे व्यवस्थापन करणारे ऑपरेटिंग सिस्टम	7
चित्र 1.9:	चतुर्भुज समीकरणाच्या मुळांचे स्वरूप शोधण्यासाठी फ्लोचार्ट	12
चित्र 1.10:	सीक्वन्स स्ट्रक्चर्ससाठी स्यूडोकोड आणि फ्लोचार्ट	12
चित्र. 1.11:	स्यूडोकोड आणि फ्लोचार्ट <i>If . . . Endif</i> selection structure च्यासाठी	13
चित्र 1.12:	स्यूडोकोड आणि फ्लोचार्ट <i>If . . . Else . . . Endif</i> selection structure च्यासाठी	13
चित्र. 1.13	एल्स इफ लाडर सिन्टाक्स	14
चित्र. 1.14:	एल्स इफ लाडर चा लॉजिक फ्लो	14
चित्र 1.15:	स्यूडोकोड आणि फ्लोचार्ट <i>While . . . Endwhile</i> iterative structure च्यासाठी	15
चित्र 1.16:	स्यूडोकोड आणि फ्लोचार्ट <i>Do . . . While</i> iterative structure च्यासाठी	15
चित्र 1.17:	दोन व्हेरिएबल्स स्वॅप करण्यासाठी फ्लोचार्ट आणि स्यूडोकोड	17
चित्र 1.18:	दिलेली संख्या सम किंवा विषम आहे हे तपासण्यासाठी फ्लोचार्ट आणि स्यूडोकोड	18
चित्र 1.19:	तीनपैकी सर्वात मोठी संख्या शोधण्यासाठी फ्लोचार्ट आणि स्यूडोकोड	18
चित्र 1.20:	कमिशनची गणना करण्यासाठी फ्लोचार्ट	20
चित्र 1.21:	एका संख्येच्या अंकांची बेरीज शोधण्यासाठी फ्लोचार्ट आणि स्यूडोकोड	21
चित्र 1.22:	दिलेला क्रमांक n पॅलिंड्रोम आहे की नाही हे तपासण्यासाठी फ्लोचार्ट आणि स्यूडोकोड	22
चित्र 1.23:	दिलेली संख्या n आर्मस्ट्रॉंग संख्या आहे की नाही हे तपासण्यासाठी फ्लोचार्ट आणि स्यूडोकोड	23
चित्र 1.24:	संख्या n प्राइम आहे की नाही हे तपासण्यासाठी फ्लोचार्ट	24
चित्र 1.25:	HCF/GCD साठी संगणकीय प्रक्रियेचे उदाहरण	25
चित्र 1.26:	दोन संख्यांच्या HCF ची गणना करण्यासाठी फ्लोचार्ट आणि स्यूडोकोड	26
चित्र 1.27:	Fibonacci अनुक्रमाच्या प्रथम n टर्म्स प्रिंटसाठी फ्लोचार्ट आणि स्यूडोकोड	27

चिल 1.28: सी प्रोग्रामची सामान्य रचना	28
चिल 1.29: कॅम्पिलेशन प्रक्रिया	32
चिल 1.30: टर्बो सी/सी ++ कंपाइलरचा स्क्रीन शॉट वाक्यरचना लुटी दर्शवतो	33
चिल 1.31: टर्बो सी/सी ++ कंपायलरचा स्क्रीन शॉट संकलन प्रक्रियेचे यश दर्शवितो	33
चिल 1.32: टर्बो सी/सी ++ कंपाइलरचा स्क्रीन शॉट दुवा साधण्याच्या प्रक्रियेचे यश दर्शवितो	34
चिल 1.33: टर्बो सी/सी ++ कंपाइलर वापरकर्ता स्क्रीन प्रोग्राम आउटपुट दर्शवित आहे	34

युनिट 3: कंडिशनल ब्रॅचिंग आणि लूप

चिल 3.1: लॉजिक फ्लो कंट्रोल आणि इफ स्टेटमेण्टचा	78
चिल 3.2: लॉजिक फ्लो कंट्रोल आणि इफ एल्स स्टेटमेण्टचा कोड	79
चिल 3.3: एल्स इफ लाडर चा कोड	82
चिल 3.4: लॉजिक फ्लो कंट्रोल एल्स इफ लाडर स्टेटमेण्टचा	83
चिल 3.5: स्विच स्टेटमेण्टचा कोड	84
चिल 3.6: लॉजिक फ्लो कंट्रोल स्विच स्टेटमेण्टचा	84
चिल 3.7: लॉजिक फ्लो कंट्रोल आणि फॉर स्टेटमेण्टचा कोड	92
चिल 3.8: लॉजिक फ्लो कंट्रोल आणि व्हाइल स्टेटमेण्टचा कोड	94
चिल 3.9: लॉजिक फ्लो कंट्रोल आणि डू - व्हाइल स्टेटमेण्टचा कोड ent	96
चिल 3.10: स्विच स्टेटमेंटमध्ये ब्रेक स्टेटमेंटची क्रिया	101
चिल 3.11: व्हाइल फॉर आणि डू - व्हाइल स्टेटमेंटमध्ये ब्रेक स्टेटमेंटची क्रिया	101
चिल 3.12: व्हाइल फॉर आणि डू - व्हाइल स्टेटमेंटमध्ये कंटिन्यू स्टेटमेंटची क्रिया	102
चिल 3.13: दीर्घ विभाजनाचा वापर करून GCD साठी संगणकीय प्रक्रियेचे उदाहरण	108

युनिट 4: अ‍ॅर्रे

चिल 4.1: mark अ‍ॅर्रे	127
चिल 4.2: 1D अ‍ॅर्रे डिक्लरेशन	128
चिल 4.3: टू डायमेशनल अ‍ॅर्रे	135
चिल 4.4: मेमरीमध्ये स्ट्रिंग साठवणे	142
चिल 4.5: कॅट्रक्टर आणि स्ट्रिंगच्या साठवणुकीतील फरक	143
चिल 4.6: स्ट्रिंग आणि कॅट्रक्टरच्या अ‍ॅर्रेमधील फरक	143
चिल 4.7: अ‍ॅर्रेच्या काही भागात स्ट्रिंग साठवणे	143

युनिट 5: मूलभूत अल्गोरिदम

चिल 5.1: बबल सॉर्ट पद्धतीचे उदाहरण	174
------------------------------------	-----

चिल 5.2:	सिलेक्शन सॉर्ट पद्धतीचे उदाहरण	177
चिल 5.3:	इंसरशन सॉर्ट पद्धतीचे उदाहरण	180
चिल 5.4:	बायसेक्शन पद्धतीचे रुट अंदाजे	182
चिल 5.5:	फलन $f(x)$ साठी बहुपद $p(x)$	190
चिल 5.6:	छायांकित क्षेत्राद्वारे दर्शविलेले निश्चित अविभाज्य	191
चिल 5.7:	ट्रॅपेझॉइडल नियमानुसार क्षेत्राचा अंदाज	191

युनिट 6: फंक्शन्स

चिल 6.1:	मल्टीफंक्शन प्रोग्रामची श्रेणीबद्ध संस्था	195
चिल 6.2:	फंक्शन डिक्लरेशन सिन्टाक्स	197
चिल 6.3:	फंक्शन डेफिनेशन सिन्टाक्स	198
चिल 6.4:	फंक्शनला अर्ग्युमेण्ट म्हणून वन डायमेशनल अर्रे पास करणे	208
चिल 6.5:	फंक्शनला अर्ग्युमेण्ट म्हणून टू डायमेशनल अर्रे पास करणे	209

युनिट 7: रिकरशन

चिल 7.1:	अर्रेच्या विभाजनाचे उदाहरण	235
चिल 7.2:	मर्ज सॉर्टच्या पायरांचे स्पष्टीकरण	238

युनिट 8: स्ट्रक्चर्स

चिल 8.1:	डिफाईनिंग टॅग स्ट्रक्चर	256
चिल 8.2:	डिक्लेअरिंग टाइप डिफाईड स्ट्रक्चर	256
चिल 8.3:	फंक्शनला स्ट्रक्चर पास करणे	264
चिल 8.4:	फंक्शन रिटर्नइंग स्ट्रक्चर	265

युनिट 9: पॉइंटर्स

चिल 9.1:	मेमरी संघटना	284
चिल 9.2:	मेमरीमध्ये व्हेरिएबलचे प्रतिनिधित्व	285
चिल 9.3:	व्हेरिएबल म्हणून पॉइंटर	285
चिल 9.4:	नोड्स 4 सह पूर्णांक मूल्यांची Linear linked list	290

युनिट 10: फाइल हँडलिंग

चिल 10.1:	फायलीमध्ये डेटा साठवण्याचे उदाहरण	307
-----------	-----------------------------------	-----

तक्त्याची सूची

युनिट 1: प्रोग्रामिंगची ओळख

Table 1.1: विविध फ्लोचार्ट चिन्हे आणि त्यांचे संक्षिप्त वर्णन	11
Table 1.2: C मधील कीवर्ड	38
Table 1.3: सामान्य Escape Sequences	39
Table 1.4: काही विरामचिन्हांची यादी आणि त्यांचे वर्णन	39
Table 1.5: बिल्ट इन डेटा प्रकार	40
Table 1.6: इन्टीजर संख्येचा प्रकार	41
Table 1.7: रिअल संख्येचा प्रकार	41
Table 1.8: I/O फंक्शन्स	44
Table 1.9: सामान्यतः वापरल्या जाणार्या फॉर्मेट स्पेसिफायर्सची यादी	46

युनिट 2: अरीथमेटिक एक्सप्रेसन आणि प्रिसिडन्स

Table 2.1: बायनरी अरीथमेटिक ऑपरेटर	55
Table 2.2: रिलेशनल (तुलना) ऑपरेटर	56
Table 2.3: लॉजिकल ऑपरेटर	56
Table 2.4: बिटवाइज ऑपरेटर	57
Table 2.5: साईझ ऑफ ऑपरेटरचा वापर	59
Table 2.6: ऍड्रेस ऑफ ऑपरेटरचा वापर	59
Table 2.7: असाइनमेंट ऑपरेटर	61
Table 2.8: काही अरीथमेटिक एक्सप्रेसनचे नमुने	62
Table 2.9: अरीथमेटिक एक्सप्रेसनच्या मूल्यांकनाचे स्पष्टीकरण	63
Table 2.10: ऑपरेटरचा प्रिसिडन्स आणि अससोसिएटिव्हिटी	65
Table 2.11: काही सामान्यपणे वापरली जाणारी गणिताची फंक्शन्स	67

युनिट 4: अरे

Table 4.1: वारंवार वापरल्या जाणाऱ्या स्ट्रिंग फंक्शन्स	146
Table 4.2: Strcmp () फंक्शनद्वारे परत केलेल्या मूल्याचे स्पष्टीकरण	148

युनिट 6: फंक्शन्स

Table 6.1: कॉल बाय व्हॅल्यू आणि कॉल बाय रेफरन्स मधील फरक-1	207
--	-----

Table 6.2: कॉल बाय व्हॅल्यू आणि कॉल बाय रेफरन्स मधील फरक -2	207
---	-----

युनिट 7: रिकरशन

Table 7.1: तुलना: रिकरशन आणि इट्रेशन	241
--------------------------------------	-----

युनिट 9: पॉईंटर्स

Table 9.1: मेमरी व्यवस्थापन फंक्शन्स	291
--------------------------------------	-----

युनिट 10: फाइल हँडलिंग

Table 10.1: टेक्स्ट फाइल VS बायनरी फाइल	307
---	-----

Table 10.2: फाइल उघडण्याचे मोड	308
--------------------------------	-----

Table 10.3: fseek () फंक्शनसाठी <i>wherefrom</i> विविध मूल्ये	321
---	-----

Table 10.4: fseek () फंक्शनचा वापर स्पष्ट करणारी काही उदाहरणे	321
---	-----

शिक्षकांसाठी मार्गदर्शक सूचना

आउटकम बेस्ड एज्युकेशन (OBE) लागू करण्यासाठी विद्यार्थ्यांचे ज्ञान स्तर आणि कौशल्य संच वाढवले पाहिजे. OBE च्या योग्य अंमलबजावणीसाठी शिक्षकांनी मोठी जबाबदारी स्वीकारली पाहिजे. OBE प्रणालीतील शिक्षकांसाठी काही जबाबदाऱ्या (मर्यादित नाहीत) खालीलप्रमाणे असू शकतात:

- वाजवी मर्यादेत, त्यांनी सर्व विद्यार्थ्यांच्या सर्वोत्तम फायद्यासाठी वेळ हाताळला पाहिजे.
- त्यांनी विद्यार्थ्यांशी भेदभाव करण्याच्या इतर कोणत्याही संभाव्य अपात्रतेचा विचार न करता केवळ काही परिभाषित निकषावर मूल्यांकन केले पाहिजे.
- त्यांनी संस्था सोडण्यापूर्वी विद्यार्थ्यांच्या शिकण्याची क्षमता एका विशिष्ट स्तरावर वाढविण्याचा प्रयत्न केला पाहिजे.
- त्यांनी हे सुनिश्चित करण्याचा प्रयत्न केला पाहिजे की सर्व विद्यार्थी त्यांचे शिक्षण संपल्यानंतर दर्जेदार ज्ञान तसेच सक्षमतेने सुसज्ज आहेत.
- त्यांनी नेहमीच विद्यार्थ्यांना त्यांची अंतिम कामगिरी क्षमता विकसित करण्यास प्रोत्साहित केले पाहिजे.
- नवीन दृष्टीकोन मजबूत करण्यासाठी त्यांनी समूह कार्य आणि संघ कार्य यांना सुलभ आणि प्रोत्साहित केले पाहिजे.
- त्यांनी मूल्यांकनाच्या प्रत्येक भागात ब्लूमस टॅक्सोनोमीचे अनुसरण केले पाहिजे.

ब्लूम वर्गीकरण

स्तर	शिक्षकांनी तपासावे	विद्यार्थी सक्षम असावा	मूल्यांकनाची संभाव्य पद्धत
निर्माण करणे	विद्यार्थी तयार करण्याची क्षमता	डिझाइन करा किंवा तयार करा	सूक्ष्म प्रकल्प
मूल्यमापन	विद्यार्थ्यांचे औचित्य सिद्ध करण्याची क्षमता	वाद घालणे किंवा बचाव करणे	असाइनमेंट
विश्लेषण करणे	विद्यार्थ्यांमध्ये फरक करण्याची क्षमता	फरक किंवा भेद करा	प्रकल्प/प्रयोगशाळा पद्धती
अर्ज करणे	विद्यार्थ्यांची माहिती वापरण्याची क्षमता	चालवा किंवा प्रात्यक्षिक करा	तात्त्विक सादरीकरण/ प्रात्यक्षिक
समजून घेणे	विद्यार्थ्यांची कल्पना स्पष्ट करण्याची क्षमता	स्पष्ट करा किंवा वर्गीकृत करा	सादरीकरण / परिसंवाद
आठवणे	विद्यार्थ्यांची आठवण करण्याची क्षमता (किंवा लक्षात ठेवणे)	व्याख्या करा किंवा आठवा	प्रश्नमंजुषा

विद्यार्थ्यांसाठी मार्गदर्शक सूचना

OBE लागू करण्यासाठी विद्यार्थ्यांनी समान जबाबदारी घ्यावी. OBE प्रणालीतील विद्यार्थ्यांसाठी काही जबाबदाऱ्या (मर्यादित नाहीत) खालीलप्रमाणे आहेत:

- प्रत्येक कोर्समध्ये युनिट सुरू होण्यापूर्वी विद्यार्थ्यांना प्रत्येक UO ची चांगली माहिती असावी.
- अभ्यासक्रम सुरू होण्यापूर्वी विद्यार्थ्यांना प्रत्येक CO ची चांगली माहिती असावी
- अभ्यासक्रम सुरू होण्यापूर्वी विद्यार्थ्यांना प्रत्येक PO ची चांगली माहिती असावी
- विद्यार्थ्यांनी योग्य चिंतन आणि कृतीसह गंभीर आणि वाजवी विचार केला पाहिजे.
- विद्यार्थ्यांचे शिक्षण व्यावहारिक आणि वास्तविक जीवनातील परिणामांशी जोडलेले आणि समाकलित केले पाहिजे.
- विद्यार्थी OBE च्या प्रत्येक स्तरावर त्यांची क्षमता जाणून घ्या.

अनुक्रमणिका

प्रास्ताविक	iii
ऋणनिर्देश	v
प्रस्तावना	vii
परिणाम आधारित शिक्षण	ix
कार्यक्रमाचे परिणाम (POs)	x
कोर्स आउटकम	xi
संक्षिप्तरूपे आणि चिन्हे	xii
आकृत्यांची सूची	xiii
तक्त्याची सूची	xvi
शिक्षकांसाठी मार्गदर्शक सूचना	xviii
विद्यार्थ्यांसाठी मार्गदर्शक सूचना	xix
1. प्रोग्रामिंगची ओळख	1-52
युनिट वैशिष्ट्ये	1
तर्कशास्त्र	1
पूर्व-आवश्यकता	1
युनिट निकाल	1
1.1 प्रोग्रामिंगची ओळख (INTRODUCTION TO COMPUTERS)	2
1.1.1 संगणक प्रणालीचे घटक (Components of a Computer System)	3
1.1.2 हार्डवेअर (Hardware)	6
1.1.3 सॉफ्टवेअर (Software)	6
1.1.4 बूट करण्याची संकल्पना (Concept of Booting)	10
1.2 अल्गोरिदमचा विचार (IDEA OF ALGORITHMS)	11
1.2.1 फ्लोचार्ट (Flowchart)	11
1.2.2 स्यूडोकोड (Pseudocode)	12
1.3 अल्गोरिदम पासून प्रोग्रॅम पर्यंत (FROM ALGORITHMS TO PROGRAM)	27
1.3.1 प्रोग्रामची रचना (Structure of a C Program)	28
1.3.2 C प्रोग्राम्सची उदाहरणे (Example C Programs)	29
1.3.3 प्रोग्राम तयार करणे, कंपायलिंग आणि एक्झिक्युट करणे (Creating, Compiling and Executing a Program)	32

1.3.4	विविध C कंपाइलर (Various C Compilers)	35
1.4	C भाषेसह प्रारंभ करणे (GETTING STARTED WITH C LANGUAGE)	35
1.4.1	C भाषेची वैशिष्ट्ये (Characteristics of C Language)	36
1.4.2	C भाषेचे ॲप्लिकेशन्स क्षेत्र (Application Areas of C Language)	37
1.4.3	C भाषेचे मूलभूत बिल्डिंग ब्लॉक्स (Basic Building Blocks of C Language)	37
	युनिट सारांश	47
	अभ्यास करा	48
	व्यक्तिनिष्ठ प्रश्न	48
	एकाधिक निवड प्रश्न	49
	संगणकीय समस्या	51
	प्रॅक्टिकल	51
	आणखी माहिती	52
	संदर्भ आणि सूचविलेले वाचन	52
2.	अरीथमेटिक एक्सप्रेशन आणि प्रिसिडन्स	53-75
	युनिट वैशिष्ट्ये	53
	तर्कशास्त्र	53
	पूर्व-आवश्यकता	53
	युनिट निकाल	53
2.1	ओळख (INTRODUCTION)	54
2.2	ऑपरेटर (OPERATORS)	54
2.2.1	अरीथमेटिक ऑपरेटर (Arithmetic Operators)	55
2.2.2	रिलेशनल (कम्पॅरिझन) ऑपरेटर (Relational (Comparison) Operators)	56
2.2.3	लॉजिकल ऑपरेटर (Logical Operators)	56
2.2.4	बिटवाईस ऑपरेटर (Bitwise Operators)	57
2.2.5	स्पेशल ऑपरेटर (Special Operators)	57
2.3	एक्सप्रेशनस (EXPRESSIONS)	61
2.3.1	अरीथमेटिक एक्सप्रेशन (Arithmetic Expressions)	61
2.3.2	अरीथमेटिक एक्सप्रेशनचे मूल्यांकन (Evaluation of Arithmetic Expressions)	62
2.3.3	टाईप कन्वर्जन (Type Conversion)	63
2.4	प्रिसिडन्स आणि अससोसिएटिव्हिटी (PRECEDENCE AND ASSOCIATIVITY)	64
2.5	लायब्ररी फंक्शन (LIBRARY FUNCTIONS)	66
	युनिट सारांश	69
	अभ्यास करा	69
	व्यक्तिनिष्ठ प्रश्न	69

एकाधिक निवड प्रश्न	70
प्रॅक्टिकल	72
आणखी माहिती	74
संदर्भ आणि सूचविलेले वाचन	75
3. कंडिशनल ब्रँचिंग आणि लूप्स	76-125
युनिट वैशिष्ट्ये	76
तर्कशास्त्र	76
पूर्व-आवश्यकता	76
युनिट निकाल	76
3.1 ओळख (INTRODUCTION)	77
3.2 कंडिशनल ब्रँचिंग (CONDITIONAL BRANCHING)	78
3.2.1 इफ स्टेटमेंट (The if Statement)	78
3.2.2 इफ - एल्स स्टेटमेंट (The if - else statement)	79
3.2.3 नेस्टेड इफ आणि इफ- एल्स स्टेटमेंट (Nested if and if - else statements)	81
3.2.4 इफ-एल्स इफ लाडर (The if-else if Ladder)	82
3.2.5 स्विच स्टेटमेंट (The switch Statement)	84
3.3 लूपिंग (LOOPING)	92
3.3.1 फॉर स्टेटमेंट (The for Statement)	92
3.3.2 व्हाइल स्टेटमेंट (The while Statement)	94
3.3.3 डू - व्हाइल स्टेटमेंट (The do - while Statement)	96
3.3.4 नेस्टेड व्हाइल, फॉर आणि डू - व्हाइल स्टेटमेंट्स (NESTED while, for and do — while STATEMENTS)	98
3.4 जम्पिंग स्टेटमेंट्स (JUMPING STATEMENTS)	100
3.4.1 ब्रेक स्टेटमेंट (The break Statement)	101
3.4.2 कंटीन्यू स्टेटमेंट (The continue Statement)	102
युनिट सारांश	109
अभ्यास करा	110
व्यक्तिनिष्ठ प्रश्न	110
एकाधिक निवड प्रश्न	112
प्रोग्रामिंग समस्या	120
प्रॅक्टिकल	120
संदर्भ आणि सूचविलेले वाचन	125
4. अरे	126-165
युनिट वैशिष्ट्ये	126
तर्कशास्त्र	126

पूर्व-आवश्यकता	126
युनिट निकाल	126
4.1 ओळख (INTRODUCTION)	127
4.2 वन डायमेंशनल अरे (One-Dimensional Arrays)	127
4.2.1 डिक्लेरेशन (Declaration)	128
4.2.2 इनिशियलिझेशन (Initialization)	129
4.2.3 ऍक्सेसइंग घटक (Accessing Elements)	129
4.2.4 इनपुट (Input)	130
4.2.5 आउटपुट (Output)	130
4.3 टू डायमेंशनल अरे (Two-Dimensional Arrays)	135
4.3.1 डिक्लेरेशन (Declaration)	135
4.3.2 इनिशियलायझेशन (Initialization)	136
4.3.3 ऍक्सेसइंग घटक (Accessing Elements)	136
4.3.4 इनपुट (Input)	136
4.3.5 आउटपुट (Output)	137
4.4 कॅरेक्टर अरे अंड स्ट्रिंग्स (CHARACTER ARRAYS AND STRINGS)	142
4.4.1 स्ट्रींग्स संग्रहित करणे (Storing Strings)	142
4.4.2 स्ट्रिंग डिलिमीटरची आवश्यकता (Need of String Delimiter)	143
4.4.3 स्ट्रिंग्स लिटरल्स (String Literals)	144
4.4.4 स्ट्रिंग व्हेरिएबल्स (String Variables)	144
4.4.5 स्ट्रिंगचे इनपुट / आउटपुट (Input/Output of Strings)	145
4.4.6 स्ट्रिंग मॅनिपुलेशन फंक्शन्स (String Manipulation Functions)	145
4.4.7 स्ट्रिंग्स अरे (Array of Strings)	151
युनिट सारांश	156
अभ्यास करा	156
व्यक्तिनिष्ठ प्रश्न	156
एकाधिक निवड प्रश्न	157
प्रोग्रामिंग समस्या	160
प्रेक्टिकल	161
संदर्भ आणि सूचविलेले वाचन	165
5. मूलभूत अल्गोरिदम	166-193
युनिट वैशिष्ट्ये	166
तर्कशास्त्र	166
पूर्व-आवश्यकता	166
युनिट निकाल	166

5.1 सर्चिंग अल्गोरिदम (SEARCHING ALGORITHMS)	167
5.1.1 लिनिअर सर्च (Linear Search)	167
5.1.2 बायनरी सर्च (Binary Search)	169
5.2 सॉर्टिंग अल्गोरिदम (SORTING ALGORITHMS)	171
5.2.1 बबल सॉर्ट (Bubble Sort)	171
5.2.2 सिलेक्शन सॉर्ट (Selection Sort)	176
5.2.3 इंसरशन सॉर्ट (Insertion Sort)	179
5.3 एखाद्या समीकरणचे मूळ शोधणे (FINDING ROOT OF AN EQUATION)	181
युनिट सारांश	184
अभ्यास करा	185
व्यक्तिनिष्ठ प्रश्	185
एकाधिक निवड प्रश्न	185
प्रोग्रामिंग समस्या	187
प्रेक्टिकल	187
संदर्भ आणि सूचविलेले वाचन	193
6. फंक्शन्स	194-226
युनिट वैशिष्ट्ये	194
तर्कशास्त्र	194
पूर्व-आवश्यकता	194
युनिट निकाल	194
6.1 परिचय (INTRODUCTION)	195
6.2 फंक्शन म्हणजे काय? (WHAT IS A FUNCTION?)	195
6.3 फंक्शन वापरण्याचे फायदे (ADVANTAGES OF USING FUNCTIONS)	196
6.4 फंक्शनचे प्रकार (TYPE OF FUNCTIONS)	196
6.5 लोकल डेटा आणि ग्लोबल डेटाची संकल्पना (CONCEPT OF LOCAL DATA & GLOBAL DATA)	196
6.6 युझर- डिफाइंड फंक्शन्स (USER-DEFINED FUNCTIONS)	197
6.7 डिक्लेअरइंग आणि डिफाईनिंग फंक्शन्स (DECLARING AND DEFINING FUNCTIONS)	197
6.7.1 डिक्लेअरइंग फंक्शन (Declaring a Function)	197
6.7.2 डिफाईनिंग फंक्शन (Defining a Function)	198
6.8 कॉलइंग फंक्शन (CALLING A FUNCTION)	200
6.9 रिटर्न स्टेटमेंट (THE return STATEMENT)	201
6.10 पाससिंग अर्ग्युमेंट्स टू फंक्शन (PASSING ARGUMENTS TO A FUNCTION)	201
6.10.1 मूल्यानुसार कॉल करा (Call by Value)	201

6.10.2	संदर्भानुसार कॉल करा (Call by Reference)	203
6.10.3	व्हॅल्यूद्वारे कॉल आणि संदर्भ द्वारे कॉल यांच्यात तुलना (Comparison between Call by Value and Call by Reference)	207
6.10.4	अर्ग्युमेण्ट म्हणून वन डायमेंशनल अरे पास करणे (Passing One-Dimensional Array as Argument)	208
6.10.5	अर्ग्युमेण्ट म्हणून टू डायमेंशनल अरे पास करणे (Passing Two-Dimensional Array as Arguments)	209
6.10.6	स्ट्रिंग फंक्शनला पास करणे (Passing String to a Function)	211
	युनिट सारांश	216
	अभ्यास करा	217
	व्यक्तिनिष्ठ प्रश्न	217
	प्रोग्रामिंग समस्या	218
	एकाधिक निवड प्रश्न	219
	प्राॅक्टिकल	221
	संदर्भ आणि सूचविलेले वाचन	226
7.	रिकरेशन 227-253	
	युनिट वैशिष्ट्ये	227
	तर्कशास्त्र	227
	पूर्व-आवश्यकता	227
	युनिट निकाल	227
7.1	ओळख (INTRODUCTION)	228
7.2	रिकर्सिव्ह फंक्शन्स (RECURSIVE FUNCTIONS)	228
7.2.1	फॅक्टोरियल फंक्शन (Factorial Function)	229
7.2.2	फिबोनाची नंबरस (Fibonacci Numbers)	230
7.2.3	अॅकर्मन फंक्शन (Ackermann Function)	232
7.3	क्विक सॉर्ट अल्गोरिदम (QUICK SORT ALGORITHM)	233
7.4	मर्ज सॉर्ट अल्गोरिदम (MERGE SORT ALGORITHM)	237
7.5	रिकरेशन, इट्रेशन किंवा? (Recursion, Iteration or ...?)	240
	युनिट सारांश	245
	अभ्यास करा	246
	व्यक्तिनिष्ठ प्रश्न	246
	एकाधिक निवड प्रश्न	246
	प्रोग्रामिंग समस्या	252
	प्राॅक्टिकल	253
	संदर्भ आणि सूचविलेले वाचन	253

8. स्ट्रक्चर्स	254-281
युनिट वैशिष्ट्ये	254
तर्कशास्त्र	254
पूर्व-आवश्यकता	254
युनिट निकाल	254
8.1 ओळख (INTRODUCTION)	255
8.2 डिफाईनिंग स्ट्रक्चर (DEFINING A STRUCTURE)	256
8.3 स्ट्रक्चर व्हेरिएबल्स डिक्लेअर करणे (DECLARING STRUCTURE VARIABLES)	257
8.4 इनिशियालाइझिंग स्ट्रक्चर (INITIALIZING STRUCTURES)	258
8.5 स्ट्रक्चर एलिमेंट्स ऍक्सेसइंग (प्रवेश करत आहे) (ACCESSING STRUCTURE ELEMENTS)	258
8.6 ऍक्सेसइंग स्ट्रक्चर्स (ASSIGNING OF STRUCTURES)	259
8.7 स्ट्रक्चर्सचे वाचन / लेखन (READING/WRITING STRUCTURES)	260
8.8 स्ट्रक्चरचा अरे (ARRAYS OF STRUCTURES)	261
8.8.1 स्ट्रक्चरचा अरे एलिमेंट्स ऍक्सेसइंग (Accessing Elements of Array of Structures)	262
8.8.2 इनिशियालाइझइंग स्ट्रक्चरचा अरे (Initializing Array of Structures)	262
8.9 फंक्शनला स्ट्रक्चर पास करणे (PASSING STRUCTURE TO A FUNCTION)	264
8.10 फंक्शन रिटर्नइंग स्ट्रक्चर (FUNCTION RETURNING A STRUCTURE)	265
युनिट सारांश	273
अभ्यास करा	273
व्यक्तिनिष्ठ प्रश्न	273
एकाधिक निवड प्रश्न	274
प्रोग्रामिंग समस्या	277
प्रेक्टिकल	278
संदर्भ आणि सूचविलेले वाचन	281
9. पॉइंटर्स	282-304
युनिट वैशिष्ट्ये	282
तर्कशास्त्र	282
पूर्व-आवश्यकता	282
युनिट निकाल	283
9.1 परिचय (INTRODUCTION)	283
9.2 पॉइंटर्सची कल्पना (IDEA OF POINTERS)	284
9.3 ऍक्सेसइंग व्हेरिएबलचा पत्ता (ACCESSING ADDRESS OF A VARIABLE)	285
9.4 पॉइंटर डिक्लेअर करा (DECLARING A POINTER)	286
9.5 पॉइंटरला ऍड्रेस असाइनइंग (ASSIGNING ADDRESS TO A POINTER)	286

9.6	पॉइंटर व्हेरिएबल वापरून ऍक्सेसइंग व्हेरिएबल (ACCESSING VARIABLE USING POINTER VARIABLE)	287
9.7	पॉइंटर अरीथमेटिक (POINTER ARITHMETIC)	287
9.8	फंक्शन अर्ग्युमेण्ट म्हणून पॉइंटर्स (POINTERS AS FUNCTION ARGUMENTS)	288
9.9	पॉइंटर्स आणि स्ट्रक्चर्स (POINTERS AND STRUCTURES)	289
9.9.1	स्वतः ची संदर्भ देणारे स्ट्रक्चर (Self-referential Structures)	290
9.10	डायनेमिक मेमरी अलोकेशन (DYNAMIC MEMORY ALLOCATION)	291
9.10.1	मेमरी वाटप (Allocating Memory)	291
9.10.2	डी-वाटप मेमरी (De-allocating Memory)	292
	युनिट सारांश	294
	अभ्यास करा	295
	व्यक्तिनिष्ठ प्रश्न	295
	एकाधिक निवड प्रश्न	296
	प्रोग्रामिंग समस्या	297
	प्रेक्टिकल	299
	संदर्भ आणि सूचविलेले वाचन	304
10.	फाईल हँडलिंग	305-332
	युनिट वैशिष्ट्ये	305
	तर्कशास्त्र	305
	पूर्व-आवश्यकता	305
	युनिट निकाल	305
10.1	ओळख (INTRODUCTION)	306
10.2	फायलीचे प्रकार (TYPES OF FILES)	306
10.3	फायलीवर प्रक्रिया करण्याच्या पायऱ्या (STEPS IN PROCESSING A FILE)	307
10.3.1	फाईल उघडणे (Opening a File)	307
10.3.2	फाईल बंद करणे (Closing a File)	309
10.3.3	चाचणी फायलीचे वाचन आणि लेखन (Reading and Writing of Text Files)	309
10.3.4	बायनरी फाइल्सचे वाचन आणि लेखन (Reading and Writing of Binary Files)	316
10.4	फाईल पोजिशनिंग फंक्शन्स (FILE POSITIONING FUNCTIONS)	320
10.4.1	रिवाइंड फाइल: rewind () फंक्शन	320
10.4.2	सद्य स्थान (Current Location): ftell () फंक्शन	321
10.4.3	स्थान सेट करा (Set Position): fseek () फंक्शन	321
10.5	फाइल स्थिती फंक्शन (FILE STATUS FUNCTIONS)	322
10.5.1	फाईलचा शेवट तपासण्यासाठी (Test End of File): feof() फंक्शन	322

10.5.2 चाचणी त्रुटी (Test Error): ferror() फंक्शन	322
10.5.3 क्लिअर त्रुटी (Clear Error): clearerr() Function	323
युनिट सारांश	326
अभ्यास करा	326
व्यक्तिनिष्ठ प्रश्न	326
एकाधिक निवड प्रश्न	327
प्रोग्रामिंग समस्या	328
प्रेक्टिकल	328
संदर्भ आणि सूचविलेले वाचन	332
पुढील शिक्षणासाठी संदर्भ	335
CO आणि PO अटेंन्मेंट तक्ता	336
सूची	337

1

प्रोग्रामिंगची ओळख

युनिट वैशिष्ट्ये

या युनिटमध्ये संगणकाची ओळख आणि त्याचे घटक, सॉफ्टवेअर आणि त्याचे प्रकार, अभ्यासक्रमाशी संबंधित महत्त्वाची सॉफ्टवेअर साधने, अल्गोरिदम विकसित करणे, अल्गोरिदमचे प्रोग्राममध्ये भाषांतर करणे आणि C प्रोग्राम तयार करणे, कंपाईल करणे आणि कार्यान्वित करण्याच्या संबंधित विविध पायऱ्यांवर चर्चा केली आहे. प्रोग्राम आणि C भाषांचे मूलभूत घटक सादर केले आहे.

तर्कशास्त्र

आपण सर्वजण या डिजिटल युगात जगत आहोत. म्हणूनच प्रत्येकास संगणक (computers) आणि तंत्रज्ञानाचा कार्यक्षमतेने उपयोग करण्यासाठी लागणारे ज्ञान आणि क्षमता यामध्ये सक्षम बनविणे आवश्यक आहे. अभियंता (Engineers) दिवसेंदिवस वास्तविक जीवनातील समस्यांसाठी थेरिझ, डिझाइन, हार्डवेअर आणि सॉफ्टवेअर सोल्यूशन्स विकसित आणि लागू करतात.

हे मूलभूत युनिट विद्यार्थ्यांना संगणक प्रणाली (computer system) आणि त्याशी संबंधित उपकरणांचे घटक आणि ते कार्य पूर्ण करण्यासाठी एकमेकांशी कसे संवाद साधतात हे समजण्यास मदत करते; लॉजिकल (logical) आणि संख्यात्मक (numerical) समस्या सोडविण्यासाठी अल्गोरिदमचा विकास आणि अल्गोरिदमला एक्जीक्यूटेबल प्रोग्राममध्ये सी (C) भाषेद्वारे रूपांतरित करण्याचे विविध चरण (Stages).

पूर्व-आवश्यकता

- संगणकांचे कार्य (Working of computers)
- मूलभूत गणित (Basic mathematics)
- लॉजिकल तर्क (Logical reasoning)
- संभाषण कौशल्य (Communication skills)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U1-O1:	सोपी अरीथमेटिक आणि लॉजिकल समस्यांसाठी अल्गोरिदम तयार करा.
U1-O2:	अल्गोरिदमचे C प्रोग्राममध्ये भाषांतर करा.

U1-O3:	दिलेल्या प्लॅटफॉर्मवर / विकसित वातावरणावर प्रोग्राम तयार, कंपाईल आणि एक्झिक्युट करा.
U1-O4:	प्रोग्रामची टेस्टिंग आणि डीबगिंग.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 1 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1 – Weak Correlation; 2 – Medium Correlation; 3 – Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U1-O1	3							
U1-O2		3						
U1-O3			3					
U1-O4			3					

1.1 प्रोग्रामिंगची ओळख (INTRODUCTION TO COMPUTERS)

संगणक एक इलेक्ट्रॉनिक मशीन आहे जे प्रोग्राम्स नावाच्या निर्देशांच्या संचानुसार कार्ये किंवा गणना करते. (Computer is an electronic machine that performs tasks or computations according to a set of instructions called *programs*.)

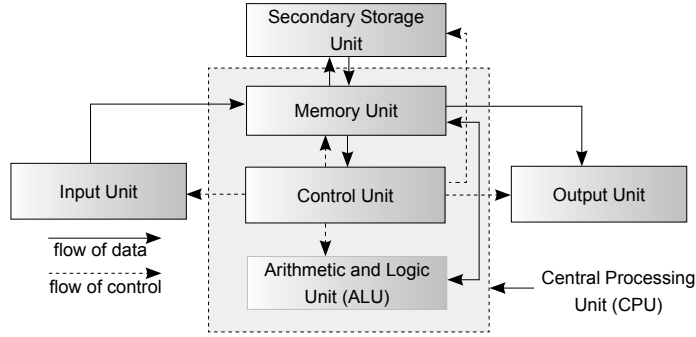
1940 च्या दशकात सुरू केलेले प्रथम पूर्णपणे इलेक्ट्रॉनिक संगणक म्हणजे प्रचंड मशीन होती ज्यांना चालवण्यासाठी लोकांच्या टीमची आवश्यकता होती. त्या सुरुवातीच्या मशीनच्या तुलनेत आजचे संगणक विस्मयकारक आहेत - ते हजारपट वेगवान आहेत आणि ते आपल्या डेस्कवर, आपल्या मांडीवर किंवा अगदी खिशातही बसू शकतात.

एक संगणक, सर्वसाधारणपणे, खाली दिलेल्या मध्ये सक्षम आहे

- विविध स्वरूपात डेटा आणि सूचना प्राप्त करणे, (Receiving the data and instructions in various forms,)
- त्याच्या मेमरीत डेटा आणि सूचना संग्रहित करत आहे, (Storing the data and instructions in its memory,)
- सेकंडरी स्टोरेज वर आधीपासून संचयित केलेला डेटा पुनर्प्राप्त करीत आहे, (Retrieving the data already stored on secondary storage,)
- निर्दिष्ट निर्देशानुसार अत्यंत वेगाने आणि मोठ्या अचूकतेसह गणित व तार्किक ऑपरेशन्स करणे, (Performing arithmetic and logical operations, according to the specified instructions, with very high speed and greater accuracy,)
- बऱ्याच प्रकारांमध्ये निकाल तयार करणे आणि (Producing the results in many forms, and)
- इतर संगणकांशी संप्रेषण करत आहे, (Communicating with other computers.)

1.1.1 संगणक प्रणालीचे घटक (Components of a Computer System)

संगणक म्हणजे हार्डवेअर आणि सॉफ्टवेअरचे संयोजन. हार्डवेअर संगणकाचे भौतिक घटक जसे की मदरबोर्ड, मेमरी डिव्हाइस, मॉनिटर, कीबोर्ड इत्यादींचे प्रतिनिधित्व करते, तर सॉफ्टवेअर प्रोग्राम किंवा निर्देशांच्या संचाचे प्रतिनिधित्व करते. दोन्ही हार्डवेअर आणि सॉफ्टवेअर एकत्र संगणक प्रणालीचे कार्य करतात.



चित्र 1.1: संगणक प्रणालीचे कार्यात्मक घटक

संगणकास दिलेली प्रत्येक कार्य इनपुट-प्रक्रिया-आउटपुट सायकल (आयपीओ सायकल) चे अनुसरण करते. यासाठी विशिष्ट इनपुट आवश्यक आहे, त्या इनपुटवर प्रक्रिया केली जाते आणि इच्छित आउटपुट तयार होते. इनपुट युनिट इनपुट घेते, केंद्रीय प्रक्रिया युनिट डेटाची प्रक्रिया करते आणि आउटपुट युनिट आउटपुट तयार करते. प्रक्रियेदरम्यान मेमरी युनिट डेटा आणि सूचना ठेवते.

संगणकाच्या कार्य-घटकांकडे एक-एक करून पाहू या.

इनपुट युनिट (Input Unit)

इनपुट युनिटमध्ये विविध इनपुट साधने (devices) असतात ज्या संगणकाशी कनेक्ट केल्या जाऊ शकतात, जसे कीबोर्ड, माऊस, लाइट पेन, मायक्रोफोन इ. कीबोर्ड हे इनपुट स्टॅण्डर्ड डिव्हाइस आहे, जे प्रत्येक नवीन संगणका बरोबर असते. कोणत्याही वेळी, एकापेक्षा जास्त इनपुट डिव्हाइस संगणकावर कनेक्ट केले जाऊ शकतात.

बाह्य जगाकडून माहिती प्राप्त करणे, त्यास बायनरी स्वरूपात रूपांतरित करणे आणि नंतर मुख्य मेमरीवर हस्तांतरित करणे हे इनपुट युनिटचे उद्दीष्ट आहे. यात काही प्रोग्रामद्वारे प्रक्रिया करण्यासाठी डेटा प्रविष्ट करणे, भाषण रेकॉर्ड करणे, देखावा किंवा छायाचित्र कॅप्चर करणे किंवा प्रदर्शनात काही वस्तू निवडणे समाविष्ट असू शकते.



(a) Typical keyboard with QWERTY layout



(b) Wireless keyboard



(c) Mouse



(d) Microphone



(e) Web Camera

चित्र 1.2: सामान्य इनपुट साधने

आउटपुट युनिट (Output Unit)

आउटपुट युनिटमध्ये विविध आउटपुट साधने (devices) असतात जी संगणकाशी कनेक्ट केली जाऊ शकतात, जसे व्हिज्युअल डिस्प्ले युनिट (व्हीडीयू) किंवा मॉनिटर, प्रिंटर, प्लॉटर्स, ऑडिओ सिस्टम इ. म्हणून त्या ओळखल्या जातात. मॉनिटर हे आउटपुट स्टॅंडर्ड डिव्हाइस आहे, जे प्रत्येक नवीन संगणका बरोबर असते. कोणत्याही वेळी, इनपुट डिव्हाइस प्रमाणेच, एकापेक्षा जास्त आउटपुट डिव्हाइस संगणकावर कनेक्ट केले जाऊ शकतात.

आउटपुट युनिटचा उद्देश मुख्य मेमरी युनिटमधून बायनरी स्वरूपात माहिती प्राप्त करणे, त्यास मानवी समजण्यायोग्य स्वरूपात रूपांतरित करणे आणि नंतर त्याचे आउटपुट देणे हे होय. यात काही संगणक प्रोग्रामचे परिणाम प्रदर्शित करणे, मुद्रण करणे किंवा प्लॉट करणे किंवा काही संगीत किंवा चित्रपट प्ले करणे यांचा समावेश असू शकतो.



(a) Monitor



(b) Printer



(c) Plotter

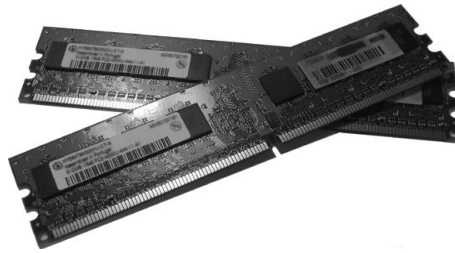


(d) Speakers

चित्र 1.3: सामान्य आउटपुट साधने

मेमरी युनिट (Memory Unit)

मेमरी युनिट संगणकाचे कोठार (warehouse) म्हणून कार्य करते, जिथे प्रोग्राम्स एक्झिक्युट होतात, इनपुट डेटा, कॉम्प्यूटेशनचे इंटरमीडिएट आणि अंतिम परिणाम संग्रहित केले जातात. संगणकात प्रविष्ट केलेली माहिती प्रथम मेमरी युनिटमध्ये संग्रहित केली जाते. त्याचप्रमाणे आऊटपुट असणारा डेटा किंवा निकाल मेमरी युनिट मधून घेतले जातात. हे मेमरी युनिट volatile मेमरीचे प्रतिनिधित्व करते, याचा अर्थ असा की पॉवर बंद केल्यावर संग्रहित माहिती गमावली जाईल.



चित्र 1.4: मेमरी चिप्स

अरीथमेटिक आणि लॉजिकल युनिट (Arithmetic and Logic Unit)

अरीथमेटिक आणि लॉजिकल युनिट (एएलयू), जसे त्याच्या नावावरून स्पष्ट होते, अंकगणित आणि लॉजिकल ऑपरेशन्स करतात. अंकगणित ऑपरेशन्समध्ये बेरीज, वजाबाकी, गुणाकार आणि भागाकार समाविष्ट आहे. लॉजिकल ऑपरेशन्स दोन डेटा आयटमची तुलना करण्यासाठी, पेक्षा कमी ($<$), पेक्षा कमी किंवा समान (\leq), पेक्षा मोठे ($>$) पेक्षा मोठे किंवा समान (\geq), समान ($=$) आणि समान नाही (\neq) ह्या सारखे ऑपरेशन्स समाविष्ट करतात.

कंट्रोल युनिट (Control Unit)

कंट्रोल युनिट संगणक प्रणालीच्या इतर सर्व भागांचे समन्वय साधते आणि नियंत्रित करते. प्रोग्रामच्या निर्देशानुसार, नियंत्रण युनिट चार मूलभूत ऑपरेशन्स करते:

- फेच (Fetch)- संगणकाच्या मेमरीमधून पुढील प्रोग्राम सूचना मिळवणे.
- डीकोड (Decode)- प्रोग्राम संगणकास काय करण्यास सांगत आहे हे शोधून काढणे.
- एक्झिक्युट (Execute)- विनंती केलेली कृती करणे, जसे की दोन संख्या जोडणे किंवा त्यापैकी एक मोठी आहे की नाही हे ठरविणे.
- राइट-बॅक (Write-Back)- परिणाम परत मेमरीवर लिहिणे.

या चार- पायऱ्या प्रक्रियेस मशीन सायकल किंवा प्रोसेसिंग सायकल असे म्हणतात आणि त्यात दोन टप्पे असतात: इंस्ट्रक्शन सायकल (फेच आणि डीकोड) आणि एक्झिक्युशन सायकल (एक्झिक्युट व राइट-बॅक).

सेकंडरी स्टोरेज युनिट (Secondary Storage Unit)

सेकंडरी स्टोरेज युनिट भविष्यातील वापरासाठी महत्त्वपूर्ण डेटा दीर्घकालीन संग्रह करते. डेटा मेमरी युनिटमधून सेकंडरी स्टोरेज युनिटमध्ये हस्तांतरित केला जातो. सेकंडरी स्टोरेज मध्ये साठवलेल्या डेटावर प्रक्रिया करण्यासाठी, ते मेमरीमध्ये हस्तांतरित केले जाते. कृपया लक्षात ठेवा सेकंडरी स्टोरेज साठवलेल्या डेटावर थेट सेकंडरी स्टोरेज प्रक्रिया केली जाऊ शकत नाही. केवळ मेमरीमध्ये असल्यास डेटावर प्रक्रिया केली जाऊ शकते.



(a) Hard disk



(b) CD/DVD



(c) Pen drive

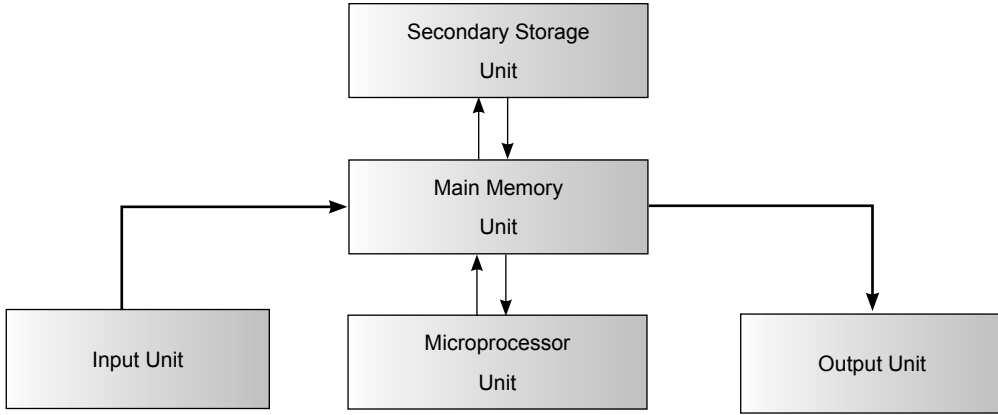
चित्र 1.5: सामान्य स्टोरेज साधने



- कंट्रोल युनिट, अरीथमेटिक आणि लॉजिकल युनिट आणि मुख्य मेमरी युनिट एकत्रितपणे सेंट्रल प्रोसेसिंग युनिट (सीपीयू) म्हणून ओळखले जातात.
- संगणकाशी कनेक्ट केलेले इनपुट डिव्हाइस आणि आउटपुट साधने यासारख्या सर्व बाह्य उपकरणांना सामान्यतः फेरीफेरल्स म्हणून ओळखले जाते.

मटेरियल आणि मायक्रोइलेक्ट्रॉनिक्सच्या क्षेत्रात प्रगतीमुळे व्हेरी लार्ज स्केल इंटीग्रेशन (व्हीएलएसआय) म्हणजे एका चिपच्या आत कंट्रोल युनिट, अरीथमेटिक आणि लॉजिकल युनिट आणि काही रेजिस्टरच्या रूपात मर्यादित हाय स्पीड मेमरी एम्बेड करणे शक्य झाले आहे. ज्याला **मायक्रोप्रोसेसर** असे म्हणतात.

औपचारिकरित्या, **मायक्रोप्रोसेसर**, जो सामान्यतः प्रोसेसर म्हणून ओळखला जातो, एक सामान्य हेतूने प्रोग्राम करण्यायोग्य डिव्हाइस आहे, जो बायनरी स्वरूपात माहिती प्राप्त करू शकतो, त्यावर प्रक्रिया करू शकतो आणि परिणाम प्रसारित करू शकतो. त्याला समजणाऱ्या सूचनांचा वापर करून विविध समस्या सोडविण्यासाठी प्रोग्राम केले जाऊ शकते. प्रोसेसर समजू शकणाऱ्या सर्व सूचनांचा सेट इन्स्ट्रक्शन सेट म्हणून ओळखला जातो.



चित्र 1.6: वैयक्तिक संगणकाचे कार्यात्मक आकृती

इतर डिजिटल युनिट्सची भूमिका सामान्य डिजिटल संगणकासारखीच आहे. लॅपटॉपसह सध्याचे बहुतेक पीसी मध्ये इंटेल किंवा एएमडी (AMD) मायक्रोप्रोसेसर असतो.

1.1.2 हार्डवेअर (Hardware)

हार्डवेअर म्हणजे आपल्याला जे संगणकाचे भाग डोळ्याने दिसतात आणि स्पर्श करू शकता.

संगणक प्रणालीच्या हार्डवेअर बनविणाऱ्या विविध घटकांमध्ये सिस्टम युनिट (ज्यामध्ये हार्डवेअर घटक जसे की मदरबोर्ड, पॉवर सप्लाय, हार्ड डिस्क.), इनपुट / आउटपुट साधने तसेच स्टोरेज साधने समाविष्ट आहेत.

1.1.3 सॉफ्टवेअर (Software)

सॉफ्टवेअर म्हणजे सूचनांचा किंवा प्रोग्रॅम्स चा एक समूह आहे जो कोणत्याही कार्याला पूर्ण करण्यासाठी सूचना देत असतो.

सॉफ्टवेअर हा डोळ्याने दिसू शकत नाही आणि तुम्ही त्याला स्पर्श सुद्धा करू शकत नाही. सॉफ्टवेअर शिवाय संगणक हा फक्त एक निर्जीव खोके आहे ज्याचा काहीही उपयोग नाही. सॉफ्टवेअर हे फक्त एक आभासी वस्तू आहे त्याला फक्त समजून घेता येते. सोप्या भाषेत पहायचे तर सॉफ्टवेअर म्हणजे लिखित स्वरूपात संगणकाला दिलेल्या सूचना. सॉफ्टवेअर प्रोग्राम्स बायनरी डेटा म्हणून साठवले जातात. जेव्हा ते इन्स्टॉल केले जाते, तेव्हा ते संगणकाच्या हार्ड ड्राईव्हवर कॉपी केले जातात. सॉफ्टवेअर आभासी आहे आणि कोणतीही भौतिक जागा घेत नसल्यामुळे संगणक हार्डवेअरपेक्षा अपग्रेड करणे बरेच सोपे (आणि बऱ्याच वेळा स्वस्त) आहे.

सॉफ्टवेअरचे सर्वात महत्वाचे उदाहरण म्हणजे **ऑपरेटिंग सिस्टम (ओएस)** जी संगणक प्रणालीची संसाधने व्यवस्थापित करते आणि एक इंटरफेस प्रदान करते ज्याद्वारे युझर संगणकाशी विविध कार्ये करण्यासाठी संवाद साधू शकतो.

हे लक्षात ठेवणे आवश्यक आहे की हार्डवेअर आणि सॉफ्टवेअर संगणक प्रणालीचे अविभाज्य भाग आहेत - सॉफ्टवेअरशिवाय हार्डवेअरचा उपयोग होत नाही आणि हार्डवेअरशिवाय सॉफ्टवेअर वापरले जाऊ शकत नाही. सॉफ्टवेअर आहे जे हार्डवेअर चालवते, म्हणजेच, सॉफ्टवेअर हे हार्डवेअरला त्याची क्षमता देते.

सॉफ्टवेअर चे पुढील प्रकारे वर्गीकरण केले जाऊ शकते.

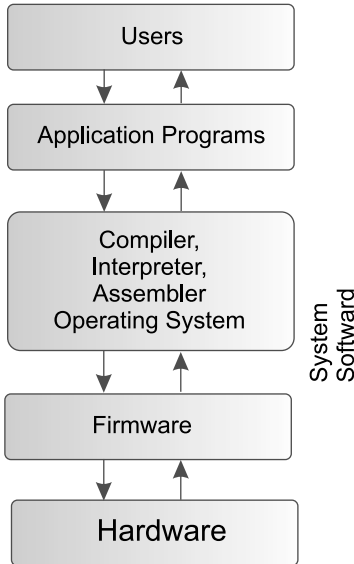
- **सिस्टम सॉफ्टवेअर (System software):** – ह्या एक प्रकारचा सॉफ्टवेअर आहे जो संगणकाचा हार्डवेअर आणि अनुप्रयोग प्रोग्राम चालविण्यासाठी डिझाइन केलेला आहे. जर आपण संगणकाच्या लेयर्ड मॉडेलचा विचार केला तर सिस्टम सॉफ्टवेअर हा हार्डवेअर आणि युझर ॲप्लिकेशन चालविण्यासाठी डिझाइन केलेला आहे ऑपरेटिंग सिस्टम आणि लैंग्वेज ट्रान्सलेटर (language translators)(असेंबलर, कंपाईलर आणि इंटरप्रीटर), लिंकर आणि लोडर सिस्टम सॉफ्टवेअरची उदाहरणे आहेत.

- **अॅप्लिकेशन सॉफ्टवेअर (Application software)** – हा एक प्रकारचा सॉफ्टवेअर आहे जो विशिष्ट हेतूसाठी डिझाइन केलेला आहे आणि एन्ड युझरद्वारे वापरला जातो. वर्ड प्रोसेसर, डेटाबेस सॉफ्टवेअर, स्प्रेडशीट सॉफ्टवेअर, करमणूक सॉफ्टवेअर, आरक्षण सॉफ्टवेअर, इन्व्हेन्टरी मॅनेजमेंट सॉफ्टवेअर इत्यादी अॅप्लिकेशन सॉफ्टवेअरची उदाहरणे आहेत.
- **युटिलिटी सॉफ्टवेअर (Utility software)** – हा सॉफ्टवेअरचा एक प्रकार आहे जो संगणकाचे विश्लेषण, कॉन्फिगरेशन, ऑप्टिमाइझ किंवा देखरेखीसाठी डिझाइन केलेले आहे. याचा उपयोग संगणकाच्या पायाभूत सुविधांना आधार देण्यासाठी केला जातो. अँटीव्हायरस सॉफ्टवेअर, कॉम्प्रेसन टूल्स, डिस्क मॅनेजमेंट टूल्स इ. युटिलिटी सॉफ्टवेअरची उदाहरणे आहेत.

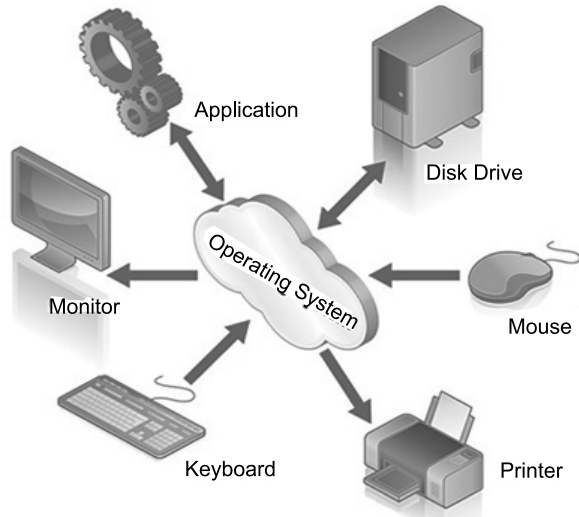
1.1.3.1 ऑपरेटिंग सिस्टम (Operating System)

कोणत्याही संगणकावरील सर्वात महत्वाचा प्रोग्राम म्हणजे ऑपरेटिंग सिस्टम किंवा फक्त ओएस. हा बऱ्याच लहान प्रोग्रामांनी बनलेला एक मोठा प्रोग्राम आहे. तो संगणक प्रणालीचे संसाधन व्यवस्थापक म्हणून कार्य करते आणि युझर व हार्डवेअर दरम्यान इंटरफेस प्रदान करते.

संगणक प्रणालीच्या संसाधनांमध्ये प्रोसेसर, आठवणी आणि I/O डिव्हाइस, युझर इत्यादींचा समावेश आहे. एक व्यवस्थापक म्हणून हे विविध स्त्रोतांच्या क्रियांचे समन्वय करते जेणेकरून युझरला आवश्यक माहिती प्रदान करता येईल.



चित्र 1.7: संगणक प्रणालीचे स्तरित आर्किटेक्चर



चित्र 1.8: ऑपरेटिंग सिस्टम संगणक प्रणालीचे विविध स्रोत व्यवस्थापित करते

पीसी लॅपटॉप्ससाठी सामान्यतः वापरल्या जाणाऱ्या ऑपरेटिंग सिस्टम


- युनिक्स (Unix) 
- लिनक्स (Linux) 
- विन्डोज (Windows) 
- सोलारिस (Solaris) 


- बीओएसएस (भारत ऑपरेटिंग सिस्टम सोल्यूशन्स) (BOSS (Bharat Operating System Solutions))



मोबाइल / हँड हेल्ड डिवाइसेससाठी सामान्यतः वापरल्या जाणाऱ्या ऑपरेटिंग सिस्टम

- अँड्रॉइड (Android) 

- आयओएस (iOS) 

- सिम्बियन (Symbian) 

1.1.3.2 लैंग्वेज ट्रांसलेटर (Language Translators)

सॉफ्टवेअरचा एक घटक जो एका भाषेमध्ये लिहिलेल्या प्रोग्रामचा दुसऱ्या भाषेत अनुवाद करतो तो लैंग्वेज ट्रांसलेटर (लैंग्वेज प्रोसेसर) म्हणून ओळखला जातो.

येथे हे नमूद करणे योग्य आहे की मशीन भाषेमध्ये भाषांतर आवश्यक नसते, परंतु सी सारख्या उच्च-स्तरीय भाषांमध्ये लिहिलेल्या प्रोग्राममध्ये भाषांतरकर्त्यास C भाषेत लिहिलेल्या सूचनांचे कार्यवाही होण्यापूर्वी भाषांतर करणे आवश्यक असते.

C भाषा वापरून लिहिलेला प्रोग्राम सौर्स कोड म्हणून ओळखला जातो. सौर्स कोडची भाषांतरित आवृत्ती मशीन भाषेमधील एक प्रोग्राम आहे, ज्याला ऑब्जेक्ट कोड म्हणून ओळखले जाते.

असेंब्ली भाषेसाठी लैंग्वेज ट्रांसलेटर एसेम्बलर म्हणून ओळखला जातो आणि उच्च-स्तरावरील लैंग्वेज ट्रांसलेटर कंपाईलर आणि इंटरप्रीटर म्हणून ओळखले जातात.

एसेम्बलर (Assembler)

एसेम्बलर एक प्रोग्राम आहे जो असेंब्ली भाषेमधील प्रोग्राम (सौर्स कोड) मशीन भाषेत (ऑब्जेक्ट कोड) भाषांतरित करतो.

एसेम्बलर प्रत्येक मशीन भाषेतील इन्स्ट्रक्शन संबंधित असेंब्ली भाषेमध्ये अनुवादित करतो.

कंपायलर (Compiler)

कंपाईलर एक प्रोग्राम आहे जो उच्च-स्तरीय भाषेत (सौर्स कोड) लिहिलेल्या प्रोग्रामला मशीन भाषेत (ऑब्जेक्ट कोड) भाषांतरित करतो.

भाषांतर करण्याची ही प्रक्रिया कंपायलेशन म्हणून ओळखली जाते. कंपायलेशन दरम्यान, कंपाईलर सौर्स कोड वाचतो. प्रत्येक लाइन आणि वाक्यरचना सिन्टाक्स एरॉरसाठी तपासणी करतो. लक्षात घ्या की प्रोग्रामिंग भाषेच्या नियमांचे कोणतेही उल्लंघन (भाषेचे व्याकरण देखील म्हुटले जाते), **सिन्टाक्स एरॉर** म्हणून ओळखले जाते.

जर तेथे सिन्टाक्स एरॉर असतील तर ती एरॉर स्थाने ओळखणारे एरॉर मेसेज दर्शविते आणि प्रोग्रामची सूची मुद्रित करते जी स्थाने आणि लुटीच्या संभाव्य कारणांवर प्रकाश टाकते आणि कोणतेही भाषांतर होत नाही. तथापि, सौर्स कोडमध्ये वाक्यरचना लुटी नसल्यास कंपाईलर सौर्स कोड पुन्हा वाचतो आणि त्यास ऑब्जेक्ट कोडमध्ये भाषांतरित करतो जे नंतर ती दुसऱ्या फाईलवर लिहितात, त्याला एक्सटेंशन obj (ऑब्जेक्टसाठी) असते.

एकदा भाषांतर झाले की त्यानंतरच्या चरणात कंपाईलरची कोणतीही भूमिका नाही. तथापि, जर सौर्स कोड सुधारित केला असेल तर ऑब्जेक्ट कोडमध्ये बदल समाविष्ट करण्यासाठी आपल्याला ते पुन्हा कंपाईल करणे आवश्यक आहे.

इंटरप्रीटर (Interpreter)

इंटरप्रीटर हा अनुवादकाचा दुसरा प्रकार आहे जो उच्च-स्तरीय भाषांमध्ये अनुवाद करण्यासाठी वापरला जातो. तथापि, इंटरप्रीटर ऑब्जेक्ट कोड तयार करत नाही. त्याऐवजी, ते एका वेळी सौर्स कोडची एक ओळ अनुवादित करतो आणि अनुवादित सूचना अंमलात आणतो. सर्व सूचनांचे भाषांतर आणि कार्यवाही होईपर्यंत हे चालूच राहते. लक्षात ठेवा की कोणतीही सूचना वारंवार कार्यान्वित करायची असल्यास, प्रत्येक वेळी त्याचे भाषांतर केले जाते.

एकदा कंपाईलरद्वारे सौर्स कोडचे ऑब्जेक्ट कोडमध्ये भाषांतर केले जाते, जे एकदाच केले जाते, ऑब्जेक्ट कोड पुढील प्रक्रियेनंतर (लिकिंग) थेट कार्यान्वित (Run) होते. याचा परिणाम प्रोग्रामच्या वेगवान अंमलबजावणीत होतो. तथापि, इंटरप्रीटर प्रथम कोड समजतो आणि नंतर सूचना अंमलात आणतो, ज्यामुळे प्रोग्रामची अंमलबजावणी हळू होते. याव्यतिरिक्त, प्रत्येक वेळी प्रोग्राम इंटरप्रीटर वापरून कार्यान्वित होईल तेव्हा भाषांतर पुन्हा होते.

इंटरप्रीटरमुळे प्रोग्रामचे भाषांतर हळू होते परंतु ते भाषा शिकण्यासाठी नवशिक्यांसाठी आणि प्रोग्राम्स डीबग करण्यासाठी उपयुक्त उपकरण आहे. जेव्हा प्रोग्रॅम लाइनद्वारे लाइन कार्यान्वित करतो तेव्हा प्रोग्रामर प्रत्येक ओळ अचूक कसे काय करते ते पाहू शकतो. येथे हे नमूद करणे महत्वाचे आहे की इंटरप्रीटरद्वारे प्रोग्राम कार्यान्वित करण्यासाठी, प्रथम आपल्याला इंटरप्रीटर रन करावे लागेल, तरच आपण युझर प्रोग्राम रन करण्यासाठी इंटरप्रीटरला सूचना देऊ शकतो. म्हणूनच, प्रत्येक वेळी आपण एखादा प्रोग्राम रन करण्यासाठी आपल्याला इंटरप्रीटर देखील रन लागतो.

1.1.3.3 प्रोग्रामिंग एनव्हायरमेंट (Programming Environment)

प्रोग्रामिंग एनव्हायरमेंट असे एनव्हायरमेंट आहे ज्यात प्रोग्राम तयार केले जातात आणि त्यांची चाचणी केली जाते, आणि ज्या ऑपरेटिंग एनव्हायरमेंटवर प्रोग्राम रन करायची अपेक्षा आहे त्यापेक्षा भाषेच्या रचनेवर त्याचा प्रभाव कमी आहे.

प्रोग्रामिंग एनव्हायरमेंट मध्ये सपोर्ट टूल आणि कमांड लैंग्वेज (कमांडचा संच) असते. प्रत्येक सपोर्ट टूल हा एक प्रोग्राम आहे जो प्रोग्रामरद्वारे प्रोग्राम तयार करण्याच्या एक किंवा अधिक टप्प्यात सहाय्यक म्हणून वापरला जाऊ शकतो.

प्रोग्रामिंग एनव्हायरमेंट मध्ये ठराविक टूल एडिटर, लैंग्वेज ट्रांसलेटर, लिंकर, लोडर्स इ. समाविष्ट असतात. लैंग्वेज ट्रांसलेटरचे आधीच्या विभागात वर्णन केले आहे, उर्वरित टूल्स चे येथे वर्णन केले आहे.

- **एडिटर (Editor)** – एडिटर एक प्रोग्राम आहे जो फायली तयार करण्यात आणि एडिट करण्यात मदत करतो. पुढे असे एडिटर आहेत जे केवळ टेक्स्ट फाइल्ससह कार्य करण्यास परवानगी देतात आणि त्यांना टेक्स्ट एडिटर म्हटले जाते. लोकप्रिय टेक्स्ट एडिटरची उदाहरणे म्हणजे विंडोज ऑपरेटिंग सिस्टमचे नोटपैड आणि एडिट, युनिक्स व लिनक्स ऑपरेटिंग सिस्टमचे Vi. एमएस वर्ड सारख्या लोकप्रिय वर्ड प्रोसेसिंग प्रोग्राममध्ये वर्ड डॉक्युमेंट्स व्यतिरिक्त टेक्स्ट फाईल तयार करण्याची परवानगी मिळते.
- **लिंकर (Linker)** – लक्षात घ्या की कंपाईलरद्वारे तयार केलेला ऑब्जेक्ट कोड संगणकाद्वारे कार्यान्वित केल्या जाणाऱ्या फॉर्ममध्ये नाही. संगणकाद्वारे कार्यान्वित करता येऊ शकेल अशा स्वरूपात ऑब्जेक्ट कोडमध्ये आणखी काही माहिती देणे आवश्यक आहे.

जेव्हा आपण एखादा प्रोग्राम लिहिता तेव्हा आपण इनपुट व आउटपुट ऑपरेशन्स आणि गणिताच्या गणनेसाठी विविध लायब्ररी फंक्शन्सचा संदर्भ घेत असतो (*sqrt, sin, cos, abs, exp* वारंवार वापरले जाणारे मॅथेमॅटिकल लायब्ररी फंक्शन्स). या लायब्ररी फंक्शनद्वारे करण्याच्या कार्याचे निर्देश मशीन लैंग्वेज (बायनरी) स्वरूपात आहेत आणि सिस्टम लायब्ररीत (एक्सटेंशन एलआयबी असलेल्या फाइल्स) कंपाईलरद्वारे पुरविल्या जातात. शिवाय, आपण कदाचित काही युझर-डिफाईन्ड फंक्शन्सचा संदर्भ देत आहात जी वेगळ्या प्रोग्राम फाइलमध्ये लिहिली गेली आहेत आणि ज्याचा ऑब्जेक्ट कोड विचाराधीन असलेल्या ऑब्जेक्ट कोडपेक्षा स्वतंत्र आहे. म्हणून, लायब्ररीचा ऑब्जेक्ट कोड तसेच युझर-डिफाईन्ड ऑब्जेक्ट कोड एकत्र करण्याची आवश्यकता आहे आणि हे कार्य लिंकर नावाच्या प्रोग्रामद्वारे पूर्ण केले गेले जाते.

लिंकर हा एक प्रोग्राम आहे जो प्रोग्रामच्या ऑब्जेक्ट कोडला लायब्ररीच्या ऑब्जेक्ट कोड आणि युझर-डिफाइनड फंक्शन्ससह इच्छित रीतीने एकत्र करतो आणि तो वेगळ्या प्रोग्राम फाईलमध्ये विस्तारासह लिहितो, त्याला एक्सटेंशन सामान्यतः *EXE*. असे असते. लिंकरद्वारे निर्मित ही फाईल ही एक्झिक्युटेबल फाइल किंवा फक्त रन फाईल म्हणून ओळखली जाते, आणि ही ती फाईल आहे जी प्रोग्राम एक्झिक्युट करण्यासाठी वापरली जाते.

- **लोडर (Loader)** – लोडर एक प्रोग्राम आहे जो एक्जीक्युटेबल फाईलला सेकंडरी स्टोरेजमधून मेमरीमध्ये हस्तांतरित करण्यास जबाबदार असतो. एक्झिक्युशन वेळी लोडर, ऑपरेटिंग सिस्टमच्या सहाय्याने प्रथम आवश्यक रकमेची फ्री मेमरी शोधतो आणि नंतर एक्झिक्युटेबल फाइल त्या मेमरीच्या त्या भागामध्ये हस्तांतरित करतो, अँड्रेस ट्रान्सलेशन सारखे योग्य पावले उचलतो आणि नंतर त्याची अंमलबजावणी सुरू करतो. त्या वेळेपासून, लोडरची भूमिका संपली आहे आणि प्रोग्रामद्वारे त्याचे नियंत्रण आपल्या ताब्यात घेण्यात आले आहे. प्रोग्रामची सूचना इच्छित कार्य करण्यासाठी कार्यान्वित केल्या जातात, परंतु ऑपरेटिंग सिस्टमच्या एकूण देखरेखीखाली. प्रोग्रामची अंमलबजावणी पूर्ण होताच, ती मेमरीमधून काढून टाकली जाते. प्रोग्रामची सूचना (Instructions) इच्छित कार्य करण्यासाठी एक्जीक्युट केल्या जातात, परंतु ऑपरेटिंग सिस्टमच्या एकूण देखरेखीखाली. प्रोग्रामची अंमलबजावणी पूर्ण होताच, ती मेमरीमधून काढून टाकली जाते.



Popular Integrated Developments Environments (IDEs), such as Turbo C/C++ and Borland C++, Visual Studio, Dev C++, CodeBlocks, Netbeans, Eclipse, *etc.*, integrate various tools (editor, compiler, linker, debugger, *etc.*) in a unified manner, so that a programmer can work with these tools with much ease and enhanced productivity.

1.1.4 बूट करण्याची संकल्पना (Concept of Booting)

सोप्या भाषेत, बूट करणे ही एक अशी प्रक्रिया आहे ज्यामध्ये आपला संगणक आरंभ होतो. या प्रक्रियेमध्ये आपल्या संगणकामधील आपले सर्व हार्डवेअर घटक आरंभ करणे आणि त्यांना एकत्र कार्य करणे आणि आपला संगणक ऑपरेटिंग बनवणारी आपली डीफॉल्ट ऑपरेटिंग सिस्टम लोड करणे हे समाविष्ट आहे.

बूटिंग प्रक्रियेचे तांत्रिक तपशील खालीलप्रमाणे दिले जाऊ शकतात:

1. संगणक चालू असतो तेव्हा, बूट प्रोग्रामची एक प्रत रॉममधून मुख्य मेमरीमध्ये आणली जाते.
2. त्यानंतर सीपीयू एक जंप इंस्ट्रक्शन चालवते जे बीआयओएस (बेसिक इनपुट आउटपुट सिस्टम) मध्ये हस्तांतरित करते आणि ते कार्यान्वित करण्यास सुरुवात करते.
3. बीओएसचे पीओएसटी (पॉवर ऑन सेल्फ टेस्ट) नावाच्या सेल्फ डायग्नोस्टिक टेस्ट घेते. या चाचण्यांमध्ये मेमरी टेस्ट, व्हिडिओ सर्किटरी कॉन्फिगर करणे आणि प्रारंभ करणे, सिस्टमचे हार्डवेअर कॉन्फिगर करणे आणि संगणकास योग्यरित्या कार्य करण्यास मदत करणारे अन्य डिव्हाइस तपासणे हे समाविष्ट आहे.
4. त्यानंतर, BIOS बूट सेक्टर लोड करण्यासाठी बूट करण्यायोग्य ड्राइव्ह शोधतो. बूट स्ट्रॅप लोडर प्रोग्राम ऑपरेटिंग सिस्टम लोड आणि एक्जीक्युट करतो.

बूट प्रक्रिया दोन प्रकारची आहे:

- **कोल्ड बूटिंग (Cold booting):** जेव्हा सिस्टम प्रारंभिक स्थितीपासून प्रारंभ होते, म्हणजेच ते चालू होते. याला **हार्ड बूटिंग** असेही म्हणतात. वर वर्णन केल्याप्रमाणे सिस्टम बऱ्याच टप्प्यातून जाते.
- **वॉर्म बूटिंग (Warm booting):** सिस्टम रीस्टार्ट करण्यासाठी रीसेट बटण किंवा Ctrl + Alt + Del की संयोजन वापरली जाते. याला **सॉफ्ट बूटिंग** असेही म्हणतात. येथे सिस्टम प्रारंभिक अवस्थेतून प्रारंभ होत नाही आणि म्हणूनच या प्रकारात सेल्फ डायग्नोस्टिक चाचण्या घेतल्या जात नाहीत.

1.2 अल्गोरिदमचा विचार (IDEA OF ALGORITHMS)

अल्गोरिदम एखाद्या विशिष्ट समस्येचे निराकरण करण्याच्या निर्देशांचे एक मर्यादित क्रम आहे. (An *algorithm* is a finite sequence of instructions defining the solution of a particular problem.)

चांगल्या अल्गोरिदमची वैशिष्ट्ये:

अल्गोरिदमची पाच महत्वाची वैशिष्ट्ये आहेत जी समस्येसाठी कोणत्याही अल्गोरिदमची रचना करताना विचारात घ्यावीत.




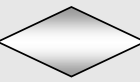
- **इनपुट (Input):** अल्गोरिदममध्ये शून्य किंवा जास्त परंतु इनपुटची मर्यादित संख्या असणे आवश्यक आहे, जे बाह्यरित्या पुरविले जाते.
शून्य इनपुट अल्गोरिदमचे उदाहरण प्रथम 100 नैसर्गिक संख्यांची बेरीज शोधणे असू शकते. येथे, युझर प्रथम 100 नैसर्गिक आकडेवारीची बेरीज शोधण्यासाठी आधीच निर्दिष्ट केलेले असल्याने बाह्य इनपुट पुरवण्याची आवश्यकता नाही. तथापि, वरील समस्या प्रथम n नैसर्गिक संख्येची बेरीज शोधून काढल्यास, युझर n साठी मूल्य दर्शविणारे सिंगल इनपुट प्रदान करणे आवश्यक आहे.
 - **आउटपुट (Output):** अल्गोरिदममध्ये कमीतकमी एक इष्ट परिणाम असणे आवश्यक आहे, म्हणजे, आउटपुट.
 - **डेफिनेटेनेस (Definiteness (No ambiguity)):** प्रत्येक स्टेप स्पष्ट असणे आणि अनेकार्थी नसणे आवश्यक आहे, म्हणजेच, फक्त एक आणि एकच अर्थ असणे.
 - **फायनायटेनेस (Finiteness):** जर आपण अल्गोरिदमच्या स्टेपचा शोध घेतला तर सर्व प्रकरणांसाठी, अल्गोरिदम एका मर्यादित संख्येने स्टेप समाप्त होणे आवश्यक आहे.
 - **इफेक्टिव्हनेस (Effectiveness):** प्रत्येक स्टेप पुरेसे मूलभूत असले पाहिजे जे तत्त्वतः केवळ कागद आणि पेन्सिल वापरून केले जाऊ शकते. याव्यतिरिक्त, प्रत्येक स्टेप केवळ निश्चितच नाही तर ते व्यवहार्य देखील असले पाहिजे.
- फ्लोचार्ट किंवा स्यूडोकोड वापरून अल्गोरिदम दर्शविला जाऊ शकतो.


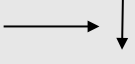
1.2.1 फ्लोचार्ट (Flowchart)

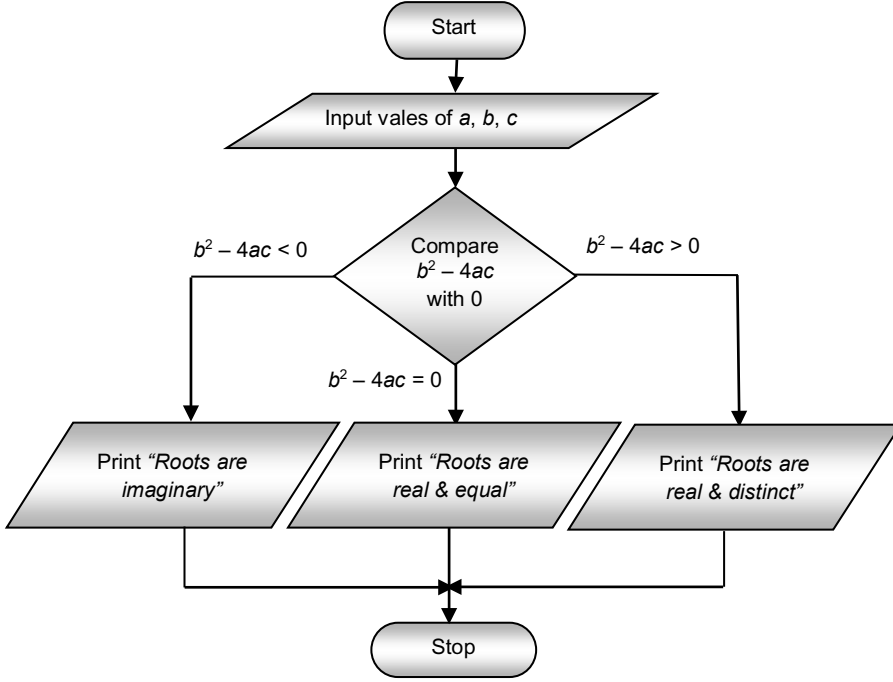
फ्लोचार्ट हे अल्गोरिदमचे चित्रमय प्रतिनिधित्व आहे. वेगवेगळ्या प्रकारच्या सूचना सूचित करण्यासाठी हे वेगवेगळ्या आकारांचा वापर करते. वास्तविक सूचना स्पष्ट आणि संक्षिप्त विधाने वापरून आकारात लिहिलेली असतात. हे आकार निर्देशित रेषांद्वारे जोडले गेले आहेत ज्या क्रमाने निर्देशांची अंमलबजावणी करायची आहे.

Table 1.1 त्यांचे नाव आणि संक्षिप्त वर्णनासह फ्लोचार्टमध्ये वापरलेली विविध चिन्हे दर्शविते.

Table 1.1: विविध फ्लोचार्ट चिन्हे आणि त्यांचे संक्षिप्त वर्णन

Symbol	Name	Purpose
	Oval	<i>Terminal</i> - to mark the beginning and end of the program logic flow.
	Parallelogram	<i>Input/Output</i> - to denote input to the program or output from the program.
	Rectangle	<i>Processing</i> - to denote arithmetic operations and movement of data.
	Diamond	<i>Decision</i> - to denote a point where decision has to be made to branch to one of the alternatives.

	Small circle	Connector - To provide a logical link between segments of a flowchart.
	Directed lines	Flow Lines - To indicate the sequence in which instructions are to be executed.



चित्र 1.9: चतुर्भुज समीकरणाच्या मुळांचे स्वरूप शोधण्यासाठी फ्लोचार्ट

1.2.2 स्यूडोकोड (Pseudocode)

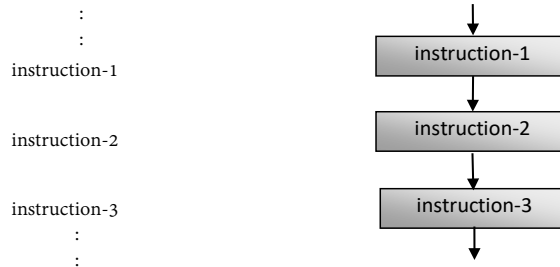
“स्यूडो” शब्दाचा अर्थ अनुकरण करणे किंवा खोटे असा आहे आणि “कोड” या शब्दाचा अर्थ प्रोग्रामिंग भाषेमध्ये लिहिलेल्या निर्देशांचा संदर्भ आहे. स्यूडोकोड, वास्तविक संगणक सूचनांचे अनुकरण आहे. स्यूडो सूचना इंग्रजीमध्ये वाक्यांशांसारखे लिहिलेले शब्द आहेत. प्रोग्रामच्या लॉजिकचे वर्णन करण्यासाठी चिन्हे वापरण्याऐवजी फ्लोचार्ट्स प्रमाणे, स्यूडोकोड एक कॉम्प्यूटर निर्देशांसारखी रचना वापरतात. कारण ते प्रोग्रामच्या डिझाईनवर जोर देते, स्यूडोकोडला **प्रोग्राम डिझाईन लॅंग्वेज** (पीडीएल) देखील म्हणतात.

स्यूडोकोड खालील मूलभूत लॉजिक स्ट्रक्चर्सपासून बनलेला आहे जो कोणत्याही संगणक प्रोग्राम लिहिण्यासाठी पुरेसा सिद्ध झाला आहे:

1. Sequence
2. Selection (*If... Endif, If... Else... Endif, If... Else If... Endif*)
3. Iteration (*While... Endwhile, Do... While*)

एकामागून एक सूचना करण्यासाठी सीक्वेन्स लॉजिकसाठी वापरले जाते. अशा प्रकारे, सीक्वेन्स लॉजिकसाठी, स्यूडोकोड सूचना ज्या क्रमाने कार्यान्वित केल्या जाव्यात त्या क्रमाने लिहिलेल्या आहेत.

लॉजिकचा प्रवाह (Flow) वरपासून खालपर्यंत आहे.

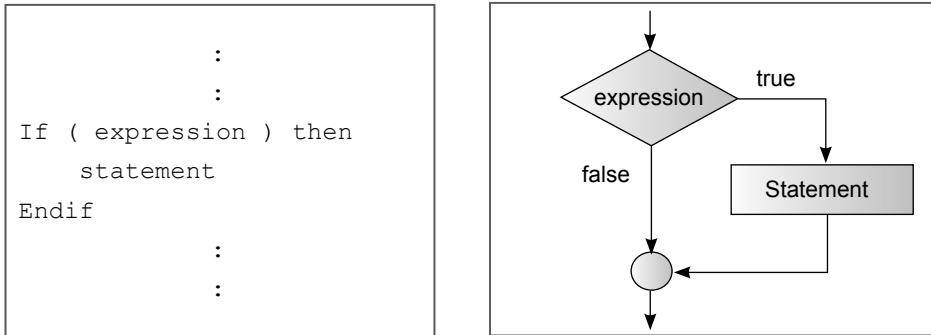


चित्र 1.10: सीक्वेन्स स्ट्रक्चरसाठी सूडोकोड आणि फ्लोचार्ट

Selection logic याला *decision logic* देखील म्हणतात, याचा उपयोग निर्णय घेण्यासाठी केला जातो. प्रोग्राम लॉजिकमधील पर्यायी मार्गांपैकी योग्य पर्याय निवडण्यासाठी याचा उपयोग केला जातो.

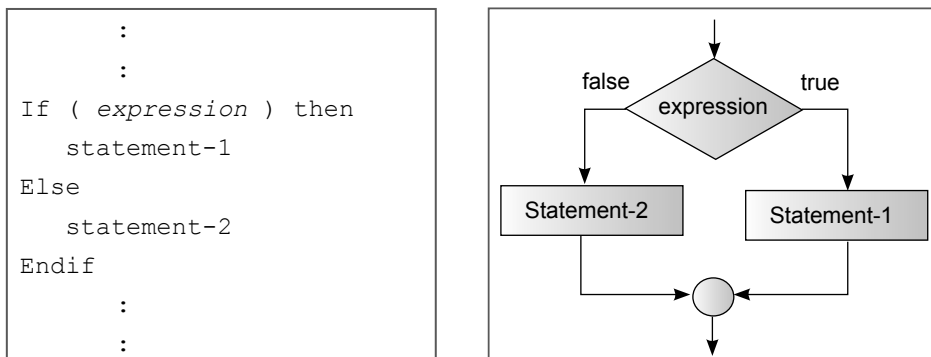
Decision logic एकतर *If... Endif* किंवा *If... Else... Endif* किंवा *If... Else If... Endif* स्ट्रक्चर.

If... Endif कन्स्ट्रक्ट सांगते की जर एक्सप्रेशन बरोबर असेल तर स्टेटमेंट एक्जीक्यूट करा अन्यथा (एक्सप्रेशन चुकीचे असल्यास) स्टेटमेंट सोडून द्या.



चित्र 1.11: सूडोकोड आणि फ्लोचार्ट *If... Endif* selection structure च्या साठी

If... Else... Endif कन्स्ट्रक्ट सांगते की जर एक्सप्रेशन बरोबर असेल तर स्टेटमेंट-1 एक्जीक्यूट करा अन्यथा (एक्सप्रेशन चुकीचे असल्यास) एक्जीक्यूट स्टेटमेंट-2.



चित्र 1.12: सूडोकोड आणि फ्लोचार्ट *If... Else... Endif* selection structure च्या साठी

जर तेथे अनेक पर्याय (एक्झिक्यूशन पाथ) असतील तर, *Else If ladder* वापरणे खूप सुलभ आहे.

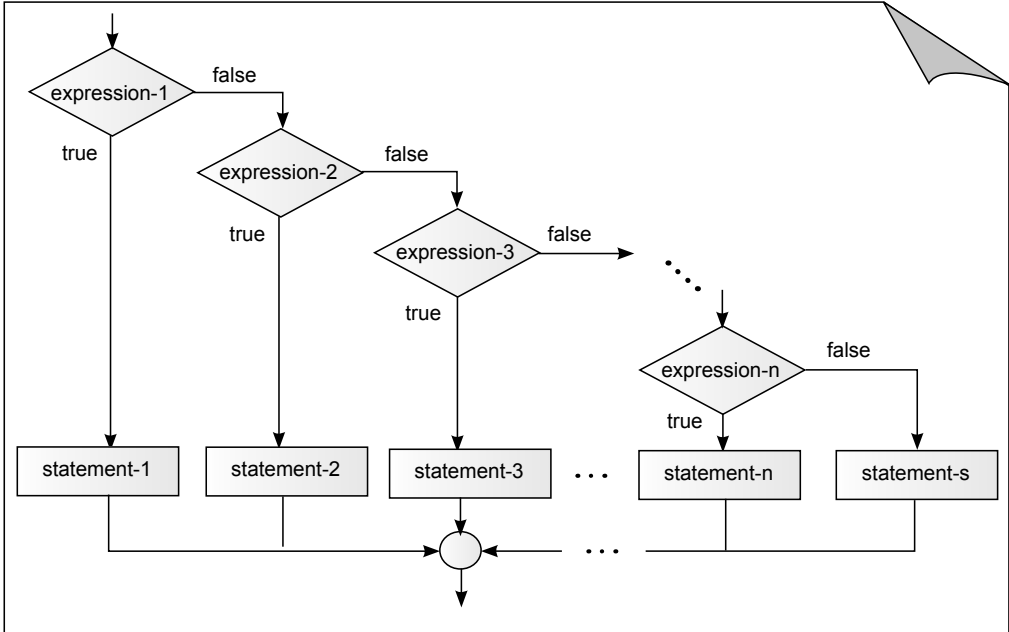
```

If ( expression-1 ) then
    statement-1
Else If ( expression-2 ) then
    statement-2
Else If ( expression-3 ) then
    statement-3
    :
Else If ( expression-n ) then
    statement-n
Else
    statement-s
Endif

```

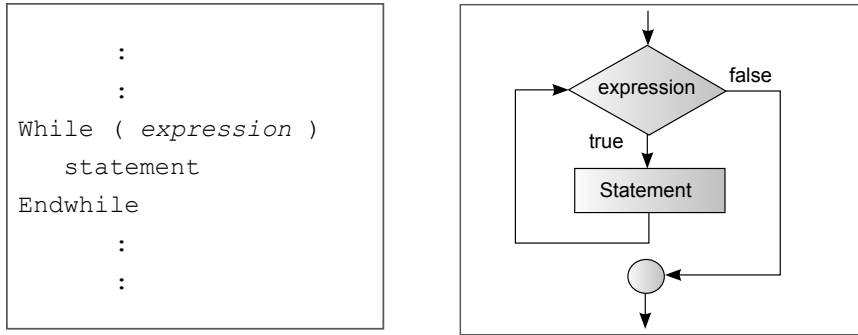
चित्र 1.13: एल्स-इफ लाडर सिंटॅक्स

क्रमाने एक्स्प्रेसनचे मूल्यांकन केले जाते, आणि जर कोणतीही एक्स्प्रेसन सत्य असेल तर संबंधित स्टेटमेंट ब्लॉक एक्जीक्यूट होईल, आणि संपूर्ण साखळी तिथे संपते. शेवटचा एल्स पार्ट वरीलपैकी कोणतेही नाही किंवा डीफॉल्ट एक्स्प्रेसन हाताळते.

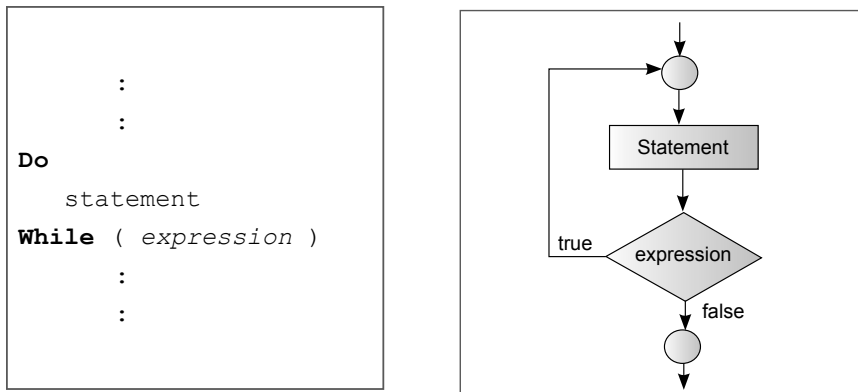


चित्र 1.14: एल्स-इफ लाडर चा लॉजिक फ्लो

जेव्हा काही एक्स्प्रेसनवर अवलंबून एक किंवा अधिक सूचना अंमलात आणल्या जातात तेव्हा लूप तयार करण्यासाठी इटरेटिव्ह लॉजिकचा वापर केला जातो. *While... Endwhile* and *Do... While* ह्या दोन रचनांचा वापर केला जातो.



चित्र 1.15: स्यूडोकोड आणि फ्लोचार्ट While... Endwhile iterative structure च्या साठी



चित्र 1.16: स्यूडोकोड आणि फ्लोचार्ट Do... While iterative structure च्या साठी

Do... While लूपच्या शेवटी एक्सप्रेशन चाचणी घेतली जाते तेव्हा लूप कमीतकमी एकदा एक्जीक्यूट केला जातो. *While... Endwhile* लूपच्या सुरवातीला एक्सप्रेशन चाचणी घेतल्या मुळे लूप एकदाही एक्जीक्यूट होणार नाही, हा या दोन्ही मधील फरक आहे.

स्यूडोकोड वर्णन (Pseudocode Description)

कमेंट्स (Comments)

प्रत्येक सूचनेनंतर एक कमेंट्स येऊ शकते. कमेंट्स डबल स्लॅशसह प्रारंभ होतात आणि अशा सूचनांचे हेतू स्पष्ट करतात, जसे की

Read: n // Enter the value of variable n

कमेंट्सचा योग्य वापर केल्यास स्यूडोकोडची वाचनीयता वाढते, यामुळे स्यूडोकोड टिकवून ठेवण्यास मदत होते.

व्हेरिएबलची नावे (Variable Names)

व्हेरिएबलच्या नावांसाठी आपण इटालिसाईज्ड लोअरकेस अक्षरे जसे की *max*, *loc*, इत्यादी वापरू शकता, तर डिफिनाईज्ड कॉन्स्टन्टसाठी आपण अपरकेस अक्षरे वापरू शकता.

असाइनमेंट स्टेटमेंट (Assignment Statement)

असाइनमेंट स्टेटमेंट नोटेशन म्हणून वापरतात

Set $max = a$

max या व्हेरिएबल ला a हे मूल्य असाईन झालेण् असाईनमेंट स्टेटमेंटच्या उजव्या बाजूला मूल्यए व्हेरिएबल किंवा एक्सप्रेशन असू शकते.

तथापि, जर अनेक असाइनमेंट स्टेटमेंट्स समान ओळीत दिसली, जसे की

Set $k = 1, loc = 1, max = a_1$

मग त्यांना डावीकडून उजवीकडे एकजीक्यूट केले जाईल.

इनपुट/आउटपुट (Input/Output)

डेटा इनपुट असू शकतो आणि खालील स्वरूपासह रीड स्टेटमेंट द्वारे व्हेरिएबल्सला असाइन केला जाऊ शकतो

Read: *Variable list*

जिथे Variable list मध्ये स्वल्पविरामांनी विभक्त केलेले एक किंवा अधिक व्हेरिएबल्स असतात.

त्याचप्रमाणे व्हेरिएबल्स आणि मेसेजेस मधील डेटा, असल्यास, तो डबल कोट्स मध्ये बंद केलेला प्रिंट स्टेटमेंटद्वारे खालील स्वरूपासह आउटपुट करू शकतो.

Print: *message and/or Variable list*

जिथे Variable list मधील मेसेज आणि व्हेरिएबल्स स्वल्पविरामाने विभक्त केले जातात.

इन्स्ट्रक्शन एक्झिक्यूशन (Execution of Instructions)

स्यूडोकोडमध्ये इन्स्ट्रक्शन एकामागून एक एकजीक्यूट केल्या जातात. तथापि, अशी उदाहरणे असू शकतात जेव्हा काही इन्स्ट्रक्शन वगळल्या गेल्या किंवा काही एक्सप्रेशनच्या परिणामी काही इन्स्ट्रक्शन पुनरावृत्ती केल्या जाऊ शकतात.

अल्गोरिदम कंपायलेशन (Completion of the Algorithm)

शेवटची इन्स्ट्रक्शन एकजीक्यूट झाल्यावर सूडोकोड पूर्ण होते. तथापि, exit इन्स्ट्रक्शन वापरून कोणत्याही दरम्यानच्या स्थितीत सूडोकोड हे संपुष्टात आणले जाऊ शकते.

The nature of roots of a quadratic equation प्रदर्शित करण्यासाठी सूडोकोड

$$ax^2 + bx + c = 0$$

provided $a \neq 0$.

Pseudocode 1.1

Begin

Read: a, b, c

Set $disc = b^2 - 4ac$

If ($disc = 0$) **then**

Print: "Roots are real and equal"

Else If ($disc > 0$) **then**

Print: "Roots are real and distinct"

Else

Print: "Roots are imaginary"

Endif

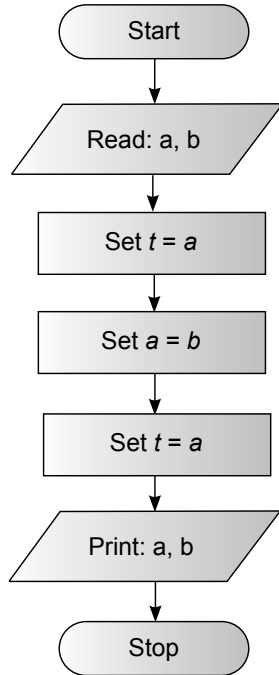
End.

स्पष्टीकरणात्मक उदाहरणे

Example 1.1: फ्लोचार्ट काढा आणि जे दोन व्हेरिएबल्स ए आणि बी स्वॅप (इंटरचेंज) करतील असा एक स्यूडोकोड लिहा. (Draw a flowchart and write a pseudocode to swap (interchange) two variables say a and b .)

Solution: परिस्थितीचा विचार करा - आपल्याकडे एका ग्लासमध्ये पाणी आणि दुसऱ्या ग्लासमध्ये रस आहे. ज्या ग्लासमध्ये रस आहे त्यात आपल्याला पाणी पाहिजे आहे. आणि ज्यात पाणी आहे त्यात रस पाहिजे आहे. हे कार्य कसे पूर्ण केले जाऊ शकते?

अशाच प्रकारे, पुढील फ्लोचार्ट आणि स्यूडोकोड मध्ये दर्शविल्या प्रमाणे दोन व्हेरिएबल्सच्या व्हॅल्यूज अदलाबदल करण्यासाठी आपल्याला तिसरा व्हेरिएबल t वापरावा लागेल.



Pseudocode 1.2

Begin

Read: a, b

Set $t = a$

Set $a = b$

Set $b = t$

Print: a, b

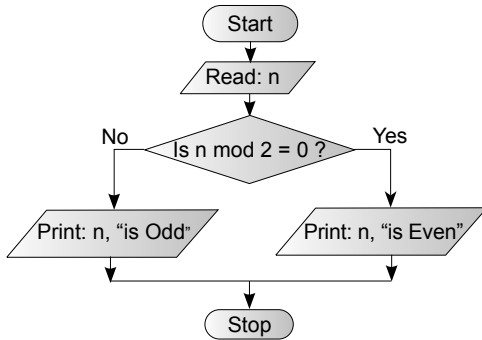
End.

चित्र 1.17: दोन व्हेरिएबल्स बदलण्यासाठी फ्लोचार्ट आणि स्यूडोकोड

Example 1.2: दिलेली नैसर्गिक संख्या " n " सम की विषम आहे की नाही हे तपासण्यासाठी फ्लोचार्ट काढा आणि स्यूडोकोड लिहा. (Draw a flowchart and write a pseudocode to test whether a given natural number ' n ' is even or odd.)

Solution: आपल्या सर्वांना हे ठाऊक असेल की कोणतीही नैसर्गिक संख्या ही 2 ने अगदी विभाजित होते उदा. 2 ने भाग केल्याने बाकीचे 0 उरते. बाकी मिळविण्याच्या ऑपरेशनला मॉड्युलो (मोड इन शॉर्ट) ऑपरेशन असे म्हणतात.

खालील फ्लोचार्ट आणि स्यूडोकोड लॉजिक प्रदर्शित करतात.



चित्र 1.18: फ्लोचार्ट आणि प्स्यूडोकोड दिलेली संख्या सम किंवा विषम आहे च्या चाचणीसाठी

Pseudocode 1.3

Begin

Read: n

If (n mod 2 = 0) **then**

Print: n, "is Even"

Else

Print: n, "is Odd"

Endif

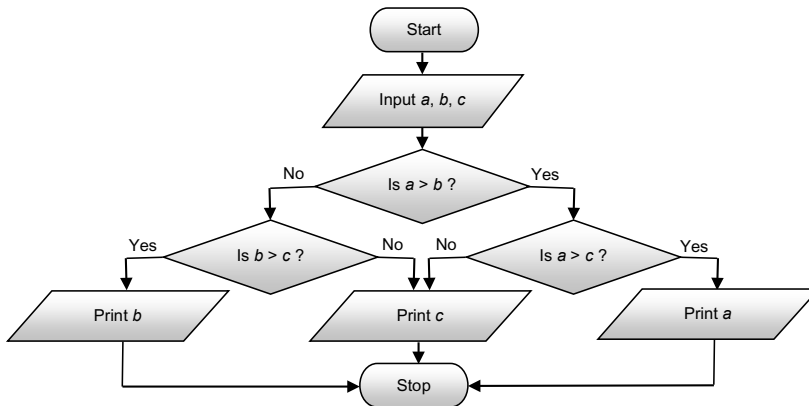
End.

Example 1.3: तीन नंबर ए, बी, सी म्हणा, यात सर्वात मोठा नंबर शोधण्यासाठी फ्लोचार्ट काढा आणि प्स्यूडोकोड लिहा. (Draw a flowchart and write a pseudocode to find the largest of three numbers, say a , b , c .)

Solution: आपण प्रथम b बरोबर a ची तुलना करू. जर a हा b पेक्षा मोठा असेल तर आपण c ची तुलना करू. जर a हा c पेक्षा मोठा असेल तर a ही सर्वात मोठी संख्या म्हणून घेतली जाईल अन्यथा आपण c ही सर्वात मोठी संख्या म्हणून घेतो.

तथापि, a हा b पेक्षा मोठे नसल्यास आपण b ची तुलना c बरोबर करू. जर b हा c पेक्षा मोठा असेल तर b ही सर्वात मोठी संख्या म्हणून घेतला जाईल अन्यथा आपण c ही सर्वात मोठी संख्या म्हणून घेतो.

खालील फ्लोचार्ट आणि प्स्यूडोकोड लॉजिक प्रदर्शित करतात.



चित्र 1.19: तीनपैकी सर्वात मोठी संख्या शोधण्यासाठी फ्लोचार्ट आणि प्स्यूडोकोड

Pseudocode 1.4

Begin

Read: a , b , c

If ($a > b$)

If ($a > c$) **then**

Print: a

Else

Print: c

Endif

```

Else
    If ( b > c ) then
        Print: b
    Else
        Print: c
    Endif
Endif
End.

```

Example 1.4: एखाद्या विषयातील गुणांच्या टक्केवारीच्या आधारे खालील परीक्षा धोरणानुसार विद्यार्थ्यास लेटर ग्रेड नियुक्त केला जातो: (Based on the percentage of marks in a subject, letter grade are assigned to a student as per the following examination policy:)

Solution:

Percentage of Marks	Grade
percentage \geq 90	A+
90 > percentage \geq 80	A
80 > percentage \geq 70	B
70 > percentage \geq 60	C
60 > percentage \geq 50	D
percentage < 50	F

ज्या विद्यार्थ्यास विषयातील गुणांची टक्केवारी दिली जाते अशा विद्यार्थ्याला लेटर ग्रेड नियुक्त करण्यासाठी प्युडोकोड लिहा.

Pseudocode 1.5

```

Begin
    Read: percentage
    If ( percentage >= 90 )
        Print: "Grade = A+"
    Else If ( percentage >= 80 )
        Print: "Grade = A"
    Else If ( percentage >= 70 )
        Print: "Grade = B"
    Else If ( percentage >= 60 )
        Print: "Grade = C"
    Else If ( percentage >= 50 )
        Print: "Grade = D"
    Else
        Print: "Grade = F"
    Endif
End.

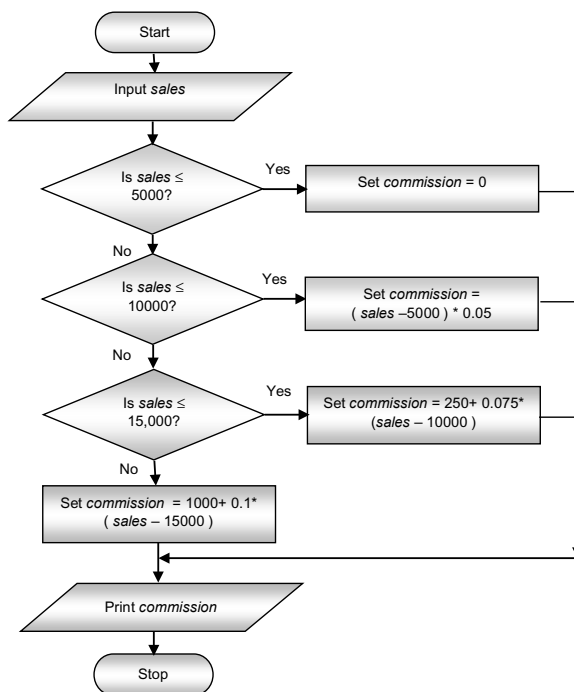
```

Example 1.5: विक्रेत्या विक्रीवरील कमिशनची गणना खालील पॉलिसीनुसार केली जाते (*Commission on sales by a salesman is calculated as per following policy:*)

Amount of Sale (in ₹)	Commission Rate
5000 – 0	Nil
10000 – 5001	5 % excess of 5000
15000 – 10001	7.5 % excess of 10000
> 15000	10 % excess of 15000

फ्लोचार्ट काढा आणि सूडोकोड लिहा जो सेल्समनने केलेला विक्री स्वीकारतो आणि कमिशन दाखवतो.

Solution:



चित्र 1.20: कमिशनची गणना करण्यासाठी फ्लोचार्ट

Pseudocode 1.6

Begin

Read: sales

If (sales <= 5000)

Set commission = 0

Else If (sales <= 10000)

Set commission = (sales - 5000) * 0.05;

Else If (sales <= 1500)

Set commission = 250 + (sales - 10000) * 0.075;

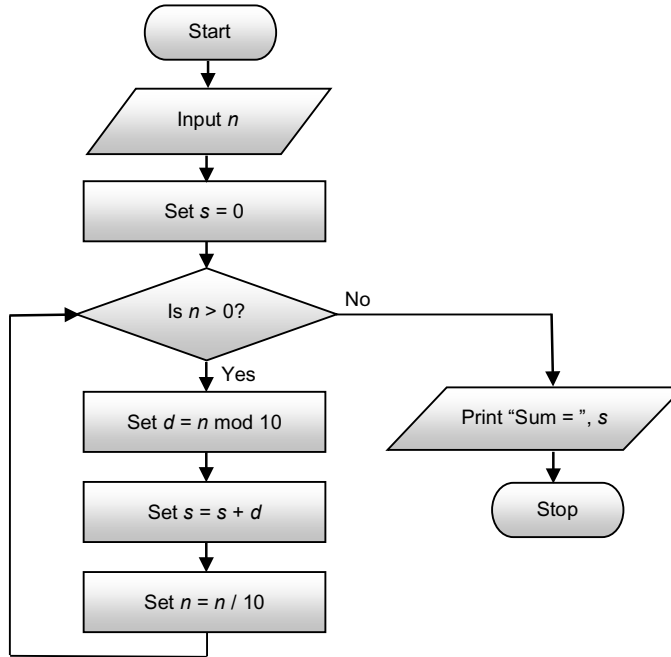

```

Else
    Set commission = 1000 + ( sales - 15000 ) * 0.1;
Endif
Print: "Computed commission = ", commission
End.

```

Example 1.6: संख्या n च्या अंकांची बेरीज शोधण्यासाठी फ्लोचार्ट काढा आणि पसूडोकोड लिहा. (Draw a flowchart and write a pseudocode to find the sum of digits of a number n .)

Solution:



चित्र 1.21: संख्यांच्या अंकांची बेरीज शोधण्यासाठी फ्लोचार्ट आणि पसूडोकोड

Pseudocode 1.7

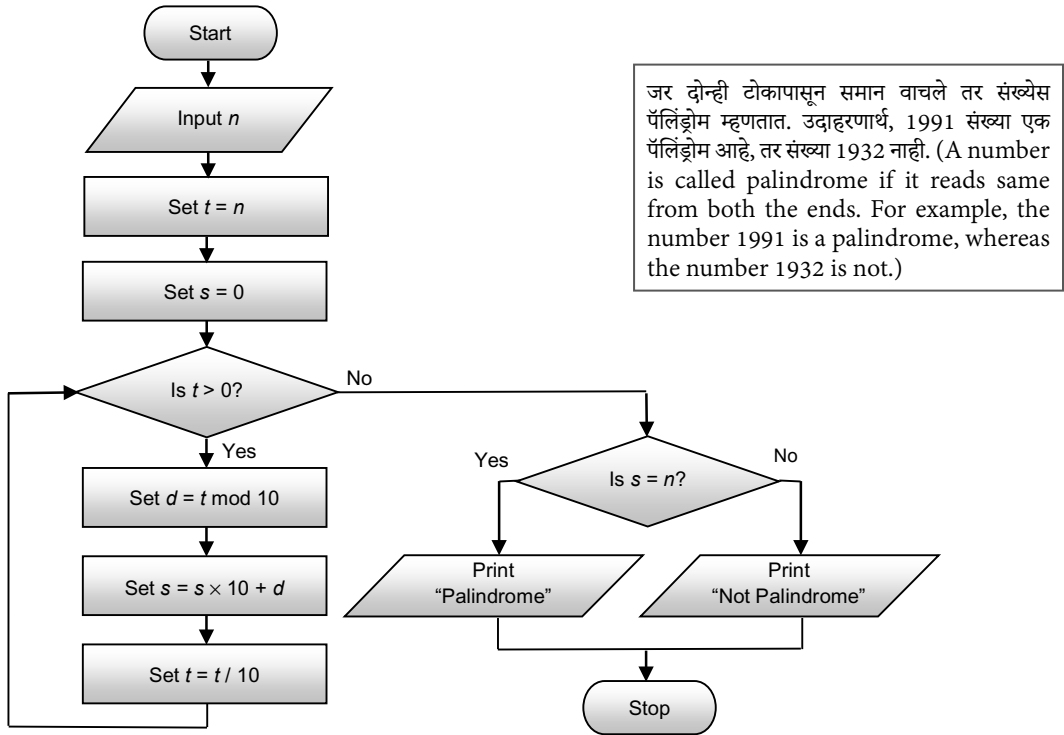
```

Begin
    Read: n
    Set s = 0
    While ( n > 0 ) do
        Set d = n mod 10
        Set s = s + d
        Set n = n / 10
    Endwhile
    Print: "Sum = ", s
End.

```

Example 1.7: दिलेला नंबर n हा पॅलिंड्रोम आहे की नाही हे तपासण्यासाठी फ्लोचार्ट काढा आणि प्स्यूडोकोड लिहा. (Draw a flowchart and write pseudocode to check whether the given number n is palindrome or not.)

Solution:



चित्र 1.22: फ्लोचार्ट आणि प्स्यूडोकोड दिलेली संख्या एन पॅलिंड्रोम आहे की नाही हे तपासण्यासाठी

Pseudocode 1.8

Begin

Read: n

Set $t = n, s = 0$

While ($t > 0$) **do**

Set $d = t \bmod 10$

Set $s = s \times 10 + d$

Set $t = t / 10$

Endwhile

If ($s = n$) **then**

Print: "Palindrome"

Else

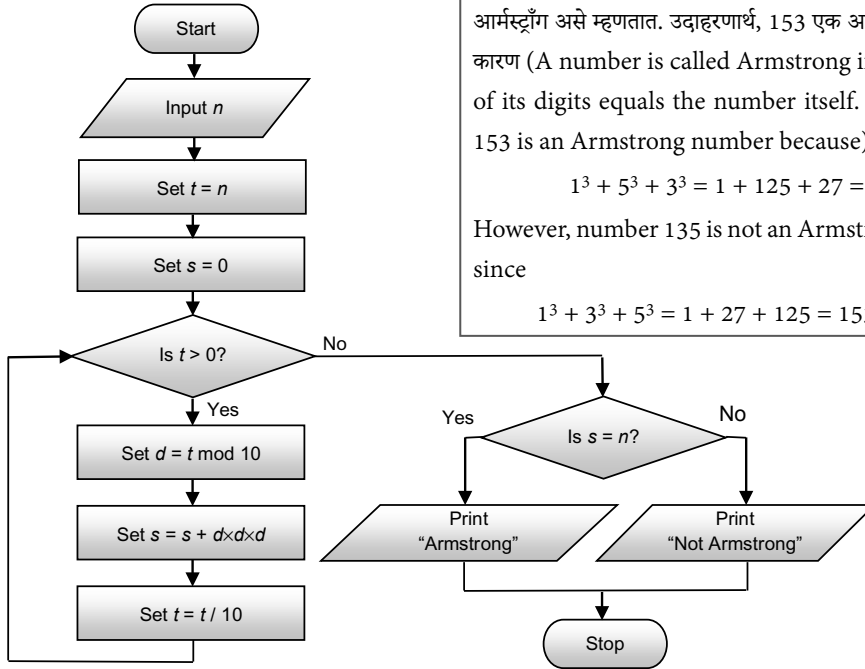
Print: "Not a Palindrome"

Endif

End.

Example 1.8: दिलेला नंबर n हा आर्मस्ट्रॉंग नंबर आहे की नाही याची तपासणी करण्यासाठी फ्लोचार्ट काढा आणि पसूडोकोड लिहा. (Draw a flowchart and write a pseudocode to check whether the given number n is an Armstrong number or not.)

Solution:



जर त्याच्या अंकांच्या घनची बेरीज ही संख्येइतकी असेल तर त्यास आर्मस्ट्रॉंग असे म्हणतात. उदाहरणार्थ, 153 एक आर्मस्ट्रॉंग नंबर आहे कारण (A number is called Armstrong if sum of cube of its digits equals the number itself. For example, 153 is an Armstrong number because)

$$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$$

However, number 135 is not an Armstrong number, since

$$1^3 + 3^3 + 5^3 = 1 + 27 + 125 = 153 \neq 135$$

चित्र 1.23: दिलेली संख्या एन आर्मस्ट्रॉंग नंबर आहे की नाही हे तपासण्यासाठी फ्लोचार्ट आणि पसूडोकोड.

Pseudocode 1.9

Begin

Read: n

Set $t = n, s = 0$

While ($t > 0$) **do**

Set $d = t \text{ mod } 10$

Set $s = s + d \times d \times d$

Set $t = t / 10$

Endwhile

If ($s = n$) **then**

Print: "Armstrong"

Else

Print: "Not Armstrong"

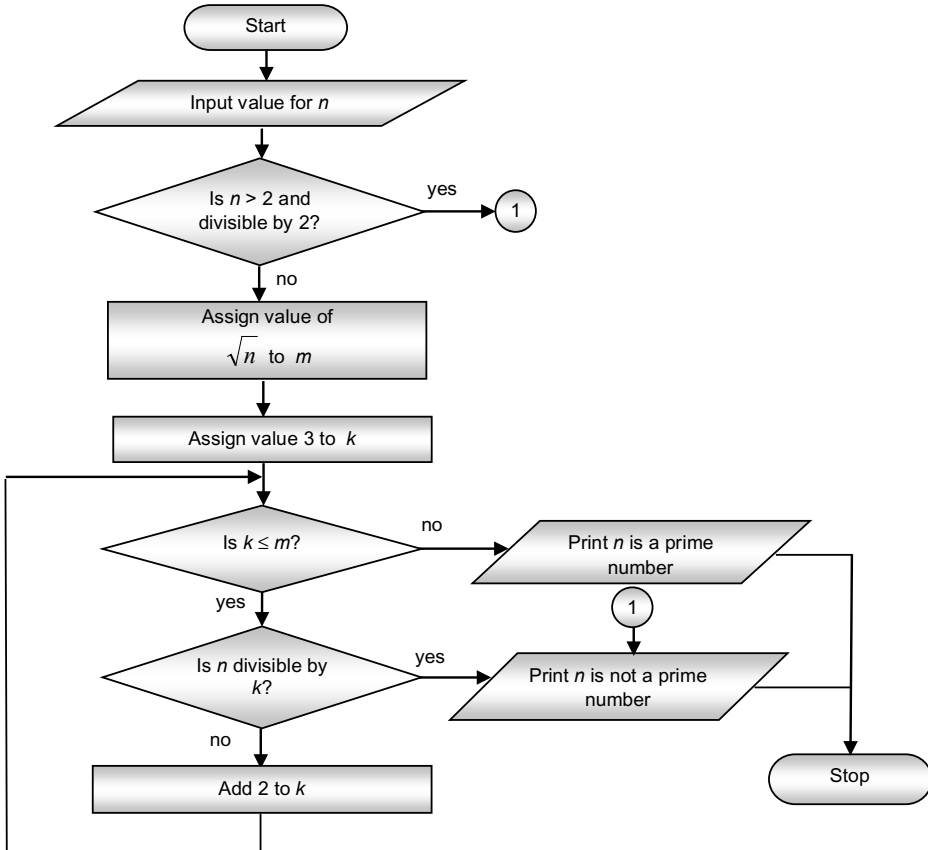
Endif

End.

Example 1.9: दिलेली नैसर्गिक संख्या n ही एक प्राइम नंबर आहे की नाही हे शोधण्यासाठी फ्लोचार्ट काढा. (Draw a flowchart to find whether the given natural number n is a prime number or not.)

Solution: नैसर्गिक संख्या प्राइम नंबर म्हटले जाते जर ती केवळ 1 ने आणि फक्त स्वतः ने विभाजित केले असेल म्हणजेच त्याचे घटक बनवता येत नाहीत. याव्यतिरिक्त, या व्याख्येनुसार, 2 वगळता सम संख्या ही प्राइम नंबर नाही. म्हणून, आमच्या चाचणी निकष

1. जर n हा 2 पेक्षा मोठा असेल आणि सम असेल तर n ही प्राइम नंबर नाही. (If n is greater than 2 and is even then n is not a prime number.)
2. पायरी 1 मधील चाचणी अयशस्वी झाल्यास आम्ही $k = 3, 5, 7, \dots \sqrt{n}$. याद्वारे घटक n विभाजित करण्याचा प्रयत्न करतो. म्हणून, जर n कोणत्याही k मूल्यांनी विभाज्य असेल तर संख्या n ही प्राइम नंबर नाही. (If test at step 1 fails, then we try to divide number n by factors $k = 3, 5, 7, \dots \sqrt{n}$. Therefore, if n is divisible by any value of k , number n is not a prime number.)
3. जर पायरी २ वर चाचणी देखील अपयशी ठरली तर n ही प्राइम नंबर आहे. (If test at step 2 also fails, then n is a prime number.)



चित्र 1.24: संख्या n प्राइम आहे की नाही हे तपासण्यासाठी फ्लोचार्ट

दिलेली सकारात्मक संख्या n ही एक प्राइम नंबर आहे की नाही हे शोधण्यासाठी खालील प्युडोकोड आहे.

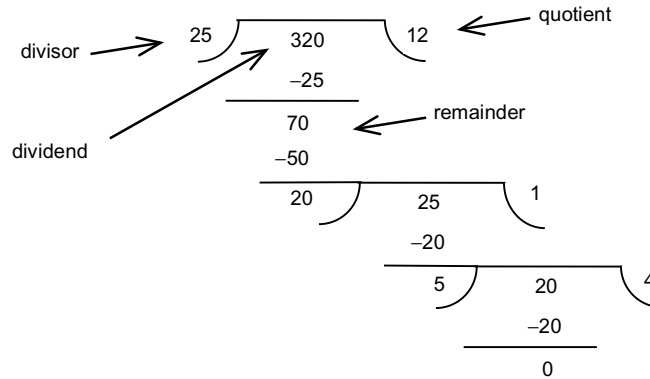
Pseudocode 1.10

```

Begin
  Read:  $n$ 
  If (  $n > 2$  and  $n \bmod 2 == 0$  ) then
    Print:  $n$ , " is not a prime number"
    Exit
  Else
    Set  $m = \sqrt{n}$ 
    For  $k = 3$  to  $m$  by 2 do
      If (  $n \bmod k == 0$  ) then
        Print:  $n$ , " is not a prime number"
        Exit
      Endif
    Endfor
    Print:  $n$ , " is a prime number"
  Endif
End.
    
```

Example 1.10: m आणि n या दोन नैसर्गिक संख्यांपैकी सर्वाधिक सामान्य घटक (एचसीएफ) / ज्यांना ग्रेअटेस्ट कॉमनल डिव्हिजर (जीसीडी) म्हणून देखील ओळखले जाते. (To find highest common factor (HCF)/, also known as greatest common divisor (GCD), of two natural numbers m and n .)

Solution:

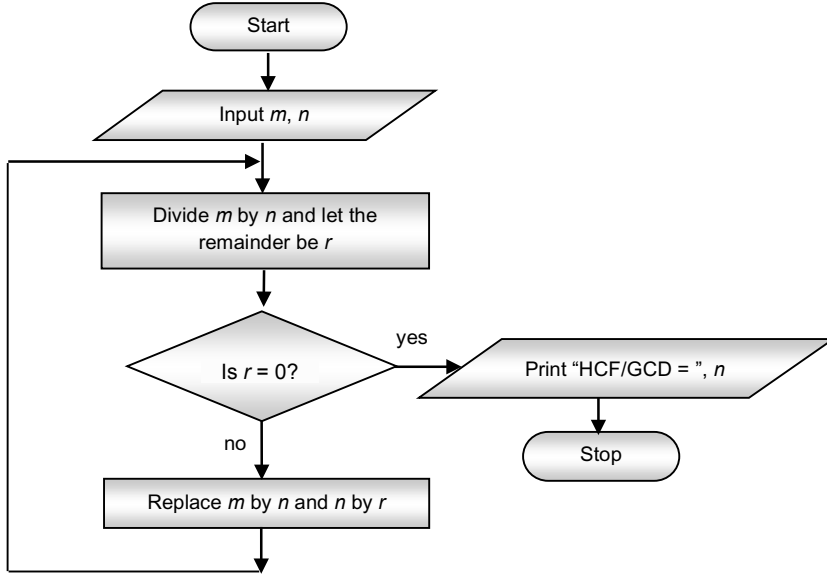


चित्र 1.25: HCF/GCD साठी संगणकीय प्रक्रियेचे उदाहरण

चित्र 1.25: दोन नैसर्गिक संख्येचे HCF/GCD शोधण्यासाठी long/continued विभागणी पद्धत दर्शवते. आपण हे लक्षात घेतले असेल की सलग विभागांमध्ये मागील विभागातील विभाजक लाभांश होतो, बाकी विभाजक होतो आणि विभागणी पुन्हा केली जाते. बाकी शून्य होईपर्यंत ही प्रक्रिया सुरू ठेवली जाते आणि वर्तमान विभाजक दिलेल्या नैसर्गिक संख्येचे HCF/GCD घेतले जाते.

पुढील पायऱ्याचा वापर करून ही प्रक्रिया राबविली जाऊ शकते

1. भागाकार करा (Perform division)
2. जर बाकी शून्य असेल तर थांबा आणि विभक्त HCF/GCD म्हणून घ्या. (If remainder is zero, then stop and take the divisor as HCF/GCD.)
3. दुभाजकाद्वारे लाभांश पुनर्स्थित करा. (Replace dividend by divisor.)
4. बाकी विभाजक द्वारे पुनर्स्थित करा. (Replace divisor by remainder.)
5. पायरी 1 पासून पुनरावृत्ती करा (Repeat from step 1.)



चित्र 1.26: दोन संख्यांच्या HCF ची गणना करण्यासाठी फ्लोचार्ट आणि पसूडोकोड

Pseudocode 1.11**Begin****Read:** m, n **Set** $r = m \bmod n$ **While** ($r \neq 0$) **do****Set** $m = n$ **Set** $n = r$ **Set** $r = m \bmod n$ **Endwhile****Print:** "HCF/GCD = ", n **End.**

Example 1.11: Fibonacci sequence च्या प्रथम n टर्म्स प्रिंट करण्यासाठी फ्लोचार्ट काढा आणि पसूडोकोड लिहा (Draw a flowchart and write a pseudocode to print first n terms of the Fibonacci sequence.)

उदाहरणार्थ, n साठी इनपुट मूल्य 8 असल्यास आउटपुट हे असावे

0 1 1 2 3 5 8 13

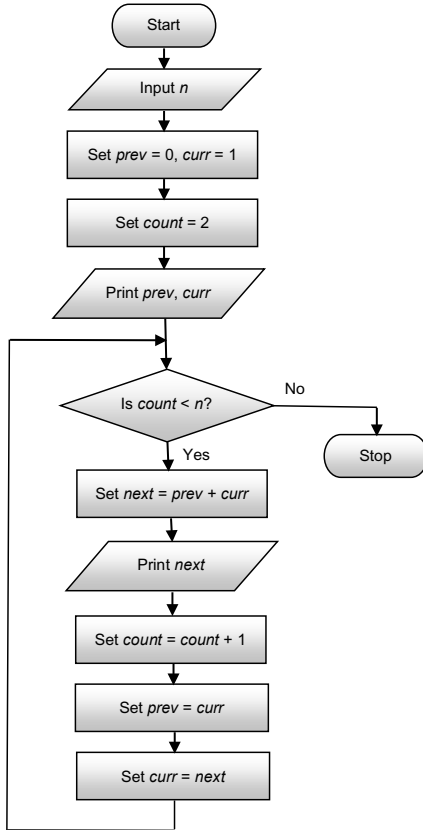
Solution: पहिल्या दोन टर्म्स सोडल्यास प्रत्येक पद हे ताबडतोब आधीच्या दोन पदांच्या बेरीजच्या रूपात प्राप्त होतो.

If we use variable *prev* for previous term, *curr* for current term, *next* for next term, and setting *prev* and *curr* to values 0 and 1, respectively, i.e., first two terms of the sequence, then the entire sequence can be generated by using the recurrence relation

$$\text{next} = \text{prev} + \text{curr}$$

replace *prev* by *curr*

replace *curr* by *next*



Pseudocode 1.12

Begin

Read: n

Set prev = 0, curr = 1

Set count = 2

Print: prev, curr

While (count < n) do

Set next = prev + curr

Print: next

Set count = count + 1

Set prev = curr

Set curr = next

Endwhile

End.

चित्र 1.27: Fibonacci अनुक्रमाच्या प्रथम n टर्म्स प्रिंटसाठी फ्लोचार्ट आणि पसुडोकोड

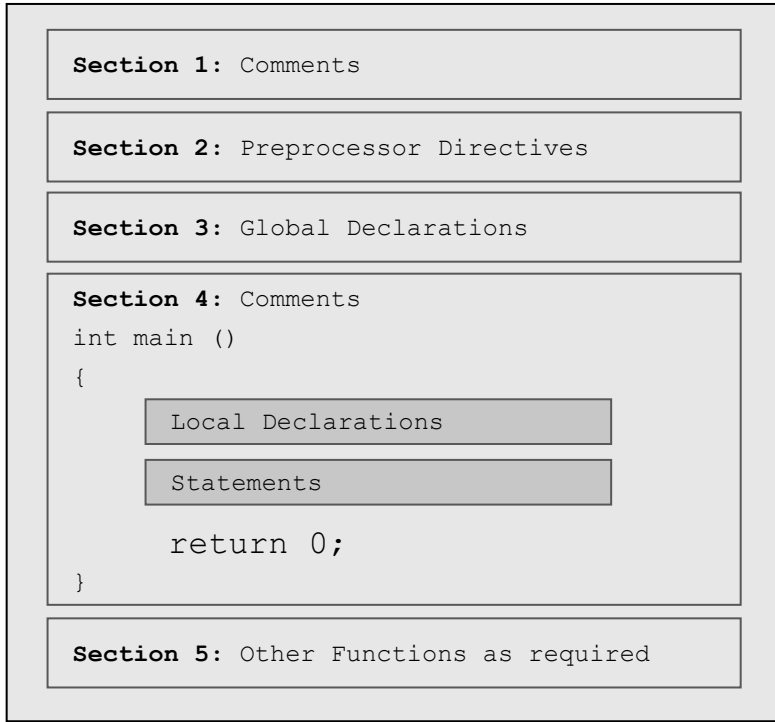
1.3 अल्गोरिदम पासून प्रोग्रॅम पर्यंत (FROM ALGORITHMS TO PROGRAM)

आत्तापर्यंत आपण शिकलात की अल्गोरिदम हा संगणक वापरून दिलेल्या समस्यांचे निराकरण करण्याचा एक मार्ग आहे. अल्गोरिदममध्ये दिलेल्या समस्येचे निराकरण करण्यासाठी लॉजिक असते. प्रोग्रामिंग भाषा वापरून हे लॉजिक प्रोग्राममध्ये रूपांतरित करणे आवश्यक आहे. या कोर्स साठी C भाषा ही आमची प्रोग्रामिंग भाषा आहे.

या विभागात आपण C भाषेची मूलभूत बाबी शिकू जी आपल्याला साध्या अल्गोरिदम चे C प्रोग्राममध्ये रूपांतरित करण्यास सक्षम करेल.

1.3.1 प्रोग्रामची रचना (Structure of a C Program)

C प्रोग्राममध्ये लिहिलेल्या सूचनाधविधानांची नेहमीची ऑर्डर खाली दर्शविल्याप्रमाणे आहे.



चित्र 1.28: सी प्रोग्रामची सामान्य रचना

C प्रोग्राम विनामूल्य स्वरूपात लिहिलेला आहे. विनामूल्य स्वरूपनात, याचा अर्थ असा आहे की एखादी इन्स्ट्रक्शन एका ओळीत कोठेही सुरू होऊ शकते आणि कोठेही समाप्त होऊ शकते. एखादी इन्स्ट्रक्शन देखील एका ओळीपेक्षा जास्त असू शकते. स्पॅसिंगचा C भाषेत काहीही परिणाम होत नाही, तो फक्त प्रोग्रामची वाचनीयता वाढविण्यासाठी वापरला जातो.

पुढे, C भाषा केस सेन्सेटिव्ह आहे, म्हणजेच, लोअरकेस आणि अपरकेस अक्षरा मध्ये फरक करते. प्रत्येक सी प्रोग्राम फक्त लोअरकेस अक्षरा मध्ये लिहिलेला असतो.

आता आपण C प्रोग्राम मध्ये प्रत्येक विभागातील भूमिका एक-एक करून समजून घेऊया.

- **Section 1** हा पर्यायी आहे. जर सादर केले तर त्यात प्रोग्राम बदलचे वर्णन आहे. या वर्णनात सामान्यतः प्रोग्रामद्वारे पूर्ण केल्या जाणाऱ्या कार्याची माहिती असते.

उदाहरणार्थ.

```

/*
 *   Program to compute the factorial of a given
 *   natural number (positive integer number).
 *
 *   Filename : prime.c
 */
  
```


लक्षात घ्या की “/” आणि “*” यामधील कोणताही मजकूर, दरम्यान स्पेस नाही, कंपाईलरकडे दुर्लक्ष केले जाते, म्हणजेच, ते मशीन कोडमध्ये भाषांतरित केलेले नाही.

- **Section 2** प्रीप्रोसेसर डिरॅक्टिव्हस आहेत. वारंवार वापरल्या जाणाऱ्या प्रीप्रोसेसर डिरॅक्टिव्हस मध्ये `#include` आणि `#define` यांचा समावेश आहे. कंपायलेशनसाठी प्रोग्रॅम कसा तयार करावा हे डिरॅक्टिव्हस प्रीप्रोसेसरला सांगतात.

प्रोग्राममध्ये कोणत्या हेडर फायली समाविष्ट कराव्यात हे `#include` डिरॅक्टिव्हस सांगते आणि प्रोग्राममध्ये बऱ्याच ठिकाणी वापरल्या जाणाऱ्या लिटरल्स (कॉन्स्टन्ट) साठी `#define` डिरॅक्टिव्हस चा वापर केला जातो.

उदाहरणार्थ,

```
#include<stdio.h>
```

प्रोग्राम फाइलमध्ये `stdio.h` नावाच्या हेडर फाइलमधील सामग्री प्रोग्रॅम फाइल मध्ये समाविष्ट करण्यास प्रीप्रोसेसरला सांगते.

त्याचप्रमाणे,

```
#define N 200
```

प्रीप्रोसेसरला identifier N ची प्रत्येक घटना 200 मूल्याद्वारे पुनर्स्थित करण्यास सांगते.

- **Section 3** हा पर्यायी आहे. विद्यमान असल्यास, त्यात ग्लोबल डिक्लरेशन केले जाते. या डिक्लरेशन मध्ये सहसा डेटा आयटम (व्हेरिएबल्स) ची डिक्लरेशन समाविष्ट असते जी प्रोग्राममधील भिन्न फंक्शन्समध्ये सामायिक केली जाते. याव्यतिरिक्त, या डिक्लरेशन मध्ये प्रोग्राममध्ये वापरल्या जाणाऱ्या इतर फंक्शन्स (प्रोटोटाइप) च्या डिक्लरेशनचा देखील समावेश असू शकतो.

- **Section 4** `main()` फंक्शन समाविष्ट करते. प्रोग्रॅमचे एक्झिक्युशन नेहमीच `main()` फंक्शनच्या एक्झिक्युशन ने सुरू होते. ते इतर फंक्शन्सना कितीही वेळा कॉल करू शकते आणि हे कॉल केलेले फंक्शन इतर फंक्शन्सना कॉल करू शकतात.

`main()` फंक्शनमधील पहिल्या विभागात तसेच इतर फंक्शन्समध्ये लोकल डिक्लरेशन केले जाते. लोकल डिक्लरेशन या अर्थाने की ते फक्त त्या फंक्शन मधील कार्याच्या आवश्यकतांशी संबंधित आहेत. दुसऱ्या विभागात इन्स्ट्रक्शन (ज्याला स्टेटमेंट्स देखील म्हणतात) जे फंक्शनद्वारे केल्या जाणाऱ्या क्रियांना परिभाषित करते.

- **Section 5** हा पर्यायी आहे. जर असल्यास त्यात इतर फंक्शन्सची व्याख्या आहे. समस्येचे निराकरण करणे सोपे आणि आकारात लहान असल्यास, कार्य पूर्ण करण्यासाठी केवळ `main()` फंक्शन पुरेसे आहे.

तथापि, जर समस्या जटिल असेल आणि समस्येचे आकार मोठे असेल तर ते लहान आणि स्वतंत्र सबप्रॉब्लम्समध्ये विभागले जाईल आणि नंतर आपण प्रत्येक सबप्रॉब्लमसाठी स्वतंत्र फंक्शन लिहितो.

`main()` फंक्शन या फंक्शन्सच्या योग्य कॉलद्वारे या फंक्शन्सची एक्झिक्युशन करते आणि या फंक्शन्समधून मिळवलेल्या सबप्रॉब्लम्सच्या सोल्यूशन्सचे संश्लेषण करते.

1.3.2 C प्रोग्राम्सची उदाहरणे (Example C Programs)

C प्रोग्रामच्या संरचनेचा अनुभव घेण्यासाठी आपण काही सोप्या C प्रोग्रामच्या उदाहरणांचा विचार करूया.

पुढील प्रोग्राम स्टँडर्ड फर्स्ट C प्रोग्राम आहे जो C च्या प्रत्येक पाठ्यपुस्तकात देण्यात आला आहे. एक्झिक्युट झाल्यावर तो हॅलो, वर्ल्ड हा संदेश दाखवेल.

Listing 1.1

```

1: /*
2: *   Program to greet the user with the message "Hello World".
3: *   File name: hello.c
4: */
5: #include<stdio.h>
6: int main()
7: {
8:     printf("Hello, World\n");
9:     return 0;
10: }
```

Test Run

Hello, World

प्रोग्रॅमचे विच्छेदन (Dissection of the Program)

संदर्भासाठी येथे लाइन क्रमांक दिलेला आहेत.

Lines 1-4 कमेंट्स आहेत.

Line 5 प्रीप्रोसेसर #include डिरेक्टिव्ह

Line 6 main नावाचे फंक्शन निर्दिष्ट करते. हे एक खास नाव आहे जे सिस्टमद्वारे ओळखले जाते. प्रोग्राम एक्झिक्युशन हे main फंक्शन पासून सुरू होते. प्रत्येक C प्रोग्राममध्ये main फंक्शन असणे आवश्यक आहे.

प्रत्येक C फंक्शनशी संबंधित रिटर्न टाइप असतो. जेथे रिटर्न टाइप हा डेटा टाइप पैकी एक आहे, जो फंक्शनद्वारे परत आलेल्या मूल्याचे टाइप निश्चित करतो. जर फंक्शन कोणतेही मूल्य परत करत नसल्यास त्याचा रिटर्न टाइप void म्हणून निर्दिष्ट केला जातो.

प्रोग्राम एक्झिक्युशन हे main फंक्शन पासून सुरू होते हे सहसा इंट टाइप घोषित केले जाते.

Line 7 कॅरेक्टर '{' मध्ये, ज्याला डावे ब्रॅकेट किंवा डावे करली ब्रॅकेट म्हटले जाते. 10 व्या ओळीत उजवा ब्रॅकेट बरोबर जुळत आहे, जो main फंक्शन च्या शेवटी दिसेल. जुळणाऱ्या कंसांची ही जोड फंक्शनचे मुख्य भाग जोडते.

Line 8 संगणकाच्या स्क्रीनवर “हॅलो, वर्ल्ड” प्रदर्शित करणारा लायब्ररी फंक्शन printf() वापरतो.

Line 9 रिटर्न स्टेटमेंट वापरते, जे एक्झिक्युशन नंतर प्रोग्रामचे एक्झिक्युशन संपुष्टात आणते आणि ऑपरेटिंग सिस्टमला नियंत्रण परत करते. याव्यतिरिक्त, ते ऑपरेटिंग सिस्टमला 0 चे मूल्य देखील परत करते जे सूचित करते की प्रोग्राम यशस्वीरित्या समाप्त झाला.

Line 10 main फंक्शनचा शेवट तसेच प्रोग्रामचा शेवट चिन्हांकित करतो कारण प्रोग्राममधील हे एकमेव फंक्शन आहे.

चला आणखी एका प्रोग्रामचे उदाहरण बघुया, जेथे युझरद्वारे काही इनपुट प्रदान केले जाते, कॉम्प्युटेशन केली जातात आणि कॉम्प्युटेशनचे परिणाम प्रदर्शित होतात. या उदाहरणातील प्रोग्राम सेल्सिअस स्केलमधील तापमान इनपुट म्हणून घेते, फॅरेनहाइट स्केलमध्ये त्याच्या समकक्ष तापमानाचे मोजणी करते आणि ते संगणकाच्या स्क्रीनवर प्रदर्शित करते.

सेल्सिअस आणि फॅरेनहाइट तापमानातला हा संबंध आहे

$$\frac{C}{100} = \frac{F - 32}{180}$$

सोपे करून

$$F = 1.8 \times C + 32$$

हे गणितीय संबंध C च्या दिलेल्या मूल्यासाठी F मोजण्यासाठी वापरले जाते.

Listing 1.2

```
1: /*
2:  * Program to convert temperature from Centigrade scale
3:  * to Fahrenheit scale
4:  */
5: #include<stdio.h>
6: int main()
7: {
8:     float f, c;
9:     printf("\nEnter temperature in Celsius scale: ");
10:    scanf("%f", &c);
11:    f = 1.8 * centigrade + 32;
12:    printf("\nEquivalent temperature in Fahrenheit scale");
13:    printf( " = %.2f\n",f);
14:    return 0;
15: }
```

Test Run

Enter temperature in Celsius scale: 30

Equivalent temperature in Fahrenheit scale = 86.00

प्रोग्रॅमचे विच्छेदन (Dissection of the Program)

Lines 1-4 कमेंट्स आहेत.

Line 5 प्रीप्रोसेसर डिरॅक्टिव्ह.

Lines 6-15 ब्रेसिस {} च्या जोड्यासह main फंक्शनची व्याख्या प्रस्तुत करते.

Line 8 फ्लोट चे दोन व्हेरिएबल्स, f आणि c घोषित करते, जे संगणक मेमरीमध्ये दोन वास्तविक संख्या (फ्लोट) दर्शवू आणि संचयित करू शकतात. व्हेरिएबल c चा वापर व्हॅल्यू ठेवण्यासाठी केला जातो, जे प्रोग्राम सेक्शन दरम्यान युझरने प्रविष्ट केलेले सेल्सिअस स्केलचे तापमान दर्शवते. व्हेरिएबल f, फॉरेनहाइट स्केलमधील तापमानाचे सेल्सिअस स्केलमध्ये दिलेल्या तपमानाचा प्रतिनिधित्व करणारे गणित मूल्य ठेवण्यासाठी वापरला जातो.

Line 9 संगणकाच्या स्क्रीनवर “Enter temperature in Celsius scale:” प्रदर्शित करणारा लायब्ररी फंक्शन printf() वापरतो. आपण या प्रकारच्या मेसेजला युझर प्रॉम्प्ट म्हणून कॉल करतो कारण ते युझर इच्छित इनपुट प्रविष्ट करण्यास मार्गदर्शन करतात. या प्रकरणात, प्रविष्ट केले जाणारे मूल्य म्हणजे सेल्सिअस स्केलमधील तापमान.

तुम्ही पाहिले असेल की दुहेरी अवतरण चिन्हात बंदिस्त अक्षराचा संपूर्ण क्रम छापलेला नाही. मग, आरंभात ‘\n’ पांदांची भूमिका काय आहे? ही दोन्ही कॅरेक्टर्स एकत्रितपणे new लाइन कॅरेक्टर दर्शविते. जरी new लाइन line कॅरेक्टर दोन कॅरेक्टरचे म्हणजेच, ‘\’ आणि ‘n’ यांचे संयोजन असले, तरी ते C कंपाईलरने एकाच कॅरेक्टर मध्ये भाषांतरित केले आहेत. new लाइन कॅरेक्टर त्यानंतरची माहिती मुद्रित करण्यापूर्वी कंपाईलरला पुढील ओळीवर जाण्यासाठी सूचना देते.

Line 10 दुसरे लायब्ररी फंक्शन `scanf()` वापरते जे युझरचे इनपुट स्वीकारते आणि व्हेरिएबल `c` मध्ये संचयित करते (खरं तर, `c` साठी आरक्षित असलेल्या मेमरी ठिकाणी). पहिला आर्ग्युमेंट एक फॉर्मेट स्ट्रिंग आहे जो दुहेरी अवतरणांमध्ये बंद केलेला आहे आणि प्रविष्ट केलेल्या मूल्याचे स्पष्टीकरण कसे करावे हे सिस्टमला सांगते. फॉर्मेट स्ट्रिंगनंतर, एक किंवा अधिक व्हेरिएबल्सची सूची आहे; प्रत्येक व्हेरिएबल ‘&’ कॅरेक्टर सह प्रीफिक्स केलेले, ज्याला अॅड्रेसऑफ (*addressof*) ऑपरेटर म्हटले जाते.

Line 11 हे असाईनमेंट स्टेटमेंट आहे जे सेल्सिअस स्केल मध्ये दिलेल्या तापमानापेक्षा फॅरनहाइट स्केल मधील तपमानाचे गणन करते आणि व्हेरिएबल `f` ला असाइन करते, म्हणजेच व्हेरिएबल `f` मध्ये स्टोअर करते.

Line 12-13 “Equivalent Temperature in fahrenheit = ” या संदेशासह लायब्ररी फंक्शन `print()` ला व्हेरिएबल `f` मधील स्टोरेज व्हॅल्यू प्रदर्शन करते. या प्रकारच्या संदेशांचा वापर करणे अनिवार्य नाही, परंतु ते उपयुक्त आहेत कारण ते आउटपुट स्पष्ट करणे सोपे करतात.

Line 14 रिटर्न स्टेटमेंट वापरते, जे प्रोग्राम बंद करते आणि ऑपरेटिंग सिस्टमला व्हॅल्यू 0 देते.

1.3.3 प्रोग्राम तयार करणे, कॉम्पयलिंग आणि एक्झिक्युट करणे (Creating, Compiling and Executing a Program)

एकदा अल्गोरिदम तयार झाल्यावर, पुढील पायरी म्हणजे अल्गोरिदमला संगणकाच्या प्रोग्राममध्ये रूपांतरित करणे. या रूपांतरण दरम्यान अल्गोरिदमची प्रत्येक पायरी एक किंवा अधिक C भाषेच्या इन्स्ट्रक्शन म्हणून कोडित केली जाते. संगणकात टाईप करण्यापूर्वी विद्यार्थ्यांनी प्रथम कागदाच्या तुकड्यावर प्रोग्राम लिहावा अशी शिफारस केली जाते.

विविध चरणांचे प्रदर्शन करण्यासाठी आम्ही Turbo C/C++ कंपाइलरचा विचार करू, जे नवशिक्यांसाठी वापरण्यास सुलभ आहे.

प्रोग्राम तयार आणि एडिटिंग करणे (Creating and Editing Programs)

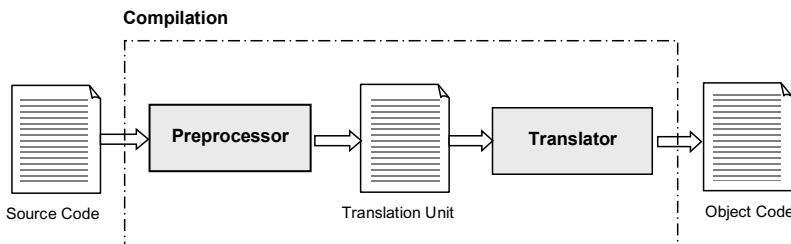
एकदा प्रोग्राम कागदावर तयार झाल्यावर आपण टेक्स्ट एडिटर वापरून त्याला कॉम्प्यूटर मेमरी टाईप करतो. टेक्स्ट एडिटर आपल्याला संगणकाच्या मेमरीमध्ये कॅरेक्टर डेटा प्रविष्ट करण्यास मदत करतो, संगणकाच्या मेमरीमधील डेटा एडिटिंग करण्यास (बदल करण्यास) आणि डिस्क फाइलमध्ये मेमरीमधून डेटा “.C” विस्तारासह दुय्यम मेमरीमध्ये जतन करण्यास अनुमती देतो.

ही संग्रहित फाईल स्रोत (*source*) फाइल म्हणून ओळखली जाते आणि त्यातील सामग्री स्रोत (*source*) कोड म्हणून ओळखली जाते. ही स्रोत (*source*) फाइल कंपाइलरसाठी इनपुट असेल.

प्रोग्रामरने काळजीपूर्वक C भाषेचे नियम पाळले पाहिजेत. भाषेच्या नियमांचे उल्लंघन केल्याने व्याकरणात्मक त्रुटी उद्भवतात, अधिक स्पष्टपणे सिन्टॅक्स एरॉर म्हणून ओळखल्या जातात. कंपाइलर सिन्टॅक्स एरॉरची तपासणी करेल. पुढे जाण्यापूर्वी या सर्व चुका दूर केल्या पाहिजेत.

कॉम्पयलिंग प्रोग्राम (Compiling Programs)

डिस्कवर स्टोअर्ड स्रोत फाइलमधील स्रोत कोड मशीन भाषेत अनुवादित करणे आवश्यक आहे. हे काम कंपाइलरद्वारे केले गेले आहे. C कंपाइलर प्रत्यक्षात दोन स्वतंत्र प्रोग्रामचे संयोजन आहे - प्रीप्रोसेसर आणि ट्रान्सलेटर.

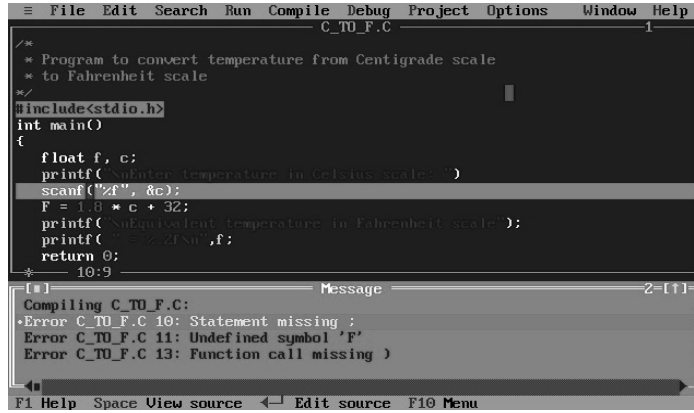


चित्र 1.29: कॉम्पयलेशन प्रक्रिया

प्रीप्रोसेसर सौर्स कोड वाचतो आणि भाषांतर करण्यासाठी तयार करतो. सौर्स कोड वाचत असताना, प्रीप्रोसेसर निर्देशांसाठी कोड स्कॅन करतो आणि त्यानुसार त्यावर प्रक्रिया करतो.

उदाहरणार्थ, जेव्हा त्याचा `#include` असलेल्या डिरेक्टिव्हशी सामना केला जातो, तेव्हा तो डिरेक्टिव्ह हेडर फाईल (जसे की `stdio.h`) च्या सामग्रीसह निर्देशित करतो आणि जेव्हा तो `#define` डिरेक्टिव्हशी सामना करतो तेव्हा तो लिटरल्स (कॉन्स्टन्ट) च्या सामग्रीसह निर्देशित करतो. प्रीप्रोसेसरचे आउटपुट एक इंटरमीडिएट फाइल आहे, ज्यास ट्रान्सलेशन युनिट म्हणून ओळखले जाते.

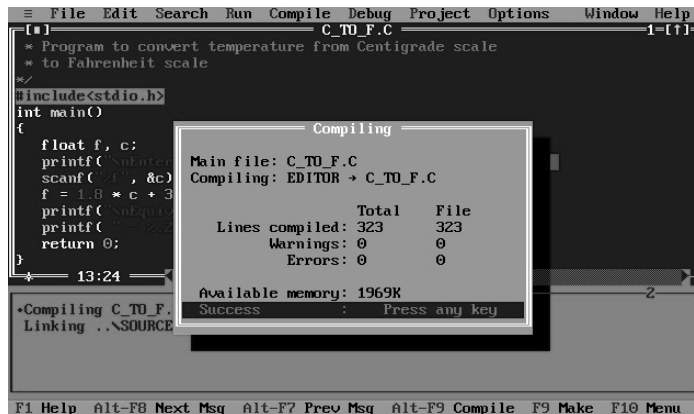
ट्रान्सलेटर भाषांतर युनिटची सूचना-दर-सूचना वाचतो आणि त्यांच्या व्याकरणाच्या अचूकतेसाठी तपासतो. कोणतीही वाक्यरचना त्रुटी असल्यास, तो एक एरॉर मेसेज ध्वजांकित करतो - ज्यास पडद्यावर diagnostic मेसेज म्हणतात. हे diagnostic मेसेज प्रोग्रामरला या त्रुटीचे कारण आणि ते ज्या ठिकाणी आहेत तेथे ओळखण्यास मदत करतात.



चित्र 1.30: टर्बो सी/सी ++ कंपायलरचा स्क्रीन शॉट सिंटॅक्स एरॉर दर्शवतो

म्हणूनच, तेथे एखादी सिंटॅक्स एरॉर देखील असल्यास, अनुवाद प्रक्रिया ही कॅम्पयलेशन म्हणून ओळखली जाते आणि संपुष्टात आणली जाते. या मध्ये, टेक्स्ट एडिटरचा वापर करून सौर्स फाइल उघडा आणि त्यात आवश्यक दुरुस्त्या करा आणि कॅम्पयलेशनची पुनरावृत्ती करा.

तथापि, ट्रान्सलेटर युनिटमध्ये सिंटॅक्स एरॉर नसल्यास, ट्रान्सलेटर सुरुवातीपासूनच सूचना पुन्हा वाचतो, त्यांचे भाषांतर मशीन भाषेत करतो आणि डिस्क फाईलवर लिहितो. सौर्स कोडची भाषांतरित आवृत्ती ऑब्जेक्ट कोड म्हणून ओळखली जाते आणि “*.obj” एक्सटेंशनसह डिस्क फाईलमध्ये संग्रहित केली जाते.



चित्र 1.31: टर्बो सी/सी ++ कंपायलरचा स्क्रीन शॉट संकलन प्रक्रियेचे यश दर्शवितो

लिंकिंग प्रोग्राम (Linking Programs)

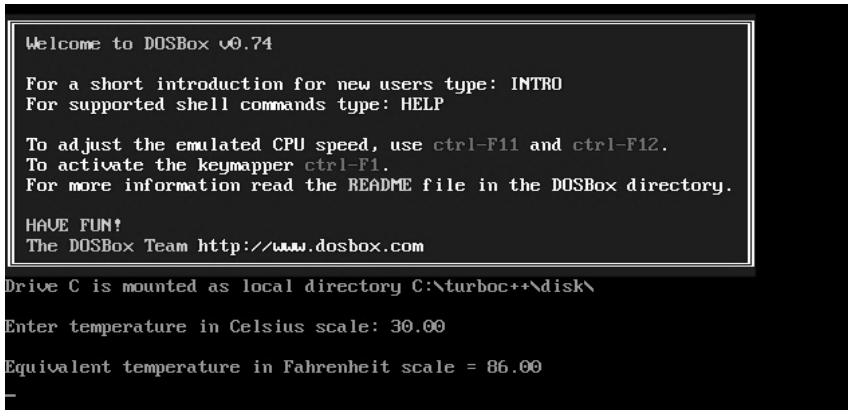
एकदा सौर्स कोडचे भाषांतर ऑब्जेक्ट कोडमध्ये झाले, जरी ते मशीन भाषेमध्ये असले तरीही ते एक्झिक्युटेबल स्वरूपात नसते. कारण असे आहे की ते लायब्ररी फंक्शन्सच्या कार्ये (लायब्ररीच्या स्वरूपात कंपाईलरसह पुरविलेले पूर्व-लेखी फंक्शन्स) संदर्भित आहे. या सर्व फंक्शन्सना अंतिम मशीन कोड मिळविण्यासाठी ऑब्जेक्ट कोडमध्ये समाविष्ट करणे देखील आवश्यक आहे, जे एक्झिक्युटेबल फॉर्ममध्ये आहे ज्याला एक्जीक्यूटेबल कोड म्हणून ओळखले जाते. आणि ते “*.exe” एक्सटेंशनसह डिस्क फाईलमध्ये संग्रहित आहे. हा एक्जीक्यूटेबल कोड प्रोग्रामचा अंतिम फॉर्म आहे जो एक्झिक्युशनसाठी तयार आहे.



चित्र 1.32: टर्बो सी/सी ++ कंपाईलरचा स्क्रीन शॉट लिंकिंग प्रक्रियेचे यश दर्शवितो

एक्झिक्युटिंग प्रोग्राम (Executing Programs)

एकदा प्रोग्राम लिंक झाल्यावर ते एक्झिक्युशनसाठी तयार आहे. प्रोग्राम एक्झिक्युट करण्यासाठी प्रोग्रॅमला संगणकाच्या मेमरीमध्ये लोड करण्यासाठी आणि एक्झिक्युट करण्यासाठी आपण ऑपरेटिंग सिस्टम आज्ञा देतो. प्रोग्रामला मेमरीमध्ये आणणे म्हणजे ऑपरेटिंग सिस्टम प्रोग्रामचे कार्य जे लोडर म्हणून ओळखले जाते. लोडर एक्झिक्युटेबल प्रोग्रामला दुय्यम स्टोरेजमध्ये शोधतो, तो वाचतो आणि त्यास संगणकाच्या मेमरीमध्ये आणतो. एकदा प्रोग्राम लोड झाल्यानंतर, ऑपरेटिंग सिस्टम प्रोग्रामवरील नियंत्रण हस्तांतरित करते आणि प्रोग्राम त्याचे एक्झिक्युशन सुरू करते.



चित्र 1.33: प्रोग्राम आउटपुट दर्शविणारी टर्बो सी/सी ++ कंपाईलरची युझर स्क्रीन

प्रोग्राम टेस्टिंग (Testing the Program)

प्रोग्राम एक्झिक्युट होत असतानाही, प्रोग्रामचे आउटपुट योग्य असू शकत नाही.

हे प्रोग्राममधील लॉजिकल चुकांमुळे असू शकते. लॉजिकल एरॉर ही एक चूक आहे जी प्रोग्रामरने समस्येचे डिझाइन करताना केली. उदाहरणार्थ, प्रोग्रामर संगणकास सांगते की वजाबाकीऐवजी एकूण वेतनातून कपात जोडून निव्वळ पगाराची गणना करा. कंपाईलर या चुका शोधू शकत नाही.

म्हणून, प्रोग्रामरला डेटाच्या संचासाठी प्रोग्राम आउटपुट काळजीपूर्वक परीक्षण करून लॉजिकल एरॉर शोधणे आणि त्या सुधारणे आवश्यक आहे ज्यासाठी त्या डेटाच्या संचाचे परिणाम आधीच माहित असणे आवश्यक आहे. अशा प्रकारच्या डेटाला टेस्ट डेटा म्हणून ओळखले जाते.



सिंटक्स एरॉर आणि लॉजिकल एरॉर एकत्रितपणे बग्स म्हणून ओळखल्या जातात. या लुटी ओळखण्याची आणि दूर करण्याची प्रक्रिया डीबगिंग म्हणून ओळखली जाते.

1.3.4 विविध C कंपाइलर (Various C Compilers)

खाली दिलेले विंडो आधारित सिस्टमवर दोन मुख्यतः वापरल्या जाणारे C कंपाइलर आहेत

- Turbo C/C++ Compiler
- Dev C++ Compiler

खालीलप्रमाणे ऑनलाईन C कॉम्प्लायर्सचे काही लिंक्स आहेत:

The following are few links to online C Compilers:

- <https://www.codechef.com/ide>
- <https://ide.geeksforgeeks.org>
- https://www.onlinegdb.com/online_c_compiler
- <https://www.programiz.com/c-programming/online-compiler/>
- https://www.tutorialspoint.com/compile_c_online.php
- <https://www.jdoodle.com/c-online-compiler/>
- <https://techiedelight.com/compiler/>

1.4 C भाषेसह प्रारंभ करणे (GETTING STARTED WITH C LANGUAGE)

सी प्रोग्रामिंग भाषा डेनिस रिची यांनी AT & T बेल प्रयोगशाळेत 1972 मध्ये विकसित केली होती आणि ती सर्वात लोकप्रिय भाषापैकी एक आहे.

संगणकाच्या अंतर्निहित हार्डवेअरशी त्यांच्या परस्परसंवादाच्या पातळीवर अवलंबून प्रोग्रामिंग भाषा दोन विस्तृत श्रेणींमध्ये विभागली जाऊ शकते:

- **लो लेवल लॅंग्वेज (Low-level Languages)** - या श्रेणी अंतर्गत आपल्याकडे मशीन भाषा आणि असेंब्ली भाषा आहे. या भाषा संगणकाचा कार्यक्षम वापर करण्यास परवानगी देतात.

परंतु या भाषांमध्ये समस्या काही आहेत:

- ही हार्डवेअर अवलंबित आहे, म्हणजेच, या भाषांचा वापर करून लिहिलेले प्रोग्राम पोर्टेबल नाहीत, म्हणजेच, इतर संगणकांवर वापरले जाऊ शकत नाहीत.

- या भाषांचा प्रोग्रामिंग करणे सोपे काम नाही. एखाद्याला संगणकाच्या आर्किटेक्चरचे सखोल ज्ञान असणे आवश्यक आहे.
- **हाय लेवल लॅंग्वेज (High-level Languages)** - या श्रेणीनुसार आपल्याकडे FORTRAN, COBOL, PASCAL इ. भाषांचा अफाट संग्रह आहे आणि सध्या C, C++, Java, आणि Python कार्यक्षमतेसाठी बनविल्या गेल्या आहेत, म्हणजे वेगवान प्रोग्राम विकासासाठी.

या भाषांचे खालील फायदे आहेत:

- प्रोग्राम सिन्टाक्स लिहिण्यासाठी वाक्यरचना इंग्रजी विधानांप्रमाणेच आहे. हे वाचकांना उच्च-स्तरीय भाषा (HLLs) लवकर शिकण्यास सक्षम करते. याव्यतिरिक्त, HLLs मध्ये लिहिलेले प्रोग्राम सहज समजू शकतात, त्यांची देखभाल सुलभरीत्या होते.
- HLLs मध्ये लिहिलेले प्रोग्राम्स हार्डवेअरवर अवलंबून नसतात. याचा अर्थ असा की एका मशीनसाठी लिहिलेला प्रोग्राम कमीतकमी बदलांसह दुसऱ्या मशीनमध्ये हस्तांतरित केला जाऊ शकतो किंवा काहीही नाही, म्हणजेच, या भाषा पोर्टेबल आहेत.

C भाषा ही दोन श्रेणींमध्ये आहे. म्हणूनच याला बऱ्याचदा मिडल लेवल लॅंग्वेज म्हटले जाते, कारण याची रचना दोन्ही जगामध्ये केली गेली होती, म्हणजेच चांगली प्रोग्रामिंग कार्यक्षमता तसेच मशीनची चांगली कार्यक्षमता.

- प्रोग्रामिंग कार्यक्षमता प्राप्त करण्यासाठी C भाषेमध्ये इतर कोणत्याही आधुनिक उच्च-स्तरीय भाषेचे सर्व घटक आहेत.
- मशीनची कार्यक्षमता प्राप्त करण्यासाठी, सी भाषेमध्ये सिस्टमच्या कोणत्याही हार्डवेअर घटकात प्रवेश करण्यासाठी, नोंदणी स्तरावर ऑपरेट करण्यासाठी आणि उच्च-स्पीड असेंब्ली भाषेच्या नित्यक्रमांसह इंटरफेस करण्यासाठी आवश्यक वैशिष्ट्ये आहेत.

1.4.1 C भाषेची वैशिष्ट्ये (Characteristics of C Language)

C भाषा ही सर्वात लोकप्रिय प्रोग्रामिंग भाषा म्हणून होती आणि राहिल.

त्याच्या लोकप्रियतेत भर घालणारी काही प्रमुख वैशिष्ट्ये अशी आहेत:

- एक सामान्य उद्देश प्रोग्रामिंग भाषा आहे आणि म्हणूनच, विविध प्रकारच्या समस्या सोडविण्यासाठी वापरली जाऊ शकते.
- ही स्ट्रक्चरिंग प्रोग्रामिंगला समर्थन देते आणि म्हणूनच त्यात विकसित केलेले प्रोग्रॅम सहज समजू शकतात.
- ही विनामूल्य फॉर्म आहे, अर्थात प्रोग्राम लिहिण्यासाठी कठोर स्वरूप नाही. कोणतीही सूचना कोठूनही सुरू होऊ शकते आणि कोठेही समाप्त होऊ शकते. याव्यतिरिक्त, एकच सूचना बऱ्याच ओळी विस्तृत करू शकते (त्यात लिहिता येते).
- ही केस सेन्सेटिव्ह आहे. हे लोअरकेस (लहान) आणि अपरकेस (कॅपिटल) वर्णमाला दरम्यान फरक करते.
- ऑपरेटरचा समृद्ध संच प्रदान करणारी ही पहिली भाषा होती.
- ही आपल्याला, पॉईंटर्सद्वारे, आपल्या प्रोग्रामद्वारे आपल्या संगणकावर कनेक्ट केलेल्या कोणत्याही स्टोरेज स्थान आणि डिव्हाइसमध्ये प्रवेश करण्याची परवानगी देते.
- ही आपल्याला अंतर्गत प्रोसेसर रजिस्टरमध्ये बदल करण्याची परवानगी देते आणि अशा प्रकारे हे लो लेवल प्रोग्रामिंगसाठी खूप उपयुक्त आहे.
- ही पोर्टेबल आहे, याचा अर्थ असा आहे की त्यामध्ये लिहिलेला कोणताही प्रोग्राम कोणत्याही संगणकावर थोडासा बदल किंवा बदल न करता वापरला जाऊ शकतो.
- ही आपल्याला आपली स्वतःची लायब्ररी विकसित करण्याची परवानगी देते ज्यास स्टँडर्ड लायब्ररी फंक्शनसारख्या कोणत्याही प्रोग्रामशी लिंक केले जाऊ शकते..

1.4.2 C भाषेचे ॲप्लिकेशन्स क्षेत्र (Application Areas of C Language)

बू भाषेची ताकद ही त्यास एक नैसर्गिक निवड बनवते

- सिस्टम प्रोग्रामिंग ज्यात लैंग्वेज ट्रांसलेटर, डिव्हाइस ड्राइव्हर्स, एडिटर, लिंक्स, लोडर इ. साठी लेखन सॉफ्टवेअर समाविष्ट आहे.
- भिन्न कम्युनिकेशन प्रोटोकॉल कार्यान्वित करण्यासाठी नेटवर्क सॉफ्टवेअर.
- ग्राफिक्स प्रोग्रामिंग ज्यात ग्राफिकल यूजर इंटरफेस (जीयूआय), वैज्ञानिक व्हिज्युअलायझेशन आणि सादरीकरण ग्राफिक्स इ.
- एम्बेडेड सिस्टम जिथे C रूटीनला वेगवान असेंब्ली लैंग्वेज रूटीनसह इंटरफेस केले जाते आणि परिणामी कोड एम्बेड केलेल्या सिस्टमचा एक भाग असलेल्या रॉम चिपमध्ये संग्रहित केला जातो.

1.4.3 C भाषेचे मूलभूत बिल्डिंग ब्लॉक्स (Basic Building Blocks of C Language)

प्रोग्रामिंग लैंग्वेज शिकण्याची एक कठीण गोष्ट म्हणजे जवळजवळ प्रत्येक गोष्ट एकमेकांशी संबंधित आहे. म्हणूनच, प्रत्येक गोष्टीबद्दल आपल्याला थोडी माहिती असण्यापूर्वी काहीही समजणे जवळजवळ अशक्य दिसते.

या विभागात आपण C भाषेच्या बिल्डिंग ब्लॉक्स म्हणून ओळखल्या जाणाऱ्या विविध कार्यात्मक घटकांबद्दल शिकूया.

1.4.3.1 कॅरेक्टर सेट (Character Set)

सर्वात मूलभूत अर्थाने, C प्रोग्राम ही कॅरेक्टरची अनुक्रम आहे. जेव्हा ही पाळे कंपाईलरकडे सबमिट केली जातात, तेव्हा त्या कॅरेक्टर, आयडेंटिफायर, कॉन्स्टन्ट आणि स्टेटमेंट म्हणून विविध संदर्भामध्ये वर्णन केल्या जातात. C सौर्स प्रोग्राममध्ये वापरली जाणारी अक्षरे अमेरिकन स्टॅंडर्ड कोड फॉर इन्फरमेशन इंटरचेंज (ASCII) या सेटची आहेत.

C भाषेतील कॅरेक्टरना खालील श्रेणींमध्ये वर्गीकृत केले आहे:

- Letters (A-Z, a-z)
- Digits (0-9)
- Special characters (, ; ~ > < # % ' ^ + - * . = ! | () { } [] / &)
- White space characters (*Blank space, horizontal tab, etc.*)



All C compilers ignore white space characters. These characters are basically used to enhance the readability and understandability of a C program.

1.4.3.2 टोकन्स (Tokens)

टोकन ही सर्वात लहान एंटीटी आहे ज्याचा स्वतःचा असा एक अर्थ असतो. C प्रोग्राम हा टोकनचा क्रम आहे. C भाषेतील टोकनचे खालील श्रेणींमध्ये वर्गीकरण केले जाऊ शकते:

- Keywords
- Identifiers
- Literals
- Punctuators
- Operators

कीवर्ड (Keywords)

C मधील प्रत्येक शब्द एकतर कीवर्ड किंवा आयडेंटिफायर्स म्हणून वर्गीकृत केला जातो.

कीवर्ड मुळात ते शब्द असतात ज्यांचे पूर्वनिर्धारित आणि निश्चित अर्थ असतात आणि हे अर्थ बदलले जाऊ शकत नाहीत. हे कीवर्ड प्रोग्राम निर्देश तयार करण्यासाठी मूलभूत बिल्डिंग ब्लॉक्स म्हणून काम करतात (स्टेटमेंट). सर्व कीवर्ड लोअरकेसमध्ये लिहिलेले आहेत. त्यांचा चुकीचा वापर सिंटॅक्स एरॉर मध्ये होतो.

Table 1.2: C मधील कीवर्ड

asm	default	for	short	
auto	do	goto	signed	union
break	double	if	sizeof	unsigned
case	else	int	static	void
char	enum	long	struct	volatile
Const	extern	register	switch	while
continue	float	return	typedef	

आयडेंटिफायर्स (Identifiers)

एक आयडेंटिफायर्स मूलतः प्रोग्राममधील एक नाव असते. व्हेरिएबल्स, अरे आणि फंक्शन्स दर्शविण्यासाठी आयडेंटिफायर्स वापरले जाऊ शकतात. या युझरने परिभाषित केलेल्या नावांमध्ये कॅरेक्टरचे अनुक्रम समाविष्टीत आहे, जिथे प्रत्येक कॅरेक्टर हा अक्षर, अंक किंवा अंडरस्कोर ‘_’ असू शकतो आणि अंकासह प्रारंभ होऊ शकत नाही. अपरकेस आणि लोअरकेस दोन्ही अक्षरे अनुमत आहेत, तथापि, लोअरकेस सामान्यतः वापरले जाते. अंडरस्कोर सामान्यतः पूर्ण नावे तयार करण्यासाठी दोन किंवा अधिक शब्दांना जोडण्यासाठी वापरला जातो.

सी भाषा केस संवेदनशील असल्याने (ती लोअरकेस आणि अपरकेसमध्ये फरक करते), म्हणूनच आयडेंटिफायर्स *price* आणि *PRICE* हे दोन भिन्न आयडेंटिफायर्स मानले जाईल.

RULES FOR IDENTIFIERS	
1.	The first character must be an alphabet or an underscore.
2.	Must consist of letters, digits or underscore only.
3.	Cannot use a keyword.
4.	Cannot contain a white space.

वैध आयडेंटिफायर्सची उदाहरणे:

myFile	roll_no	date_of_birth	_chk
file10	N1TO10	area	AbCdE

लिटरल्स (Literals)

लिटरल्स हे (बहुतेक वेळा कॉन्स्टंट म्हणून ओळखले जाते) निश्चित मूल्याचा संदर्भ देते जे प्रोग्रामच्या एक्झिक्युशन दरम्यान बदलत नाहीत.

- **इंटीजर लिटरल्स (Integer Literals)** – दशांश बिंदूशिवाय संख्या. इंटीजर संख्येमध्ये वैकल्पिक चिन्हाच्या आधीचे अंकांचा अनुक्रम, अधिक (+) किंवा वजा (−) असते.

- **रिअल लिटरल्स (Real Literals)** – आंशिक भाग असलेली संख्या.
- **सिंगल कॅरेक्टर लिटरल्स (Single Character Literals)** – ‘A’ सारख्या एका कोट्स (') मध्ये बंद असलेल्या एका अक्षराचा समावेश आहे. अंतर्गत, प्रत्येक कॅरेक्टर एक अविभाज्य मूल्य दर्शविला जातो जो वर्णाच्या ASCII कोडचे प्रतिनिधित्व करतो. उदाहरणार्थ, वर्ण ‘A’ चे अखंड मूल्य 65, वर्ण ‘B’ 66 द्वारे दर्शविले जाईल.

C भाषा काही विशिष्ट कॅरेक्टर देते जी कीबोर्डमधून थेट प्रविष्ट करणे/टाइप करणे शक्य नाही जसे की बॅकस्पेस, टॅब, कॅरेज रिटर्न, न्यूलाईन (*backspace, tab, carriage return, newline*) इ. ही अक्षरे \ (बॅक स्लॅश) ने सुरू होणाऱ्या अनुक्रमांद्वारे दर्शविली जातात आणि *escape sequences* म्हणून ओळखल्या जातात.

Table 1.3: सामान्य *escape sequences*

Escape Sequence	Represents	Effect
\n	Newline	Subsequent output starts from new line.
\t	Horizontal Tab	Moves over to the next eight-space-wide field.
\r	carriage return	Carriage return.
\0	Null character	Terminates a string.

- **स्ट्रिंग (मल्टि - कॅरेक्टर) लिटरल्स (String (multi-character) Literals)** – दुहेरी अवतरण मध्ये बंद कॅरेक्टरचा क्रम आहे. कॅरेक्टर सेट मधील कोणतेही कॅरेक्टर्स असू शकतात लक्षात घ्या की ‘A’ सारखे सिंगल कॅरेक्टर लिटरल्स हे “A” सारख्या स्ट्रिंगलिटरल्स अक्षरांच्या सारखे नाही.

विरामचिन्हे (पंकचूएटर्स , Punctuators)

विरामचिन्हे यांना विभाजक देखील म्हणतात. C मध्ये वापरलेले विविध विरामचिन्हे खाली दिलेल्या तक्त्यात वर्णन आहेत.

Table 1.4: काही विरामचिन्हांची यादी आणि त्यांचे वर्णन

Punctuators	Description
Brackets []	Used to enclose array subscripts.
Parentheses ()	Used to enclose arguments in function declaration as well as function definition, parameters in function calls and expressions.
Braces { }	Used to enclose a block of statements. Also used to enclose list of elements while initializing arrays.
Comma,	Used to separate arguments in function definition, and parameters in function call.
Semicolon ;	Used to terminate a statement.

ऑपरेटर (Operators)

ऑपरेटर हे C भाषेचे क्रियापद आहेत जे युझरच्या मूल्यांवर कार्य करतात. C भाषेचा ऑपरेटरचा समृद्ध असा संच आहे आणि त्याच्या विशिष्ट वैशिष्ट्यांपैकी एक आहे.

हे ऑपरेटर पुढील श्रेणींमध्ये मोजले जाऊ शकतात:

- अरीथमेटिक ऑपरेटर (+, -, *, /, %)
- रिलेशनल ऑपरेटर (<, <=, >, >=, ==, !=)
- लॉजिकल ऑपरेटर (!, &&, ||)
- बिटवाईस ऑपरेटर (~, >>, <<, &, |)

- स्पेशल ऑपरेटर
 - इन्क्रीमेंट आणि डिक्रीमेंट ऑपरेटर (++, --)
 - साईझऑफ ऑपरेटर
 - ऍड्रेसऑफ ऑपरेटर (&)
 - इंडिरेक्शन/डी - रेफरेंस ऑपरेटर (*)
 - टर्नरी/कंडिशनल ऑपरेटर (? :)

1.4.3.3 डेटा टाइपची संकल्पना (Concept of Data Type)

डेटा टाइप म्हणजे बिट्सच्या स्ट्रिंगला लागू केलेला अर्थ. औपचारिकरित्या, डेटा टाइपना मूल्यांच्या परिष्कृत संचासह परिभाषित केले जाते तसेच या मूल्यांवर कार्य केले जाऊ शकतात अशा ऑपरेशन्ससाठी नियम परिभाषित केले जाते.

C मधील डेटा टाइप चे दोन प्रकारचे आहेत:

- बिल्ट-इन डेटा टाइप (Built-in Data Types)
 - बिल्ट-इन डेटा टाइप हा C मधील सर्वात मूलभूत डेटा टाइप आहे.
 - बिल्ट-इन म्हणजे ते C मध्ये पूर्व परिभाषित आहेत आणि प्रोग्राममध्ये ते थेट वापरले जाऊ शकतात.
 - उदाहरणार्थ : *char, int, float, and double*.
 - याशिवाय आपल्याकडे *void* डेटा टाइपही आहे.
- Derived डेटा टाइप (Data Types)
 - हे विद्यमान डेटा प्रकारांद्वारे (बिल्ट-इन किंवा युझर- डिफाइंड) derived केले आहेत.
 - उदाहरणार्थ : *arrays and pointers*.
- युझर-डिफाइंड डेटा टाइप (User-defined Data Types)
 - हे युझर्सने त्यांची आवश्यकता पूर्ण करण्यासाठी तयार केले आहेत.
 - उदाहरणार्थ : *structure, union, and enumeration*.

या विभागात आपण केवळ बिल्ट-इन डेटा प्रकारांबद्दल शिकूया. उर्वरित डेटा प्रकार, अभ्यासक्रमाच्या आवश्यकतेनुसार योग्य ठिकाणी चर्चा केली जाईल.

C अंतर्गत विविध बिल्ट-इन डेटा प्रकारांना मूलभूत डेटा प्रकार किंवा मूलभूत डेटा प्रकार असे देखील म्हटले जाते, सारांश 1.5 मध्ये सारांशित केले आहेत

Table 1.5: बिल्ट-इन डेटा प्रकार

Name	Description
Integer <i>int</i>	Integer numbers
Real/Floating-point <i>float double</i>	Single precision floating-point numbers (<i>precision is 6 decimal places</i>) Double precision floating-point numbers (<i>precision is 12 decimal places</i>)
Character <i>char</i>	A Single Character

इंट डेटा प्रकार पूर्णांकांसाठी वापरला जातो आणि इन्टिजरचा सबसेट बनलेला असतो. Table 1.6 विविध प्रकारचे पूर्णांक संख्या, त्यांची मेमरी आवश्यकता आणि प्रत्येक प्रकारच्या मूल्यांची श्रेणी दर्शविते.

Table 1.6: इन्टिजर संख्येचा प्रकार

Type	Memory Requirements (in bytes)		Range of Values
	16-bit Compiler (Turbo C/C++)	32-bit Compiler (Dev C/C++)	
short	2	2	for -2byte integer 32768_ to 32767+ for -4byte integer 2147483648_ to 2147483647+
int	2	4	
long	4	4	

दशांश बिंदू असलेल्या संख्येसाठी फ्लोट आणि डबल डेटा प्रकार वापरले जातात. फ्लोट आणि डबल डेटा प्रकारांमधील फरक एवढाच आहे की श्रेणी आणि अचूकता, डबल डेटा प्रकारात फ्लोट डेटा प्रकारापेक्षा जास्त अचूकता आहे.

वास्तविक मूल्य म्हणजे काही विशिष्ट दशांश स्थितीत इच्छित वास्तविक संख्येचे अचूक परिमाण. प्रकारच्या फ्लोटचे वास्तविक मूल्य सहा दशांश स्थानांवर अचूक असते; जेथे डबल प्रकाराचे वास्तविक मूल्य बारा दशांश स्थानांवर अचूक आहे.

Table 1.7 विविध प्रकारच्या वास्तविक संख्या, त्यांची मेमरी आवश्यकता आणि प्रत्येक प्रकारच्या मूल्यांची श्रेणी दर्शविते.

Table 1.7: रियल संख्यांचा प्रकार

Type	Bytes	Range
float	4	$38-10 \times 3.4$ to $38+10 \times 3.4$
double	8	$308-10 \times 1.7$ to $308+10 \times 1.7$

char डेटा प्रकार कॅरेक्टरसाठी वापरला जातो आणि त्यास केवळ 1-बाइट मेमरी आवश्यक आहे. char डेटा प्रकाराबद्दल येथे एक विशेष गोष्ट आहे - अक्षरे त्यांच्या ASCII कोड वापरून मेमरीमध्ये संग्रहित केले जातात जे संख्यात्मक, म्हणजेच पूर्णांक आहेत.

void शब्दाचा अर्थ रिक्त आहे, म्हणून आपण असे म्हणू शकतो की void डेटा प्रकाराचे कोणतेही मूल्य नाही.

हे बऱ्याच परिस्थितींमध्ये उपयुक्त आहे. अशा परिस्थितींपैकी एक ही आहे - जे फंक्शन मूल्य परत करत नाही अशा फंक्शनसाठी रिटर्न प्रकार म्हणून वापरले जाते.

1.4.3.4 कॉन्स्टन्ट (Constants)

प्रोग्रॅमच्या एक्झिक्युशन दरम्यान बदलत नसलेल्या स्थिर मूल्यांना कॉन्स्टन्ट म्हणतात. कॉन्स्टन्ट पुढील मार्गांनी हाताळले जाऊ शकतात:

- लिटरल्स वापरणे (Using literal) – अरीथमेटिक एक्सप्रेशन मध्ये मूल्य थेट एन्कोडिंग
- सिम्बॉलिक कॉन्स्टन्ट वापरणे (Using symbolic constants)

सिम्बॉलिक कॉन्स्टन्ट म्हणजे असे नाव जे लिटरल्स ऐवजी बदलते.

उदाहरणार्थ,

```
#define PI 3.142
```

Here, the `#define` pre-processor directive associates the name PI with the literal 3.142.

कंपाइलिंग दरम्यान, प्रीप्रोसेसर PI नावाच्या प्रत्येक घटकास अक्षरशः 3.142 सह पुनर्स्थित करेल.

- कॉन्स्टन्ट म्हणून घोषित केलेले व्हेरिएबल वापरणे (Using a variable declared as constant)

पुढील चर्चा केल्याप्रमाणे एक व्हेरिएबल असे काहीतरी असते ज्याचे मूल्य प्रोग्राम अंमलबजावणी दरम्यान बदलू शकते. तथापि, कीवर्ड `const` वापरून व्हेरिएबलला कॉन्स्टन्ट म्हणून चिन्हांकित केले जाऊ शकते.

उदाहरणार्थ,

```
const float pie = 3.142;
```

येथे डेटा प्रकार फ्लोटशी संबंधित `pie` नावाचे व्हेरिएबल 3.141 मूल्यासह प्रारंभ केले गेले आहे आणि कॉन्स्टन्ट म्हणून चिन्हांकित केले आहे. त्यानंतर, आपण आपल्या प्रोग्राममध्ये व्हेरिएबलचे मूल्य वापरू शकतो, परंतु आपण त्याचे मूल्य बदलू शकत नाही, म्हणजेच आता व्हेरिएबल `pie` स्थिरतेने वागेल.

1.4.3.5 व्हेरिएबल्स (Variables)

व्हेरिएबल आपल्याला नामित स्टोरेज प्रदान करतो जो संपूर्ण प्रोग्रामच्या मध्ये आपण पुन्हा लिहू शकतो, पुनर्प्राप्त करू शकतो आणि त्याद्वारे हाताळू शकतो.

दुसऱ्या शब्दात, व्हेरिएबल्स संगणकाच्या मेमरीमधील डेटा असतात. व्हेरिएबल मधील मूल्य आपण बदलू शकतो म्हणून नाव व्हेरिएबल. व्हेरिएबल्समध्ये विविध प्रकारचे डेटा असतो आणि प्रोग्रामच्या एक्झिक्युशन दरम्यान भिन्न मूल्ये ठेवू शकतात.

C भाषेमधील प्रत्येक व्हेरिएबल विशिष्ट डेटा प्रकाराशी संबंधित असतो, जो त्याशी संबंधित मेमरीचा आकार आणि लेआउट, त्या मेमरीमध्ये संग्रहित केल्या जाणाऱ्या मूल्यांची श्रेणी आणि त्यावर लागू होणाऱ्या ऑपरेशन्सचा संच निर्धारित करतो.

डिक्लेअरिंग व्हेरिएबल्स (Declaring Variables)

C भाषेत, प्रोग्राममधील प्रत्येक व्हेरिएबल डिक्लेअर करणे आवश्यक आहे. व्हेरिएबल डिक्लेअर करण्यासाठी आणि डिफाईंड करण्यासाठी सिंटॅक्स खालील दिल्या प्रमाणे आहे.

```
type v1, v2, ..., vn;
```

जेथे $v1, v2, \dots, vn$ अद हे स्वल्पविरामाने विभक्त केलेल्या व्हेरिएबलची नावे आहेत.

खाली डिक्लेअर व्हेरिएबलची आणि डिफाईंडची काही उदाहरणे दिली आहेत:

```
int count, m, n;
float value, sum;
char ch;
double deviation;
```

प्रथम स्टेटमेंट व्हेरिएबल्स बवन्नदजए `count`, `m` आणि `n` ह्या `int` डिक्लेअर करते. कंपाईलर या प्रत्येक व्हेरिएबल्ससाठी 2 किंवा 4 बाइट्स (कंपाईलरवर अवलंबून) राखून ठेवते.

दुसरे विधान व्हेरिएबल `value` आणि `sum` हे `float` डिक्लेअर करते. कंपाईलर या प्रत्येक व्हेरिएबल्ससाठी 4-बाइट राखून ठेवते.

तिसरे स्टेटमेंट `char` प्रकारच्या व्हेरिएबल `ch` डिक्लेअर करते. कंपाईलरने यासाठी 1-बाइट आरक्षित केले.

चौथे स्टेटमेंट टाईप `double` प्रकारच्या व्हेरिएबल `deviation` डिक्लेअर करते. कंपाईलरने त्यासाठी 8-बाइट राखीव ठेवल्या आहेत.

इनिशियलिझिंग व्हेरिएबल (Initializing Variables)

व्हेरिएबल डिक्लेअर करण्याव्यतिरिक्त व्हेरिएबलला इनिशियल व्हॅल्यू देखील दिली जाऊ शकते.

ते करण्यासाठी '=' कॅरेक्टर नंतर व्हेरिएबलला व्हॅल्यू द्यावी लागेल. पुढील विधानाचा विचार करा

```
int sum = 0;
```

हे `int` व्हेरिएबल `sum` चे मूल्या 0 पासून सुरू करते.

1.4.3.6 एक्सप्रेशन्स (Expressions)

एक्सप्रेशन्स मूल्य मोजण्यासाठी एक सूत्र आहे. यात ऑपरेंड (operands) आणि ऑपरेटरचा क्रम असतो. ऑपरेंडमध्ये फंक्शन संदर्भ, व्हेरिएबल आणि कॉन्स्टन्ट असू शकतात. ऑपरेटर ऑपरेंडवर कार्य करण्यासाठी निर्दिष्ट करतात.

पुढील एक्सप्रेशन्स मध्ये

```
a + b
```

प्लस (+) एक ऑपरेटर आहे आणि a, b ऑपरेंड्स आहेत.

C भाषा खालील प्रकारच्या एक्सप्रेशन्सचे समर्थन करते:

- अरीथमेटिक एक्सप्रेशन (Arithmetic expressions)
- रिलेशनल एक्सप्रेशन (Relational expressions)
- लॉजिकल एक्सप्रेशन (Logical expressions)

प्रत्येक प्रकारचे एक्सप्रेशन्स विशिष्ट प्रकारचे ऑपरेंड घेतात आणि ऑपरेटरचा एक विशिष्ट संच वापरतात. प्रत्येक एक्सप्रेशन्सचे मूल्यांकन विशिष्ट प्रकारच्या मूल्याचे उत्पादन करते. लक्षात ठेवा की एक्सप्रेशन्स ही विधाने नसतात परंतु विधानांचे घटक असू शकतात.

उदाहरणार्थ पुढील मजकूराला ओळ विचारात घ्या

```
x = 2.0/3.0 + a * b;
```

संपूर्ण ओळ एक विधान आहे परंतु समान चिन्हा नंतरचा भाग म्हणजे एक एक्सप्रेशन. विशेषतः, मजकूरची ही ओळ असाईनमेंट स्टेटमेंट दर्शवते जी एक्सप्रेशनचे मूल्य व्हेरिएबल x ला प्रदान करते.

1.4.3.7 स्टेटमेंट्स (Statements)

स्टेटमेंट्स असंख्य कार्ये करतात जसे की संगणन करणे, संगणनाचा परिणाम संग्रहित करणे, नियंत्रणाचा प्रवाह बदलणे, वाचन करणे आणि डिव्हाइस/फायली वरून किंवा डिव्हाइस/फायली वर लिहिणे आणि कंपाइलरसाठी माहिती पुरवणे.

1.4.3.8 इनपुट/आउटपुट हाताळणे (HANDLING INPUT/OUTPUT)

जवळजवळ प्रत्येक प्रोग्राममध्ये युझर्सला काही डेटा इनपुट करावा लागतो जो मेमरीमध्ये संग्रहित केला जाईल आणि त्यानंतर प्रक्रिया केली जाईल. संगणनाचे इंटरमिजिएट आणि अंतिम परिणाम मेमरीमध्ये देखील संग्रहित केले जातात. आणि अखेरीस, संगणनाचे निकाल आउटपुट करण्याची आवश्यकता आहे जेणेकरून युझर्सला त्यांचा रोजच्या कामात त्याचा उपयोग करू शकतील.

सामान्य इनपुट डिव्हाइस म्हणजे कीबोर्ड आणि आउटपुट डिव्हाइस एक संगणक स्क्रीन आहे, ज्यास कधीकधी मॉनिटर देखील म्हटले जाते. कीबोर्ड आणि मॉनिटरचा समावेश असलेल्या या उपप्रणालीला कन्सोल म्हणून संदर्भित केले जाते.

I/O फंक्शन्सची संपूर्ण श्रेणी वापरण्यासाठी आपल्याला सर्व प्रोग्राममध्ये studio.h नावाची हेडर फाईल समाविष्ट करणे आवश्यक आहे.

Table 1.8: I/O फंक्शन्स

Function Category	Function Name	Description
Unformatted	getchar()	Returns a character that has been recently typed. The typed character is echoed to the computer screen. After typing the appropriate character, the user is required to press <i>Enter</i> key.
	getche()	Returns a character that has been recently typed. The typed character is also echoed to the computer screen. But the user is not required to press <i>Enter</i> .
	getch()	Returns a character that has been recently typed. But, neither the user is required to press <i>Enter</i> key nor the typed character is echoed to the computer screen.
	putchar()	Display a character on the screen.
	gets()	Accepts a string from the keyboard.
	puts()	Display a string on the screen.
Formatted	scanf()	Accepts formatted data from the keyboard.
	printf()	Displays formatted data on the screen.



The only difference between these functions is that the formatted functions permit the input from the keyboard or output sent to a screen to be formatted as per the requirements.

For example, if different values are to be displayed, how many columns on screen to be used, and how much space between two values is to be given. If a value to be displayed is of real type, then how many decimal places to output.

Listing 1.3

```
/*
    Program to demonstrate working of character I/O functions
*/
#include <stdio.h>
int main()
{
    char ch;
    printf( "\nEnter any character: " );
    ch = getchar();
    printf( "\nCharacter you typed is " );
    putchar(ch);
    printf( "\nEnter any character: " );
    ch = getche();
```



```
printf( "\nCharacter you typed is " );
putchar(ch);
printf( "\nEnter any character: " );
ch = getch();
printf( "\nCharacter you typed is " );
putchar(ch);
return 0;
}
```

Test Run

```
Enter any character: A↵
Character you typed is A
Enter any other character: x
Character you typed is x
Enter any other character:
Character you typed is H
```

Listing 1.4

```
/*
   Program to illustrate working of string I/O functions
*/
#include <stdio.h>
int main()
{
    char str[31];
    printf( "\nEnter string of length <= 30: " );
    gets( str );
    printf( "\nString you typed is " );
    puts( str );
    return 0;
}
```

Test Run

```
Enter string of length <= 30: Wel Come
String you typed is Wel Come
```

`scanf()` फंक्शन आम्हाला निर्दिष्ट स्वरूपात डेटा इनपुट करण्यास अनुमती देते. त्याचा सिन्टॅक्स हा आहेण्

```
scanf( "format string ", list of addresses of variables );
```

जेथे *format string* मध्ये फॉर्मॅट स्पेसिफायर्स असतात जे '%' कॅरेक्टर पासून सुरू होतात आणि स्पेस किंवा स्वल्पविरामाने विभक्त होतात.

व्हेरिएबलच्या अॅड्रेसची यादी वापरली जाते जेणेकरून `scanf()` फंक्शन कीवर्डमधून प्राप्त केलेला डेटा ठेवू शकेल. व्हेरिएबलचा पत्ता अॅड्रेस ऑपरेटर (कॅरेक्टर &) वापरून आणि `addressof` ऑपरेटर म्हणून घोषित केला जातो.

Table 1.9: सामान्यतः वापरल्या जाणार्या फॉर्मॅट स्पेसिफायर्सची यादी

Format Specifier	Used For
%d	signed decimal integer
%f	single precision floating real number
%lf	double precision floating real number
%c	single character
%s	single-word string

एन्क्वायरी करताना, इनपुट डेटा निर्दिष्ट स्वरूपात स्ट्रिंगनुसार काटेकोरपणे प्रविष्ट करणे आवश्यक आहे अन्यथा परिणाम फार विचित्र असू शकतात.

`printf()` फंक्शन एका निर्दिष्ट फॉर्मॅटमध्ये डेटा आउटपुट करण्यास अनुमती देते. त्याचा सिन्टॅक्स हा आहे.

```
printf( "format string ", list of variables );
```

जेथे *format string* मध्ये *format specifiers*, *escape sequences* टेक्स्ट डेटासह आउटपुट असणारा मजकूर असतो.

`scanf ()` फंक्शनसह वापरलेले सर्व फॉर्मॅट स्पेसिफायर `printf ()` फंक्शनसाठी वैध आहेत.

Listing 1.5

```
/*
   Program to demonstrate working of formatted I/O functions
*/
#include <stdio.h>
int main(void)
{
    int a;
    printf( "\nEnter values for a : " );
    scanf ( "%d", &a );
    printf( "\nValue of a = %d\n", a );
    return 0;
}
```

Test Run

Enter values for a : 150

Value of a = 150

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- संगणक एक इलेक्ट्रॉनिक मशीन आहे जे प्रोग्राम्स नावाच्या निर्देशांच्या संचानुसार कार्ये किंवा गणना करते.
- संगणक विविध प्रकारांमध्ये इनपुट प्राप्त करू शकतो, निर्देशानुसार या इनपुटवर प्रक्रिया करू शकतो आणि विविध प्रकारच्या फॉर्ममध्ये निकाल सादर करू शकतो.
- संगणक प्रणालीतील घटकांमध्ये इनपुट युनिट, आउटपुट युनिट, मेमरी युनिट, कंट्रोल युनिट, अरीथमेटिक आणि लॉजिक युनिट आणि सेकंडरी स्टोरेज युनिट असते.
- कंट्रोल युनिट, अरीथमेटिक आणि लॉजिक युनिट आणि मेमरी युनिट एकत्रितपणे सेंट्रल प्रोसेसिंग युनिट (सीपीयू) म्हणून ओळखले जातात.
- संगणकाशी कनेक्ट केलेले इनपुट डिव्हाइस आणि आउटपुट साधने यासारख्या सर्व बाह्य उपकरणांना फेरीफेरल्स या सामान्य नावाने ओळखले जाते.
- हार्डवेअर संगणकाच्या त्या भागास सूचित करते ज्यास आपण पाहू शकता आणि स्पर्श करू शकता, त्यामधील केस आणि त्यातील प्रत्येक गोष्टीसह.
- सॉफ्टवेअर, सर्वसाधारण अर्थाने सूचना किंवा प्रोग्रामचा एक संच आहे जे हार्डवेअरला काय करावे ते सांगते.
- ऑपरेटिंग सिस्टम (ओएस) संगणक प्रणालीची संसाधने व्यवस्थापित करते, आणि एक इंटरफेस प्रदान करते ज्याद्वारे युझर्स संगणकासह विविध कार्ये करण्यासाठी संवाद साधू शकतो.
- सॉफ्टवेअरला सिस्टम सॉफ्टवेयर, ॲप्लिकेशन सॉफ्टवेअर आणि युटिलिटी सॉफ्टवेयर म्हणून वर्गीकृत केले जाऊ शकते.
- बूट करणे ही एक प्रक्रिया आहे जी सिस्टमला आरंभ करण्यास दर्शवते.
- अल्गोरिदम एखाद्या विशिष्ट समस्येचे निराकरण करण्याच्या निर्देशांचे एक मर्यादित क्रम आहे.
- एक अल्गोरिदम फ्लोचार्ट आणि पसेडोकोड वापरून दर्शविला जाऊ शकतो.
- फ्लोचार्ट हे अल्गोरिदमचे एक आरेखात्मक प्रतिनिधित्व आहे, जिथे वेगवेगळ्या प्रकारचे भूमितीय आकार वेगवेगळ्या प्रकारच्या ऑपरेशन्सचे प्रतिनिधित्व करण्यासाठी वापरले जातात.
- पसेडोकोड हे अल्गोरिदममधील चरणांचे साधे भाषांतर आहे.
- सिंटॅक्स एरॉर भाषा नियमांच्या चुकीच्या वापराचा परिणाम आहेत.
- लॉजिकल एरॉर ही समस्या किंवा त्याचे निराकरण न समजल्यामुळे होते.
- C भाषा डेनिस रिची यांनी विकसित केली आहे.
- C भाषा ही एक मिडल लेवल भाषा आणि पोर्टेबल आहे.
- C भाषा ही केस संवेदनशील आहे.
- C प्रोग्रामच्या सामान्य संरचनेत पाच विभाग असतात - कमेंट्स, प्रीप्रोसेसर डिरॅक्टिव्ह, ग्लोबल डिक्लरेशन, main () फंक्शन आणि आवश्यकतेनुसार इतर फंक्शनस.
- C भाषेच्या मुख्य ॲप्लिकेशन्स क्षेत्रांमध्ये सिस्टम प्रोग्रामिंग, नेटवर्क प्रोग्रामिंग, जीयूआय, वैज्ञानिक व्हिज्युअलायझेशन, एम्बेडेड सिस्टम इत्यादींचा समावेश आहे.

- प्रोग्राम बनवण्याच्या वेगवेगळ्या पायऱ्या आहेत - क्रीएशन कॅम्पयलेशन, लिंकिंग आणि एक्झिक्युट.
- टेक्स्ट एडिटर हा एक प्रोग्राम आहे जो संगणक स्मृतीत प्रोग्राम प्रविष्ट करण्यास आणि बदलण्यात आणि शेवटी डिस्क फाइलमध्ये सेव्ह करण्यास मदत करतो.
- सौर्स कोड हा प्रोग्राम हा उच्च स्तरीय भाषेमध्ये सीसारखा लिहिला जातो आणि सौर्स फाइल म्हणून ओळखल्या जाणाऱ्या डिस्क फाइलमध्ये संग्रहित केला जातो..
- कंपाईलर एक प्रोग्राम आहे जो C प्रोग्रामचे भाषांतर मशीन भाषेत करतो. यात दोन घटक आहेत - प्रीप्रोसेसर आणि ट्रान्सलेटर. प्रीप्रोसेसर प्रोग्रामवर प्रक्रिया करतो आणि भाषांतरासाठी तयार करतो. ट्रान्सलेटर अंतिम भाषांतर करतो.
- सौर्स प्रोग्रामचे ऑब्जेक्ट प्रोग्राममध्ये भाषांतर करण्याची प्रक्रिया कॅम्पयलेशन म्हणून ओळखली जाते
- ऑब्जेक्ट कोड मशीन भाषेमधील प्रोग्राम आहे जो कंपाईलरद्वारे तयार केला जातो आणि ऑब्जेक्ट फाइल म्हणून ओळखल्या जाणाऱ्या फाइलमध्ये संग्रहित केला जातो.
- लिंकर एक प्रोग्राम आहे जो सिस्टम लायब्ररी आणि युझर्सद्वारे परिभाषित फंक्शनमधून लायब्ररी फंक्शन्सची मशीन भाषा जोडतो आणि अंतिम एक्झिक्युटेबल प्रोग्राम तयार करतो.
- एक्झिक्युटेबल कोड मशीन भाषेमधील प्रोग्राम आहे जो लिंकरद्वारे तयार केला जातो आणि रन फाइल म्हणून ओळखल्या जाणाऱ्या फाइलमध्ये संग्रहित केला जातो जो एक्झिक्युशन साठी तयार आहे.
- लोडर एक प्रोग्राम आहे जो एक्झिक्युटेबल प्रोग्रामला डिस्क फाइलमधून दुय्यम स्टोरेजवरील कॉम्प्यूटर मेमरीमध्ये स्थानांतरित करतो आणि एक्झिक्युशन साठी तयार करतो.
- प्रोग्रामच्या संभाव्य इनपुटसाठी प्रोग्राम योग्यरित्या कार्य करतो हे सुनिश्चित करण्यासाठी टेस्टिंग प्रक्रिया आहे..
- डीबगिंग हे दोष ओळखणे, शोधणे आणि निराकरण करण्याची प्रक्रिया आहे. प्रोग्राम डेव्हलपमेंट प्रक्रियेत ही स्वतंत्र क्रिया नाही; हे नेहमीच टेस्टिंगशी संबंधित असते.
- प्रोग्राम एक्झिक्युशनच्या वेळी बदलत नसलेल्या आयडेंटिफायरद्वारे थेट कोड केलेले किंवा प्रस्तुत केलेल्या मूल्याला कॉन्स्टन्ट म्हटले जाते, जिथे ती मूल्ये बदलू शकतात म्हणून नाव व्हेरिएबल.
- डेटा टाईप अनुज्ञेय ऑपरेटरच्या संचासह मूल्यांचा संच आहेण.
- C भाषेद्वारे समर्थित विविध डेटा प्रकारांना बिल्ट-इन डेटा प्रकार, derived डेटा प्रकार आणि युझर- डिफाइंड डेटा प्रकार म्हणून नावे दिलेली आहेत.
- प्रोग्रामद्वारे प्रक्रिया केलेली प्रत्येक डेटा आयटम डिक्लेअर करणे आवश्यक आहे..
- C प्रोग्राममधील इनपुट/आउटपुट फॉर्मॅटेड किंवा अनफॉर्मॅटेड केले जाऊ शकते.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. संगणक म्हणजे काय ?
2. ब्लॉक डायग्रामच्या मदतीने संगणकाच्या कार्यात्मक घटकांचे कार्य स्पष्ट करा.
3. जेव्हा आपण म्हणतो की मेमरी volatile आहे, तर याचा अर्थ काय आहे ?
4. हार्डवेअर व सॉफ्टवेअर मधील फरक लिहा.
5. ऑपरेटिंग सिस्टम म्हणजे काय ?
6. टर्म बूटिंगद्वारे आपणास काय समजते ?

7. सॉफ्टवेअरचे प्रकार काय आहेत?
8. अल्गोरिदम म्हणजे काय? चांगल्या अल्गोरिदमची इष्ट वैशिष्ट्ये कोणती?
9. फ्लोचार्ट म्हणजे काय? विविध फ्लोचार्ट चिन्हे वर्णन करा..
10. स्यूडोकोड म्हणजे काय?
11. एखादी निवड दिल्यास तुम्हाला फ्लोचार्ट किंवा स्यूडोकोड वापरून अल्गोरिदमचे प्रतिनिधित्व करायचे आहे का?
12. C प्रोग्रामच्या सामान्य रचनेचे वर्णन करा.
13. C प्रोग्रामच्या विकासाच्या विविध चरणांचे वर्णन करा.
14. C भाषेला मिडल लेवल भाषा का म्हणतात?
15. टोकन म्हणजे काय? सी मधील विविध टोकनचे वर्णन करा.
16. डेटा प्रकार म्हणजे काय? सी भाषेद्वारे समर्थित विविध डेटा प्रकारांची नावे द्या.
17. कॉन्स्टन्ट आणि व्हेरिएबल मधील फरक लिहा.
18. व्हेरिएबल्स घोषित (डिक्लेअर) करण्यास आणि प्रारंभ (इनिशियालाइज) करण्यासाठी सिंटॅक्सचे वर्णन करा.
19. C मधील इनपुट/आउटपुट हाताळण्यासाठी विविध फंक्शनस द्या.

एकाधिक निवड प्रश्न

1. जर एखाद्या प्लॅटफॉर्मवर कोणत्याही भाषेमध्ये लिहिलेला प्रोग्राम बदल न करता किंवा कमीतकमी बदलांसह दुसऱ्या प्लॅटफॉर्मवर नेला जाऊ शकतो, तर त्याला _____ भाषा असे म्हणतात.
 - (a) समजण्यायोग्य
 - (b) वाचनीय
 - (c) पोर्टेबल
 - (d) देखभाल करण्यायोग्य
2. सी भाषा कोणी विकसित केली?
 - (a) वॉन न्युमन
 - (b) डेनिस रिची
 - (c) पीटर नॉर्टन
 - (d) केन थॉम्पसन
3. C प्रोग्राममधील निर्देश समाप्त करण्यासाठी खालील पैकी कोणते कॅरेक्टर वापरले जाते?
 - (a) , (comma)
 - (b) : (colon)
 - (c) ; (semicolon)
 - (d) . (period)
4. Which पुढीलपैकी कोणते आयडेंटिफायरचे पहिले अक्षर असू शकत नाही?
 - (a) digit
 - (b) underscore
 - (c) alphabet
 - (d) uppercase alphabet
5. फंक्शनचा मुख्य भाग बंद करण्यासाठी खालीलपैकी कोणते वापरले जातात?
 - (a) []
 - (b) { }
 - (c) ()
 - (d) < >
6. C खालीलपैकी कोणत्या प्लॅटफॉर्मवर सी भाषा वापरली जाऊ शकते?
 - (a) Unix
 - (b) Windows
 - (c) Linux
 - (d) All of the above
7. C भाषा ही एक _____ भाषा आहे.
 - (a) हाय
 - (b) लो
 - (c) मिडल
 - (d) सिम्बॉलिक
8. परिवर्णी शब्द ANSI म्हणजे _____.
 - (a) अमेरिकन नॅशनल स्टँडर्ड इंटरनॅशनल
 - (b) अमेरिकन नॅशनल सॉफ्टवेअर इनकॉर्पोरेशन
 - (c) अमेरिकन नॅशनल स्टँडर्ड इंस्टिट्यूट
 - (d) अमेरिकन नॅशनल स्टँडर्ड इन्स्ट्रक्शन्

9. विषम पद शोधा
 (a) सौर्स कोड (b) ऑब्जेक्ट कोड (c) एक्झिक्यूटेबल कोड (d) ASCII कोड
10. याची खात्री करण्यासाठी प्रोग्रामची टेस्टिंग केली जाते
 (a) त्यात कोणतीही सिंटॅक्स एरॉर असू नये (b) हे त्याचे इच्छित कार्य करते
 (c) हे यशस्वीरित्या कॅम्पयलेशन करते (d) हे अमर्यादपणे चालवू नये
11. बग्स शब्द म्हणजे _____ होय.
 (a) सिंटॅक्स एरॉर (b) लॉजिकल एरॉर (c) रनटाइम एरॉर (d) वरील सर्व
12. परिवर्णी शब्द ASCII म्हणजे _____.
 (a) अमेरिकन स्टॅंडर्ड कोड फॉर इंटरनॅशनल इन्फॉर्मेशन
 (b) अमेरिकन सिस्टिम फॉर कंपायलर इन्फॉर्मेशन इंटरनॅशनल
 (c) अमेरिकन सिस्टिम फॉर कोड इन्फॉर्मेशन इंटरनॅशनल
 (d) अमेरिकन स्टॅंडर्ड कोड फॉर इन्फॉर्मेशन इंटरचेंज
13. खालीलपैकी कोणती विधाने सी भाषेबद्दल खरी नाहीत?
 (a) प्रत्येक सूचना अर्धविराम द्वारे संपुष्टात आणली जाते
 (b) हे असंवेदनशील आहे
 (c) कमेंट्स प्रोग्राम कोडमध्ये कोठेही ठेवल्या जाऊ शकतात
 (d) यात हाय लेवल आणि लो लेवल भाषेची वैशिष्ट्ये आहेत
14. डीबगिंग टर्म म्हणजे _____ प्रक्रियेस संदर्भित करते.
 (a) ऑब्जेक्ट कोडमध्ये सौर्स कोड भाषांतरित करणे (b) ऑब्जेक्ट कोडला सिस्टम लायब्ररीत जोडणे
 (c) बग्स ओळखणे आणि त्यांचे निराकरण करणे (d) वरीलपैकी कोणतेही नाही
15. पोर्टेबिलिटी या शब्दाचा अर्थ _____ असा आहे.
 (a) प्रोग्राम कोड समजण्यायोग्य आहे (b) प्रोग्राम कोड देखभाल करण्यायोग्य आहे
 (c) प्रोग्राम कोड बग फ्री आहे
 (d) प्रोग्राम कोड दुसऱ्यामध्ये कमीतकमी बदलांशिवाय/न बदलांशिवाय ट्रान्सपोर्ट केला जाऊ शकते.
16. C प्रोग्राम _____ वापरून मशीन भाषेत रूपांतरित केले गेले.
 (a) Assembler (b) Interpreter
 (c) Compiler (d) Operating system
17. टर्बो C/C++ कंपाइलरद्वारे निर्मित मशीन कोड असलेल्या फाईलचा एक्सटेंशन _____ आहे.
 (a) *.com (b) *.exe (c) *.c (d) *.obj
18. खालीलपैकी कोणता कीवर्ड नाही?
 (a) for (b) main (c) break (d) else
19. पुढीलपैकी कोणता C भाषेमध्ये वैध डेटा प्रकार नाही?
 (a) Char (b) float (c) long (d) double
20. पुढील पैकी कोणते C मधील primitive डेटा मूल्य दर्शवत नाही?
 (a) "a" (b) 'k' (c) 35.25 (d) 14

ANSWERS													
1.	(c)	2.	(b)	3.	(c)	4.	(a)	5.	(b)	6.	(d)	7.	(c)
8.	(c)	9.	(d)	10.	(b)	11.	(d)	12.	(d)	13.	(b)	14.	(c)
15.	(d)	16.	(c)	17.	(d)	18.	(b)	19.	(a)	20.	(a)		

संगणकीय समस्या

अल्गोरिदम विकसित करा आणि फ्लोचार्ट आणि/किंवा स्यूडोकोड वापरून त्याचे प्रतिनिधित्व करा:

1. दिलेले वर्ष लीप वर्ष आहे की नाही याची चाचणी करणे.
2. दिलेली तारीख वैध आहे की नाही याची चाचणी घेणे.
3. नैसर्गिक संख्येतील सर्वात मोठा अंक 'n' शोधण्यासाठी.
4. नैसर्गिक संख्येतील सर्वात छोटा अंक शोधण्यासाठी 'n'.
5. nth प्राइम नंबर शोधा.
6. लिकोणा 'a', 'b' आणि 'c' लांबीच्या तीन रेषाखंड तयार केले जाऊ शकतात का हे तपासण्यासाठी (मूल्य हे cms च्या)
7. डिग्री मध्ये 'a', 'b' आणि 'c' उपायांसह दिलेल्या तीन कोनात लिकोण तयार केला जाऊ शकतो की नाही हे तपासण्यासाठी.
8. दोन ओळी एकमेकांना छेदतात की नाही याची दोन चाचणी करा. एका ओळीवर दोन बिंदू A (xa, ya) आणि B (xb, yb) आणि दुसऱ्या ओळीवर C (xc, yc) आणि D (xd, yd) दिले.
9. बॉडी मास इंडेक्स (BMI) एखाद्या व्यक्तीचा काढला जातो म्हणून $BMI = \frac{W}{H^2}$, जेथे W वजन किलोग्रॅम (kilograms) आणि H मीटर उंची आहे.

BMI	Classification
< 18.5	Under weight
24.9-18.5	Normal weight
29.9-25.0	Overweight
34.9-30.0	Class I obesity
39.9-35.0	Class II obesity

ज्याचे वजन आणि उंची दिली आहे अशा व्यक्तीचे लढपणाचे वर्गीकरण करा.

10. मासिक टेलिफोन बिल खालीलप्रमाणे मोजले जाईल:

Minimum ₹ 200 for upto 100 calls

plus ₹ 0.60 per call for next 50 calls

plus ₹ 0.50 per call for next 50 calls

plus ₹ 0.40 per call for any call beyond 200 calls.

दिलेल्या कॉलच्या संख्येसाठी मासिक बिल मोजा.

प्रॅक्टिकल

1. पीसी/लॅपटॉपच्या क्रियांशी परिचित होणे: बूट करण्याची संकल्पना, डेस्कटॉप कॉन्फिगर करणे, फायली आणि फोल्डर्ससह कार्य करणे, सॉफ्टवेअर स्थापित करणे आणि विविध ॲप्लिकेशन्स चालवणे.

2. प्रोग्रामिंग वातावरणाशी परिचित होणे: C प्रोग्राम तयार करणे आणि संपादित करणे, एडिटिंग करणे आणि एक्झिक्युट करणे.

आणखी माहिती

समस्या सोडवणे हे एक कौशल्य आहे आणि हे कौशल्य रात्रीतून शिकता येत नाही. म्हणूनच, विद्यार्थ्यांमध्ये आवश्यक कौशल्ये विकसित करण्यासाठी, शिक्षकांनी योग्य उदाहरणे घेऊन आणि निराकरण करण्यासाठी विद्यार्थ्यांना सामील करून समस्येचे निराकरण करण्याचा दृष्टीकोन दर्शविला पाहिजे. शेवटी, शिक्षकांनी संबंधित मुबलक समस्या द्याव्या जेणेकरून विद्यार्थ्यांनी पूर्वी पाहिलेल्या किंवा सोडवलेल्या समस्यांचे निराकरण करण्यासाठी शिकलेल्या संकल्पना लागू करू शकतील.

संदर्भ आणि सूचविलेले वाचन

1. R.G. Dromey, How to solve it by Computer, Pearson Education, 2008.
2. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
3. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi..
4. V. Anton Spraul, Think Like a Programmer, No Starch Press.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

2

अरीथमेटिक एक्सप्रेसन आणि प्रिसिडन्स

युनिट वैशिष्ट्ये

या युनिटमध्ये विविध प्रकारचे ऑपरेटर, त्यांची प्रिसिडन्स आणि अससोसिएटिव्हिटी, विविध प्रकारचे एक्सप्रेसन, एक्सप्रेसन चे मूल्यमापन नियंत्रित करणारे नियम आणि लायब्ररी च्या कार्याशी संबंधित विषयांवर चर्चा केली आहे.

तर्कशास्त्र

विज्ञान आणि अभियांत्रिकीशी संबंधित समस्यांमध्ये प्रोग्रामरला वेगवेगळ्या अंकगणित अभिव्यक्तींचा सामना करावा लागतो ज्यामध्ये विविध अंकगणित ऑपरेटर्स आणि मानक गणितीय आणि त्रिकोणमितीय कार्य असू शकतात. त्या बीजगणित अभिव्यक्त्यांना C मधील त्यांच्या समकक्ष अभिव्यक्तींमध्ये रूपांतरित करण्यासाठी आणि त्यांचे मूल्यांकन क्रम नियंत्रित करण्यासाठी प्रोग्रामरला सीद्वारे समर्थित विविध ऑपरेटर, अभिव्यक्तींचे मूल्यांकन कसे केले जाते आणि इतर संबंधित प्रकरणांचे ज्ञान असावे.

हे युनिट विद्यार्थ्यांना ऑपरेटर, एक्सप्रेसन आणि त्यांचे मूल्यांकन यांच्याशी संबंधित विविध पैलू समजण्यास मदत करते.

पूर्व-आवश्यकता

- रेखीय बीजगणित (Linear algebra)
- मानक गणिती कार्ये (Standard mathematical functions)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U2-O1:	विविध ऑपरेटर आणि त्यांचा उपयोग समजावून सांगा.
U2-O2:	बीजगणितिक एक्सप्रेसन C एक्सप्रेसन मध्ये रूपांतरित करा.
U2-O3:	एक्सप्रेसन चे मूल्यांकन कसे केले जाते ते सांगा.
U2-O4:	ऑपरेटरमध्ये प्रिसिडन्स आणि अससोसिएटिव्हिटी ही संकल्पना समजावून सांगा.
U2-O5:	असाईनमेंट आणि एक्सप्रेसन्स मध्ये टाइप कन्वर्जन समजावून सांगा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 2 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U2-O1	1							
U2-O2		1						
U2-O3			2					
U2-O4			2					
U2-O5			1					

2.1 ओळख (INTRODUCTION)

ऑपरेटर भाषेचे क्रियापद असतात जे युझर्स मूल्यांवर मोजणी करू देतात. आपण त्या ऑपरेटर्सचा क्रियापद म्हणून आणि ऑपरेन्ड्सचा ऑब्जेक्ट आणि सब्जेक्ट म्हणून विचार करू शकता. C भाषेचा ऑपरेटरचा समृद्ध संच त्याच्या विशिष्ट वैशिष्ट्यांपैकी एक आहे.

एक्सप्रेशन हे मूल्य मोजण्यासाठी ऑपरेन्ड्स आणि ऑपरेटर एकत्र जोडलेले एक सूत्र आहे. संगणकीय मूल्ये भविष्यातील वापरासाठी व्हेरिएबलमध्ये संग्रहित केली जाणे आवश्यक आहे. हे असाईनमेंट स्टेटमेंटद्वारे केले जाते.

संगणक प्रोग्राममध्ये, हातातील समस्येच्या आवश्यकतेनुसार विविध कंप्यूटेशन्स केले जातात. तरीही, काही रूटीन प्रकारांची संगणने आहेत जी बहुधा प्रत्येक प्रोग्रामचा भाग असू शकतात. म्हणून एक सोयीसाठी, प्रत्येक आधुनिक प्रोग्रामिंग भाषा लायब्ररी फंक्शन्सच्या रूपात या रूटीन प्रकारांच्या संगणनांसाठी समर्थन प्रदान करते.

हे युनिट C भाषेच्या या अर्थपूर्ण सिमॅंटिक (semantic) युनिट्सचे वर्णन करण्याचा प्रयत्न करते.

2.2 ऑपरेटर (OPERATORS)

ऑपरेटर कोणत्याही प्रोग्रामिंग भाषेचा पाया असतात. ऑपरेटरच्या वापराशिवाय कोणत्याही प्रोग्रामिंग भाषेची कार्यक्षमता अपूर्ण आहे.

ऑपरेटर सामान्यतः दोन प्रकारचे असतात:

- **युनेरी ऑपरेटर** - ऑपरेटर जे सिंगल ऑपरेन्डवर चालतात. युनिअरी ऑपरेटर त्यांच्या ऑपरेन्डसह प्रीफिक्स केलेले असतात. उदाहरणार्थ, $-x$, हे एकरी वजा $-$ ऑपरेटर त्याच्या ऑपरेन्ड x च्या आधी (एक संख्यात्मक मूल्य ठेवलेला असावा) आणि त्याचे मूल्य दुर्लक्षित करते, म्हणजेच त्याचे चिन्ह बदलते.
- **बायनरी ऑपरेटर** - ऑपरेटर जे दोन ऑपरेन्डसह ऑपरेट करतात. बायनरी ऑपरेटर त्यांच्या ऑपरेन्ड्स दरम्यान एम्बेड केलेले आहेत.

उदाहरणार्थ, $a + b$, येथे बायनरी प्लस $+$ ऑपरेटर त्याच्या ऑपरेन्ड्स a आणि b दरम्यान दिसून येईल (दोन्ही अंकीय मूल्ये धारण करतात) आणि व्हेरिएबल b चे मूल्य जोडेल ऑपरेटरचा C भाषेचा समृद्ध संच त्याच्या विशिष्ट वैशिष्ट्यांपैकी एक आहे.

हे ऑपरेटर पुढील श्रेणींमध्ये मोजले जाऊ शकतात:

- अरीथमेटिक ऑपरेटर ($+$, $-$, $*$, $/$, $\%$)
- रिलेशनल ऑपरेटर ($<$, $<=$, $>$, $>=$, $==$, $!=$)
- लॉजिकल ऑपरेटर ($!$, $\&\&$, $||$)
- बिटवाईस ऑपरेटर (\sim , $>>$, $<<$, $\&$, $|$, \wedge)
- स्पेशल ऑपरेटर
 - इन्क्रीमेंट आणि डिक्रीमेंट ऑपरेटर ($++$, $--$)
 - साईझऑफ ऑपरेटर
 - ऍड्रेसऑफ ऑपरेटर ($\&$)
 - इंडिरेक्शनध डी - रेफरेंस ऑपरेटर ($*$)
 - टर्नरी/कंडिशनल ऑपरेटर ऑपरेटर ($?:$)

या युनिटमध्ये, आम्ही प्रामुख्याने अंकगणित मोजणीत वापरल्या जाणाऱ्या ऑपरेटरवर लक्ष केंद्रित करू, विषयातील पूर्णतेसाठी इतर प्रकारचे ऑपरेटर सादर केले जातील.

2.2.1 अरीथमेटिक ऑपरेटर (Arithmetic Operators)

अरीथमेटिक ऑपरेटर गणिती ऑपरेशन जसे की बेरीज, वजाबाकी, गुणाकार, भागाकार इ. करण्यासाठी करतात.

Table 2.1: बायनरी अरीथमेटिक ऑपरेटर

Operator	Symbol	Form	Operation
Addition	+	$x + y$	Adds value of y to value of x .
Subtraction	-	$x - y$	Subtracts value of y from value of x .
Multiplication	*	$x * y$	Multiplies value of x by value of y .
Division	/	x / y	Divides value of x by value of y .
Modulus	%	$x \% y$	Divides value of x by value of y and gives remainder.

इन्टिजर अरीथमेटिक (Integer Arithmetic)

जेव्हा दोन्ही ऑपरेंड पूर्णांक असतात तेव्हा ऑपरेशनसना इन्टिजर अरीथमेटिक म्हटले जाईल आणि नेहमी इन्टिजर मूल्य मिळेल.

पुढील विधानांचा विचार करा

`int x = 13, y = 5;`

मग आमच्याकडे खालील परिणाम आहेत:

$x - y = 8$

$x + y = 18$

$x * y = 65$

$x / y = 2$ (decimal part truncated)

$x \% y = 3$ (remainder of division)

रिअल अरीथमेटिक (Real Arithmetic)

जेव्हा दोन्ही ऑपरेंड रिअल संख्या असतात तेव्हा ऑपरेशनसला रिअल अरीथमेटिक म्हटले जाईल आणि नेहमीच रिअल मूल्य मिळेल. रिअल संख्या (फ्लोटिंग-पॉइंट व्हॅल्यूज) ही राऊंडेड संख्या असल्याने अरीथमेटिक क्रियांचा निकाल योग्य परिणामाचे अंदाजे मूल्य आहे.

ते आहे

$6.0 / 7.0 = 0.857143$

$-1.0 / 3.0 = -0.333333$

$3.0 / -2.0 = -1.500000$

मॉड्यूलस ऑपरेशन रिअल ऑपरेंडसह वापरले जाऊ शकत नाही..

मिक्सड-मोड अरीथमेटिक (Mixed-mode Arithmetic)

जेव्हा एक ऑपरेंड रिअल असेल आणि दुसरा इन्टिजर असेल तेव्हा ऑपरेशनसला मिक्सड-मोड अरीथमेटिक म्हटले जाईल आणि नेहमीच रिअल मूल्य मिळेल. इन्टिजर ऑपरेंड रिअल ऑपरेंडमध्ये रूपांतरित होते आणि नंतर रिअल अरीथमेटिक केले जाते ज्यायोगे रिअल मूल्य प्राप्त होते.

ते आहे

$$6 / 7.0 \rightarrow 6.0 / 7.0 = 0.857143$$

$$-1.0 / 3 \rightarrow -1.0 / 3.0 = -0.333333$$

लक्षात ठेवा की

$$15 / 10.0 = 1.5$$

$$15.0 / 10 = 1.5$$

जेथे

$$15 / 10 = 1$$

2.2.2 रिलेशनल (कम्पॅरिझन) ऑपरेटर (Relational (Comparison) Operators)

हे ऑपरेटर निर्णय घेण्यास सुलभ करण्यासाठी सुसंगत असणे आवश्यक असलेल्या ऑपरेंडच्या मूल्यांची तुलना करण्यासाठी वापरले जातात. जर तुलना यशस्वी झाली तर ते मूल्य 1 परत करेल (बुलियन मूल्य true) आणि जर तुलना अयशस्वी झाली तर ते मूल्य 0 (बुलियन मूल्य false) परत करेल.

Table 2.2: रिलेशनल (कम्पॅरिझन) ऑपरेटर

Operator	Symbol	Form	Meaning
Greater than	>	$x > y$	Return True if x is greater than y.
Greater than or equal to	>=	$x >= y$	Return True if x is greater than or equal to y.
Less than	<	$x < y$	Return True if x is less than y.
Less than or equal to	<=	$x <= y$	Return True if x is less than or equal to y.
Equal to	==	$x == y$	Return True if x and y are equal.
Not equal to	!=	$x != y$	Return True if x and y are not equal.



Note that in C language, the non-zero value is treated as equivalent to Boolean True and zero value as equivalent to Boolean False. Further, there is no support for Boolean data as such in C language in old versions of C language. The new version of C language, known as C99 standard, supports the Boolean type.

2.2.3 लॉजिकल ऑपरेटर (Logical Operators)

हे ऑपरेटर रिलेशनल ऑपरेटरच्या सहाय्याने तयार केलेल्या दोन किंवा अधिक सोप्या परिस्थितीत सामील होऊन कंपाऊंड स्थिती तयार करण्यासाठी वापरले जातात.

Table 2.3: लॉजिकल ऑपरेटर

Operator	Symbol	Form	Meaning
Logical AND	&&	$x \&\& y$	Return True if both the operands are True.
Logical OR		$x y$	Return True if either of the operands is True.
Logical NOT	!	! x	Return True if operand is false, and False if operand True (complements the operand).

2.2.4 बिटवाईस ऑपरेटर (Bitwise Operators)

बिटवाईस ऑपरेटर ऑपिन्डर्सवर कार्य करतात जणू ते बायनरी अंकांचे स्ट्रिंग्स असतात. हे बिट-बिट वर ऑपरेट करते, म्हणूनच बिटवाईस हे नाव. लक्षात घ्या की हे ऑपरेटर केवळ अविभाज्य (integral) ऑपरेंडसह वापरले जातात. बिटवाइज ऑपरेटरचे कार्य दर्शविण्यासाठी, $x = 10$ (बायनरीमध्ये 0000 1010) आणि $y = 4$ (बायनरीमध्ये 0000 0100) असे गृहित धरू की 8 बिट्स इन्टिजर संख्येचे प्रतिनिधित्व करण्यासाठी वापरले जातात.

Table 2.4: बिटवाईस ऑपरेटर

Operator	Symbol	Form	Meaning	Example
Bitwise AND	&	$x \& y$	Return 1 if corresponding bits are 1 else 0.	$x = 0000\ 1010$ (10) $y = 0000\ 0100$ (4) $x \& y = 0000\ 0000$ (0)
Bitwise OR		$x y$	Return 1 if either or both of corresponding bits are 1 else 0.	$x = 0000\ 1011$ (11) $y = 0000\ 0101$ (5) $x y = 0000\ 1111$ (15)
Bitwise XOR	^	$x \wedge y$	Return 1 if corresponding bits are not equal.	$x = 0000\ 1011$ (11) $y = 0000\ 0101$ (5) $x \wedge y = 0000\ 1110$ (14)
Bitwise NOT	~	$\sim x$	Inverts 1 to 0 and 0 to 1	$x = 0000\ 1011$ (11) $\sim x = 1111\ 0100$ (-12)
Bitwise right shift	>>	$x \gg y$	Shifts the bits of left operand to right by number of bits specified by the right operand. The right most bit is shifted out after each shift, while the left most bit is restored. Note: Each right shift by 1-bit is equivalent to integral division by 2.	$x = 0000\ 1011$ (11) $x \gg 1 = 0000\ 0101$ (5)
Bitwise left shift	<<	$x \ll y$	Shifts the bits of left operand to left by number of bits specified by the right operand. The left most bit is shifted out after each shift, while the right most bit is filled with 0. Note: Each left shift by 1-bit is equivalent to multiplication by 2.	$x = 0000\ 1011$ (11) $x \ll 1 = 0001\ 0110$ (22)

2.2.5 स्पेशल ऑपरेटर (Special Operators)

चला त्या प्रत्येकाबद्दल थोडक्यात चर्चा करूया आणि त्यांची उपयुक्तता तुम्हाला त्यानंतरच्या विभागांत व अध्यायांत समजेल.

2.2.5.1 इन्क्रीमेंट आणि डिक्रीमेंट ऑपरेटर (Increment & Decrement Operators)

C भाषा दोन अतिशय उपयुक्त ऑपरेटरला समर्थन देते - इन्क्रीमेंट ऑपरेटर (++) आणि डिक्रीमेंट ऑपरेटर (--). हे ऑपरेटर सर्व मूलभूत डेटा प्रकारांसह वापरले जाऊ शकतात. इन्क्रीमेंट ऑपरेटर ऑपरेंडमध्ये 1 जोडते, तर डिक्रीमेंट ऑपरेटर ऑपरेंडमधून 1 वजा करते.

दोन्ही ऑपरेटर युनिअरी ऑपरेटर आहेत आणि खाली दर्शविल्याप्रमाणे प्रिफिक्स तसेच पोस्टफिक्स नोटेशन मध्ये वापरले जाऊ शकतात:

```
++k;      or      k++;
--k;      or      k--;
```

येथे ++ k (तसेच k++) $k = k + 1$ किंवा $k += 1$ हे दोन्ही समान आहे आणि k-- (तसेच k--) $k = k - 1$ किंवा $k -= 1$ हे दोन्ही समान आहेत.

हे ऑपरेटर कंट्रोल व्हेरिएबल्स म्हणून असताना आणि लूपसाठी सर्वाधिक वापरले जातात.

जेव्हा ++ k किंवा k++ चा अर्थ असा असतो जेव्हा ते स्वतंत्रपणे विधान तयार करतात तेव्हा ते इतर एक्झिक्युशनचा भाग म्हणून वापरतात तेव्हा ते भिन्न वर्तन करतात.

पुढील विधानांचा विचार करा

```
int y, k = 5;
y = ++k;
```

या प्रकरणात प्रथम k चे मूल्य वाढवून नंतर y ला दिले जाईल आणि म्हणून y आणि k ची मूल्ये 6 असेल. अशा प्रकारे, वरील विधाने खालील विधानांच्या बरोबरीची असतात.

```
int y, k = 5;
++k;
y = k;
```

तथापि, जर आपण वरील विधाने म्हणून लिहिले तर

```
int y, k = 5;
y = k++;
```

प्रथम k चे वर्तमान मूल्य (मूल्य 5) y ला दिले जाईल आणि नंतर k चे मूल्य वाढविले जाईल आणि म्हणून y चे मूल्य 5 असेल तर k चे मूल्य 6 असेल.

अशा प्रकारे वरील विधाने खालील विधानांच्या बरोबरीची असतात

```
int y, k = 5;      int y, k = 5;
y = k;              or      y = k;
++k;                k++;
```

2.2.5.2 साईझऑफ ऑपरेटर (The sizeof Operator)

साईझऑफ ऑपरेटर दिलेल्या ऑपरेंडचा साईझ बाइटमध्ये परत करतो. साईझऑफ ऑपरेटरचा सिंटॅक्स हा आहे.

```
sizeof( exp )
```

जेथे exp डेटा प्रकार (बिल्ट-इन किंवा युझर-डिफाइंड), लिटरल्स/कॉन्स्टन्ट किंवा व्हेरिएबल दर्शवते.

उदाहरणार्थ

```
sizeof( float )
```

रिटर्न मूल्य 4

Table 2.5: साईझऑफ ऑपरेटरचा वापर

sizeof Operator used as	Returns	Justification
sizeof(12)	2	Integer constant by default belong to data type <i>int</i> , and size for <i>int</i> is 2 on Turbo C/C++ and 4 on Dev C++.
sizeof('a')	1	Size of Character constant is 1.
sizeof(int)	2	Size of data type <i>int</i> is 2 on Turbo C/C++ and 4 on Dev C++.
sizeof(125.25)	8	Real constant by default belong to data type <i>double</i> .
sizeof(125.25F)	4	Real constant is of data type <i>float</i> , and size for <i>float</i> is 4.
double x; sizeof(x);	8	Variable <i>x</i> is of type <i>double</i> , and size for <i>double</i> is 8.

2.2.5.3 ऍड्रेसऑफ ऑपरेटर (The *addressof* Operator)

कॅरेक्टर ‘&’ सह प्रिफिक्स केल्यावर व्हेरिएबल मेमरी स्थानांचा पत्ता परत करते, म्हणूनच त्याचे नाव ऍड्रेसऑफ ऑपरेटर आहे. हे scanf() फंक्शनमध्ये आणि पॉइंटर्स प्रारंभ करण्यासाठी वापरले जाते. ऍड्रेसऑफचा वापर (&) ऑपरेटर मध्ये Table 2.6.

दर्शविला आहेण्

Table 2.6: ऍड्रेसऑफ ऑपरेटरचा वापर

Addressof (&) Operator used as	Task Performed
int y = 10; int *py; py = &y;	First statement declares and defines variable <i>y</i> of type <i>int</i> . It also initializes it with value 10. Second statement declares and defines a pointer variable <i>py</i> that can hold an address of a memory location reserved for storing value of type <i>int</i> . Third statement assigns the address of variable <i>y</i> to pointer variable <i>py</i> .
int x, y; scanf("%d %d", &x, &y);	First statement declares and defines two variable <i>x</i> and <i>y</i> of type <i>int</i> . Second statement takes two integer values as input from the keyboard and stores those values at addresses of variable <i>x</i> and <i>y</i> , respectively.

2.2.5.4 इंडिरेक्शन ऑपरेटर (Conditional/Ternary Operator)

‘पॉइंटर व्हेरिएबल’ सह प्रीफिक्स केल्यावर कॅरेक्टर ‘*’ मेमरी स्थानात संग्रहित केलेले मूल्य परत मिळवते ज्यांचा पत्ता पॉइंटर व्हेरिएबलमध्ये ठेवला आहे. म्हणजेच पॉइंटर व्हेरिएबलच्या माध्यमातून अप्रत्यक्षपणे व्हॅल्यू मिळू शकते, म्हणून त्याचे नाव इंडिरेक्शन ऑपरेटर.

पुढील विधानांवर विचार करा

```
int y = 10, x;
int *py;
py = &y;
x = *py;
```

“*py” एक्सप्रेसन *py* ऍड्रेस ची व्हॅल्यू मिळवते जी पॉइंटर व्हेरिएबल *py* मध्ये असते, म्हणजेच व्हेरिएबलची व्हॅल्यू 10 जी व्हेरिएबल *x* ला दिलेली असते.

2.2.5.5 कंडिशनल किंवा टर्नरी ऑपरेटर (Conditional/Ternary Operator)

स्यूडोकोडमध्ये लिहिलेल्या पुढील विभागाचा विचार करा

```
if ( a > b ) then
    set big = a
else
    set big = b
endif
```

हे विभाग a आणि b चे जास्तीत जास्त मूल्य निर्दिष्ट करते. हे स्पष्ट आहे की व्हेरिएबल big नियुक्त केलेले मूल्य चाचणी स्थिती “a > b” च्या परिणामावर अवलंबून असेल.

अशा एक्सप्रेशनना कंडिशनल एक्सप्रेशन म्हणून ओळखले जाते आणि कंडिशनल ऑपरेटरच्या सहाय्याने लिहिले जाऊ शकते. टर्नरी ऑपरेटर वापरण्याचा सिन्टॅक्स हा आहे.

```
exp1 ? exp2 : exp3;
```

जेथे exp1, exp2, exp3 हे एक्सप्रेशन असते.

exp 1 एक्सप्रेशन चे मूल्यांकन प्रथम केले जाते. जर ते शून्य (सत्य) नसेल तर एक्सप्रेशन exp 2 चे मूल्यांकन केले जाईल आणि ही कंडिशनल एक्सप्रेशन चे मूल्य आहेत; अन्यथा एक्सप्रेशन exp 3 चे मूल्यांकन केले जाते आणि हे कंडिशनल एक्सप्रेशन मूल्य आहे. लक्षात घ्या की exp 2 आणि exp 3 यापैकी केवळ एकाचे मूल्यमापन केले गेले आहे.

कंडिशनल ऑपरेटर वापरून वरील सूडोकोड सेगमेंट असे लिहिले जाऊ शकते.

```
big = ( a > b ) ? a : b;
```

कंडिशनल एक्सप्रेशनच्या प्रथम एक्सप्रेशनच्या आसपास पॅरेंथेसिस आवश्यक नसतात कारण ‘?:’ चे प्रिसिडन्स खूप कमी आहे. तथापि, कंस वापरण्याची शिफारस केली जाते कारण ते अट पाहणे सुलभ करतात.

2.2.5.6 असाइनमेंट ऑपरेटर (Assignment operators)

असाइनमेंट ऑपरेटर व्हेरिएबल्सला मूल्य (व्हॅल्यूज) देण्यासाठी वापरतात. उदाहरणार्थ,

```
a = 5
```

येथे ‘=’ एक सोपा असाइनमेंट ऑपरेटर आहे जो व्हेरिएबल a साठी 5 ची मूल्य देतो. हे दुसऱ्या व्हेरिएबलचे किंवा एक्सप्रेशनचे मूल्य प्रदान करण्यासाठी देखील वापरले जाऊ शकते.

C प्रमाणे विविध कंपाऊंड ऑपरेटर आहेत जसे

```
a += 5
```

हे व्हेरिएबलला जोडेल आणि नंतर तेच असाइन करेल. ते ह्या सारखे आहे.

```
a = a + 5
```

‘+ =’ प्रकारच्या ऑपरेटरला शॉर्टहँड असाइनमेंट ऑपरेटर देखील म्हणतात.

Table 2.7: असाइनमेंट ऑपरेटर

Operator	Example	Equivalent to	Operator	Example	Equivalent to
=	x = 5	x = 5	&=	x &= 5	x = x & 5
+=	x += 5	x = x + 5	=	x = 5	x = x 5
-=	x -= 5	x = x - 5	^=	x ^= 5	x = x ^ 5
*=	x *= 5	x = x * 5	>>=	x >>= 5	x = x >> 5
/=	x /= 5	x = x / 5	<<=	x <<= 5	x = x << 5
%=	x %= 5	x = x % 5			

2.3 एक्सप्रेसनस (EXPRESSIONS)

एक्सप्रेसन म्हणजे मूल्य मोजण्यासाठी एक किंवा अधिक ऑपरेण्ड्स आणि शून्य किंवा अधिक ऑपरेटर एकत्र जोडलेले सूत्र असते. ऑपरेंड फंक्शन संदर्भ, व्हेरिएबल, और घटक किंवा कॉन्स्टन्ट असू शकतो.

उदाहरणार्थ एक्सप्रेसन मध्ये

```
a + b
```

प्लस कॅरेक्टर '+' एक ऑपरेटर आहे आणि a आणि b ऑपरेण्ड्स आहेत.

C मध्ये चार प्रकारचे एक्सप्रेसन आहेत. हे आहेत ते

1. अरीथमेटिक एक्सप्रेसनस (Arithmetic expressions)
2. रिलेशनल एक्सप्रेसनस (Relational expressions)
3. लॉजिकल एक्सप्रेसनस (Logical expressions)
4. कंडिशनल एक्सप्रेसनस (Conditional expressions)

प्रत्येक प्रकारचे एक्सप्रेसन विशिष्ट प्रकारचे ऑपरेंड घेतात आणि ऑपरेटरचा एक विशिष्ट संच वापरतात. प्रत्येक एक्सप्रेसनचे मूल्यांकन विशिष्ट प्रकारच्या मूल्याचे उत्पादन करते. एक्सप्रेसन ही विधाने नसतात परंतु विधानांचे घटक असू शकतात.

उदाहरणार्थ, ही ओळ विचारात घ्या

```
x = 2.0/3.0 + a * b;
```

संपूर्ण ओळ एक विधान आहे, परंतु असाइनमेंट ऑपरेटर नंतरचा भाग म्हणजे एक एक्सप्रेसन.

या विभागात, आमची चर्चा अरीथमेटिक एक्सप्रेसन पुरती मर्यादित असेल.

2.3.1.1 अरीथमेटिक एक्सप्रेसन (Arithmetic Expressions)

अरीथमेटिक एक्सप्रेसन हे ऑपरेण्ड्स आणि अरीथमेटिक ऑपरेटरची बनलेली असते. हे इंट, फ्लोट किंवा डबल (int, float or double) प्रकाराचे मूल्य निर्माण करते. जेव्हा एखाद्या एक्सप्रेसनमध्ये केवळ इन्टिजर ऑपरेण्डचा समावेश असतो तेव्हा ते शुद्ध इन्टिजर (pure integer) म्हणून ओळखले जाते, जेव्हा त्यात केवळ रीअल ऑपरेण्ड्स असतात तेव्हा ती शुद्ध रीअल (pure real) एक्सप्रेसन म्हणून ओळखली जाते आणि जेव्हा त्यात इन्टिजर आणि रीअल दोन्ही ऑपरेण्ड्स असतात तेव्हा ते मिश्रित एक्सप्रेसन म्हणून ओळखले जाते.

Table 2.8: काही अरीथमेटिक एक्सप्रेसनचे नमुने

Mathematical/Algebraic Expressions	The C Arithmetic Expressions
$\frac{a+b}{2}$	<code>(a+b) / 2</code>
$a + \frac{b}{c} + d$	<code>a+b/c+d</code>
$\frac{ab}{c-d^2} + d$	<code>(a*b) / (c-d*d) + d</code>
$1 + \frac{a}{b + \frac{1}{c}}$	<code>1+a/ (b+1/c)</code>
$\frac{\frac{a}{c+b} - e}{d}$	<code>a/ ((c+b) / d) - e</code>
$ax^2 + bx + c$	<code>a*x*x+b*x+c</code>
$ut + \frac{1}{2} at^2$	<code>u*t+0.5*a*t*t</code>
$m\left(a \times h + \frac{v^2}{2}\right)$	<code>m* (a*h+ (v*v) / 2)</code>

गणितीय/बीजगणितय एक्सप्रेसनचे C अरीथमेटिक एक्सप्रेसन मध्ये रूपांतरित करताना, कंस वापरले जाऊ शकतात अससोसिएटिव्हिटी नियंत्रित करण्यासाठी आणि त्या क्रमाने ऑपरेटरचे मूल्यांकन केले जाते. कंस एखाद्या एक्सप्रेसन मध्ये उपस्थित असल्यास प्रथम कंसातील एक्सप्रेसन चे मूल्यमापन केले जाते आणि कंसात प्रिंसिडन्स पहिले जाते, त्या क्रमाने ऑपरेटरचे मूल्यांकन केले जाते. जर कोष्टकांचे (कोष्टक अंतर्गत कंस) नेस्टिंग असतील तर सर्वात आधी आतील कंसांचे मूल्यांकन केले जाईल.

2.3.2 अरीथमेटिक एक्सप्रेसनचे मूल्यांकन (Evaluation of Arithmetic Expressions)

आपल्याला माहिती आहे की एकावेळी एक ऑपरेशन करून एक्सप्रेसनचे मूल्यांकन केले जाते आणि जेव्हा संगणकाद्वारे एक्सप्रेसनचे मूल्यांकन केले जाते तेव्हा असे होते. स्वतंत्र ऑपरेशन्सच्या मूल्यांकनाचा ऑर्डर ऑपरेटरच्या प्राथमिकता आणि असोसिएटिव्हिटीद्वारे नियंत्रित केला जातो.

तथापि, जेव्हा स्वतंत्र ऑपरेशन्स केल्या जातात तेव्हा खालील प्रकरणे येऊ शकतात:

- जेव्हा दोन्ही ऑपरेंड इन्टिजर संख्येचे असतात तेव्हा इन्टिजर अरीथमेटिक केले जाईल आणि ऑपरेशनचे परिणाम इन्टिजर मूल्य असेल.

उदाहरणार्थ, अपूर्णाक दुर्लक्षित केल्यामुळे $5/2$ मधून 2.5 नाही 2 हे मूल्य मिळतील.

- जेव्हा दोन्ही ऑपरेंड रिअल आहेत, रिअल अरीथमेटिक केले जाईल आणि ऑपरेशनच्या परिणामाचे रिअल मूल्य मिळेल.

उदाहरणार्थ, $5.0 / 2.0$, 2.5 हे मूल्य देईल.

तथापि, जेव्हा एक ऑपरेंड प्रकार इन्टिजर असेल आणि दुसरा प्रकार रिअल असेल तेव्हा मिश्रित गणित केले जाईल. या प्रकरणात, प्रथम इन्टिजर ऑपरेंड रिअल ऑपरेंडमध्ये रूपांतरित होईल आणि नंतर रिअल अरीथमेटिक केले जाईल आणि ऑपरेशनचे परिणाम रिअल मूल्य होतील. उदाहरणार्थ, $5/2.0$ किंवा $5.0/2$ मध्ये 2.5 उत्पन्न मिळेल, कारण पहिल्या प्रकरणात

मूल्य 5 मध्ये 5.0 मध्ये रूपांतरित होईल आणि नंतर विभागणी केली जाईल, तर दुसऱ्या प्रकरणात मूल्य 2 मध्ये 2.0 मध्ये रूपांतरित होईल आणि नंतर विभागणी होईल. केले जाऊ.

अरीथमेटिक एक्सप्रेसनचे मूल्यांकन कसे केले जाते हे स्पष्ट करण्यासाठी खालील एक्सप्रेसन विचारात घ्या.

$$5 * 4 / (1 + 5 * 2 / 3 + 6) + 8 * (7 / 4)$$

खाली दर्शविलेल्या तक्त्यात दाखविल्याप्रमाणे या एक्सप्रेसनचे मूल्यांकन केले जाते:

Table 2.9: अरीथमेटिक एक्सप्रेसनच्या मूल्यांकनचे स्पष्टीकरण

Operation by Operation Evaluation of Expression	Description of Each Operation
$5 * 4 / (1 + 5 * 2 / 3 + 6) + 8 * (7 / 4)$	Given expression
$5 * 4 / (1 + 10 / 3 + 6) + 8 * (7 / 4)$	5 is multiplied by 2, giving 10
$5 * 4 / (1 + 3 + 6) + 8 * (7 / 4)$	10 is divided by 3 using integral division, giving 3
$5 * 4 / (4 + 6) + 8 * (7 / 4)$	3 is added to 1, giving 4
$5 * 4 / 10 + 8 * (7 / 4)$	6 is added to 4, giving 10
$5 * 4 / 10 + 8 * 1$	7 is divided by 4 using integral division, giving 1
$20 / 10 + 8 * 1$	5 is multiplied by 4, giving 20
$2 + 8 * 1$	20 is divided by 10 using integral division, giving 2
$2 + 8$	8 is multiplied by 1, giving 8
10	8 is added to 2, giving 10, which is the final value of the expression.

2.3.3 टाईप कन्वर्जन (TYPE CONVERSION)

जेव्हा एक्सप्रेसनमध्ये भिन्न प्रकारची मूल्ये मिसळली जातात, तेव्हा कोणतीही कार्यवाही करण्यापूर्वी ते समान प्रकारात रूपांतरित केली जातात. मूल्ये एका प्रकारातून दुसऱ्या प्रकारात रूपांतरित करण्याची ही प्रक्रिया टाईप कन्वर्जन म्हणून ओळखली जाते.

C भाषा खालील दोन प्रकारांमध्ये रूपांतरण सुलभ करते:

- इम्प्लिसिट टाईप कन्वर्जन (Implicit type conversion)
- एक्सप्लिसिट टाईप कन्वर्जन (Explicit type conversion)

2.3.3.1 इम्प्लिसिट टाईप कन्वर्जन (Implicit Type Conversion)

प्रोग्रामरच्या हस्तक्षेपाशिवाय कंपाइलरद्वारे इम्प्लिसिट टाईप कन्वर्जन स्वयंचलितपणे केले जाते. एक्सप्रेसनच मिश्रित मोडमध्ये असते तेव्हा, माहिती गमावू नये म्हणून हे केले जाते.

एक्सप्रेसनचे मूल्यांकन मध्ये रूपांतरण (Conversion in Evaluation of Expressions)

जेव्हा वेगवेगळ्या प्रकारच्या ऑपरेशन्सवर ऑपरेशन केले जाणे आवश्यक असते, तेव्हा कमी साईझचे ऑपरेंड एका प्रकारात रूपांतरित केले जाते जे उच्च साईझच्या ऑपरेंडशी जुळते. एकदा हे रँकचे रूपांतरण झाल्यावर ऑपरेशन केले जाईल आणि ऑपरेशनचा परिणाम उच्च साईझचे असेल.

उदाहरणार्थ, समजा व्हेरिएबल्स x आणि y हे float आहेत आणि व्हेरिएबल्स i आणि j हे पदज प्रकारचे आहेत:

```
x / i + y * j
```

एक्सप्रेसन “x/i” चे मूल्यांकन करताना, इन्टिजर ऑपरेंड i ला फ्लोट प्रकारात रूपांतरित केले जाईल आणि नंतर ऑपरेंड x ला i द्वारा विभाजित केले जाईल आणि एक्सप्रेसनचा परिणाम टाइप फ्लोटचे मूल्य असेल.

असाइनमेंट स्टेटमेंट मधील कन्वर्जन (Conversions in Assignment Statements)

असाइनमेंट स्टेटमेंटमध्ये असाइनमेंट ऑपरेटर (=) आणि दोन ऑपरेंड असतात. असाइनमेंट ऑपरेटरच्या डाव्या बाजूला ऑपरेंड एक व्हेरिएबल आहे तर उजव्या बाजूला ऑपरेंड एक लिटरल्स/कॉन्स्टन्ट, व्हेरिएबल किंवा एक्सप्रेसन असू शकतो.

ऑपरेशन्सच्या साईझच्या फरकावर अवलंबून C सिस्टम डावी ऑपरेंड (व्हेरिएबल) शी जुळण्यासाठी उजवी ऑपरेंडला बदती देण्याची किंवा डिमोशन करण्याचा प्रयत्न करते. उजव्या ऑपरेंडचा साईझ कमी असल्यास पदोन्नती उद्भवते; उजव्या ऑपरेंडचा साईझ अधिक असल्यास डिमोशन उद्भवते.

2.3.3.2 एक्सप्लिसिट टाइप कन्वर्जन (Explicit Type Conversion)

एक्सप्लिसिट टाइप कन्वर्जन हे युझर- डिफाईंड आहे. जे ऑपरेंड किंवा एक्सप्रेसनला विशिष्ट प्रकारची सक्ती करते. एक्सप्लिसिट टाइप कन्वर्जनची प्रक्रिया ही **टाईप कास्टिंग** म्हणून देखील ओळखली जाते.

एक्सप्लिसिट टाइप कन्वर्जन किंवा प्रकार निर्णायक साठी सिन्टाक्स हा आहे.

```
(type) operand
```

येथे कंसातील प्रकाराला कास्ट ऑपरेटर असे संबोधले जाते, ज्याचे प्रिसिडन्स 2 आहे. सिंटॅक्स मध्ये दर्शविल्या प्रमाणे, एका प्रकारामधून दुसऱ्या प्रकारात डेटा कन्व्हर्ट करण्यासाठी, आपण त्याचे कन्वर्जन करू आधी नवीन कंसात या इच्छित मूल्य निर्दिष्ट करतो.

उदाहरणार्थ, x चा टाइप int व float प्रकारात रूपांतरित करण्यासाठी, आपण एक्सप्रेसन असे लिहितो

```
(float) x
```

येथे हे लक्षात घेणे महत्वाचे आहे की या ऑपरेशनमध्ये इतर युनिअरी ऑपरेशन प्रमाणेच, x मध्ये संग्रहित केलेले मूल्य अद्याप int टाइपचे आहेत, परंतु एक्सप्रेसनचे मूल्य float टाइप करण्यासाठी बदली दिले जाते.

कास्ट ऑपरेटरचा एक वापर म्हणजे इन्टिजर ऑपरेशन्सवर रिअल ऑपरेशनचा परिणाम खऱ्या संख्येने होतो याची खात्री करून घ्या.

उदाहरणार्थ, जर आपल्याला सरासरीची गणना करायची असल्यास ज्यामध्ये अपूर्णाक देखील असू शकतात, कास्ट ऑपरेटरशिवाय नसेल तर, इन्टिजर भागाकारामुळे निकाल इन्टिजर संख्या मिळेल.

रिअल निकालाची सक्ती करण्यासाठी आपण गणन सरासरी चे खाली दिलेल्या प्रमाणे स्टेटमेंट लिहू शकतो

```
average = (float) total / n;
```

या विधानात, total ते float चे एक्सप्लिसिट कन्वर्जन आहे आणि त्यानंतर n ते float चे इम्प्लिसिट कन्वर्जन आहे. आता रिअल भागाकार होईल आणि इच्छितेनुसार रिअल संख्या येईल. जी average ला नियुक्त केली जाईल.

2.4 प्रिसिडन्स आणि अससोसिएटिव्हिटी (PRECEDENCE AND ASSOCIATIVITY)

ऑर्डर निश्चित करण्यासाठी प्रिसिडन्स वापरले जाते ज्यामध्ये एक्सप्रेसनमध्ये भिन्न ऑपरेटरचे मूल्यांकन केले जाते.

ऑर्डर निश्चित करण्यासाठी असोसिएटिव्हिटी वापरली जाते ज्यामध्ये समान एक्सप्रेसनसह भिन्न ऑपरेटरचे एका एक्सप्रेसनमध्ये मूल्यांकन केले जातात.

कोणत्या क्रमाने एक्सप्रेसनचे मूल्यांकन केले जाते ते निश्चित करण्यासाठी असोसिएटिव्हिटीपूर्वी प्रिसिडन्स लागू केले जाते. आवश्यक असल्यास असोसिएटिव्हिटी नंतर लागू केली जाते.

प्रिसिडन्स (Precedence)

गणिताच्या विषयामध्ये प्रिसिडन्स संकल्पना चांगल्या प्रकारे परिभाषित केली गेली आहे. आपल्याला नियम BODMAS - कंस, विभाग, गुणाकार, बेरीज आणि वजाबाकीबद्दल माहिती असणे आवश्यक आहे. बीजगणितात, बेरीज व वजाबाकी करण्यापूर्वी भागाकार व गुणाकार केला जातो.

खाली प्रिसिडन्सचे एक साधे उदाहरण दिलेले आहे:

$$5 + 3 * 4$$

या एक्सप्रेसनमध्ये बेरीज आणि गुणाकार ऑपरेटरचा समावेश आहे. तुम्हाला माहिती आहे की बीजगणितात गुणाकाराला बेरीज पेक्षा जास्त प्राधान्य आहे, बेरीज पूर्वी गुणाकार केला जातो.

म्हणूनच, एक्सप्रेसनचे मूल्यांकन असे केले जाईल

$$(5 + (3 * 4)) \rightarrow (5 + 12) \rightarrow 17$$

संपूर्ण एक्सप्रेसनचे मूल्य म्हणून 17 देत आहे.

असोसिएटिव्हिटी (Associativity)

जेव्हा सारख्याच प्रिसिडन्सचे एक किंवा एकापेक्षा जास्त ऑपरेटर एक्सप्रेसनमध्ये वापरले जातात तेव्हा असोसिएटिव्हिटी लागू केली जाते. असोसिएटिव्हिटी डावीकडून उजवीकडे किंवा उजवीकडून डावीकडे असू शकते. डावीकडून उजवी असोसिएटिव्हिटी डावीकडून प्रारंभ करून आणि उजवीकडे एक्सप्रेसनचे मूल्यांकन करते तर उजवीकडून डावीकडील असोसिएटिव्हिटी उजवीकडून प्रारंभ करून आणि डावीकडे एक्सप्रेसनचे मूल्यांकन करते.

असोसिएटिव्हिटीचे एक साधे उदाहरण खाली दिले आहे:

$$2 + 5 + 7$$

या एक्सप्रेसनमध्ये दोन आडिशन (+) ऑपरेटर आहेत. येथे असोसिएटिव्हिटी निर्धारित करते की उप एक्सप्रेसन एकत्रित कसे केल्या जातात. आडिशन (+) ऑपरेटरकडे डावीकडून उजवी असोसिएटिव्हिटी असल्याने एक्सप्रेसन म्हणून खाली दिलेले वर्गीकरण केले जाईल

$$((2 + 5) + 7) \rightarrow (7 + 7) \rightarrow 14$$

संपूर्ण एक्सप्रेसनचे मूल्य म्हणून 14 हे येईल.

Table 2.10: ऑपरेटरचा प्रिसिडन्स आणि असोसिएटिव्हिटी

Operator	Description	Precedence Level (Rank)	Associativity
()	Function call	1	Left-to-right
[]	Array element reference		
→	Structure operator used with pointer to a structure		
.	Structure operator used with structure variable		

Operator	Description	Precedence Level (Rank)	Associativity
!	Logical NOT operator	2	Right-to-left
~	1's complement		
++	Increment		
--	Decrement		
+	Unary plus		
-	Unary minus		
*	Pointer reference (indirection)		
&	Address of		
(type)	Type cast		
sizeof	Size of an operand		
*	Multiplication	3	Left-to-right
/	Division		
%	Modulus (remainder)		
+	Addition	4	Left-to-right
-	Subtraction		
<<	Left shift	5	Left-to-right
>>	Right shift		
<	Less than	6	Left-to-right
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equal to	7	Left-to-right
!=	Not equal to		
&	Bitwise AND	8	Left-to-right
^	Bitwise exclusive OR	9	Left-to-right
	Bitwise inclusive OR	10	Left-to-right
&&	Logical AND	11	Left-to-right
	Logical OR	12	Left-to-right
?:	Condition (ternary)	13	Right-to-left
= += -= *= /= %= &= ^= != <<= >>=	Assignment operators	14	Right-to-left

2.5 लायब्ररी फंक्शन (LIBRARY FUNCTIONS)

बऱ्याचदा वारंवार वापरल्या जाणाऱ्या अनेक गणितीय कार्यांचे मूल्यांकन करणे आवश्यक असते, जसे की स्क्वेअर रूट, लॉगरिदम, एक्सपोनेन्शियल, ट्रायगोनोमेट्रिक फंक्शन्स इ. प्रत्येक युझर्सने या कार्ये करण्यासाठी स्वतःचा कोड लिहिण्याऐवजी, C कंपाईलर त्यांना सुविधा म्हणून बिल्ट इन लायब्ररी प्रदान करते.

युझर्स प्रोग्राम युनिटमध्ये *math.h* ही हेडर फाइल समाविष्ट करावी लागेल. या फंक्शन्स व्यतिरिक्त, इतरही बरेच लायब्ररी फंक्शन्स आहेत जे खास फंक्शन्स करतात आणि वेगवेगळ्या हेडर फाइल्समध्ये परिभाषित केल्या जातात. उदाहरणार्थ स्ट्रिंग्स हाताळणारे लायब्ररी फंक्शन्स *string.h*

फंक्शनच्या व्याख्येनुसार काही लायब्ररी फंक्शन्सचे तर्क मर्यादित असतात. उदाहरणार्थ, नकारात्मक संख्येचे लॉगॅरिथम गणितीयदृष्ट्या परिभाषित केलेले नाही. त्याचप्रमाणे, नकारात्मक संख्येचा वर्ग मूळ गणितीयदृष्ट्या अपरिभाषित असतो, आणि म्हणूनच याला परवानगी नाही.

Table 2.11: काही सामान्यपणे वापरल्या गेलेल्या गणिताची फंक्शन्स

Function	Description
<code>pow(x, y)</code>	Returns x raised to power y , i.e., x^y .
<code>pow10(p)</code>	Returns the value 10^p .
<code>exp(x)</code>	Returns e to the x th power, i.e., e^x .
<code>log(x)</code>	Returns the value of $\ln(x)$.
<code>log10(x)</code>	Returns the value of $\log_{10}(x)$.
<code>sqrt(x)</code>	Returns the value of \sqrt{x} .
<code>sin(x)</code>	Returns the value sine of x , where the angle x is in radians.
<code>ceil(x)</code>	Returns the smallest integer greater than or equal to x .
<code>floor(x)</code>	Returns largest integer less than or equal to x .

स्पष्टीकरणात्मक उदाहरणे

या विभागात, आपण काही प्रसंग आणि / किंवा समस्या विचारात घेऊया ज्या अपल्याला ऑपरेटर आणि एक्सप्रेसन वापरण्याची संधी देतील. त्यापूर्वी ड्राई रनची संकल्पना जाणून घेऊ.

ड्राई रन म्हणजे मुळात आम्ही संगणकाचे अनुकरण करू आणि पेपर पेन्सिलच्या सहाय्याने केलेल्या सूचनांच्या एक्झिक्युशनचा शोध घेऊ. प्रक्रियेदरम्यान, व्हेरिएबल्सची एक्झिक्युशन दरम्यान प्रारंभिक व्हॅल्यूजसहित बदल झाल्यामुळे आम्ही त्यांची व्हॅल्यू रेकॉर्ड करतो. लक्षात घ्या की जर व्हेरिएबल आरंभ केला नसेल तर त्याचे मूल्य कॅरेक्टर “?” सह दर्शविले जाईल.

स्पष्टीकरणात्मक उदाहरणामध्ये आपणास प्रोग्रामची ड्राई रन आणि / किंवा प्रोग्राम कोडचा विभाग दिसेल, जिथेही मुख्य संकल्पना स्पष्ट करणे आवश्यक आहे असे समजले जाईल.

Example 2.1: थर्ड व्हेरिएबल “t” चा वापर करून दोन ‘a’ व ‘b’ व्हेरिएबल्सच्या व्हॅल्यूज स्वॅप करणे.

Solution:

```
int a=10,b=20,t;          /* 1 */
t = a;                     /* 2 */
a = b;                     /* 3 */
b = t;                     /* 4 */
```

कोडच्या शेवटच्या तीन ओळी देखील C प्रमाणे लिहिता येतील

```
t = a, a = b, b = t;
```

Dry Run

Statement No.	a	b	t
1	10	20	?
2	10	20	10
3	20	20	10
4	10	20	10

Example 2.2: तृतीय व्हेरिएबल न वापरता आणि अंकगणित व्यतिरिक्त बेरीज (+) आणि वजाबाकी (-) ऑपरेशन्स न वापरता 'a' आणि 'b' या दोन व्हेरिएबल्सच्या व्हॅल्यूज स्वॅप करणे.

Solution:

```
int a=10,b=20;          /* 1 */
a = a + b;               /* 2 */
b = a - b;               /* 3 */
a = a - b;               /* 4 */
```

हे कार्य खालील एकाच स्टमेंटद्वारे देखील लागू केले जाऊ शकते:

```
a = ( a + b ) - ( b = a );
```

Dry Run

Statement No.	a	b
1	10	20
2	30	20
3	30	10
4	20	10

Example 2.3: तृतीय व्हेरिएबल वापरल्याशिवाय आणि अंकगणित गुणाकार (*) आणि भागाकार (/) ऑपरेशन्स न वापरता 'a' आणि 'b' या दोन व्हेरिएबल्सच्या व्हॅल्यूज स्वॅप करणे.

Solution:

```
int a=10,b=20;          /* 1 */
a = a * b;               /* 2 */
b = a / b;               /* 3 */
a = a / b;               /* 4 */
```

हे कार्य खालील एकाच स्टमेंटद्वारे देखील लागू केले जाऊ शकते:

```
a = ( a * b ) / ( b = a );
```

Dry Run

Statement No.	a	b
1	10	20
2	200	20
3	200	10
4	20	10

Example 2.4: a, b आणि c मध्ये तीनपैकी सर्वात मोठी संख्या शोधण्यासाठी

Solution:

```
int a=10,b=30,c=25, big;
big = (a>b)?((a>c)?a:c):((b>c)?b:c);
```

येथे, a हा b पेक्षा मोठे असल्यास एक्सप्रेशन $((a>c)?a:c)$ चे मूल्यांकन केले जाते; अन्यथा एक्सप्रेशन $((b>c)?b:c)$ चे मूल्यांकन केले जाईल.

एक्सप्रेशन $((a>c)?a:c)$ a हा c पेक्षा मोठी असल्यास a परत करेल अन्यथा c,

एक्सप्रेशन $((b>c)?b:c)$ b हा c पेक्षा मोठी असल्यास b परत करेल अन्यथा c,

Example 2.5: n मध्ये संग्रहित केलेल्या 4-अंकी क्रमांकाच्या उजव्या-अंकातील आणि डावीकडील-अंकांची बेरीज शोधण्यासाठी योग्य विधाने लिहा.

Solution:

```
left_most_digit = n / 1000;
right_most_digit = n % 1000;
sum = left_most_digit + right_most_digit;
```


Example 2.6: सेल्सिअस स्केलमधील तापमान फॅरेनहाइट स्केलमध्ये रूपांतरित करण्यासाठी योग्य विधाने लिहा.

Solution: सेल्सिअस स्केल आणि फॅरेनहाइट स्केलमधील तापमानामधील संबंध आहे

$$\frac{C}{100} = \frac{F - 32}{180} \Rightarrow F - 32 = \frac{180}{100} C \Rightarrow F = 1.8C + 32$$

```
float c, f;
```

```
f = 1.8 * C + 32;
```

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- ऑपरेटर भाषेचे क्रियापद असतात जे युझर मूल्यांवर मोजणी करू देतात.
- ऑपरेटर युनिअरी किंवा बायनरी असू शकतात
- ऑपरेटरमध्ये समृद्ध असलेली C भाषा ही पहिली भाषा होती.
- ऑपरेटर प्रामुख्याने अरीथमेटिक ऑपरेटर, रिलेशनल ऑपरेटर, लॉजिकल ऑपरेटर, बिट-वाईस ऑपरेटर आणि स्पॅसिअल ऑपरेटरमध्ये विभागले जातात.
- अरीथमेटिक ऑपरेटर्सचे इन्टिजर अरीथमेटिक, रिअल अरीथमेटिक आणि मिश्रित मोड अरीथमेटिक म्हणून वर्गीकृत केले जाऊ शकते.
- असाइनमेंट ऑपरेटर व्हेरिएबल्सला व्हॅल्यूज देण्यासाठी वापरतात.
- एक्सप्रेसन हे मूल्य मोजण्यासाठी ऑपरेन्ड्स आणि ऑपरेटर एकत्र जोडलेले एक सूत्र आहे.
- एक्सप्रेसन प्रामुख्याने अरीथमेटिक एक्सप्रेसनस, रिलेशनल एक्सप्रेसनस, लॉजिकल एक्सप्रेसनस, आणि कंडिशनल एक्सप्रेसनस मध्ये विभागल्या जातात.
- एका प्रकारासाठी मूल्ये दुसऱ्या प्रकारात रूपांतरित करण्याच्या प्रक्रियेस टाईप कन्वर्जन असे म्हणतात.
- प्रोग्रामरच्या हस्तक्षेपाशिवाय कंपाइलरद्वारे इम्प्लिसिट टाईप कन्वर्जन स्वयंचलितपणे केले जाते.
- एक एक्सप्लिसिट टाईप कन्वर्जन युझर- डिफाइंड आहे जे ऑपरेंड किंवा
- एक्सप्रेसनला विशिष्ट प्रकारची सक्ती करते.
- एक्सप्लिसिट टाईप कन्वर्जनची प्रक्रिया टाईप कास्टिंग म्हणून देखील ओळखली जाते.
- ऑर्डर निश्चित करण्यासाठी प्रिसिडन्स वापरले जाते ज्यामध्ये एक्सप्रेसनमध्ये भिन्न ऑपरेटरचे मूल्यांकन केले जाते.
- असोसिएटिव्हिटीचा क्रम ऑर्डर निश्चित करण्यासाठी केला जातो ज्यामध्ये समान अग्रभागी असलेल्या भिन्न ऑपरेटरचे मूल्यांकन म्हणून व्यक्त केले जाते.
- लायब्ररी फंक्शन्स विविध रूटीन प्रकारच्या ऑपरेटर्सची एक्झिक्युशन करण्यासाठी पूर्व-लिखित कार्ये आहेत आणि कॅम्पयलेशन भाग म्हणून प्रदान केली जातात. त्यांचा वापर करण्यासाठी, आपल्याला आपल्या प्रोग्राममध्ये योग्य हेडर फाईल समाविष्ट करण्याची आवश्यकता आहे.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. ऑपरेटर म्हणजे काय? योग्य अंकांसह विविध अरीथमेटिक ऑपरेटरच्या वापराचे वर्णन करा.
2. अरीथमेटिक ऑपरेटरपैकी ऑपरेटरची विस्तृत श्रेणी कोणती आहे?

3. मॉड्यूलस ऑपरेटर '%' वर काही बंधन आहे का?
4. कंडिशनल ऑपरेटरच्या कामाचे योग्य उदाहरण देऊन वर्णन करा.
5. साईझऑफ ऑपरेटरचे कार्य समजावून सांगा.
6. बीजगणितीय एक्सप्रेसनच्या मूल्यांकनावर आधारीत असलेल्या नियमांचे वर्णन करा.
7. ऑपरेटरच्या प्रिंसिडन्सचा अर्थ काय? विविध अरीथमेटिक ऑपरेटरमध्ये प्रिंसिडन्स काय आहे?
8. ऑपरेटरच्या असोसिएटिव्हिटी म्हणजे काय?
9. ज्याचे ऑपरेशन भिन्न प्रकारचे आहेत अशा एक्सप्रेसनना लागू असलेल्या नियमांचे सारांश.
10. अरीथमेटिक अंकात कंस कधी वापरावे? किमान एक उदाहरण द्या.
11. लायब्ररी प्रत्यक्षात "C" भाषेचा भाग आहेच स्पष्ट करणे.
12. कंडिशनल ऑपरेटरच्या कामाचे योग्य उदाहरण देऊन वर्णन करा.
13. बुलियन false बरोबर किती इन्टिजर मूल्य आहे?
14. एक्सप्रेसन म्हणजे काय?
15. इन्क्रीमेंट आणि डिक्रीमेंट ऑपरेटरचे कार्य स्पष्ट करा.
16. टाईप कन्वर्जन म्हणजे काय?
17. इम्प्लिसिट टाईप कन्वर्जन होते त्या परिस्थितीची नोंद करा.
18. एक्सप्लिसिट टाईप कास्टिंग कसे केले जाते? उदाहरण द्या.
19. गणिताची लायब्ररी फंक्शन वापरण्यासाठी आपल्याला आवश्यक असलेली हेडर फाईल चे नाव द्या.
20. खालील बीजगणित एक्सप्रेसनच्या समकक्ष C एक्सप्रेसन लिहा:

$$(a) a - \frac{a}{a^{-x}} + b$$

$$(b) 1 + \frac{1}{a + \frac{1}{a}} + b$$

$$(c) x + \frac{y^{-z}}{a + \frac{1}{a}} - \sqrt{x}$$

$$(d) x^3 - \frac{y^7}{-z} + e^x$$

$$(e) \left(\frac{1-x}{1+x} \right) / \left(\frac{1+y}{1-y} \right)$$

$$(f) \frac{1}{r} = \frac{1}{r_1} + \frac{1}{r_2}$$

$$(g) \sqrt{(x^2 + y^2 + z^2)^3}$$

$$(h) \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

एकाधिक निवड प्रश्न

1. पुढील पैकी कोणते अंकगणित ऑपरेटर नाही?

(a) +	(b) &	(c) %	(d) *
-------	-------	-------	-------
2. ऑपरेटरची प्राथमिकता कोणते ऑपरेटर ठरवते

(a) प्रथम वापरली जाते	(b) महत्त्वाचे आहे
(c) सर्वात मोठ्या संख्येने कार्य करते	(d) जलद चालवते

3. ऑपरेटर % ला लागू केले जाऊ शकते
 - (a) फ्लोट व्हॅल्यूज
 - (b) डबल व्हॅल्यूज
 - (c) इन्टिजर व्हॅल्यूज
 - (d) वरील सर्व
4. एक्सप्रेसन $x : y$ हे या समान आहे
 - (a) $(x - (x/y))$
 - (b) $(x - (x/y) * y)$
 - (c) $(y - (y/x))$
 - (d) $(y - (x/y) * x)$
5. खालील पैकी कोणते एक बिटवाईस ऑपरेटर नाही?
 - (a) >
 - (b) >>
 - (c) ^
 - (d) &
6. $5/6/3 + 8/3$ एक्सप्रेसनचे मूल्य काय असेल?
 - (a) 4
 - (b) 2
 - (c) 2.333 (approx.)
 - (d) वरीलपैकी काहीही नाही
7. x/y एक्सप्रेसनमध्ये, y चे मूल्य असावे
 - (a) integer
 - (b) non-zero
 - (c) non-negative
 - (d) positive
8. x हा प्रकार *int* चे व्हेरिएबल असल्यास खालील पैकी कोणते वेगळे आहे?
 - (a) $x < 1$
 - (b) $x = x * 2$
 - (c) $x * = 2$
 - (d) $x < = 1$
9. C भाषेत खालील विधानाचा विचार करा

$$x = (a > b) ? ((a > c) ? a : c) : (b > c) ? b : c;$$

$a = 3, b = -5, c = 2$ असल्यास x चे मूल्य किती असेल?

 - (a) 2
 - (b) 3
 - (c) -5
 - (d) वरीलपैकी काहीही नाही
10. जेव्हा रिलेशनल एक्सप्रेसन चुकीचे असते तेव्हा त्याचे मूल्य असते _____.
 - (a) 1
 - (b) 0
 - (c) -ve
 - (d) +ve
11. जर $a = 15$ असेल तर विधान $b = a < 2$; परिणामी होईल
 - (a) 30
 - (b) 60
 - (c) 7
 - (d) वरीलपैकी काहीही नाही
12. पुढील कोड विभागाचा विचार करा


```
int i, j = 10;
i = j++;
```

या विभागाच्या शेवटी i आणि j चे मूल्य असेल

 - (a) 10, 10
 - (b) 10, 11
 - (c) 11, 10
 - (d) 11, 11
13. पुढील कोड विभागाचा विचार करा


```
int x = 5.234, y;
y = sizeof(x);
```

y चे मूल्य असेल

 - (a) 4
 - (b) 2
 - (c) 8
 - (d) वाक्यरचनेची चूक

14. खालीलपैकी कोणता ऑपरेटर अविभाज्य (integral) ऑपरेटर भागाकार 2 करण्यासाठी वापरला जाऊ शकतो?
 (a) / (b) >>
 (c) << (d) दोन्ही (a) आणि (b)
15. खालीलपैकी कोणते विधान ++ ऑपरेटरबद्दल योग्य नाही?
 (a) हे युनिअरी ऑपरेटर आहे. (b) ऑपरेटर आधी किंवा नंतर ऑपरेंड येऊ शकतो.
 (c) हे एक्सप्रेसनवर लागू केले जाऊ शकते. (d) त्याचे अससोसिएट्स उजवीकडून डावीकडे.

ANSWERS															
1.	(b)	2.	(a)	3.	(c)	4.	(b)	5.	(a)	6.	(b)	7.	(b)	8.	(a)
9.	(b)	10.	(b)	11.	(b)	12.	(b)	13.	(b)	14.	(b)	15.	(c)		

प्रॅक्टिकल

1. साधे आणि चक्रवाढ व्याज शोधण्यासाठी प्रोग्राम लिहा.

Solution: साधे आणि चक्रवाढ व्याज मोजण्याचे सूत्र आहेत

$$\text{Simple interest} = \frac{p \times r \times t}{100},$$

$$\text{and Compound interest} = p \left[\left(1 + \frac{r}{100} \right)^t - 1 \right]$$

जिथे p ही मूळ रक्कम असते, r वार्षिक व्याज दर असतो आणि t वर्षांमध्ये ठेवीची वेळ असते.

Listing 2.1

```
/* program to compute simple and compound interest */
#include <stdio.h>
#include <math.h>
int main()
{
    float p, r, si, ci;
    int t;
    printf("\nEnter principle amount : ");
    scanf("%f", &p);
    printf("Enter rate of interest : ");
    scanf("%f", &r);
    printf("Enter time : ");
    scanf("%d", &t);
    si = (p*r*t)/100;
```

```

ci = p*(pow(1+r/100,t)-1);
printf("\nsimple interest = Rs. %.2f", si);
printf("\ncompound interest = Rs. %.2f", ci);
return 0;
}

```

Test Run

```

Enter principle amount : 1000
Enter rate of interest : 5
Enter principle amount : 4
Simple interest = Rs. 200.00
Compound interest = Rs. 215.51

```

2. त्रिकोणाचे क्षेत्रफळ शोधण्यासाठी प्रोग्राम लिहा, ज्याच्या तीन बाजूंचे नाव अनुक्रमे a , b आणि c असे दिले आहेत. a , b आणि c च्या मूल्यांनी ही अट पूर्ण केली पाहिजे $a + b > c$ and $b + c > a$ and $c + a > b$

Solution: त्रिकोणाचे क्षेत्रफळ मोजण्याचे सूत्र हे आहे

$$s = \frac{a + b + c}{2},$$

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

Listing 2.2

```

/* program to compute area of a triangle whose sides are given */
#include <stdio.h>
#include <math.h>
int main()
{
    float a, b, c, s, area;
    printf("\nEnter value for a : ");
    scanf("%f", &a);
    printf("Enter value for b : ");
    scanf("%f", &b);
    printf("Enter value for c : ");
    scanf("%f", &c);
    s = (a+b+c)/2;
    area = sqrt(s*(s-a)*(s-b)*(s-c));
    printf("\nArea of triangle = %.2f Sq. Units", area);
    return 0;
}

```

Test Run

```
Enter value for a : 5
Enter value for b : 7
Enter value for c : 6
Area of triangle = 14.70 Sq. Units
```

3. एका इमारतीत 10 मजले आहेत आणि मजल्याची उंची प्रत्येकी 3 मीटर आहे. इमारतीच्या वरच्या बाजूस एक बॉल सोडला जातो. प्रत्येक मजल्यावर जाण्यासाठी बॉलने घेतलेला वेळ शोधण्यासाठी एक प्रोग्राम लिहा.

Solution: प्रत्येक मजल्याची उंची 3 मीटर आहे, म्हणूनच 10 मजल्यासह इमारतीची उंची (h) 30 मीटर आहे.

येथे आपण गतिमानाचे समीकरण वापरू.

$$s = ut + \frac{1}{2} at^2$$

येथे s चे स्थानांतर h आणि a द्वारा g ने केले आहे (9.8 m/s^2).

चेंडू वरून (10^{th}) मजला खाली सोडला जात आहे, $u = 0$. म्हणून, हे सूत्र काम करते.

$$h = \frac{1}{2} gt^2$$

$$\Rightarrow t = \sqrt{\frac{2h}{g}}$$

Listing 2.3

```
/* program to compute the time taken by ball to reach
   the ground floor, when it is released from the top floor
*/
#include <stdio.h>
#include <math.h>
int main()
{
    float h = 30, g = 9.8, t;
    t = sqrt(2*h/g);
    printf("\nTime taken by ball = %.2f seconds", t);
    return 0;
}
```

Test Run

```
Time taken by ball = 2.47 seconds
```

आणखी माहिती

विज्ञान आणि अभियांत्रिकी संबंधित समस्यांसाठी अरीथमेटिक एक्सप्रेसन हा प्रोग्रेशनसचा आधार आहे. अशा समस्यांमध्ये गुंतागुंतीचे गणितीय आणि बीजगणित एक्सप्रेसन असतात. म्हणून, अशा एक्सप्रेसनना C एक्सप्रेसनमध्ये रूपांतरित करण्यासाठी, ऑपरेटरचे

प्रिसिडन्स आणि असोसिएटिव्हिटी, अरीथमेटिकचे प्रकार आणि एक्सप्रेसनच्या मूल्यांकनादरम्यान कोणत्या प्रकारचे रूपांतरण घडते हे माहित असले पाहिजे.

शिक्षकाने एक्सप्रेसन फॉर्मेशन आणि एक्सप्रेसनचे मूल्यांकन आणि इतर संबंधित समस्यांविषयी समज विकसित करणे अपेक्षित आहे.

शिक्षकांच्या विद्यार्थ्यांच्या सक्रिय सहभागासह विविध प्रकारच्या एक्सप्रेसनची निर्मिती आणि मूल्यांकन देखील प्रदर्शित केले पाहिजे.

संदर्भ आणि सूचविलेले वाचन

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

3

कंडिशनल ब्रँचिंग आणि लूप्स

युनिट वैशिष्ट्ये

हे युनिट विविध नियंत्रण विधानांशी संबंधित विषयांवर चर्चा करते. ही कंट्रोल स्टेटमेंट प्रोग्रामरला स्टेटमेंटचा क्रम बदलण्याची किंवा प्रोग्राममधील स्टेटमेंटची पुनरावृत्ती नियंत्रित करण्याची परवानगी देते. या युनिटमध्ये स्पष्ट केलेल्या विविध कंट्रोल स्टेटमेंटमध्ये इफ स्टेटमेंट, इफ - एल्स स्टेटमेंट, नेस्टेड इफ आणि इफ- एल्स स्टेटमेंट, इफ-एल्स इफ लाडर, स्विच स्टेटमेंट, फॉर, व्हईल, डू..व्हईल, ब्रेक आणि कंटिन्यू यांचा योग्य उदाहरणांसह वापर दर्शविला आहे

तर्कशास्त्र

संगणक प्रोग्राम संगणकासाठी सूचनांचा एक सेट आहे. या सूचना क्रमानुसार अंमलात आणल्या जातात, म्हणजेच, प्रोग्रॅम मध्ये आल्याप्रमाणे एकामागून एक. निवेदनाची अंमलबजावणी करण्याचा हा आदेश सोप्या समस्यांसाठी चांगला कार्य करतो जेथे निर्णय घेण्याची कोणतीही प्रक्रिया किंवा निर्देशांची पुनरावृत्ती होत नाही.

तथापि, सराव मध्ये, बहुतेकदा सूचनांच्या एक्झिक्युशनचा क्रम बदलणे किंवा ज्ञात संख्येसाठी किंवा काही विशिष्ट अटी पूर्ण होईपर्यंत सूचनांचे समूह पुन्हा करणे आवश्यक असते. अशा परिस्थितीत ही विधाने ज्या क्रमाने एक्झिक्युट केली जातील त्या क्रमाने नियंत्रित करणे आवश्यक आहे.

हे युनिट विद्यार्थ्यांना वाक्यरचना समजून घेण्यासाठी आणि वास्तविक जीवनातील समस्यांचे निराकरण करण्यासाठी कंडिशनल ब्रँचिंग आणि लूप्स एक्झिक्युट करण्यासाठी वापरल्या जाणाऱ्या विविध विधानांची रचना समजण्यास मदत करते.

पूर्व-आवश्यकता

- रिलेशनल ऑपरेटर (Relational operators)
- लॉजिकल ऑपरेटर (Logical operators)
- इन्क्रीमेंट आणि डिक्रीमेंट ऑपरेटर (Increment/decrement operators)
- रिलेशनल / लॉजिकल एक्सप्रेशनचे मूल्यांकन (Evaluation of relational/logical expressions)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U3-O1:	कंडिशनल ब्रँचिंग आणि लूपिंगची आवश्यकता स्पष्ट करा.
U3-O2:	दिलेल्या समस्येवर आधारित योग्य प्रकारचे कंडिशनल ब्रँचिंग स्टेटमेंट निवडा.

U3-O3:	दिलेल्या समस्येवर आधारित लूपचा योग्य प्रकार निवडा.
U3-O4:	ब्रेक आणि कंटीन्यू कीवर्ड वापरा.
U3-O5:	कंडिशनल ब्रँचिंग आणि लूप वापरून वास्तविक जीवनातील अडचणी सोडविण्यासाठी प्रोग्राम तयार करा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit-3 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1 – Weak Correlation; 2 – Medium Correlation; 3 – Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U3-O1	1							
U3-O2				2				
U3-O3				2				
U3-O4		1		2				
U3-O5				3				

3.1 ओळख (INTRODUCTION)

विधानांच्या एक्झिक्युशनचा क्रम बदलणे म्हणजेच - नियंत्रणाचा प्रवाह बदलणे, पुढीलपैकी एक परिस्थितीत सामील होऊ शकते.

- ब्रँचिंग — निर्णयाच्या निकालावर अवलंबून निर्देशांचा एक गट एक्झिक्युट करणे.
- लूपिंग — दिलेल्या संख्येसाठी किंवा आवश्यक अट पूर्ण होईपर्यंत पुनरावलोकनात्मक सूचनांच्या गटाची एक्झिक्युशन करणे.
- जम्पिंग — समान प्रोग्राम युनिटमधील एका पॉइंट पासून दुसऱ्या पॉइंट वर नियंत्रण हस्तांतरित करणे.

C लॅंग्वेज स्टेटमेंट्सच्या एक्झिक्युशनच्या क्रमावर नियंत्रण ठेवण्यासाठी सुविधा पुरवते, ज्यास फ्लो कंट्रोल स्टेटमेंट्स किंवा फक्त कंट्रोल स्टेटमेंट्स म्हणून संबोधले जाते.

विविध फ्लो कंट्रोल विधान खालील श्रेणींमध्ये एकत्रित केली आहेत:

1. कंडिशनल ब्रँचिंग — या वर्गात, C भाषा खालील विधाने प्रदान करते.
 - if statement
 - if - else statement
 - else if ladder
 - switch statement
2. लूपिंग — या वर्गात, C भाषा खालील विधाने प्रदान करते.
 - for statement
 - while statement
 - do - while statement
3. जम्पिंग — या वर्गात, C भाषा खालील विधाने प्रदान करते.
 - break statement
 - continue statement

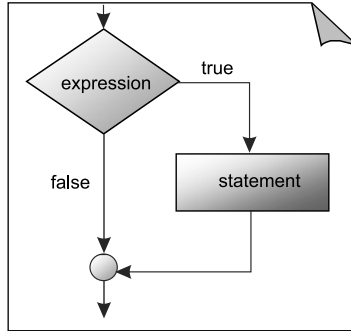
या युनिटमध्ये, आपण या विधानांबद्दल चर्चा करतो आणि चर्चेचे निराकरण केलेल्या बऱ्याच प्रोग्रामद्वारे समर्थित करतो आहे.

3.2 कंडिशनल ब्रॅंचिंग (CONDITIONAL BRANCHING)

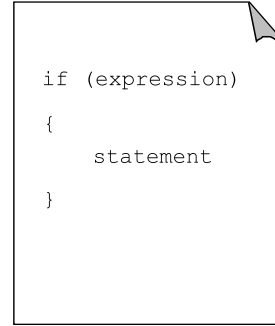
या श्रेणीतील विविध विधाने विशिष्ट निर्णय निकषांवर आधारित निवडक विधानांची एक्झिक्युशन करण्यास परवानगी देतात. आम्ही त्यांचे कार्य चांगल्या प्रकारे डिझाइन केलेल्या स्पष्टीकरणात्मक उदाहरणांच्या द्वारे पाहूया.

3.2.1 इफ स्टेटमेंट (The if Statement)

इफ स्टेटमेंटचा सिन्टॅक्स



(a) Logic Flow Control of if Statement



(b) Code of if Statement

चित्र 3.1: लॉजिक फ्लो कंट्रोल आणि इफ स्टेटमेंटचा कोड

एक्सप्रेशन रिलेशनल एक्सप्रेशन, लॉजिकल एक्सप्रेशन, न्यूमेरिक व्हेरिएबल किंवा न्यूमेरिक कॉन्स्टन्ट दर्शवते. निर्दिष्ट एक्सप्रेशन एक सिम्पल एक्सप्रेशन किंवा कंपाऊंड एक्सप्रेशन असू शकते.

C मधील एक्सप्रेशनचे मूल्यांकन 0 किंवा 1 असे करते. जर एक्सप्रेशनचे मूल्य 1 असेल तर स्टेटमेंट-ब्लॉकमधील स्टेटमेंट्स एक्झिक्युट केली जातात; अन्यथा त्यांना बायपास केले जाते.

Example 3.1: दिलेली इन्टिजर संख्या नकारात्मक किंवा सकारात्मक आहे की नाही हे तपासण्यासाठी प्रोग्राम.

Listing 3.1

```

#include<stdio.h>
int main()
{
    int n;
    printf( "\nEnter integer number : " );
    scanf( "%d", &n );
    if ( n < 0 ) {
        printf( "\nNumber is negative.\n" );
        return 0;
    }
    printf( "\nNumber is positive.\n" );
    return 0;
}
  
```

Test Runs**First Run**

Enter integer number : -25

Number is negative.

Second Run

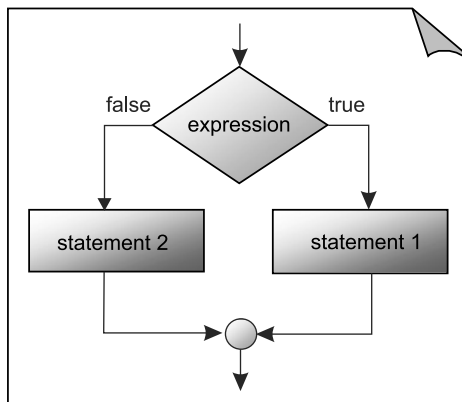
Enter integer number : 30

Number is positive.

3.2.2 इफ - एल्स स्टेटमेंट (The if - else statement)

निर्दिष्ट केलेले एक्सप्रेशन शून्य नसलेल्या मूल्याचे मूल्यांकन करते तर इफ स्टेटमेंट सिंगल स्टेटमेंट (सिम्पल किंवा कंपाऊंड) एक्झिक्युट करते. जेव्हा ते शून्य मूल्याचे मूल्यांकन करते तेव्हा ते काहीही करत नाही. जर एक्सप्रेशनने शून्य मूल्याशिवाय मूल्यमापन केले तर दुसरे विधान शून्य मूल्याचे मूल्यांकन केल्यास दुसरे विधान एक्झिक्युट होईल का? उत्तर होय आहे.

हे उद्दीष्ट इफ - एल्स स्टेटमेंट वापरून साध्य केले जाते.



(a) Logic Flow Control of if - else statement

```

if (expression)
{
    statement-1
}
else
{
    statement-2
}
  
```

(b) Code of if - else statement

चित्र 3.2: लॉजिक फ्लो कंट्रोल आणि इफ-एल्स स्टेटमेंटचा कोड

जर एक्सप्रेशनचे मूल्यांकन 1 असेल तर स्टेटमेंट -1 एक्झिक्युट होईल आणि स्टेटमेंट -2 बायपास केले जाईल. तथापि, जर एक्सप्रेशनचे मूल्यांकन 0 असेल तर स्टेटमेंट -1 बायपास केले जाईल आणि स्टेटमेंट -2 एक्झिक्युट होईल.

Example 3.2: नैसर्गिक संख्या सम किंवा विषम आहे की नाही हे तपासण्यासाठी प्रोग्राम.

Listing 3.2

```

#include<stdio.h>
int main()
{
    int n;
    printf( "\nEnter any natural number : " );
    scanf( "%d", &n );
  
```

```

    if ( n % 2 == 0 )
        printf( "\nNumber entered is EVEN\n" );
    else
        printf( "\nNumber entered is ODD\n" );
    return 0;
}

```

Test Runs

First Run

Enter any whole number: 22
 Number entered is EVEN

Second Run

Enter any number: 15
 Number entered is ODD

Example 3.3: दोन संख्यांमधील मोठा शोधण्याचा प्रोग्राम.

Listing 3.3

```

include<stdio.h>
int main()
{
    int a, b;
    printf( "\nEnter value for a : " );
    scanf( "%d", &a );
    printf( "\nEnter value for b : " );
    scanf( "%d", &b );
    if ( a > b )
        printf( "\n%d is the larger.\n", a );
    else
        printf( "\n%d is the larger.\n", b );
    return 0;
}

```

Test Runs

First Run

Enter value for a : 20
 Enter value for b : 10
 20 is the larger.

Second Run

```
Enter value for a : 15
Enter value for b : 25
25 is the larger.
```

3.2.3 नेस्टेड इफ आणि इफ- एल्स स्टेटमेंट (Nested if and if - else statements)

इफ स्टेटमेंट्स नेस्ट केले जाऊ शकतात, म्हणजेच, इफ स्टेटमेंट दुसऱ्या इफ स्टेटमेंट मध्ये असू शकते. जर बाह्य एक्सप्रेशनचे मूल्य शून्य नसल्यास त्याच्या आतील इफ स्टेटमेंट एक्झिक्युट होईल.

त्याचप्रमाणे इफ- एल्स स्टेटमेंट्स नेस्ट केले जाऊ शकतात. येथे इफ स्टेटमेंट नेस्टेड असेल तर इफ भाग किंवा एल्स भाग नेस्टेड असेल.

Example 3.4: दिलेले वर्ष लीप वर्ष आहे की नाही हे ठरवण्यासाठी प्रोग्राम लिहा.

पुढीलपैकी कोणतीही एक परिस्थिती पूर्ण झाल्यास दिलेले वर्ष हे लीप वर्ष आहे:

1. वर्ष 4 ने समान रीतीने विभाजीत केले जाते आणि 100 ने विभाजित केले जाऊ शकत नाही.
2. वर्ष 4 ने समान रीतीने विभाजीत केले जाऊ शकते, 100 ने समान रीतीने विभाजीत केले जाऊ शकते तसेच 400 ने समान रीतीने विभाजित केले जाऊ शकते.

इतर सर्व प्रकरणांमध्ये, दिलेले वर्ष लीप वर्ष नसते.

उदाहरणार्थ

1. 1988, 1992, आणि 1996 ही वर्ष लीप वर्षे आहेत कारण ती 4 ने विभाजीत केली आहेत पण 100 ने विभाजित केले जाऊ शकत नाही.
2. 1700, 2100, आणि 2500 ही वर्षे लीप वर्षे नसतात कारण ती 100 आणि 4 द्वारा विभाजित केली जातात पण 400 ने विभाजित केले जाऊ शकत नाही.
3. 1600, 2000, आणि 2400 ही वर्ष लीप वर्षे आहेत कारण ती 100 व 400 ने विभाजित केली जातात.

Listing 3.4

```
#include <stdio.h>
int main()
{
    int year;
    printf("Enter given year : ");
    scanf("%d", &year);
    if ( year % 4 == 0 ) {
        if ( year % 100 == 0 ) {
            if ( year % 400 == 0 )
                printf("\nYear %d is a leap year.\n", year);
            else
                printf("\nYear %d is not a leap year.\n", year);
        }
    }
```

```

        else
        {
            printf("\nYear %d is a leap year.\n", year);
        }
    }
    else
    {
        printf("\nYear %d is not a leap year.\n", year);
    }
    return 0;
}

```

Test Runs

First Run

```

Enter any year : 2000
Year 2000 is a leap year.

```

Second Run

```

Enter any year : 2008
Year 2008 is a leap year.

```

Third Run

```

Enter any year : 2006
Year 2006 is not a leap year.

```

3.2.4 इफ-एल्स इफ लाडर (The if-else if Ladder)

इफ-एल्स इफ लाडर हे इफ-एल्स स्टेटमेंटचा विस्तार आहे. हे अशा परिस्थितीत वापरले जाते जिथे भिन्न परिस्थितींमध्ये अनेक प्रकरणे सादर केली जातात.

इफ-एल्स इफ लाडर स्टेटमेंटचा सिन्टॅक्स

```

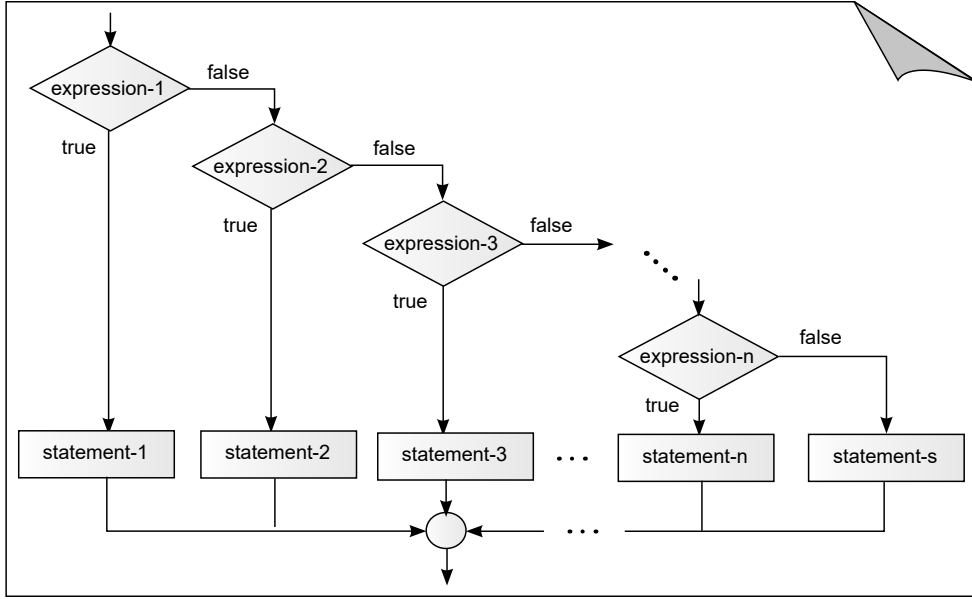
if ( expression-1 )
    statement-1
else if ( expression-2 )
    statement-2
else if ( expression-3 )
    statement-3
:
else if ( expression-n )
    statement-n
else
    statement-s

```

एक्सप्रेशनचे क्रमाक्रमाने मूल्यांकन केले जाते आणि जर कोणतीही एक्सप्रेशन सत्य असेल तर संबंधित स्टेटमेंट ब्लॉक एक्झिक्युट होईल आणि यामुळे संपूर्ण साखळी संपुष्टात येईल.

शेवटचा एल्स पार्ट वरीलपैकी कोणतेही नाही किंवा डीफॉल्ट एक्सप्रेशन हाताळते.

चित्र 3.3: इफ-एल्स इफ लाडर चा कोड



चित्र 3.4: लॉजिक फ्लो कंट्रोल इफ-एल्स इफ लाडर स्टेटमेंटचा

Example 3.5: दिगुणांची टक्केवारी दिली आहे. पुढील धोरणानुसार विद्यार्थ्यांचा ग्रेड मोजला जातो.

Percentage of Marks	Grade
percentage ≥ 90	A
$90 > \text{percentage} \geq 75$	B
$75 > \text{percentage} \geq 60$	C
$60 > \text{percentage} \geq 50$	D
percentage < 50	F

विद्यार्थ्यांच्या गुणांची टक्केवारी दिल्यावर प्रिंट करण्यासाठी प्रोग्राम लिहा.

Listing 3.5

```

#include<stdio.h>
int main()
{
    float percentage;
    printf("\nEnter percentage of marks : ");
    scanf("%f", &percentage);
    if ( percentage >= 90 )
        printf("\nGrade is A\n");
    else if ( percentage >= 75 )
        printf("\nGrade is B\n");

```

```

else if ( percentage >= 60 )
    printf("\nGrade is C\n");
else if ( percentage >= 50 )
    printf("\nGrade is D\n");
else
    printf("\nGrade is E\n");
return 0;
}

```

Test Run

```

Enter percentage of marks : 93
Grade is A

```

3.2.5 स्विच स्टेटमेंट (The switch Statement)

स्विच स्टेटमेंट व्हेरिएबल / एक्सप्रेशनच्या वेगवेगळ्या मूल्यांसाठी केलेल्या अनेक प्रकरणांना पर्याय उपलब्ध करते.

स्विच स्टेटमेंटचा सिन्टॅक्स चित्र 3.5

मध्ये दर्शविले आहे.

val-1, *val-2*, *val-3*, ..., *val-n*, वरून एक्सप्रेशनचे कोणतेही मूल्य घेतल्यास नियंत्रण त्या योग्य केस मध्ये स्थानांतरित केले जाते. प्रत्येक बाबतीत स्टेटमेंट्स एक्झिक्युट केली जातात आणि नंतर ब्रेक स्टेटमेंट स्विच स्टेटमेंटच्या बाहेर नियंत्रण स्थानांतरित करते.

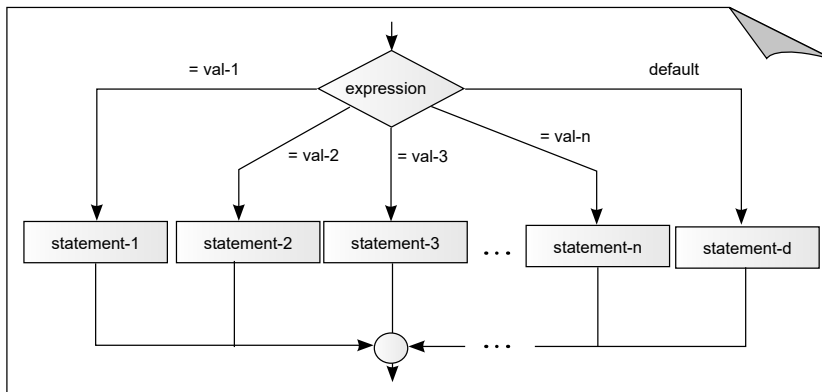
एखाद्या केस नंतर ब्रेक स्टेटमेंट वापरले नसल्यास डीफॉल्ट कीवर्डच्या अनुपस्थितीत शेवटचे वगळता, त्याचे नियंत्रण पुढील केस वर येईल. जर एक्सप्रेशनचे मूल्य कोणत्याही केस मूल्यांशी जुळत नसेल तर नियंत्रण डीफॉल्ट कीवर्डकडे जाते, जे सहसा स्विच स्टेटमेंटच्या शेवटी असते. डीफॉल्ट कीवर्डचा वापर मोठ्या सोयीसाठी होऊ शकतो. डीफॉल्ट कीवर्ड नसल्यास, कोणताही केस मूल्यांशी जुळत नसेल तर संपूर्ण स्विच स्टेटमेंट सहजपणे वगळले जाते.

```

switch ( expression )
{
    case val-1 :
        statement-1
        break;
    case val-2 :
        statement-2
        break;
    :
    case val-n :
        statement-n
        break;
    default :
        statement-d
}

```

चित्र 3.5: स्विच स्टेटमेंटचा कोड



चित्र 3.6: स्विच स्टेटमेंटचे लॉजिक फ्लो कंट्रोल

Example 3.6: कॅल्क्युलेटर चे चार फंक्शन लागू करण्यासाठी प्रोग्राम लिहा.

Listing 3.6

```
#include<stdio.h>
int main()
{
    int a, b;
    char op;
    printf("\nEnter expression such as 5 + 3 : ");
    scanf("%d %c %d", &a, &op, &b );
    switch ( op )
    {
        case '+' : printf("\n%d %c %d = %d\n", a, op, b, a+b);
                    break;
        case '-' : printf("\n%d %c %d = %d\n", a, op, b, a-b);
                    break;
        case '*' : printf("\n%d %c %d = %d\n", a, op, b, a*b);
                    break;
        case '/' : printf("\n%d %c %d = %d\n", a, op, b, a/b);
                    break;
        default : printf("\nWrong input\n");
    }
    return 0;
}
```

Test Run

```
Enter expression such as 5 + 3 : 20 / 2
20 / 2 = 10
```

अशा परिस्थिती उद्भवू शकतात जेव्हा आपल्याला असे हवे होते की एक्सप्रेशनच्या एकापेक्षा जास्त मूल्यांसाठी समान विधान एक्झिक्युट केले जावे.

अशी परिस्थिती हाताळण्यासाठी आम्ही या केसंना एकामागून एक कोड करतो आणि पुढीलप्रमाणे:

```
switch ( expression )
{
    :
    case val-4 :
    case val-5 :
    case val-6 : statement;
                    break;
    :
}
```

जेव्हा एक्सप्रेशनचे मूल्य val-4, val-5, किंवा val-6 असते तेव्हा स्टेटमेंट-ब्लॉक एक्झिक्युट होईल.

Example 3.7: वर्णमाला स्वर किंवा व्यंजनात्मक (vowel or consonant) आहे की नाही हे तपासण्यासाठी प्रोग्राम लिहा.

Listing 3.7

```
#include<stdio.h>
int main()
{
    char ch;
    printf("\nEnter an alphabet : ");
    ch = getche();
    switch ( ch )
    {
        case 'a' :
        case 'A' :
        case 'e' :
        case 'E' :
        case 'i' :
        case 'I' :
        case 'o' :
        case 'O' :
        case 'u' :
        case 'U' : printf("\nAlphabet is vowel.\n");
                    break;
        default :
                    printf("\nAlphabet is consonant.\n");
    }
    return 0;
}
```

Test Run

```
Enter an alphabet : A
Alphabet is vowel.
```

कंडिशनल ब्रँचिंग चे स्पष्टीकरणात्मक उदाहरणे

Example 3.8: चतुर्भुज समीकरणाची मुळे (roots of a quadratic equation) शोधण्याचा प्रोग्राम लिहा.

$ax^2 + bx + c = 0$ provided $a \neq 0$.

Solution: roots of the quadratic equation चे सूत्र खाली दिले आहे:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

$b^2 - 4ac$ एक्स्प्रेसन भेदभाव करणारा म्हणून ओळखली जाते.

भेदभावाच्या चिन्हावर अवलंबून, रूट्ससाठी तीन शक्यता आहेत:

1. जर $b^2 - 4ac < 0$, तर मूळ कल्पित आहे आणि आपण वास्तविक भाग आणि काल्पनिक भागाची स्वतंत्रपणे गणना करू शकतो (If $b^2 - 4ac < 0$, then the roots are imaginary, and we can compute real part and imaginary part separately as)

$$\text{real part} = -\frac{b}{2a} \quad \text{imaginary part} = \frac{\sqrt{-(b^2 - 4ac)}}{2a}$$

2. जर $b^2 - 4ac = 0$, मग मुळे वास्तविक आणि समान असतात. (If $b^2 - 4ac = 0$, then the roots are real and equal, and root is simply computed as)

$$\text{root} = -\frac{b}{2a}$$

3. जर $b^2 - 4ac > 0$, मग मुळे वास्तविक आणि वेगळ्या असतात. (If $b^2 - 4ac > 0$, then the roots are real and distinct and are computed as)

$$\text{root one} = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad \text{root two} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Listing 3.8

```
#include<stdio.h>
#include<math.h>
int main()
{
    float a, b, c, disc, root1, root2;
    float realPart, imagPart;
    printf( "\nEnter values of a,b,c: " );
    scanf( "%f,%f,%f", &a, &b, &c);
    if ( a == 0 ) {
        printf( "\nIt is not an quadratic equation\n" );
        return 0;
    }
    disc = b * b - 4.0 * a * c;
    if ( disc < 0 )
    {
        realPart = - b / (2*a);
```

```

    imagPart = sqrt((double)-disc) / (2*a);
    printf( "\nRoots are imaginary" );
    printf( "\nReal part = %.3f", realPart );
    printf( "\nImaginary part = %.3f\n", imagPart );
}
else if ( disc == 0 )
{
    root1 = - b / (2*a);
    printf( "\nRoots are real and equal" );
    printf( "\nEach root = %.3f\n", root1 );
}
else
{
    root1 = (-b-sqrt((double)disc )) / (2*a);
    root2 = (-b+sqrt((double)disc )) / (2*a);
    printf( "\nRoots are real and distinct" );
    printf( "\nRoot 1 = %.3f", root1 );
    printf( "\nRoot 2 = %.3f\n", root2 );
}
return 0;
}

```

Test Run

Enter values of a,b,c: 2,3,5

Roots are imaginary

Real part = -0.750

Imaginary part = 1.392

Example 3.9: समजा, व्यक्तींसाठी इनकम टॅक्स खालीलप्रमाणे स्लॅब दरांवर मोजला गेला आहे:

Income	Tax Payable
Upto ₹ 1,00,000/-	Nil
From ₹ 1,00,001/- to ₹ 2,00,000/-	10% of the excess over ₹ 1,00,000/-
From ₹ 2,00,001/- to ₹ 3,00,000/-	20% of the excess over ₹ 2,00,000/-
Above ₹ 3,00,000/-	30% of the excess over ₹ 3,00,000/-

एक प्रोग्रॅम लिहा जो इनकम वाचतो आणि इनकम टॅक्स प्रिंट करतो.

Solution: पुढील गोष्टी लक्षात घ्या:

- जर उत्पन्न पर्यंत ₹ 1,00,000/- फक्त असेल, मग कर *nil* आहे.
- जर उत्पन्न श्रेणीत ₹ 1,00,001/- ते ₹ 2,00,000/- पर्यंत असल्यास, मग कर 10% पेक्षा जास्त रकमेसाठी ₹ 1,00,000/- आहे.
- जर उत्पन्न श्रेणीत ₹ 2,00,001/- ते ₹ 3,00,000/- पर्यंत असल्यास, मग कर 10% of ₹ 1,00,000/- (₹ 10,000/-) ₹ 1,00,001 ते ₹ 2,00,000/- च्या स्लॅबसाठी अधिक 20% पेक्षा जास्त रकमेसाठी ₹ 2,00,000/- आहे.
- जर उत्पन्न जास्त असल्यास ₹ 3,00,000/-, मग कर 10% of ₹ 1,00,000/- (₹ 10,000/-) च्या स्लॅबसाठी ₹ 1,00,001 ते ₹ 2,00,000/- पर्यंत अधिक 20% of ₹ 1,00,000/- (₹ 20,000/-) च्या स्लॅबसाठी ₹ 2,00,001 ते ₹ 3,00,000/- पर्यंत अधिक 30% पेक्षा जास्त रकमेसाठी ₹ 3,00,000/- आहे.

जर दिलेले उत्पन्न असेल तर आपण करांची गणना करूया

(a) ₹ 2,50,000/- (b) ₹ 1,75,000/- (c) ₹ 3,20,000/-

(a) ₹ 2,50,000/- च्या उत्पन्नासाठी कर मोजणे

For first ₹ 1,00,000/-	Tax is Nil
For next ₹ 1,00,000/-	Tax is 10,000/- (@ 10%)
For next ₹ 50,000/-	Tax is 10,000/- (@ 20%)
Total Tax due	₹ 20,000/-

(b) ₹ 1,75,000/- च्या उत्पन्नासाठी कर मोजणे

For first ₹ 1,00,000/-	Tax is Nil
For next ₹ 75,000/-	Tax is 7,500/- (@ 10%)
Total Tax due	₹ 7,500/-

(c) ₹ 3,20,000/- च्या उत्पन्नासाठी कर मोजणे

For first ₹ 1,00,000/-	Tax is Nil
For next ₹ 1,00,000/-	Tax is 10,000/- (@ 10%)
For next ₹ 1,00,000/-	Tax is 20,000/- (@ 20%)
For next ₹ 20,000/-	Tax is 6,000/- (@ 30%)
Total Tax due	₹ 36,000/-

Listing 3.9

```
#include<stdio.h>

void main()
{
    float income, tax;
    printf( "\nEnter gross income (in Rs.): " );
    scanf( "%f", &income );
    if ( income <= 100000.0 )
        tax = 0;
```

```

else if ( income <= 200000.0 )
    tax = ( income - 100000.0 ) * 0.1;
else if ( income <= 300000.0 )
    tax = 10000 + ( income - 200000.0 ) * 0.2;
else
    tax = 30000 + ( income - 300000.0 ) * 0.3;
printf( "\nTax due = Rs. %.2f\n", tax );
}

```

Test Run

```

Enter gross income : 250000
Tax due = Rs. 20000.00

```

Example 3.10: आठवड्याचा दिवस नंबर म्हणून स्वीकारणारा असा प्रोग्रॅम लिहा आणि संबंधित दिवसाचे नाव प्रिंट करा, म्हणजेच 0 रविवारसाठी, सोमवारसाठी 1, ..., 6 शनिवार.

Listing 3.10

```

#include <stdio.h>
int main()
{
    int day;
    printf("\nEnter day of week as number ( 0 - 6 ) : ");
    scanf("%d", &day);
    switch ( day )
    {
        case 0 : printf("\nDay of week is Sunday." );
                 break;
        case 1 : printf("\nDay of week is Monday." );
                 break;
        case 2 : printf("\nDay of week is Tuesday." );
                 break;
        case 3 : printf("\nDay of week is Wednesday." );
                 break;
        case 4 : printf("\nDay of week is Thursday." );
                 break;
        case 5 : printf("\nDay of week is Friday." );
                 break;
    }
}

```

```

        case 6 : printf("\nDay of week is Saturday." );
                break;
        default : printf("\nWrong input" );
    }
    printf("\n");
    return 0;
}

```

Test Run

```

Enter day of week as number ( 0 - 6 ) : 2
Day of week is Tuesday

```

Example 3.11: ऑपरेशनद्वारे मॉड्यूलस न वापरता नैसर्गिक संख्या सम किंवा विषम आहे की नाही हे तपासण्यासाठी प्रोग्राम लिहा.

Listing 3.11

```

#include <stdio.h>
int main()
{
    int n;
    printf("\nEnter any natural number : ");
    scanf("%d", &n);
    switch ( n%10 )
    {
        case 0 :
        case 2 :
        case 4 :
        case 6 :
        case 8 : printf("\n%d is even.\n");
                break;
        default : printf("\n%d is even.\n");
    }
    return 0;
}

```

Test Runs

First Run

```

Enter any natural number : 120
120 is even.

```

Second Run

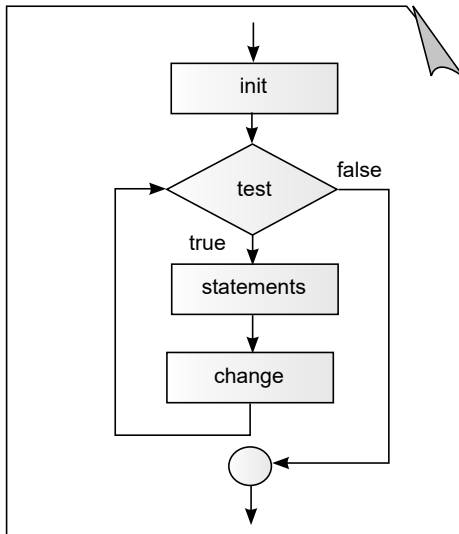
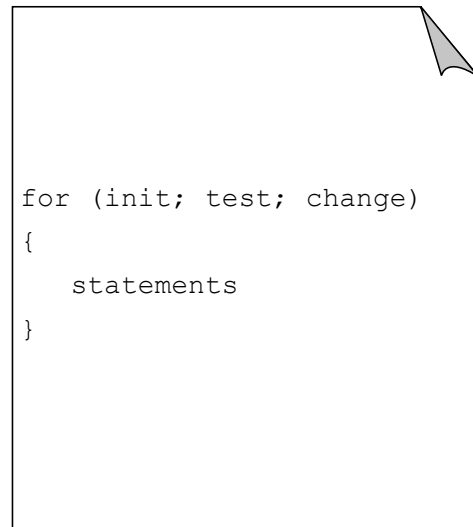
```
Enter any natural number : 75
75 is odd.
```

3.3 लूपिंग (LOOPING)

या श्रेणीतील विविध विधाने दिलेल्या संख्येने किंवा काही अटी पूर्ण होईपर्यंत निवेदनांच्या गटाच्या पुनरावृत्ती कार्याची परवानगी देतात. आम्ही त्यांचे कार्य चांगल्या प्रकारे डिझाइन केलेल्या स्पष्टीकरणात्मक उदाहरणांच्या द्वारे पाहूया.

3.3.1 फॉर स्टेटमेंट (The for Statement)

फॉर स्टेटमेंट अशा समस्यांसाठी उपयुक्त आहे जेथे स्टेटमेंट किंवा स्टेटमेंट-ब्लॉक किती वेळा अंमलात आणले जाईल याची अगोदर माहिती असेल.

(a) Logic Flow Control of *for* Statement(b) Code of *for* Statement**चित्र 3.7:** लॉजिक फ्लो कंट्रोल आणि फॉर स्टेटमेंटसाठी कोड

जेथे *init* हे एक्सप्रेशन आहे ते काउंटर इनिशियलाइज करण्यासाठी आहे, *test* हे एक्सप्रेशन आहे ते पुनरावृत्ती कधी थांबवायची हे पाहते, आणि *change* हे एक्सप्रेशन आहे ते लूपच्या प्रत्येक पाससाठी काउंटर बदलण्यासाठी आहे. *init* आणि *change* भागांमध्ये स्वल्पविरामाने विभक्त केलेले एकापेक्षा जास्त विधान असू शकतात.

for स्टेटमेंटच्या वापराचे प्रदर्शन करण्यासाठी, चला तर *for* स्टेटमेंटचा वापर करून काही प्रोग्राम लिहूया.

Example 3.12: प्रथम एन नैसर्गिक संख्यांची बेरीज शोधण्यासाठी प्रोग्राम लिहा.

Listing 3.12

```
#include<stdio.h>
int main(void)
{
```



```

int i, n, sum = 0;
printf("\nEnter value for n : " );
scanf("%d", &n);
for ( i = 1; i <= n; i++ ) {
    sum = sum + i;
}
printf("\nSum of first %d natural numbers = %d\n", n, sum );
return 0;
}    printf("\nEnter value for n : " );
scanf("%d", &n);
for ( i = 1; i <= n; i++ ) {
    sum = sum + i;
}
printf("\nSum of first %d natural numbers = %d\n", n, sum );
return 0;
}

```

Test Run

```

Enter value for n : 10
Sum of first 10 natural numbers = 55

```

Example 3.13: n नैसर्गिक क्रमांकाचे फॅक्टोरियल शोधण्यासाठी प्रोग्राम लिहा.

Listing 3.13

```

#include<stdio.h>
int main(void)
{
    int i, n, prod = 1;
    printf("\nEnter value for n : " );
    scanf("%d", &n);
    for ( i = 1; i <= n; i++ ) {
        prod = prod * i;
    }
    printf("\nFactorial %d = %d\n", n, prod );
    return 0;
}

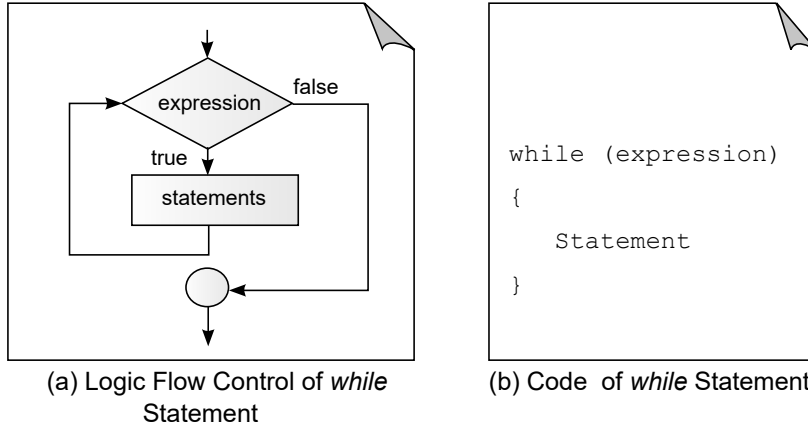
```

Test Run

```
Enter value for n : 5
Factorial 5 = 120
```

3.3.2 व्हाइल स्टेटमेंट (The while Statement)

व्हाइल स्टेटमेंट अशा समस्यांसाठी उपयुक्त आहे जेथे स्टेटमेंट किंवा स्टेटमेंट-ब्लॉक किती वेळा एक्झिक्युट होईल हे आधीपासूनच माहित नसते.



चित्र 3.8: लॉजिक फ्लो कंट्रोल आणि व्हाइल स्टेटमेंटचा कोड

येथे एक्सप्रेशन व्हेरिएबल, कॉन्स्टन्ट किंवा एक्सप्रेशन असू शकते. एक्सप्रेशनचे मूल्यांकन 1 असेपर्यंत स्टेटमेंटची वारंवार अंमलबजावणी केली जाते. जेव्हा एक्सप्रेशनचे मूल्यांकन 0 होते तेव्हा व्हाइल स्टेटमेंट संपेल आणि कंट्रोल नंतर व्हाइल स्टेटमेंटच्या बाहेर स्थानांतरित केला जाईल.

व्हाइल स्टेटमेंट वापरताना खालील बाबी लक्षात घेतल्या पाहिजेत:

- व्हाइल स्टेटमेंट निवेदनापूर्वी विधान असणे आवश्यक आहे जे एक्सप्रेशनला इनिशियलाइज करते.
- स्टेटमेंट-ब्लॉकमध्ये, एक्सप्रेशनला सुधारित करणारे विधान असणे आवश्यक आहे.

व्हाइल स्टेटमेंटचा वापर निदर्शनास आणण्यासाठी, व्हाइल स्टेटमेंटचा वापर करून काही प्रोग्राम बनवूया.

Example 3.14: n नैसर्गिक संख्येच्या अंकांची बेरीज शोधण्यासाठी प्रोग्राम लिहा.

Listing 3.14

```
#include <stdio.h>
int main()
{
    int i, n, sum = 0, temp, digit;
    printf("\nEnter any natural number : ");
    scanf("%d", &n);
    temp = n;
    while ( temp > 0 )
```

```

{
    digit = temp % 10;
    sum = sum + digit;
    temp = temp / 10;
}
printf( "\nSum of digits of %d = %d\n", n, sum );
return 0;
}

```

Test Run

```

Enter any natural number : 2375
Sum of digits of 2375 = 17

```

Example 3.15: परिच्छेदामधील वर्ण, स्वर आणि शब्दांची (characters, vowels, and words) संख्या मोजण्यासाठी प्रोग्राम लिहा.

Listing 3.15

```

#include <stdio.h>
int main()
{
    int vowelCount = 0, characterCount = 0, wordCount = 0;
    char ch;
    printf( "Type in the paragraph and terminate by ENTER key\n\n" );
    while ( ( ch = getche() ) != '\r' )
    {
        characterCount++;
        switch ( ch )
        {
            case ' ' :
            case '\t' :
                wordCount++;
                break;
            case 'a' :
            case 'A' :
            case 'e' :
            case 'E' :

```

```

        case 'i' :
        case 'I' :
        case 'o' :
        case 'O' :
        case 'u' :
        case 'U' :

            vowelCount++;
            break;

    }
}
wordCount++;
printf( "\nCharacter count = %d", characterCount);
printf( "\nVowel count = %d", vowelCount);
printf( "\nWord count = %d\n", wordCount);
return 0;
}

```

Test Run

Type in the paragraph and terminate by ENTER key

Programming is the way to instruct computer to do a particular task.

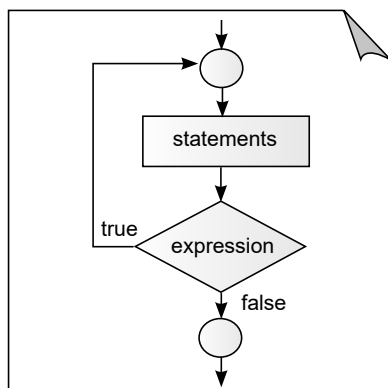
Character count = 68

Vowel count = 20

Word count = 12

3.3.3 डू - व्हाइल स्टेटमेंट (The *do - while* Statement)

डू- व्हाइल स्टेटमेंट, जसे की व्हाइल स्टेटमेंट, हे देखील अशा समस्यांसाठी उपयुक्त आहे जेथे स्टेटमेंट किती वेळा एक्झिक्युट होईल हे आधीच माहित नसते.



(a) Logic Flow Control of *do-while* Statement

```

do
{
    Statement
}
while (expression);
  
```

(b) Code of *do-while* Statement

चित्र 3.9: लॉजिक फ्लो कंट्रोल आणि डू- व्हाइल स्टेटमेंट कोड

येथे एक्सप्रेशन व्हेरिएबल, कॉन्स्टन्ट किंवा एक्सप्रेशन असू शकते. एक्सप्रेशनचे मूल्यांकन 1 असे पर्यंत वारंवार स्टेटमेंट एक्झिक्युट केले जाते.

Example 3.16: अज्ञात संख्यांची सरासरी मोजण्यासाठी एक प्रोग्राम लिहा.

Listing 3.16

```
#include<stdio.h>
int main()
{
    int n = 0;
    char yesno;
    float number, sum = 0, average;
    do {
        printf( "\nEnter number %d: ", n+1 );
        scanf( "%f", &number );
        n++;
        sum += number;
        printf( "\nAny more number [yn]?: " );
        yesno = getchar();
    }
    while ( ( yesno == 'y' ) || ( yesno == 'Y' ) );
    average = sum / n;
    printf( "\n\nAverage of given numbers = %.2f\n", average );
    return 0;
}
```

Test Run

```
Enter number 1: 2.5
Any more number [yn]?: y
Enter number 2: 12.0
Any more number [yn]?: y
Enter number 3: 10.25
Any more number [yn]?: y
Enter number 4: 8.75
Any more number [yn]?: y
Enter number 5: 11.0
Any more number [yn]?: n
Average of given numbers = 8.90
```

3.3.4 नेस्टेड व्हाइल, फॉर आणि डू – व्हाइल स्टेटमेंट्स (NESTED WHILE, FOR AND DO — WHILE STATEMENTS)

जसे इफ स्टेटमेंट नेस्टेड असू शकतात, तसेच हे स्टेटमेंट्स देखील नेस्टेड असू शकतात. बाह्य लूपच्या प्रत्येक पुनरावृत्तीसाठी आतील लूप सुरुवातीपासूनच एक्झिक्युट केले जाते.

कोडचे खालील विभाग त्यांच्या स्वतःच्या प्रकारात नेस्टेड लूपिंग स्टेटमेंट दर्शवित आहेत.

<code>for (i=0; i<m; i++)</code>	<code>i=0;</code>	<code>i=0;</code>
<code>{</code>	<code>while (i<m)</code>	<code>do</code>
<code>:</code>	<code>{</code>	<code>{</code>
<code>for (j=0; j<n; j++)</code>	<code>:</code>	<code>:</code>
<code>{</code>	<code>j=0;</code>	<code>j=0;</code>
<code>:</code>	<code>while (j<n)</code>	<code>do</code>
	<code>{</code>	<code>{</code>
<code>}</code>	<code>:</code>	<code>:</code>
<code>}</code>	<code>j++;</code>	<code>j++;</code>
	<code>}</code>	<code>} while (j<n);</code>
	<code>i++;</code>	<code>i++;</code>
	<code>}</code>	<code>} while (i<m);</code>

सर्वसाधारणपणे, व्हाइल आणि डू-व्हाइल स्टेटमेंट्सच्या आत फॉर स्टेटमेंट नेस्ट करणे शक्य असते आणि फॉर स्टेटमेंटच्या आत व्हाइल आणि डू-व्हाइल स्टेटमेंट्स नेस्ट करणे शक्य असते, म्हणजेच सर्व प्रकारच्या नेस्टिंग एकात्रित करण्यास परवानगी आहे.

नेस्टेड लूपचा वापर दर्शविण्यासाठी, चला तर नेस्टेड लूप वापरून काही प्रोग्रॅम विकसित करू या.

Example 3.17: खालील नमुना प्रिंट करण्यासाठी प्रोग्राम लिहा.

```
1
2 2
3 3 3
```

Listing 3.17

```
#include <stdio.h>
int main()
{
    int i, j;
    printf("\n");
    for ( i = 1; i <= 3; i++ )
    {
        for ( j = 1; j <= i; j++ )
        {
            printf("%4d", i);
```

```

    }
    printf("\n");
}
return 0;
}

```

Example 3.18: खालील नमुना प्रिंट करण्यासाठी प्रोग्राम लिहा.

```

3 3 3
2 2
1

```

Listing 3.18

```

#include <stdio.h>
int main()
{
    int i, j;
    printf("\n");
    for ( i = 3; i >= 1; i-- )
    {
        for ( j = 1; j <= i; j++ )
        {
            printf("%4d", i);
        }
        printf("\n");
    }
    return 0;
}

```

Example 3.19: खालील नमुना प्रिंट करण्यासाठी प्रोग्राम लिहा.

```

1
1 2
1 2 3

```

Listing 3.19

```

#include <stdio.h>
int main()
{
    int i, j;

```

```

printf("\n");
for ( i = i; i <=3; i++ )
{
    for ( j = 1; j <= i; j++ )
    {
        printf("%4d", j);
    }
    printf("\n");
}
return 0;
}

```

Example 3.20: खालील नमुना प्रिंट करण्यासाठी प्रोग्राम लिहा.

```

*
* *
* * *

```

Listing 3.20

```

#include <stdio.h>
int main()
{
    int i, j;
    printf("\n");
    for ( i = i; i <= 3; i++ )
    {
        for ( j = 1; j <= i; j++ )
        {
            printf("    *");
        }
        printf("\n");
    }
    return 0;
}

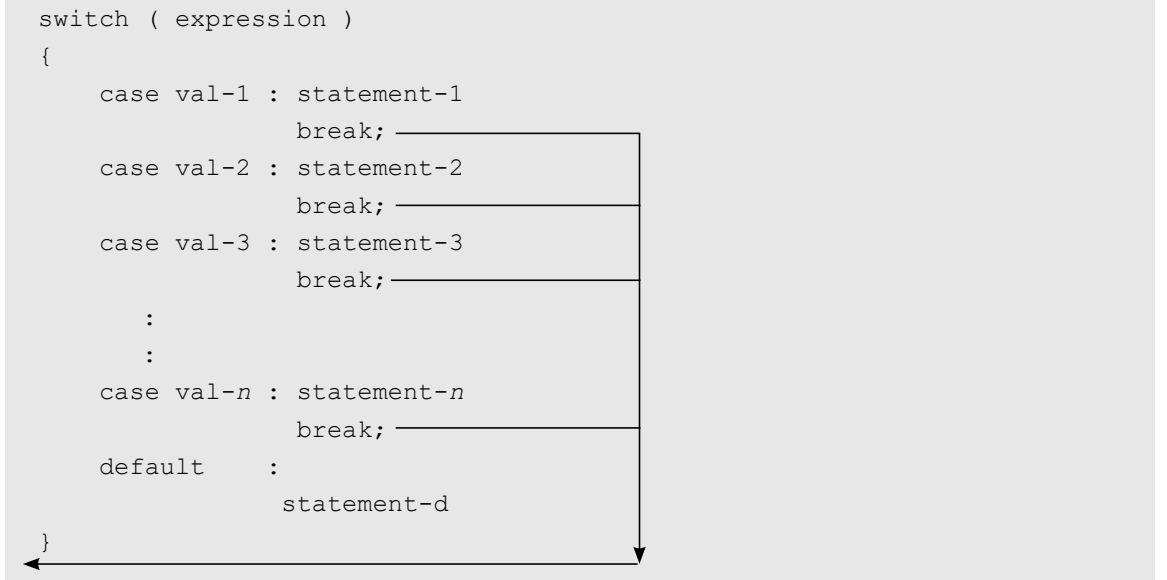
```

3.4 जम्पिंग स्टेटमेंट्स (JUMPING STATEMENTS)

ही विधाने प्रोग्रामच्या एका भागातून दुसऱ्या भागात नियंत्रण हस्तांतरित करतात. या विभागात, आपण त्यांचे कार्य पाहू या.

3.4.1 ब्रेक स्टेटमेंट (The break Statement)

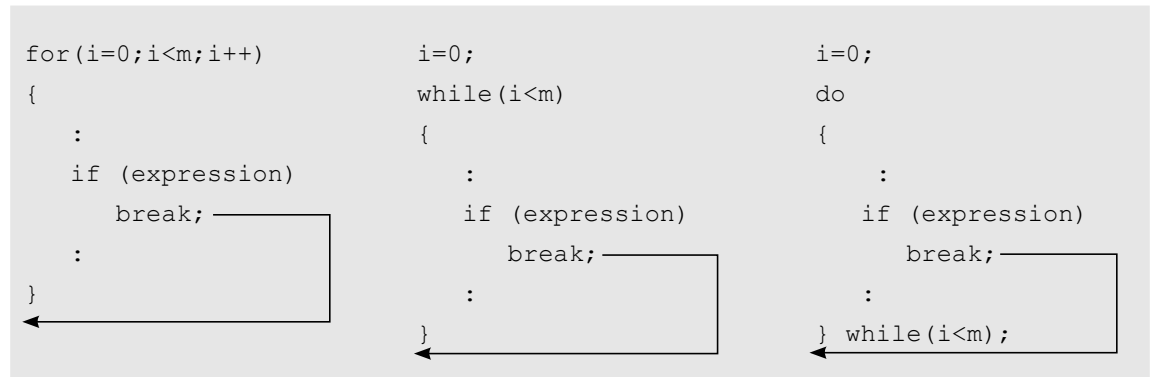
ब्रेक स्टेटमेंट नेहमीच स्विक स्टेटमेंटच्या आणि लूपिंग स्टेटमेंटच्या मुख्य भागात वापरली जाते.



चित्र 3.10: स्विक स्टेटमेंटमध्ये ब्रेक स्टेटमेंटची क्रिया

स्विक स्टेटमेंटमध्ये ब्रेक स्टेटमेंट हे शेवटची केस प्रकरण वगळता प्रत्येक केसचे शेवटचे विधान म्हणून वापरले जाते. जेव्हा अंमलात आणले जाते, तेव्हा हे स्विक स्टेटमेंटच्या बाहेर नियंत्रण हस्तांतरित करते आणि स्विक स्टेटमेंटनंतर च्या स्टेटमेंटपासून प्रोग्रामचे एक्झिक्युशन सुरू करते.

ब्रेक स्टेटमेंट हे व्हाइल, फॉर आणि डू - व्हाइल स्टेटमेंट्स मध्ये हे नेहमी इफ स्टेटमेंटच्या सहाय्याने वापरले जाते. लक्षात ठेवा लूपिंग स्टेटमेंटच्या मुख्य भाग नसल्यास इफ स्टेटमेंटसह ब्रेक कधीही वापरला जात नाही. एक्झिक्युट केल्यावर, हे लूपिंग स्टेटमेंटच्या बाहेरचे नियंत्रण स्थानांतरित करते आणि लूपिंग स्टेटमेंटनंतर च्या स्टेटमेंट पासून प्रोग्रामचे एक्झिक्युशन सुरू होते.



चित्र 3.11: व्हाइल, फॉर, डू - व्हाइल स्टेटमेंटमध्ये ब्रेक स्टेटमेंटची क्रिया

सोप्या भाषेत सांगायचे तर एक्झिक्युट केल्यावर ब्रेक लूपचे एक्झिक्युशन थांबवते.

Example 3.21: लूपमध्ये ब्रेक स्टेटमेंटचा वापर दाखवण्यासाठी प्रोग्राम लिहा.

Listing 3.21

```
#include <stdio.h>

int main()
{
    int i;
    for ( i = 1; i < 10; i++ )
    {
        if ( i % 5 == 0 )
            break;
        printf("%d ", i);
    }
    return 0;
}
```

Test Run

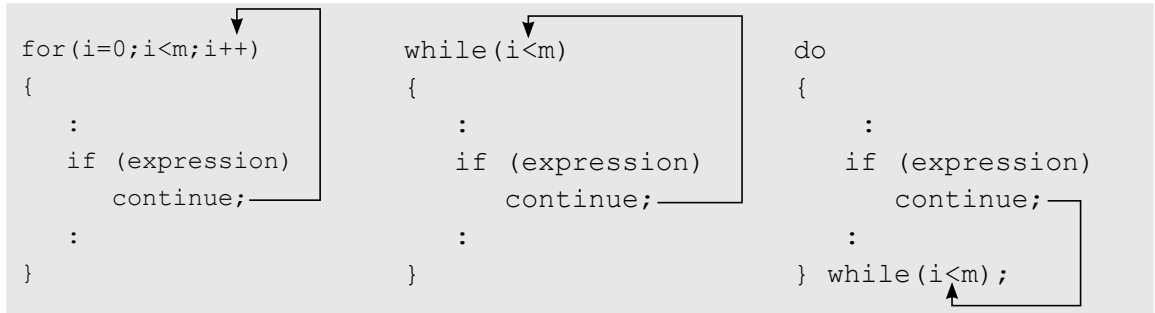
1 2 3 4

इफ स्टेटमेंटच्या सहाय्याने ब्रेक स्टेटमेंटचा वापर 5 ने भागाकार असलेल्या i च्या व्हॅल्यूसाठी for लूप बंद करेल.

3.4.2 कंटिन्यू स्टेटमेंट (The continue Statement)

लूपिंग स्टेटमेंटच्या मुख्य भागात कंटिन्यू स्टेटमेंट वापरले जाते. अशी परिस्थिती उद्भवू शकते जेव्हा दिलेल्या स्टेटमेंटपासून पुढे लूपच्या शेवटच्या स्टेटमेंटपर्यंतची उर्वरित स्टेटमेंट वगळली पाहिजे. हे कंटिन्यू स्टेटमेंट वापरून पूर्ण केले जाते.

कंटिन्यू स्टेटमेंट कंट्रोल स्थानांतरित करते लूपच्या पुढील पुनरावृत्तीच्या सुरुवातीस, जेणेकरून अद्याप एक्झिक्युट न झालेल्या स्टेटमेंट्स बायपास होतील. लक्षात ठेवा कंटिन्यू स्टेटमेंट नेहमीच if स्टेटमेंटच्या सहाय्याने वापरले जाते.



चित्र 3.12: व्हाइल, फॉर, डू – व्हाइल स्टेटमेंटमध्ये कंटिन्यू स्टेटमेंटची क्रिया

सोप्या भाषेत आपण असे म्हणू शकतो की एक्झिक्युट केल्यावर कंटिन्यू स्टेटमेंट चालू असलेले लूपची पुनरावृत्ती समाप्त करेल.

Example 3.22: लूपमध्ये कंटिन्यू स्टेटमेंटचा वापर दाखवण्यासाठी एक प्रोग्राम लिहा.

Listing 3.22

```
#include <stdio.h>
int main()
{
    int i;
    for ( i = 1; i < 10; i++ )
    {
        if ( i % 5 == 0 )
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

Test Run

```
1 2 3 4 6 7 8 9
```

इफ स्टेटमेंटच्या सहाय्याने कंटिन्यू स्टेटमेंटचा वापर 5 ने भागाकार असलेल्या i च्या वॅल्यूसाठी for लूपच्या उर्वरित स्टेटमेंट्स वगळेल.

फॉर लूपचे स्पष्टीकरणात्मक उदाहरणे

Example 3.23: दिलेल्या नंबर आणि टेबलमधील पंक्तींच्या(rows) संख्येसाठी गुणाकार प्रिंट करण्यासाठी प्रोग्राम लिहा.

उदाहरणार्थ, 5 संख्या साठी आणि पंक्ती (row) = 3 साठी, आउटपुट हे असावे:

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
```

Listing 3.23

```
#include <stdio.h>
int main()
{
    int num, rows, i, j;
    printf( "\nEnter number whose table to print : " );
    scanf( "%d", &num );
    printf( "\nEnter rows to print : " );
    scanf( "%d", &rows );
```

```

for ( i = 1; i <= rows; i++ )
{
    printf("\n%d x %d = %d", num, i, num*i) ;
}
return 0;
}

```

Test Run

```

Enter number whose table to print : 5
Enter rows to print : 3
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15

```

Example 3.24: दिलेली नैसर्गिक संख्या n एक प्राइम नंबर आहे की नाही हे शोधण्यासाठी प्रोग्राम लिहा.

नैसर्गिक संख्या प्राइम नंबर म्हटले जाते जर ती केवळ 1 ने आणि फक्त स्वतः ने विभाजित केले असेल म्हणजेच त्याचे घटक बनवता येत नाहीत. याव्यतिरिक्त, या व्याख्येनुसार, 2 वगळता सम संख्या ही प्राइम नंबर नाही. म्हणून, आमच्या चाचणी निकष

1. जर n हा 2 पेक्षा मोठा असेल आणि सम असेल तर n ही प्राइम नंबर नाही. (If n is greater than 2 and is even then n is not a prime number.)
2. पायरी 1 मधील चाचणी अयशस्वी झाल्यास आम्ही $k = 3, 5, 7, \dots \sqrt{n}$. याद्वारे घटक n विभाजित करण्याचा प्रयत्न करतो. म्हणून, जर n कोणत्याही k मूल्यांनी विभाज्य असेल तर संख्या n ही प्राइम नंबर नाही. (If test at step 1 fails, then we try to divide number n by factors $k = 3, 5, 7, \dots$. Therefore, if n is divisible by any value of k , number n is not a prime number.)
3. जर पायरी २ वर चाचणी देखील अपयशी ठरली तर n ही एक प्राइम नंबर आहे. (If test at step 2 also fails, then n is a prime number.)

पुढील प्रोग्रॅम या निकषाची एक्झिक्युशन करतो.

Listing 3.24

```

#include<stdio.h>
#include<math.h>
int main()
{
    int n, k, m;
    printf( "\nEnter a positive integer number: " );
    scanf("%d", &n);
    if ( ( n > 2 ) && ( ( n % 2 ) == 0 ) )
    {
        printf( "\n%d is not a prime number.\n", n );
    }
}

```

```

        return 0;
    }
    m = sqrt( n );
    for ( k = 3; k <= m; k += 2 )
    {
        if ( n % k == 0 )
        {
            printf( "\n%d is not a prime number.\n", n );
            return 0;
        }
    }
    printf( "\n%d is a prime number.\n", n );
    return 0;
}

```

Test Runs

First Run

```

Enter a positive integer number: 43
43 is a prime number.

```

Second Run

```

Enter a positive integer number: 92
92 is not a prime number.

```

Example 3.25: 1991 संख्या एक पॅलिंड्रोम आहे कारण पुढे वा मागे वाचताना तीच संख्या असते. दिलेली संख्या पॅलिंड्रोम आहे की नाही हे तपासण्यासाठी प्रोग्राम लिहा.

येथे प्रथम दिलेल्या संख्येचे अंक उलट करून नवीन संख्या बनवितो आणि नंतर दिलेल्या संख्येची उलट केलेल्या संख्येशी तुलना करू. जर ते जुळत असतील तर दिलेली संख्या पॅलिंड्रोम आहे अन्यथा ती पॅलिंड्रोम नाही.

Listing 3.25

```

#include<stdio.h>

int main()
{
    int sum = 0, digit;
    int number, temp;
    printf( "\nEnter any positive integer number: " );
    scanf( "%d", &number );

```

```

temp = number;
while ( temp > 0 )
{
    digit = temp % 10;
    temp /= 10;
    sum = sum * 10 + digit;
}
if ( number == sum )
    printf( "\n%d is a palindrome number.\n", number );
else
    printf( "\n%d is not a palindrome number.\n", number );
return 0;
}

```

Test Runs

First Run

```

Enter any positive integer number: 1991
1991 is a palindrome number.

```

Second Run

```

Enter any positive integer number: 1234
1234 is not a palindrome number.

```

Example 3.26: दिलेली नैसर्गिक संख्या आर्मस्ट्रॉंग नंबर आहे की नाही हे तपासण्यासाठी प्रोग्राम लिहा.

जर त्याच्या अंकांच्या घनची बेरीज ही संख्येइतकी असेल तर त्यास आर्मस्ट्रॉंग असे म्हणतात. उदाहरणार्थ, 153 एक आर्मस्ट्रॉंग नंबर आहे

$$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$$

Listing 3.26

```

#include<stdio.h>
int main()
{
    int n, t, s = 0, d;
    printf( "\nEnter 3-dgit natural number : " );
    scanf("%d", &n);
    t = n;
    while ( t > 0 )

```

```

{
    d = t % 10;
    s = s + d * d * d;
    t = t / 10;
}
if ( s == n )
    printf("\n%d is an Armstrong number.\n", n);
else
    printf( "\n%d is not an Armstrong number,\n", n);
return 0;
}

```

Test Runs

First Run

```

Enter 3-dgit natural number : 153
153 is an Armstrong number.

```

Second Run

```

Enter 3-dgit natural number : 135
135 is not an Armstrong number.

```

Example 3.27: फिबोनेकी अनुक्रम खालीलप्रमाणे परिभाषित केला आहे: अनुक्रमातील प्रथम आणि द्वितीय संज्ञा 0 आणि 1 आहेत. पहिल्या दोन टर्म्स सोडल्यास प्रत्येक पद हे ताबडतोब आधीच्या दोन पदांच्या बेरीजच्या रूपात प्राप्त होतो.

उदाहरणार्थ, n साठी इनपुट मूल्य 10 असल्यास आउटपुट हे असावे

0 1 1 2 3 5 8 13 21 34

फिबोनेकी अनुक्रमच्या प्रथम n टर्म्स प्रिंट करण्यासाठी प्रोग्राम लिहा.

Listing 3.27

```

#include<stdio.h>
int main()
{
    int prev = 0, curr = 1, next, n, count;
    printf( "\nEnter value for n(>2) : " );
    scanf( "%d", &n );
    printf( "%d %d", prev, curr );
    count = 2;
    while ( count < n )

```

```

{
    next = prev + curr;
    printf( "%d  ", next );
    count++;
    prev = curr;
    curr = next;
}
printf( "\n" );
return 0;
}

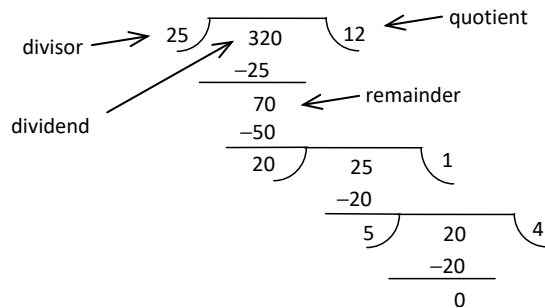
```

Test Run

Enter value for n(>2) : 10

0 1 1 2 3 5 8 13 21 34

Example 3.28: खालील आकृती दीर्घ विभाजन वापरून 25 आणि 320 या दोन सकारात्मक पूर्णांकांच्या सर्वात मोठ्या सामान्य विभाजक (GCD) ची गणना करण्याचा मार्ग दर्शवते.



चित्र 3.13: दीर्घ विभाजनाचा वापर करून GCD साठी संगणकीय प्रक्रियेचे उदाहरण

वरील आकृतीवरून, तुम्ही हे पाहिले असेल की सलग विभागांमध्ये, मागील भागाचे विभाजक लाभांश बनतात, उर्वरित भागाकार होतात आणि विभाजन पुन्हा चालते. बाकी शून्य होईपर्यंत ही प्रक्रिया चालते, आणि वर्तमान विभाजक दिलेल्या पूर्णांक संख्यांच्या GCD म्हणून घेतले जाते.

दोन सकारात्मक इन्टिजर m आणि n चा सर्वात मोठा सामान्य विभाजक (जीसीडी) शोधण्यासाठी प्रोग्राम लिहा.

Listing 3.28

```

#include<stdio.h>
void main()
{
    int m, n, r;
    printf( "\nEnter value for m : " );
    scanf( "%d", &m );

```



```

printf( "Enter value for n : " );
scanf( "%d", &n );
while ( 1 )
{
    r = n % m;
    if ( r == 0 )
    {
        printf( "\nGCD = %d\n", n );
        return;
    }
    n = m;
    m = r;
}
}

```

Test Run

```

Enter value for m : 25
Enter value for n : 320
GCD = 5

```

युनिट सारांश

या अध्यायात, आपण हे शिकलो आहोत.

- कंट्रोल स्टेटमेंट्स स्टेटमेंटच्या एक्झिक्युशनच्या क्रमास नियंत्रित करण्यास परवानगी देतात..
- विधानांच्या नियंत्रणामध्ये वैकल्पिक विधानांमधून विधान निवडणे किंवा काही निवडलेली विधाने वारंवार अंमलात आणणे यांचा समावेश असू शकतो..
- इफ-एल्स आणि स्विच स्टेटमेंट्स दिलेल्या अटच्या परिणामावर अवलंबून असलेल्या पर्यायांमधून विधान निवडण्याची परवानगी देताना निर्णय घेणारे विधान (Decision making statement) म्हणून ओळखले जातात.
- स्विच स्टेटमेंट केवळ अशा परिस्थितीत कार्य करते जिथे एखादी स्थिती अविभाज्य मूल्ये घेऊ शकते.
- फॉर, व्हाइल आणि डू-व्हाइल स्टेटमेंट्स पुनरावृत्ती किंवा लूपिंग स्टेटमेंट म्हणून ओळखले जातात कारण ते निवडलेल्या स्टेटमेंट्सची पुनरावृत्ती (iterate) करतात.
- फॉर स्टेटमेंटला अशा परिस्थितीत प्राधान्य दिले जाते जिथे आम्हाला स्टेटमेंट्स किती वेळा एक्झिक्युट करायचे हे आधीच माहित असते.
- व्हाइल आणि डू-व्हाइल स्टेटमेंट्स तेव्हा प्राधान्य दिले जाते जेव्हा किती वेळा स्टेटमेंट्स एक्झिक्युट करायची असतात हे विशिष्ट अटीच्या समाधानावर अवलंबून असते.
- व्हाइल आणि डू-व्हाइल स्टेटमेंट्स मध्ये फक्त एकच फरक आहे डू-व्हाइल स्टेटमेंट किमान एकदाच अंमलात आणली जातात तर व्हाइल लूप अजिबात अंमलात आणले जाऊ शकते किंवा नाही.

- ब्रेक स्टेटमेंट स्विक आणि सर्व लूपिंग स्टेटमेंट्स सह वापरले जाते. त्याच्या एक्झिक्युशन, ते ब्लॉकच्या बाहेर नियंत्रण हस्तांतरित करते.
- जेव्हा स्विक स्टेटमेंट मध्ये ब्रेक स्टेटमेंट वापरले जाते, तेव्हा ब्रेक स्टेटमेंट स्विक स्टेटमेंटच्या व्याप्तीच्या बाहेरचे नियंत्रण घेते.
- जेव्हा लूपिंग स्टेटमेंट मध्ये ब्रेक स्टेटमेंट वापरले जाते, तेव्हा ब्रेक स्टेटमेंट लूपच्या बाहेरचे नियंत्रण घेते, म्हणजेच लूपची एक्झिक्युशन थांबवते.
- कंटिन्यू स्टेटमेंट फक्त लूपिंग स्टेटमेंटसह वापरला जातो. त्याच्या खालील विधानांचे एक्झिक्युशन वगळते, म्हणजेच ती वर्तमान पुनरावृत्ती संपुष्टात आणते आणि पुढील पुनरावृत्ती नव्याने सुरू होते.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. जेव्हा इफ स्टेटमेंटला एल्स नसते, जेव्हा अट शून्य मूल्याचे मूल्यांकन करते तेव्हा काय होते?
2. स्विक स्टेटमेंटचा एल्स-इफ स्टेटमेंटपेक्षा काय फायदा आहे?
3. नेस्टिंग म्हणजे काय?
4. पुढील प्रोग्राममध्ये काही चूक आहे काय?

```
void main()
{
    char ch;
    if ( ( ch = getch() ) == 'a' )
        printf("\nYou typed character a\n");
}
```

5. ब्रेक स्टेटमेंट कोठे वापरले जाते आणि त्याचे कार्य काय आहे?
6. स्विक स्टेटमेंटचा वापर करून खालील कोड फ्रेगमेंट पुन्हा लिहा:

```
char code;
code = getchar();
if ( code == 'A' )
    printf("\nAccountant\n");
else if ( code == 'C' || code == 'G' )
    printf("\nGrade IV\n");
else if ( code == 'F' )
    printf("\nFinancial Advisor\n");
else
    printf("\nIncorrect code\n");
```

7. इफ-एल्स स्टेटमेंट्स वापरून खालील कोड फ्रेगमेंट पुन्हा लिहा:

```
int month;
scanf("%d", &month);
switch ( month )
{
    case 1 :
    case 3 :
    case 5 :
    case 7 :
    case 8 :
    case 10:
    case 12: printf("\nIt is a month having 31 days\n");
             break;
    case 4 :
    case 6 :
    case 9 :
    case 11: printf("\nIt is a month having 30 days\n");
             break;
    case 2:  printf("\nIt is a month having 28/29 days\n");
    default: printf("\nIncorrect month\n");
}
}
```

8. स्विच स्टेटमेंटचा वापर करून खालील कोड फ्रेगमेंट पुन्हा लिहा

if (ch == 'N')	if (ch == 'O')
north++;	Outstanding++;
if (ch == 'S')	else if (ch == 'E')
south++;	Excellent++;
if (ch == 'E')	else if (ch == 'G')
east++;	Good++ ;
if (ch == 'W')	else if (ch == 'P')
west++;	Poor++;
else	else
unknown++;	Unknown++ ;

9. स्विच आणि इफ-एल्समधील फरक लिहा.

10. स्विच स्टेटमेंटमध्ये ब्रेक स्टेटमेंटच्या अनुपस्थितीचा काय परिणाम होतो?

11. जर व्हाइल स्टेटमेंट मधील एक्सप्रेशन सुरुवातीस चुकिचे असल्यास काय होते?

12. व्हाइल आणि डू- व्हाइल स्टेटमेंट मधील फरक लिहा.

13. केव्हा व्हाइल स्टेटमेंट पेक्षा डू- व्हाइल स्टेटमेंटला प्राधान्य दिले जाते?
14. फॉर स्टेटमेंटच्या इनक्रेमेंट पार्ट मध्ये मल्टिपल इनक्रेमेंट एक्सप्रेशन असू शकतात का?
15. केव्हा फॉर स्टेटमेंट पेक्षा व्हाइल स्टेटमेंटला प्राधान्य दिले जाते?
16. कंटिन्यू स्टेटमेंटची काय कृती आहे?
17. ब्रेक आणि कंटिन्यू स्टेटमेंट मधील फरक लिहा.

एकाधिक निवड प्रश्न

1. पुढील कोडचा विचार करा:

```
switch (ch)
{
    case 'a': printf("a");
    case 'b': printf("b");
    default: printf("error");
}
```

कॅरेक्टर व्हेरिएबल ch ला 'a' व्हॅल्यू दिल्यास आउटपुट मिळेल

- (a) a (b) ab (c) aberror (d) Error

2. पुढील कोडचा विचार करा:

```
int a = 6, b = 6;
if ( b == 6 || --a )
{
    statement1;
    statement2;
}
```

समजा स्टेटमेंट 1 आणि स्टेटमेंट 2 मध्ये a आणि b चे मूल्य बदलत नसेल तर वरील कोड लागू केल्यावर a चे मूल्य किती असेल?

- (a) 6 (b) 5 (c) 7 (d) Error

3. स्विच स्टेटमेंटबद्दल खालीलपैकी कोणते विधान सत्य आहे?

- (a) यात शून्य किंवा अधिक केस असू शकतात.
- (b) कॉन्स्टन्ट एक्सप्रेशन ही वैध केस मूल्ये आहेत.
- (c) प्रत्येक प्रकरणातील स्टेटमेंट ब्लॉकमध्ये ब्रेक स्टेटमेंट असणे आवश्यक असते कारण नंतरचे केस मध्ये नियंत्रण येऊ नये म्हणून.
- (d) वरील सर्व.

4. जर स्विच स्टेटमेंटमध्ये डीफॉल्ट विधान वगळले गेले असेल आणि केस व्हॅल्यूज बरोबर कोणतेही जुळले नसेल तर

- (a) स्विच ब्लॉकमधील कोणतेही विधान एक्झिक्युट झाले नाही.
- (b) स्विच ब्लॉकमधील सर्व विधाने एक्झिक्युट करा.

(c) केवळ शेवटच्या केस ब्लॉकमध्ये स्टेटमेंट्स एक्झिक्युट करा.

(d) रन टाईम लुटी उद्भवते.

5. खालील प्रोग्राम कंपाईल आणि रन केल्यामुळे काय होईल?

```
#include <stdio.h>
void main()
{
    int c;
    printf( "\nEnter value of c: " );
    scanf( "%d", &c);
    switch(c);
    {
        case 1 : printf( "\nHello 1\n" );
                break;
        case 2 : printf( "\nHello 2\n" );
                break;
        default : printf( "\nInvaild value\n" );
                 break;
        case 3 : printf( "\nHello 3\n" );
                 break;
        case 4 : printf( "\nHello 4\n" );
    }
}
```

(a) प्रोग्राम कंपाईल करण्यात अयशस्वी होईल कारण सर्व वैध केस नंतर डीफॉल्ट केवळ शेवटी दिसू शकते.

(b) प्रोग्राम कंपाईल आणि एक्झिक्युट करेल.

(c) प्रोग्राम कंपाईल करण्यात अयशस्वी होईल आणि कंपाईलर वाक्यरचनातील लुटीचा अहवाल म्हणून “*Case outside of switch statement in function main*” नंतर “*Misplaced break in function main*” सर्व केस साठी आणि ब्रेक स्टेटमेंटसाठी करेल.

(d) वरीलपैकी कोणतेही नाही.

6. पुढील प्रोग्रामचा विचार करा

```
#include <stdio.h>
void main()
{
    int a, b = 10;
    scanf( "%d", &a );
    if ( a = 0 )
```

```
        b *= 2;
    else
        b /= 2;
    printf( "%d", b );
}
```

प्रोग्राम 6 नंबर या युझरच्या इनपुटसह अंमलात आणल्यास काय होईल?

- (a) 20 (b) 10 (c) 5 (d) None

7. खालीलपैकी कोणते स्विच स्टेटमेंटमध्ये वैध प्रकारचे एक्सप्रेशन नाही?

- (a) character (b) integer (c) float (d) enum

8. पुढील प्रोग्रामचे आउटपुट काय असेल?

```
void main() {
    char val=1;
    if(val--==0)
        printf("TRUE");
    else
        printf("FALSE");
}
```

- (a) TRUE (b) FALSE
(c) Error (d) None of above

9. पुढील प्रोग्रामचे आउटपुट काय असेल?

```
#define TRUE 1
void main()
{
    if(TRUE)
        printf("1");
        printf("2");
    else
        printf("3");
        printf("4");
}
```

- (a) Error (b) 1 (c) 12 (d) 34

10. पुढील प्रोग्रामचे आउटपुट काय असेल?

```
void main()
{
```

```

int a=10;
switch(a) {
    case 5+5:
        printf("Hello\n");
    default:
        printf("OK\n");
}

```

- (a) Hello OK (b) OK (c) Hello (d) Error

11. पुढील प्रोग्रामचे आउटपुट काय असेल?

```

void main()
{
    int a=2;
    switch(a)
    {
        printf("Message\n");
        default:
            printf("Default\n");
        case 2:
            printf("Case-2\n");
        case 3:
            printf("Case-3\n");
    }
    printf("Exit from switch\n");
}

```

- (a) Case-2 (b) Message
(c) Message (d) Case-2
Case- 2 Case- 3
Exit from switch

12. पुढील प्रोग्रामचे आउटपुट काय असेल?

```

void main()
{
    short day=2;
    switch(day)

```

- (a) 2nd (b) 22nd (c) Error (d) 2nd
22nd

13. पुढील प्रोग्रामचे आउटपुट काय असेल?

- (a) One (b) Two (c) Error (d) Else

14. पुढील प्रोग्रामचे आउटपुट काय असेल?

- (a) Hello (b) No output (c) Garbage value (d) Error

15. पुढील प्रोग्रामचे आउटपुट काय असेल?

```
void main()
{
    int x;
    float y = 5.5;
    switch(x=y+1)
    {
        case 6: printf("It's Eight."); break;
        default: printf("Oops No choice here.");
    }
}
```

- (a) Oops No choice here. (b) It's Eight.Oops No choice here!!!
(c) It's Eight. (d) Error

16. पुढील कोडचा विचार करा:

```
while ( ++i <= n );
```

जेव्हा लूप पूर्ण होईल तेव्हा i चे मूल्य किती असेल?

- (a) n (b) $n-1$ (c) $n+1$ (d) $n+2$

17. पुढील कोडचा विचार करा: त्याचे आउटपुट काय आहे?

```
for ( i = 1; i <= 5; i++ )
{
    printf( "%d", i += 2 );
}
```

- (a) 1 2 3 4 5 (b) 3 4 5 6 7 (c) 3 5 7 9 11 (d) 3 6

18. पुढील विधानांपैकी कोणते विधान कंटिन्यू स्टेटमेंट बद्दल खरे नाही?

- (a) हे स्वीच स्टेटमेंटच्या संयुक्त रूपात वापरले जाऊ शकते.
(b) हे लूप संपवते.
(c) हे वर्तमान पुनरावृत्ती समाप्त करते.
(d) हे सर्व लूपिंग स्टेटमेंट्स तसेच स्विच स्टेटमेंटसह वापरले जाऊ शकते.

19. खालीलपैकी कोणते विधान स्विच स्टेटमेंटबद्दल खरे नाही?

- (a) यात शून्य किंवा अधिक केस असू शकतात.
(b) कॉन्स्टंट एक्सप्रेशन ही वैध केस मूल्ये आहेत.
(c) स्विच स्टेटमेंट मधील अभिव्यक्ती प्लोट प्रकारातील असू शकते.
(d) प्रत्येक प्रकरणातील स्टेटमेंट ब्लॉकमध्ये ब्रेक स्टेटमेंट असणे आवश्यक असते कारण नंतरचे केस मध्ये नियंत्रण येऊ नये म्हणून.

20. कंटिन्यू स्टेटमेंट _____ यूज्ड करण्यासाठी.

- (a) लूप स्टेटमेंटची पुढील पुनरावृत्ती सुरू ठेवा.
- (b) लूप स्टेटमेंटच्या ब्लॉकमधून बाहेर पडा.
- (c) सर्वात बाह्य ब्लॉकमधून बाहेर पडा जरी तो सर्वात आतल्या ब्लॉकमध्ये वापरला जातो.
- (d) प्रोग्रॅमची एक्झिक्युशन सुरू ठेवा अगदी लुटी उद्धवते तरी.

21. पुढील प्रोग्रामचा विचार करा

```
void main()
{
    int i, j;
    for ( i = 0; j = 10; i < j; i++, j-- );
    printf( "x" );
}
```

‘x’ अक्षर किती वेळा छापले जाईल?

- (a) 5
- (b) 1
- (c) 10
- (d) 4

22. पुढील प्रोग्रामचा विचार करा

```
void main()
{
    unsigned char ch;
    for ( ch = 0; ch <= 256; ch++ )
        printf( "%d = %c", ch, ch );
}
```

फॉर लूप किती वेळा एक्झिक्युट होईल?

- (a) 256
- (b) 255
- (c) 257
- (d) Infinitely

23. पुढील प्रोग्राममध्ये किती वेळा व्हाइल लूप एक्झिक्युट होईल?

```
void main()
{
    int j = 1;
    while ( j <= 100 );
    {
        printf("\nj = %d, j*j = %d", j, j*j );
        j++;
    }
}
```

- (a) Infinite times
- (b) 100 times
- (c) 99 times
- (d) 101 times

24. पुढील प्रोग्रामचे आउटपुट काय असेल?

```
void main()
{
    int i;
    for( i = 0; i < 5; i++ )
    {
        int j = 3;
        printf("%d ", i*j);
    }
    printf("%d", j);
}
```

- (a) प्रोग्राम कंपाईल करेल आणि यशस्वीरित्या एक्झिक्युट होईल.
- (b) प्रोग्राम कंपाईल करण्यात अयशस्वी होईल.
- (c) प्रोग्राम 0 3 6 9 12 3 म्हणून आउटपुट देईल.
- (d) वरीलपैकी काहीही नाही.

25. पुढील प्रोग्रामचे आउटपुट काय असेल?

```
void main()
{
    int i = 0;
    for(;;)
    {
        if ( i++ == 4 ) break;
        continue;
    }
    printf("i = %d", i);
}
```

- (a) i = 4
- (b) i = 5
- (c) i = 0
- (d) Error

ANSWERS									
1.	(c)	2.	(a)	3.	(d)	4.	(a)	5.	(c)
6.	(c)	7.	(c)	8.	(a)	9.	(c)	10.	(c)
11.	(d)	12.	(c)	13.	(b)	14.	(b)	15.	(c)
16.	(c)	17.	(d)	18.	(d)	19.	(c)	20.	(a)
21.	(b)	22.	(d)	23.	(a)	24.	(b)	25.	(b)

प्रोग्रामिंग समस्या

1. कोणताही अंकगणित ऑपरेटर न वापरता दिलेली नैसर्गिक संख्या EVEN किंवा ODD आहे की नाही हे तपासण्यासाठी एक प्रोग्राम लिहा.
2. a , b आणि c या तीन रेषाखंडांचे मोजमाप दिल्यास हे लाइन विभाग त्रिकोण तयार करण्यासाठी वापरता येऊ शकतात की नाही हे तपासण्यासाठी प्रोग्राम लिहा.
3. ABC त्रिकोणाच्या तीन कोनांचे अनुक्रमे a , b आणि c अंश दिले तर या कोनातून त्रिकोण तयार होऊ शकतो की नाही हे तपासण्यासाठी प्रोग्राम लिहा.
4. दिलेली वर्णमाला स्वर किंवा व्यंजन (vowel or a consonant) आहे की नाही हे तपासण्यासाठी स्विच स्टेटमेंटचा वापर करून एक प्रोग्राम लिहा.
5. कोनातून अंश आणि अँटी-क्लॉकवाइव्ह दिशानिर्देश दिल्यास कोनचा प्रकार शोधण्यासाठी प्रोग्राम लिहा.
6. $A(x_1, y_1)$, $B(x_2, y_2)$ and $C(x_3, y_3)$, असे तीन बिंदू दिले तर ते कॉलिनर म्हणजेच, समान रेषेवरील आहेत की नाही ते निश्चित करा.
7. Given points (x_1, y_1) & (x_2, y_2) on line AB , and points (x_3, y_3) & (x_4, y_4) on line CD , ओळी एबी आणि सीडी एकमेकांना छेदतात की नाही हे ठरवण्यासाठी प्रोग्राम लिहा.
8. दिलेल्या तारखेला आठवड्याचा दिवस शोधण्यासाठी एखादा प्रोग्राम लिहा.
9. आपली जन्मतारीख आणि आजची तारीख दिले जाते तेव्हा आपले वय शोधण्यासाठी प्रोग्राम लिहा, दोन्ही $dd/mm/yyyy$ स्वरूपात.
10. 12 तासांच्या सिस्टममधून 24 तासांच्या सिस्टममध्ये वेळ रूपांतरित करण्यासाठी प्रोग्राम लिहा.
11. 24 तासांच्या सिस्टममधून 12 तासांच्या सिस्टममध्ये वेळ रूपांतरित करण्यासाठी एक प्रोग्राम लिहा.
12. सुरुवातीची वेळ आणि समाप्तीची वेळ दोन्ही $hh:mm:ss$ स्वरूपात असताना वेळेत फरक शोधण्यासाठी एखादा प्रोग्राम लिहा.
13. n th प्राइम नंबर मिळवण्यासाठी प्रोग्राम लिहा.
14. प्रथम n प्राइम नंबर प्रिंट करण्यासाठी प्रोग्राम लिहा.
15. फिबोनेची अनुक्रमांची n th term शोधण्यासाठी प्रोग्राम लिहा.
16. आपणास माहित आहे की सम संख्या ही एक संख्या आहे जी 2 ने भागली जाते. तथापि दिलेली संख्या एक आहे की नाही हे तपासण्यासाठी आणखी एक दृष्टिकोन वापरला जाऊ शकतो. दृष्टीकोन युनिट अंकाची तपासणी करून आहे - जर युनिट अंक 0, 2, 4, 6 किंवा 8 असेल तर संख्या समान आहे. म्हणून दिलेली संख्या 2 ने भाग न घेता दिलेली संख्या समान आहे की नाही हे तपासण्यासाठी आपल्याला प्रोग्राम लिहिणे आवश्यक आहे.

प्राॅक्टिकल

1. चतुर्भुज समीकरणाची मुळे (roots of a quadratic equation) शोधण्यासाठी प्रोग्राम लिहा.
Refer to Example 3.8.
2. दिलेला नैसर्गिक क्रमांक हा प्राइम नंबर आहे की नाही याची चाचणी घेण्यासाठी प्रोग्राम लिहा.
Refer to Example 3.24.

3. दिलेला नैसर्गिक क्रमांक पॅलिंड्रोम नंबर आहे की नाही याची चाचणी घेण्यासाठी प्रोग्राम लिहा.

Refer to Example 3.25.

4. *dd/mm/yyyy* स्वरूपात दिलेली तारीख वैध आहे की नाही हे तपासण्यासाठी प्रोग्राम लिहा.

Listing 3.29

```
/*
    Program to check whether date given in format dd/mm/yyyy
    is valid or not.
*/
#include<stdio.h>
int main()
{
    int mm, dd, yyyy;
    char ch;
    printf( "\nEnter date in format dd/mm/yyyy : ");
    scanf( "%2d%c%2d%c%4d", &dd, &ch, &mm, &ch, &yyyy );
    if ( mm < 1 || mm > 12 ) {
        printf( "\nDate is Invalid.\n");
        return 0;
    }
    if ( mm == 2 ) {
        /* condition to check for leap year */
        if ( (yyyy%400==0)|| ( (yyyy%4==0)&&(yyyy%100!=0) ) )
        {
            if ( dd > 1 && dd <= 29 )
                printf( "\nDate is valid.\n");
            else
                printf( "\nDate is invalid.\n");
        } else {
            if ( dd > 1 && dd <= 28 )
                printf( "\nDate is valid.\n");
            else
                printf( "\nDate is invalid.\n");
        }
    } else if ( mm == 4 || mm == 6 || mm == 9 || mm == 11 ) {
        if ( dd > 1 && dd <= 30 )
```

```

        printf( "\nDate is valid.\n");
    else
        printf( "\nDate is invald.\n");
} else {
    if ( dd > 1 && dd <= 31 )
        printf( "\nDate is valid.\n");
    else
        printf( "\nDate is invald.\n");
}
return 0;
}

```

Test Runs

First Run

```

Enter date in format dd/mm/yyyy : 29/2/2020
Date is valid.

```

Second Run

```

Enter date in format dd/mm/yyyy : 29/2/2021
Date is invalid.

```

5. सर्व आर्मस्ट्रॉंग नंबर प्रिंट करण्यासाठी एक प्रोग्राम लिहा.

लक्षात घ्या की आर्मस्ट्रॉंग 3 – अंकांची संख्या. म्हणून, आम्ही 100 पासून सुरू (प्रथम 3-अंकी संख्या) आणि 999 (शेवटचा 3-अंक क्रमांक) पर्यंत पुढे जाऊ आणि मार्गावर प्रत्येक आर्मस्ट्रॉंग नंबर आहे की नाही हे तपासण्यासाठी चाचणी करतो.

Listing 3.30

```

/*
    Program to print all Armstrong numbers
*/

#include<stdio.h>
int main()
{
    int i, t, s, d;
    printf( "\nFollowing is list of all Armstrong numbers.\n\n");
    for ( i = 100; i <= 999; i++ )
}

```

```

{
    t = i;
    s = 0;
    while ( t > 0 )
    {
        d = t % 10;
        s = s + d * d * d;
        t = t / 10;
    }
    if ( s == i )
        printf("%d ", i);
}
printf( "\n");
return 0;

```

Test Run

Following is list of all Armstrong numbers.
153 370 371 407

6. m....n श्रेणीत सर्व प्राइम नंबर प्रिंट करण्यासाठी एक प्रोग्राम लिहा. m आणि n साठीची मूल्ये एक्झिक्युशनच्या वेळी युझरद्वारे पुरविली जातील.

Listing 3.31

```

/*
    Program to find prime numbers in the range m..n. Value
    of m and n will be supplied by user at execution time
*/
#include<stdio.h>
#include<math.h>
int main()
{
    int i, m, n, k, mm;
    printf("\nProgram to find prime numbers in range m..n\n");
    printf( "\nEnter value of m : ");
    scanf( "%d", &m );

```

```

printf( "\nEnter value of n : ");
scanf( "%d", &n );
printf( "\n\nPrime numbers in range %d..%d are\n\n", m,
n);
for ( i = m; i <= n; i++ )
{
    if ( ( i > 2 ) && ( ( i % 2 ) == 0 ) )
        continue;
    mm = sqrt(i);
    for ( k = 3; k <= mm; k += 2 )
    {
        if ( i % k == 0 )
            break;
    }
    if ( k > mm )
        printf( "%d ", i );
}
printf( "\n");
return 0;
}

```

Test Run

```

Program to find prime numbers in range m..n
Enter value of m : 10
Enter value of n : 50
Prime numbers in range 10..50 are
11 13 17 19 23 29 31 37 41 43 47

```

आणखी माहिती

कंडिशनल ब्रॅचिंग आणि लूप्स हा विषय प्रोग्रामिंगचा मुख्य भाग आहे. त्यामुळे विद्यार्थ्यांकडून या विषयावर जोरदार पकड असणे अपेक्षित आहे.

शिक्षकांच्या अपेक्षेनुसार तर्कशास्त्र विकसित करण्यासाठी चर्चा केलेल्या समस्येवर

विद्यार्थ्यांचा सक्रिय सहभाग अपेक्षित आहे.

शिक्षकांनी समस्येचे निराकरण करण्याचे निर्देश देऊ नये, म्हणूनच विद्यार्थ्यांनी पुस्तकांमध्ये सूचीबद्ध प्रोग्रामचा संदर्भ म्हणून वापर करून स्वतः हून प्रोग्राम विकसित केले पाहिजे.

प्रोग्रामची शुद्धता सुनिश्चित करण्यासाठी शिक्षकांनी चाचणी व डीबगिंगची प्रक्रिया देखील दर्शविली पाहिजे.

संदर्भ आणि सूचविलेले वाचन

1. R.S. Salaria, Problem Solving & programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi..
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

4

अरे

युनिट वैशिष्ट्ये

हे युनिट अरेशी संबंधित विषयांवर चर्चा करते. हे अरे न्यूमरिक किंवा कॅरेक्टर डेटा असू शकतात. C भाषेत स्ट्रिंग हाताळण्यासाठी कॅरेक्टरचा अरे वापरला जातो. हे युनिट अरे आणि स्ट्रिंगचे विविध पैलू स्पष्ट करते आणि योग्य उदाहरणांसह त्यांचा वापर प्रदर्शित करते.

तर्कशास्त्र

वास्तविक जीवनातील अडचणींमध्ये, वास्तविक जीवनातील परिस्थिती उद्भवते जिथे आपल्याला एकसंध किंवा विवादास्पद असू शकतात अशा डेटा संकलनाशी सामना करावा लागतो. यापुढे डेटा संकलन संख्यात्मक किंवा अ-संख्यात्मक असू शकते. म्हणूनच, विशेषतः एकसंध, डेटा संकलित करताना वास्तविक जीवनातील समस्यांचे निराकरण करण्यासाठी आम्हाला ते संग्रह स्मृतीत संग्रहित करण्यासाठी योग्य डेटा स्ट्रक्चर आवश्यक आहे आणि समस्येचे निराकरण होण्यापर्यंत त्यांची प्रक्रिया सुलभ करणे आवश्यक आहे.

अरे एक असे डेटा स्ट्रक्चर आहे ज्यात एकसंध डेटा संग्रहित करणे तसेच प्रक्रिया करणे सोपे आहे. हे युनिट विद्यार्थ्यांना अरेशी संबंधित विविध पैलू समजून घेण्यास आणि एकसमान डेटा संग्रहणासह वास्तविक जीवनातील समस्यांसाठी प्रोग्राम विकसित करण्यास मदत करेल.

पूर्व-आवश्यकता

- मूलभूत डेटा प्रकार (Basic data types)
- लूप्स आणि नेस्टेड लूप्स (Loops and nested loops)
- कंडिशनल ब्रँचिंग (Condition branching)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील.

U4-O1:	अरेची संकल्पना समजावून सांगा.
U4-O2:	डिक्लेअर, इनिशियालाइज आणि अरेचे इनपुट / आउटपुट करा.
U4-O3:	अल्गोरिदम आणि प्रोग्राम तयार करण्यासाठी अरे वापरा.
U4-O4:	मॅट्रिक्सशी संबंधित समस्या सोडवण्यासाठी अरे वापरा.
U4-O5:	स्ट्रिंग्स अरे म्हणून स्पष्टीकरण द्या.
U4-O6:	स्टँडर्ड स्ट्रिंग हँडलिंग फंक्शन्सच्या वापराचे प्रदर्शन करा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 4 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U4-O1	1							
U4-O2			1					
U4-O3						3		
U4-O4						3		
U4-O5			1					
U4-O6						2		

4.1 ओळख (INTRODUCTION)

अरे म्हणजे एकसमान डेटा घटकांचे संग्रह (म्हणजे समान डेटा प्रकाराचे) एकाच नावाने वर्णन केले आहे आणि अरेच्या प्रत्येक घटकाचा संदर्भ सबस्क्रिप्टेड व्हेरिएबलद्वारे केला जातो, जो अरे नावाला चिकटवून तयार केलेले ब्रॅकेट्स मध्ये सबस्क्रिप्ट किंवा इंडेक्स असतो.

सबस्क्रिप्ट या शब्दाचा अर्थ गणिताच्या अंकात सारखाच आहे.

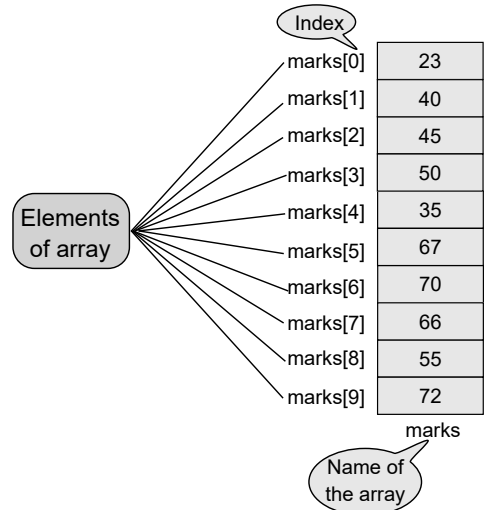
अरेचा प्रकार (Type of Arrays)

- **वन डायमेंशनल अरे (One-dimensional (1-D) array)** - हा अरेचा एक प्रकार आहे जिथे त्यातील घटक ऍक्सेस करण्यासाठी आपल्याला केवळ सिंगल सबस्क्रिप्ट आवश्यक आहे. वन डायमेंशनल अरे हा लिनिअर अरे म्हणून देखील ओळखला जातो. हा अरेचा सर्वाधिक वापर केला जाणारा प्रकार आहे
- **टू डायमेंशनल अरे (Two-dimensional (2-D) array)** - हा अरेचा एक प्रकार आहे जिथे त्याच्या घटकांमध्ये प्रवेश करण्यासाठी आपल्याला दोन सबस्क्रिप्टची आवश्यकता आहे. या प्रकारच्या अरेचा संग्रह डेटा संग्रहित करण्यासाठी केला जातो जिथे त्याचे घटक आयताकृती फॅशनमध्ये तयार केले जातात, म्हणजेच पंक्ती आणि स्तंभांमध्ये. व्यवसायाच्या संज्ञेमध्ये आपण याला एक टेबल म्हणतो आणि गणिताच्या परिभाषा मध्ये आम्ही त्याला मॅट्रिक्स म्हणतो.
- **मल्टि डायमेंशनल अरे (Multi-dimensional arrays)** - हा अरेचा एक प्रकार आहे जिथे त्याच्या घटकांमध्ये प्रवेश करण्यासाठी आपल्याला दोनपेक्षा अधिक सबस्क्रिप्टची आवश्यक आहेत. या प्रकारच्या अरेचा वापर जटिल अभियांत्रिकी समस्यांशी संबंधित डेटा संचयित करण्यासाठी केला जातो.

या युनिटमध्ये आमची चर्चा 1-डी आणि 2-डी पर्यंत मर्यादित राहिल. याव्यतिरिक्त, आम्ही त्या स्ट्रिंग्स बदल देखील चर्चा करू जे कॅरेक्टरच्या अरेचे विशेष केस आहेत.

4.2 वन डायमेंशनल अरे (ONE-DIMENSIONAL ARRAYS)

चित्र 4.1 मध्ये वन डायमेंशनल अरेचे विविध पैलू रेखाटले आहेत, ज्याचे नाव marks आहे, जे 10 विद्यार्थ्यांचे गुण संग्रहित करतात



चित्र 4.1: marks अरे

4.2.1 डिक्लेरेशन (Declaration)

अरे वापरण्यापूर्वी डिक्लेर करणे आवश्यक आहे. अरे डिक्लेरेशन कंपाईलरला अरेचे नाव, त्यातील घटकांचे प्रकार आणि अरेमधील घटकांची साईझ किंवा नंबर सांगते. अरेची साईझ कॉन्स्टन्ट असते आणि कॅम्पयलेशनच्या वेळी त्याचे मूल्य असणे आवश्यक आहे.

वन डायमेंशनल अरे डिक्लेर करण्यासाठी सिंटॅक्स आहे

```
type arrayName[arraySize];
```

चित्र 4.2 तीन वेगवेगळ्या अरे चे डिक्लेरेशन दर्शविते, एक इन्टिजर, एक फ्लोटिंग-पॉइंट नंबर आणि एक कॅरेक्टरसाठी. अरेचे घटक याक्षणी केवळ पूर्णतेसाठी दर्शविले आहेत.

marks[0]	56
marks[1]	76
marks[2]	54
marks[3]	77
marks[4]	50
marks[5]	80
marks[6]	68
marks[7]	43
marks[8]	69
marks[9]	80

```
int marks[10];
```

(a)

cgpa[0]	5.0
cgpa[1]	4.5
cgpa[2]	7.0
cgpa[3]	8.0
cgpa[4]	6.5
cgpa[5]	7.0
cgpa[6]	5.5
cgpa[7]	6.0
cgpa[8]	8.5
cgpa[9]	9.0

```
float cgpa[10];
```

(b)

name[0]	A
name[1]	n
name[2]	n
name[3]	i
name[4]	
name[5]	S
name[6]	u
name[7]	d
name[8]	\0
name[9]	

```
char name[10];
```

(c)

चित्र 4.2: 1-डी अरे डिक्लेरेशन

डिक्लेरेशन स्टेटमेंट

```
int marks[10];
```

इन्टिजरच्या प्रत्येक घटकासाठी 20 बाइट्स (टर्बो सी / सी ++ कंपाइलर, जे एक 16 बिट कंपाइलर आहे) चे contiguous मेमरी ब्लॉकचे वाटप करते. पहिले 2 बाइट अरेच्या पहिल्या घटकासाठी वापरले जातात (marks[0]), अरेच्या दुसऱ्या घटकासाठी पुढील 2-बाइट (marks[1]), आणि अशाच प्रकारे. शेवटचे 2-बाइट (marks[9]) घटकासाठी वापरले जातात.

डिक्लेरेशन स्टेटमेंट

```
float cgpa[10];
```

फ्लोटच्या प्रत्येक घटकासाठी 40 बाइट्सचा contiguous मेमरी ब्लॉक वाटप करते. पहिले 4-बाइट अरेच्या पहिल्या घटकासाठी वापरले जातात (cgpa[0]), अरेच्या दुसऱ्या घटकासाठी पुढील 4-बाइट (cgpa[1]), आणि अशाच प्रकारे. शेवटचे 4-बाइट (cgpa[9]), घटकासाठी वापरले जातात.

डिक्लेरेशन स्टेटमेंट

```
char name[10];
```

कॅरेक्टरच्या प्रत्येक घटकासाठी 10 बाइटचे contiguous मेमरी ब्लॉकचे वाटप करते. पहिला बाइट अरेच्या पहिल्या घटकासाठी (name[0])वापरला जातो, अरेच्या दुसऱ्या घटकासाठी पुढील बाइट (name[1])आणि अशाच प्रकारे. शेवटचे बाइट (name[9]) घटकासाठी वापरले जाते

4.2.2 इनिशियलिझेशन (Initialization)

ज्याप्रमाणे डिक्लेरेशनसह सामान्य व्हेरिएबल्स इनिशिएलाइझ केले जाऊ शकतात तसेच अरेचे घटक देखील इनिशिएलाइझ केले जाऊ शकतात. अरेमधील प्रत्येक घटकासाठी आम्ही मूल्य प्रदान करतो. फरक फक्त इतकाच आहे की ही मूल्ये ब्रेसिसमध्ये बंद केलेली असणे आवश्यक आहे, आणि जर एकापेक्षा जास्त असल्यास स्वल्पविरामाने विभक्त केले जावे.

पुढील उदाहरणे वन डायमेंशनल अरे इनिशिएलाइझ करण्याच्या सर्व संभाव्य पद्धती प्रदान करतात
खालील विधान

```
int b[10]={12, 0, 14, -4, 7, 8, 10, 11, 9, 15};
```

इनिशिएलाइझ घटक $b[0]$ to 12, $b[1]$ to 0, $b[2]$ to 14, $b[3]$ to -4, $b[4]$ to 7, $b[5]$ to 8, $b[6]$ to 10, $b[7]$ to 11, $b[8]$ to 9, $b[9]$ to 15 करते.

जर अरे अशा प्रकारे इनिशिएलाइझ केला असेल

```
int b[] = {12, 0, 14, -4, 7};
```

अरेची साईझ निर्दिष्ट केलेला नसल्यामुळे कंपाईलर आरंभिक यादीतील घटकांची संख्या मोजतो आणि अरेचा साईझ निश्चित करतो.

खालील घोषणा विधानातून आपण काय अपेक्षा करू शकता?

```
int b[10] = {1, 2, 3};
```

इनिशिएलाइझ घटक $b[0]$ to 1, $b[1]$ to 2, $b[2]$ to 3 करते, आणि उर्वरित सर्व घटक 0 इनिशिएलाइझ करते. आम्ही असे म्हणू शकतो की अंशतः आरंभिक अरेमध्ये उर्वरित सर्व घटक 0 ने इनिशिएलाइझ केले आहेत.

खालील घोषणा विधानातून आपण काय अपेक्षा करू शकता?

```
int a[6]={12, 0, 14, -4, 7, 15, -20, 22, 25};
```

इनिशियालाइज सूचीत डिक्लेर साईझ पेक्षा अधिक घटक असल्यामुळे कंपाईलर एक लुटी संदेश ध्वजांकित करेल.

4.2.3 ऍक्सेसइंग घटक Accessing Elements

वन डायमेंशनल अरेमध्ये स्वतंत्र घटकांवर प्रवेश करण्यासाठी सिंगल इंडेक्स वापरले जाते. निर्देशांक एक अविभाज्य (integral) मूल्य किंवा एक्सप्रेसन असणे आवश्यक आहे जे अविभाज्य मूल्याचे मूल्यांकन करते.

उदाहरणार्थ, चित्र 4.2 (a) मधील *marks* अरे दिले तर आपण पहिल्या घटकास ऍक्सेस असे करू

```
marks[0]
```

marks मधील सर्व घटकांवर प्रक्रिया करण्यासाठी, खालील कोड प्रमाणेच एक लूप वापरला जाईल:

```
for (i = 0; i < 10; i++ )  
    process ( marks[i] );
```

4.2.4 इनपुट(Input)

डिक्लेरेशनचा विचार करा

```
int a[50];
```

समजा, वन डायमेन्शनल अरे मधील घटकांची वास्तविक संख्या n ($n \leq 50$) आहे. एन्ट्रिक्शनच्या वेळी युझर n चे मूल्य पुरवेल.

पुढील विभाग वन डायमेन्शनल अरेमध्ये डेटा कसा प्रवेश केला आहे हे दर्शवितो.

```
for ( i = 0; i < n; i++ )
{
    scanf( "%d", &a[i] );
}
```

पुढील गोष्टींचे निरीक्षण करा:

1. आपण i इंडेक्स 0 ने प्रारंभ करतो आणि नंतर तो $(n - 1)$ पर्यंत जातो.
2. जरी आपण अरे घटकांशी व्यवहार करत असलो तरीही *scanf()* फंक्शन कॉलमध्ये **address of** ऑपरेटर (&) आवश्यक आहे

स्पेस / आडवा टॅब किंवा प्रत्येक ओळीत एक घटक विभक्त करून अरेचे घटक समान लाइनवर प्रविष्ट केले जाऊ शकतात.

4.2.5 आउटपुट (Output)

अरेच्या घटकांची व्हॅल्यू आउटपुट करण्यासाठी आपण लूप वापरू.

```
for ( i = 0; i < n; i++ ) {
    printf( "%d ", a[i] );
}
printf( "\n" );
```

सर्व व्हॅल्यूज दोन स्पेस ने विभक्त करून एका ओळीवर प्रदर्शित होतील.

तथापि, जर सर्व मूल्ये एका ओळेत दर्शविली जाऊ शकत नाहीत त्यांच्या मूल्यांच्या विशालतेमुळे किंवा मूल्यांच्या संख्येमुळे, नंतरच्या ओळींमध्ये त्या प्रदर्शित केल्या जातील. नंतर, for लूप पूर्ण झाल्यावर, *printf()* फंक्शन कॉल कर्सरला पुढच्या ओळीवर नेतो

खालील for लूप प्रत्येक ओळ वर एक मूल्य प्रदर्शित करते.

```
for ( i = 0; i < n; i++ ) {
    printf( "\n%d", a[i] );
}
printf( "\n" );
```

1-डी अरेची स्पष्टीकरणात्मक उदाहरणे

वन डायमेन्शनल अरेचे विविध पैलू समजून घेतल्यानंतर, जटिल समस्या हाताळण्यासाठी वन डायमेन्शनल अरेची शक्ती दर्शविण्यासाठी काही प्रोग्राम लिहू या.

Example 4.1: साईझ $n(\leq 50)$ चे, a नामित एक अरे दिले, ज्यांचे घटक `int` प्रकारचे आहेत. एक प्रोग्राम लिहा जो अरेचे त्या घटकांना आउटपुट करेल.

Listing 4.1

```
/*
   Program that outputs those elements of a 1D array that are EVEN
*/
#include<stdio.h>
int main()
{
    int a[50], i, n;
    printf("\nEnter size of array n(<=50) : ");
    scanf("%d", &n);
    printf("\nEnter %d natural numbers as elements of array\n", n);
    for ( i = 0; i < n; i++ )
        scanf("%d", &a[i]);
    printf("\nEVEN elements of array are\n");
    for ( i = 0; i < n; i++ )
    {
        if ( a[i] % 2 == 0 )
            printf("%d  ", a[i]);
    }
    printf("\n");
    return 0;
}
```

Test Run

```
Enter size of array n(<=50) : 10
Enter 10 natural numbers as elements of array
10 7 15 14 24 23 77 35 70 12
EVEN elements of the array are
10 14 24 70 12
```

Example 4.2: साईझ $n(\leq 50)$ चे, a नामित एक अरे दिले, ज्यांचे घटक `int` प्रकारचे आहेत. सर्वात मोठा घटक, सर्वात लहान घटक आणि 1-डी अरेच्या घटकांची सरासरी शोधण्यासाठी प्रोग्राम लिहा.

Listing 4.2

```

/*
    Program to find the largest element, smallest element,
    and the average of elements of 1D array
*/
#include <stdio.h>
int main()
{
    int a[20];
    int i, j, n, largest, smallest, sum;
    float average;
    printf("\nEnter size of array n(<=20) : ");
    scanf("%d", &n);
    printf("\nEnter %d element of 1D array\n", n);
    for ( i = 0; i < n; i++ )
        scanf("%d", &a[i]);
    /* code segment to find largest element */
    largest = a[0];
    for ( i = 1; i < n; i++ ) {
        if ( a[i] > largest )
            largest = a[i];
    }
    /* code segment to find smallest element */
    smallest = a[0];
    for ( i = 1; i < n; i++ ) {
        if ( a[i] < smallest )
            smallest = a[i];
    }
    /* code segment to find average of elements */
    sum = 0;
    for ( i = 0; i < n; i++ )
        sum = sum + a[i];
    average = (float) sum/n;
    /* code segment to output the results */
    printf("\nLargest element    = %d", largest );
    printf("\nSmallest element    = %d", smallest );
    printf("\nAverage of elements = %.2f\n", average );
    return 0;
}

```


Test Run

```

Enter size of array n(<=20) : 10
Enter 10 element of 1D array
10 24 33 15 19 21 35 20 40 16
Largest element      = 40
Smallest element     = 10
Average of elements  = 23.30

```

Example 4.3: डेसिमल नंबर n दिला आहे. डेसिमल नंबर बायनरी नंबर मध्ये रूपांतरित करण्यासाठी प्रोग्राम लिहा.

Listing 4.3

```

/*
   Program to convert a decimal number 'n' into its equivalent
   binary number. The program uses 1D array to store remainders
   during the process of conversion and then prints this array in
   reverse order.
*/
#include <stdio.h>
int main()
{
    int a[20];
    int i, j, t, n;
    printf("\nEnter decimal number n : ");
    scanf("%d", &n);
    t = n;
    i = 0;
    while ( t > 0 )
    {
        a[i] = t % 2;
        i++;
        t = t / 2;
    }
    printf("\nDecimal number %d is equivalent to ", n);
    for ( j = i-1; j >= 0; j-- )
        printf("%d", a[j]);
    printf(" binary.\n");
    return 0;
}

```

Test Run

Enter decimal number n : 10

Decimal number 10 is equivalent to 1010 binary.

Example 4.4: साईझ $n(\leq 50)$ चे, a नामित एक अरे दिले, ज्यांचे घटक int प्रकारचे आहेत. 1D अरेचे समीप घटक स्वॅप करण्यासाठी प्रोग्राम लिहा.

Listing 4.4

```
/*
    Program to swap adjacent elements of a 1D array.
    Size of array is even number.
*/
#include <stdio.h>

int main()
{
    int a[20];
    int i, n, t;
    printf("\nEnter size of array n(<=20) : ");
    scanf("%d", &n);
    printf("\nEnter %d element of 1D array\n", n);
    for ( i = 0; i < n; i++ )
        scanf("%d", &a[i]);
    /* code segment to swap adjacent elements */
    for ( i = 0; i < n-1; i += 2 ) {
        t = a[i];
        a[i] = a[i+1];
        a[i+1] = t;
    }
    printf("\nElements of array after swapping adjacent elements\n");
    for ( i = 0; i < n; i++ )
        printf("%d  ", a[i]);
    printf("\n\n");
    return 0;
}
```

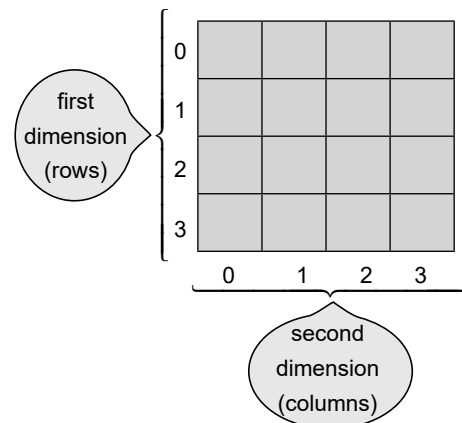
Test Run

```
Enter size of array n(<=20) : 10
Enter 10 element of 1D array
10 24 33 15 19 21 35 20 40 16
Elements of an array after swapping adjacent elements
14 10 15 33 21 19 20 35 16 40
```

4.3 टू डायमेंशनल अरे (TWO-DIMENSIONAL ARRAYS)

आता पर्यंत आपण चर्चा केली त्या अरेला वन डायमेंशनल अरे म्हणून ओळखले जाते कारण डेटा केवळ एका दिशेने (dimension) मध्ये रेषेतारित्या आयोजित केला जातो. बऱ्याच अनुप्रयोगांना डेटा एकापेक्षा जास्त परिमाणात आयोजित करण्याची आवश्यकता असते. एक सामान्य उदाहरण म्हणजे एक मॅट्रिक्स (टेबल), एक अरे आहे ज्यामध्ये पंक्ती आणि स्तंभ आहेत.

चित्र 4.3 मध्ये 4 पंक्ती आणि 4 स्तंभ असलेले एक मॅट्रिक्स (table) दर्शविलेले आहे जे सामान्यतः टू डायमेंशनल अरे म्हणून वापरले जाते.



चित्र 4.3: टू डायमेंशनल अरे

4.3.1 डिक्लेरेशन (Declaration)

टू डायमेंशनल अरे, वन डायमेंशनल अरे प्रमाणे, अरे वापरण्यापूर्वी डिक्लेर करणे आवश्यक आहे. अरे डिक्लेरेशन कंपाईलरला अरेचे नाव, त्यातील घटकांचे प्रकार आणि अरेमधील घटकांची साईझ किंवा नंबर सांगते

टू डायमेंशनल अरे डिक्लेर करण्यासाठी हा सिंटॅक्स आहे.

```
type arrayName[RowSize][ColSize];
```

संमेलनाद्वारे, पहिले परिमाण अरे मधील पंक्तींची संख्या निर्दिष्ट करते तर दुसरा परिमाण प्रत्येक पंक्तीतील स्तंभांची संख्या निर्दिष्ट करतो.

डिक्लेरेशनचा विचार करा.

```
int a[3][3];
```

हे डिक्लेरेशन स्टेटमेंट 18 बाइट, प्रत्येक पंक्तीसाठी 6-बाइटचे contiguous मेमरी ब्लॉकचे वाटप करते. 6-बाइटचा पहिला सेट पहिल्या रांगेतील घटक साठवण्यासाठी वापरला जातो, 6-बाइटचा दुसरा सेट दुसऱ्या रांगेत घटक साठवण्यासाठी वापरला जातो, आणि 6-बाइटचा तिसरा सेट तिसऱ्या घटकांचे संग्रहित करण्यासाठी वापरला जातो (शेवटची) पंक्ती. प्रत्येक पंक्तीमध्ये, प्रथम 2 बाइट्स पहिल्या स्तंभातील घटक संग्रहित करण्यासाठी वापरले जातात, पुढील 2-बाइट्स दुसऱ्या स्तंभातील घटक संग्रहित करण्यासाठी वापरले जातात, आणि अंतिम 2-बाइट्स तृतीय (शेवटचे) घटक संचयित करण्यासाठी वापरले जातात.



Elements of a two-dimensional array are stored row-wise, i.e., in the allocated contiguous block of memory, the first elements of first row are stored, then elements of the second row, and finally elements of the third row, and so on.

4.3.2 इनिशियलायझेशन(Initialization)

टू डायमेंशनल अ‍ॅर्रे इनिशियलायझ करण्याचा एक मार्ग खाली दर्शविल्याप्रमाणे आहे.

```
int a[3][3] = { 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3 };
```

हे पहिल्या पंक्तीचे सर्व घटक मूल्य 0 सह प्रारंभ करेल, मूल्य 1 सह दुसऱ्या पंक्तीचे घटक, 2 सह तृतीय पंक्तीचे घटक आणि 3 मूल्यासह चौथ्या पंक्तीचे घटक.

जरी वरील इनिशियलायझेशन मध्ये कोणतीही अडचण नसली तरी nest braces दोन टू डायमेंशनल अ‍ॅर्रे नेमके स्वरूप दर्शविण्यासाठी वापरण्याची शिफारस केली जाते.

खाली दर्शविल्याप्रमाणे अ‍ॅर्रे अधिक चांगला इनिशियलायझ केला जातो:

```
int a[3][3] = { { 0, 0, 0 },
                { 1, 1, 1 },
                { 2, 2, 2 },
                { 3, 3, 3 },
                };
```

वरील आरंभिक प्रक्रियेमध्ये आपण कंसात बंदिस्त असलेल्या तीन घटकांचा वन डायमेंशनल अ‍ॅर्रे म्हणून प्रत्येक पंक्तीस आरंभ करतो. तीन पंक्तींच्या अ‍ॅर्रेमध्ये त्याचे ब्रेसेसचे सेट देखील आहेत. लक्षात ठेवा आपण स्तंभांमधील घटकांमधील स्वल्पविराम आणि पंक्तींमध्ये स्वल्पविराम वापरू.

4.3.3 ऍक्सेसइंग घटक (Accessing Elements)

i th पंक्तीच्या jth घटक ऍक्सेस करण्यासाठी आपण लिहित आहोत.

```
a[i][j]
```

a च्या पंक्तीनुसार सर्व घटकांवर process करण्यासाठी, खालील कोड प्रमाणेच नेस्टेड लूप वापरले जातील:

```
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
    {
        process ( a[i][j] );
    }
}
```

4.3.4 इनपुट (Input)

खालील विभाग नेस्टेड फॉर लूप्स वापरून घटकांना रो वार्डज वाचण्याचे मार्ग दर्शवितात.

```
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
    {
```

```

scanf( "%d", &a[i][j] );
}
}

```

सर्वसाधारणपणे, जर टू डायमेशनल अरेची साईझ $m \times n$ असेल तर प्रथम लूप 0 ते $(m - 1)$ पर्यंत आणि दुसरे लूप 0 ते $(n - 1)$ पर्यंत बदलते.

4.3.5 आउटपुट (Output)

टू डायमेशनल अरेच्या घटकांच्या मूल्यांचे आउटपुट करण्यासाठी आपण नेस्टेड लूप्स खालीलप्रमाणे वापरू:

```

for ( i = 0; i < 3; i++ )
{
    for ( j = 0; j < 3; j++ )
    {
        printf( "%d ", a[i][j] );
    }
    printf( "\n" );
}

```

वरील विभागाद्वारे टू डायमेशनल अरेचे घटक रो वाईजनुसार आउटपुट करतील. प्रथम लूप पंक्ती नियंत्रित करतो तर दुसरा लूप स्तंभ नियंत्रित करतो.

2-डी अरेची स्पष्टीकरणात्मक उदाहरणे

Example 4.5: ऑर्डर $m \times n$ च्या matrix A च्या ट्रान्सपोजची गणना करण्यासाठी प्रोग्राम.

आपल्याला माहित आहे की जर आपण पंक्ती कॉलमसह बदलली तर आपल्याला दिलेल्या मॅट्रिक्सचे ट्रान्सपोज मिळेल, B ऑर्डर $n \times m$ म्हणा, म्हणजे,

$$b_{ji} = a_{ij}$$

Listing 4.5

```

/*
    Program to compute transpose of matrix A(mxn)
*/
#include<stdio.h>
int main()
{
    int a[10][10], b[10][10];
    int n, m, i, j;
    printf( "Enter the size of matrix A as m,n: " );
    scanf( "%d,%d", &m, &n);
}

```

```

printf("\nEnter %d elements of matrix A row-wise\n",m*n);
for ( i = 0 ; i < m; i++ ) {
    for ( j = 0; j < n; j++ ) {
        scanf( "%d", &a[i][j] );
    }
}
for ( i = 0 ; i < m ; i++ ) {
    for ( j = 0 ; j < n ; j++ ) {
        b[j][i] = a[i][j];
    }
}
printf( "\nTranspose is\n\n" );
for ( i = 0 ; i < n; i++ ) {
    for ( j = 0; j < m; j++ ) {
        printf( "%5d", b[i][j] );
    }
    printf ( "\n" );
}
return 0;
}

```

Test Run

```

Enter the size of matrix A as m,n: 3,3
Enter 9 elements of matrix A row-wise
1 2 3
4 5 6
7 8 9
Transpose is
1   4   7
2   5   8
3   6   9

```

Example 4.6: A आणि B हे दोन मॅट्रिक्सची (एँड) बेरीज करण्याचा प्रोग्राम, प्रत्येक ऑर्डर $m \times n$.

आम्हाला माहित आहे की दोन मॅट्रिक्स बेरीज करण्यासाठी, परिणामी मॅट्रिक्सचा घटक मिळविण्यासाठी आपण संबंधित घटकांची बेरीज करतो, C ऑर्डर $m \times n$ म्हणा,

$$c_{ij} = a_{ij} + b_{ij}$$

Listing 4.6

```

/*
    Program to add matrix A and B, each of order mxn
*/
#include<stdio.h>
int main()
{
    int a[10][10], b[10][10], c[10][10];
    int n, m, i, j;
    printf( "Enter the size of matrices as m,n: " );
    scanf( "%d,%d", &m, &n);
    printf( "Enter %d elements of matrix A row-wise\n", m*n );
    for ( i = 0 ; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
            scanf( "%d", &a[i][j] );
        }
    }
    printf( "Enter %d elements of matrix B row-wise\n", m*n );
    for ( i = 0 ; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
            scanf( "%d", &b[i][j] );
        }
    }
    for ( i = 0 ; i < m ; i++ ) {
        for ( j = 0 ; j < n ; j++ ) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    printf( "\nSum A+B is\n\n" );
    for ( i = 0 ; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
            printf( "%4df", c[i][j] );
        }
        printf ( "\n" );
    }
    return 0;
}

```

Test Run

```

Enter size of matrices as m,n: 3,3
Enter 9 elements of matrix A row-wise
1 1 1
1 1 1
1 1 1
Enter 9 elements of matrix B row-wise
2 2 2
2 2 2
2 2 2
Sum A+B is
3 3 3
3 3 3
3 3 3

```

Example 4.7: A आणि B दोन मॅट्रिक्सचा गुणाकार करण्याचा प्रोग्राम. मॅट्रिक्स A ची ऑर्डर $m \times n$ आहे आणि मॅट्रिक्स B ची ऑर्डर $p \times q$ आहे, जर $n = p$.

आपल्याला A आणि B दोन मॅट्रिक्सचा गुणाकार करायचा आहे. मॅट्रिक्स A ची ऑर्डर $m \times n$ आहे आणि मॅट्रिक्स B ची ऑर्डर $p \times q$ आहे, उत्पादन मॅट्रिक्स, मॅट्रिक्स C म्हणा, मॅट्रिक्स ची C ऑर्डर $m \times p$ आहे, एंट्री c_{ij} सह, जी i th पंक्ती आणि j th स्तंभात दिसते खालीलप्रमाणे दिली आहे:

$$\begin{aligned}
 c_{ij} &= \text{sum of the products of the entries in the } i\text{th row of matrix A with} \\
 &\quad \text{the corresponding entries in the } j\text{th column of matrix B} \\
 &= a_{i0} \times b_{0j} + a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \dots + a_{i(n-1)} \times b_{(n-1)j} \\
 &= \sum_{k=0}^{n-1} a_{ik} b_{kj}
 \end{aligned}$$

Listing 4.7

```

/*
   Program to multiply matrix A of size mxn by matrix B of
   size pxq. The program also checks for the feasibility of product.
*/
#include<stdio.h>
int main()
{
    int a[10][10], b[10][10], c[10][10];
    int n, m, p, q, i, j, k;
    printf( "Enter the size of matrix A as m,n: " );

```



```

scanf( "%d,%d", &m, &n);
printf( "Enter the size of matrix B as p,q: " );
scanf( "%d,%d", &p, &q);
if ( n != p ) {
    printf( "\nMatrix Product AxB is not feasible\n" );
    return 1;
}
printf("\nEnter %d elements of matrix A row-wise\n", m*n);
for ( i = 0 ; i < m; i++ ) {
    for ( j = 0; j < n; j++ ) {
        scanf( "%a", &a[i][j] );
    }
}
printf("\nEnter %d elements of matrix B row-wise\n", p*q);
for ( i = 0 ; i < p; i++ )
{
    for ( j = 0; j < q; j++ )
    {
        scanf( "%d", &b[i][j] );
    }
}
for ( i = 0 ; i < m ; i++ ) {
    for ( j = 0 ; j < q ; j++ ) {
        c[i][j] = 0;
        for ( k = 0; k < n; k++ ) {
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}
printf( "\nProduct AxB is\n\n" );
for ( i = 0 ; i < m; i++ ) {
    for ( j = 0; j < q; j++ ) {
        printf( "%4d", c[i][j] );
    }
    printf ( "\n" );
}
return 0;
}

```

Test Run

```

Enter the size of matrix A as m,n: 3,3
Enter the size of matrix B as p,q: 3,3
Enter 9 elements of matrix A row-wise
1 1 1
1 1 1
1 1 1
Enter 9 elements of matrix B row-wise
2 2 2
2 2 2
2 2 2
Product AxB is
  6   6   6
  6   6   6
  6   6   6

```

4.4 कॅरेक्टर अरे अंड स्ट्रिंग्स (CHARACTER ARRAYS AND STRINGS)

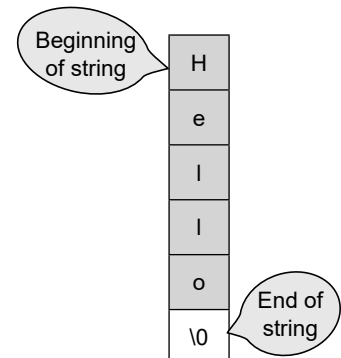
आतापर्यंत सचित्र केलेल्या प्रत्येक अरेमध्ये संख्यात्मक घटक आहेत. तथापि, आपल्याकडे अक्षरे देखील असू शकतात. कॅरेक्टरचा अरे वापरून आपण स्ट्रिंग डेटा साठवू शकतो. परंतु लक्षात ठेवा की स्ट्रिंग्स C भाषेमध्ये थेट समर्थित नाहीत, जरी कधीकधी आपण कॅरेक्टरच्या अरेला स्ट्रिंग म्हणतो.

C भाषेमधील स्ट्रिंग ही कॅरेक्टरचा अरे आहे जो नल कॅरेक्टर ('\0') द्वारे मर्यादित असते. सामान्यतः स्ट्रिंग असलेले कॅरेक्टर फक्त प्रिंट करण्यायोग्य कॅरेक्टरमधून निवडले जातात; तथापि, C भाषा नल कॅरेक्टर वगळता इतर कोणत्याही कॅरेक्टरच्या वापरास परवानगी देत नाही. खरं तर, स्ट्रिंगमध्ये टॅब, फॉर्मेट स्पेसिफायर इत्यादी फॉर्मॅटिंग कॅरेक्टर वापरणे सामान्य आहे.

या विभागात, आपण C भाषेत स्ट्रिंग हाताळण्याशी संबंधित विविध पैलूंचा चर्चा करू या.



The C language uses variable length, delimited strings. The null character '\0' is used as a delimiting character. Though the null character '\0' looks like a sequence of two characters, '\' and '0', but in C language, it is treated as a single character.

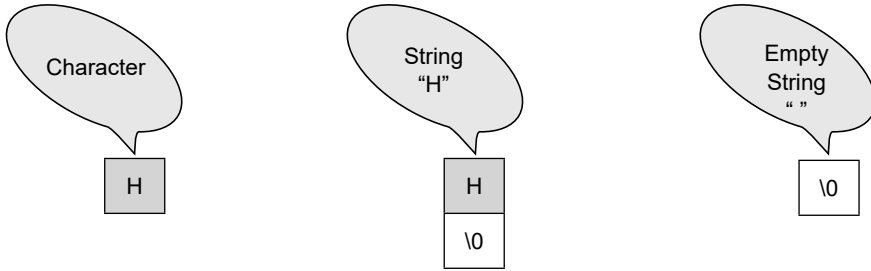
**4.4.1 स्ट्रिंग्स संग्रहित करणे (Storing Strings)**

C भाषेमध्ये, कॅरेक्टरच्या अरेमध्ये स्ट्रिंग संग्रहित केली जाते. हे नल कॅरेक्टर ('\0') नुसार समाप्त केले जाते. चित्र 4.4 मेमरीमध्ये "हॅलो" स्ट्रिंग कशी संग्रहित करते ते दर्शविते. एखादी स्ट्रिंग अरेमध्ये संचित केलेली असल्यामुळे अरेचे नाव आणि म्हणून स्ट्रिंग, हे स्ट्रिंगच्या सुरुवातीस सूचक आहे.

चित्र 4.5 एक कॅरेक्टरचे साठवण आणि एक-कॅरेक्टर स्ट्रिंगमधील फरक दर्शवितो. कॅरेक्टरला 1-बाइटचे सिंगल स्टोरेज आवश्यक आहे तर एक-कॅरेक्टर स्ट्रिंगला प्रत्येकी

चित्र 4.4: मेमरीमध्ये स्ट्रिंग साठवणे

1-बाइटची दोन स्टोरेज ठिकाणे आवश्यक आहेत; एक कॅरेक्टर आणि एक डिलिमीटरसाठी. रिकामी स्ट्रिंग कशी संग्रहित केली जाते हे देखील आकृती दर्शवते. रिक्त स्ट्रिंगमध्ये केवळ नल कॅरेक्टर असतात.

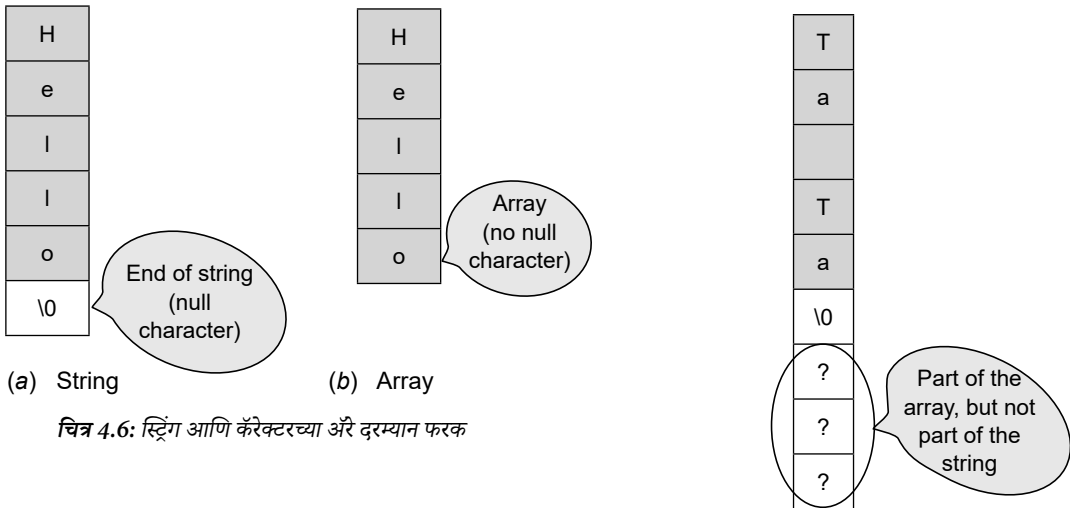


चित्र 4.5: कॅरेक्टर आणि स्ट्रिंगच्या साठवणुकीतील फरक

4.4.2 स्ट्रिंग डिलिमीटरची आवश्यकता (Need of String Delimiter)

प्रश्न - स्ट्रिंग संपुष्टात आणण्यासाठी आपल्याला नल कॅरेक्टर गरज का आहे? - कदाचित आपल्या मनात येत असेल. या प्रश्नाचे उत्तर असे आहे की स्ट्रिंग हा डेटा प्रकार नसतो; ती डेटा स्ट्रक्चर आहे - अरे. म्हणून, स्ट्रिंगची अंमलबजावणी लॉजिकल आहे, फिजिकल नाही. फिजिकल स्ट्रक्चर अरे असते ज्यात स्ट्रिंग स्टोअर असते. स्ट्रिंगची लेन्थ वेगवेगळी असल्याने, फिजिकल संरचनेत डेटाची लॉजिकल समाप्ती ओळखण्याची आवश्यकता असते.

दुसऱ्या दृष्टिकोनातून स्ट्रिंग डिलिमीटरची आवश्यकता पाहू. जर डेटा निश्चित लेन्थचा असेल तर आपल्याला त्यास संचयित करण्यासाठी स्ट्रिंग डेटा स्ट्रक्चरची आवश्यकता नाही उलट ती अरेमध्ये सहजपणे संग्रहित केली जाऊ शकते. जेव्हा डेटा अरेमध्ये संग्रहित केला जातो, तेव्हा डेटाचा शेवट अरेच्या शेवटच्या घटकाद्वारे दर्शविला जातो. परंतु, डेटा निश्चित लेन्थचा नसल्यास (म्हणजे भिन्न लेन्थ) तर डेटाचा शेवट निश्चित करण्यासाठी आपल्याला आणखी काही मार्ग आवश्यक आहेत. स्ट्रिंगचा शेवट चिन्हांकित करण्यासाठी नल कॅरेक्टर वापरला जातो.



चित्र 4.6: स्ट्रिंग आणि कॅरेक्टरच्या अरे दरम्यान फरक

चित्र 4.7: अरेच्या काही भागात स्ट्रिंग साठवणे

कारण स्ट्रिंग्स व्हेरिबल- लेन्थच्या स्ट्रक्चर्स असतात, आपल्याकडे नल कॅरेक्टर (डिलिमीटर) साठी प्लस वन ठेवण्यासाठी जास्तीत जास्त लेन्थच्या स्ट्रिंगसाठी पुरेशी जागा राखीव असते

बऱ्याच परिस्थितींमध्ये हे शक्य आहे की संपूर्ण अरे भरला नसेल आणि अरेच्या मध्यभागी नल कॅरेक्टर असू शकेल. त्या प्रकरणात, सुरुवातीपासून नल कॅरेक्टर पर्यंत अरेला स्ट्रिंग मानले जाते आणि अरेच्या उर्वरित घटकांकडे दुर्लक्ष केले जाते. याचा अर्थ असा की कॅरेक्टरच्या अरेचा भाग स्ट्रिंग म्हणून मानला जाऊ शकतो जर चित्र 4.11 मध्ये दर्शविल्याप्रमाणे एखाद्या नल कॅरेक्टर सह समाप्त होते.

4.4.3 स्ट्रिंग्स लिटरल्स (String Literals)

एक स्ट्रिंग लिटरल्स, ज्याला स्ट्रिंग कॉन्स्टेंट देखील म्हटले जाते, हा दुहेरी अवतरण मध्ये बंद केलेल्या कॅरेक्टरचा अनुक्रम आहे. खाली काही स्ट्रिंग लिटरल्सची उदाहरणे दिली आहेत.

```
"Hello! you are wel come"
"Hari\'s Book"
""
"A"
```

लक्षात घ्या की एम्बेड केलेली स्पेस महत्त्वपूर्ण आहेत. आणि कोट्स, सिंगल (') किंवा डबल ("), स्ट्रिंगचा भाग असल्यास, ते एस्केप सिकव्हेन्ससह वापरले जाणे आवश्यक आहे.

4.4.4 स्ट्रिंग व्हेरिएबल्स (String Variables)

स्ट्रिंग व्हेरिएबल हे कॅरेक्टरचे अरे असते.

4.4.4.1 डिक्लेअरइंग स्ट्रिंग व्हेरिएबल (Declaring String Variable)

स्ट्रिंग व्हेरिएबल खालील स्टेटमेंटमध्ये दाखवल्याप्रमाणे डिक्लेअर केले जाते.

```
char str[20];
```

हे विधान स्ट्रिंग व्हेरिएबल स्ट्रिंग लेन्थ 20 डिक्लेअर करते. खरं तर, शेवटचे अक्षर नेहमीच नल कॅरेक्टर असल्यामुळे संग्रहित केले जाणारे जास्तीत जास्त वर्ण 19 आहेत. खरं तर वरील डिक्लेरेशन मध्ये अरेच्या पहिल्या घटकाचा पत्ता दर्शविणारा अरे नेम स्ट्रिंगसह 20 बाइटचा मेमरी ब्लॉक राखीव आहे.

4.4.4.2 इनिशियलिझिंग स्ट्रिंग व्हेरिएबल (Initializing String Variables)

डिक्लेरेशन दरम्यान स्ट्रिंग व्हेरिएबल देखील इनिशियलाइज केले जाऊ शकते. पुढील डिक्लेरेशन अशाच एका दृष्टिकोनाचे वर्णन करते.

```
char mess[] = {'Y', 'o', 'u', ' ', 'a', 'r', 'e', ' ',
               'w', 'e', 'l', ' ', 'c', 'o', 'm', 'e', '\0'};
```

येथे स्ट्रिंग व्हेरिएबल mess ची सुरुवात कॅरेक्टर कॉन्स्टंट च्या सीक्वेन्स म्हणून होते. या शैलीमध्ये नल कॅरेक्टर अंतिम वर्ण म्हणून निर्दिष्ट करणे प्रोग्रामरचे कार्य आहे. या व्यतिरिक्त, आपण पाहू शकता की अरेचा साईझ निर्दिष्ट केलेला नाही. mess ची साईझ काय असेल याचा अंदाज लावू शकता? खरं तर ती नल कॅरेक्टर सह निर्दिष्ट केलेल्या कॅरेक्टर ची संख्या आहे.

स्ट्रिंग बहुतेक वेळा वापरल्या जात असल्याने, C भाषा स्ट्रिंग्स इनिशियलाइज करण्यासाठी एक लहान आणि कार्यक्षम दृष्टिकोन प्रदान करते.

उदाहरणार्थ, वरील विधानांचे कार्य असे देखील पूर्ण केले जाऊ शकते.

```
char mess[] = "You are wel come";
```

लक्षात घ्या की या दृष्टिकोनात कंपाईलर आपोआप नल कॅरेक्टर जोडेल.

4.4.5 स्ट्रिंगचे इनपुट / आउटपुट (Input/Output of Strings)

स्ट्रिंग वाचली आणि लिहिली जाऊ शकते. C भाषा लायब्ररी फंक्शन *gets ()* हे इनपुटसाठी आणि स्ट्रिंग डेटाच्या आउटपुटसाठी *puts ()*.

Listing 4.8

```
/*
    Program to perform I/O of string data with string I/O functions
*/
#include<stdio.h>
int main()
{
    char str[30];
    printf( "\nEnter string of length <= 29" );
    printf( "\nand terminate with ENTER key\n\n" );
    gets( str );
    printf( "\nYou have entered\n\n" );
    puts( str );
    return 0;
}
```

Test Run

```
Enter string of length <= 29
and terminate with ENTER key
There is no substitute of hard work..
You have entered
There is no substitute of hard work.
```

4.4.6 स्ट्रिंग मॅनिपुलेशन फंक्शन्स (String Manipulation Functions)

जवळजवळ प्रत्येक C कंपाईलर स्ट्रिंग्स हाताळण्यासाठी मोठ्या प्रमाणात लायब्ररी फंक्शन्स प्रदान करते. Table 4.1 मध्ये वारंवार वापरल्या जाणाऱ्या स्ट्रिंग फंक्शन्सची यादी दिली आहे.

Table 4.1: वारंवार वापरल्या जाणाऱ्या स्ट्रिंग फंक्शन्स

String Function	Operation Performed
<code>strlen(str)</code>	Return length of string <i>str</i> .
<code>strcat(str1, str2)</code>	Concat or joins string <i>str2</i> with string <i>str1</i> . The result is stored in string <i>str1</i> .
<code>strcpy(str1, str2)</code>	Copies the contents of string <i>str2</i> to string <i>str1</i> .
<code>strcmp(str1, str2)</code>	Compares the first string <i>str1</i> with second string <i>str2</i> , and returns the value <ul style="list-style-type: none"> • < 0, if string <i>str1</i> comes earlier in dictionary order than <i>str2</i> • = 0, if both strings <i>str1</i> and <i>str2</i> are same • > 0, if string <i>str1</i> comes later in dictionary order than <i>str2</i>
<code>strrev(str)</code>	Reverses the string <i>str</i> .
<code>strlwr(str)</code>	Converts alphabets in string <i>str</i> to lowercase.
<code>strupr(str)</code>	Converts alphabets in string <i>str</i> to uppercase.

यापैकी काही स्ट्रिंग फंक्शन्सची आपण एक-एक चर्चा करू या.

4.4.6.1 The `strlen()` Function

हे फंक्शन एक अर्ग्युमेण्ट घेते जो स्ट्रिंग कॉन्स्टन्ट किंवा व्हेरिएबल असू शकतो. हे स्ट्रिंगमध्ये असलेल्या कॅरेक्टरची संख्या मोजते. लक्षात ठेवा की नल कॅरेक्टर '\0' हे त्या कॅरेक्टरचा भाग नाही; हे फक्त स्ट्रिंगच्या शेवटी चिन्हांकित करण्यासाठी वापरले जाते, म्हणून ते मोजले जात नाही.

Listing 4.9

```
/*
    Program to illustrate the use of strlen() function
*/
#include <stdio.h>
#include <string.h>
int main()
{
    char str[30];
    printf( "\nEnter string : " );
    gets (str);
    printf( "Length of string = %d\n", strlen(str) );
    return 0;
}
```

Test Run

```
Enter string : Programming
Length of string = 11
```

4.4.6.2 The *strcpy* () Function

हे फंक्शन दोन वितर्क(अर्ग्युमेण्ट) घेते - एक स्ट्रिंग व्हेरिएबल आणि दुसरे स्ट्रिंग कॉन्स्टन्ट किंवा व्हेरिएबल असू शकते. हे पहिल्या वितर्कात दुसऱ्या वितर्काची कॅरेक्टर कॉपी करते.

Listing 4.10

```
/*
    Program to illustrate the use of strcpy() function
*/
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[30], str2[30];
    printf( "\nEnter string str1 : ");
    gets(str1);
    strcpy( str2, str1 );
    printf( "String str2 = %s\n", str2 );
    return 0;
}
```

Test Run

```
Enter string str1 : Hey, how are you?
String str2 = Hey, how are you?
```

4.4.6.3 The *strcat* () Function

हे फंक्शन दोन वितर्क (अर्ग्युमेण्ट) घेते - एक स्ट्रिंग व्हेरिएबल आणि दुसरे स्ट्रिंग कॉन्स्टन्ट किंवा व्हेरिएबल असू शकते. हे पहिल्या अर्ग्युमेण्ट च्या शेवटी दुसऱ्या वितर्काचे कॅरेक्टर जोडते.

Listing 4.11

```
/*
    Program to illustrate the use of strcat() function
*/
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[30], str2[30];
    printf( "\nEnter string str1 : " );
```

```

gets(str1);
printf( "\nEnter string str2 : " );
gets(str2);
strcat( str1, str2 );
printf("String str1 after concatenation with str2 = %s\n", str1);
return 0;
}

```

Test Run

```

Enter string str1 : naughty
Enter string str2 : boy
String str1 after concatenation with str2 = naughtyboy

```

4.4.6.4 The *strcmp* () Function

हे फंक्शन दोन अर्ग्युमेण्ट घेते - दोन्ही स्ट्रिंग व्हेरिएबल्स किंवा कॉन्स्टन्ट असू शकतात. हे प्रत्येक कॅरेक्टरची तुलना करते पण एका वेळी एका आणि तुलनाचा परिणाम दर्शविणारे मूल्य मिळवते. उदाहरणार्थ,

```
strcmp ( str1, str2 );
```

Table 4.2. मध्ये वर्णन केल्यानुसार परत केलेल्या मूल्याचे अर्थ असतील.

Table 4.2: Strcmp () फंक्शनद्वारे परत केलेल्या मूल्याचे स्पष्टीकरण

Value Returned	Meaning
< 0	String <i>str1</i> comes earlier in dictionary order than <i>str2</i> .
= 0	Strings <i>str1</i> and <i>str2</i> are same (identical).
> 0	String <i>str1</i> comes later in dictionary order than <i>str2</i> .

Listing 4.12

```

/*
   Program to illustrate the use of strcmp() function
*/
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[30], str2[30];
    int value;
    printf( "\nEnter string str1 : " );
    gets(str1);
    printf( "Enter string str2 : " );

```



```

gets(str2);
value = strcmp( str1, str2 );
if ( value > 0 ) {
    printf("\n%s comes after %s", str1, str2);
    printf(" in dictionary order\n" );
} else if ( value < 0 ) {
    printf("\n%s comes before %s", str1, str2);
    printf(" in dictionary order\n" );
} else
    printf("\nBoth strings are same\n" );
return 0;
}

```

Test Run

```

Enter string str1 : software
Enter string str2 : hardware
hardware comes before software in dictionary order

```

4.4.6.5 The *strrev* () Function

हे फंक्शन स्ट्रिंग व्हेरिएबलला अर्ग्युमेण्ट म्हणून घेते. हे स्ट्रिंगमधील कॅरेक्टरचा क्रम बदलवते.

Listing 4.13

```

/*
    Program to illustrate the use of strrev() function
*/
#include <stdio.h>
#include <string.h>
int main()
{
    char str[50];
    printf( "\nEnter string : " );
    gets(str);
    strrev( str );
    printf( "\nReverse string = %s\n", str );
    return 0;
}

```

Test Run

```
Enter string : programming
Reverse string = gnimmarginorp
```

4.4.6.6 The *strupr* () Function

हे फंक्शन स्ट्रिंग व्हेरिएबलला अर्ग्युमेण्ट म्हणून घेते. हे स्ट्रिंग मधील अक्षरांना अप्पर केस रूपांतरित मध्ये करते.

Listing 4.14

```
/* Program to illustrate the use of strupr)function */
#include <stdio.h>
#include <string.h>
int main()
{
    char str[50];
    printf( "\nEnter string in lowercase : " );
    gets(str);
    strupr( str );
    printf( "\nGiven string in UPPERCASE = %s\n", str );
    return 0;
}
```

Test Run

```
Enter string in lowercase : programming
Given string after conversion to UPPERCASE = PROGRAMMING
```

4.4.6.7 The *strlwr* () Function

हे फंक्शन स्ट्रिंग व्हेरिएबलला अर्ग्युमेण्ट म्हणून घेते. हे स्ट्रिंग मधील अक्षरांना लोअरकेसमध्ये रूपांतरीत करते.

Listing 4.15

```
/*
    Program to illustrate the use of strupr)function
*/
#include <stdio.h>
#include <string.h>
int main()
{
    char str[50];
```

```

printf( "\nEnter string in UPPERCASE : " );
gets(str);
strlwr( str );
printf( "\nGiven string in Lowercase = %s\n", str );
return 0;
}

```

Test Run

```

Enter string in UPPERCASE : PROGRAMMING
Given string in Lowercase = programming

```

4.4.7 स्ट्रिंग्स अरे (Array of Strings)

शेवटच्या विभागात आपण स्ट्रिंग व्हेरिएबल्सची चर्चा केली जे एकावेळी एक स्ट्रिंग व्हॅल्यू साठवू शकतात. समजा आपल्याला अशा स्ट्रिंगची यादी (अरे) हँडल करायची असेल तर उपाय काय आहे? टू डायमेंशनल अरेचा वापर करून या प्रश्नाचे उत्तर सापडेल. टू डायमेंशनल अरेमध्ये, प्रथम पंक्ती प्रथम स्ट्रिंग संग्रहित करते; दुसरी पंक्ती दुसरी स्ट्रिंग संचयित करते, वगैरे वगैरे.

समजा, आम्हाला 4 लोकांच्या नावांची यादी हँडल करायची आहे, जिथे प्रत्येक व्यक्तीचे नाव जास्तीत जास्त 30 कॅरेक्टरचे असू शकते, खालील आवश्यक डिक्लरेशन असेल:

```
char names[4][31];
```

खालील विधान स्ट्रिंगची यादी इनिशियालाइज करण्याचे मार्ग दर्शविते.

```

char names[4][31] = { "Ram Parkash",
                      "Inder Mohan Singh Sidhu",
                      "Amanpreet",
                      "Rajan"
                    };

```

Listing 4.16

```

/*
   Program to illustrates input/output of array of strings
   represented as two-dimensional array of characters
*/
#include <stdio.h>
int main()
{
    char names[50][31];
    int i, n;

```

```

printf( "Enter number of strings : " );
scanf( "%d", &n );
fflush(stdin); /* clear keyboard buffer */
for ( i = 0; i < n; ++i )
{
    printf( "Enter string %d: ", i+1 );
    gets( names[i] );
}
printf( "\nList of given strings\n\n" );
for ( i = 0; i < n; ++i )
    puts ( names[i] );
return 0;
}

```

Test Run

```

Enter number of strings : 5
Enter string 1 : Hari
Enter string 2 : Santosh
Enter string 3 : Balwinder
Enter string 4 : Ram
Enter string 5 : Sham
List of given strings
Hari
Santosh
Balwinder
Ram
Sham

```

स्पष्टीकरणात्मक उदाहरणे

Example 4.8: स्ट्रिंगमध्ये दिलेल्या अक्षराची वारंवारता शोधण्यासाठी प्रोग्राम लिहा.

Listing 4.17

```

/*
    Program to find the frequency of a given character in a string
*/
#include <stdio.h>
int main()

```

```

{
    char str[81], ch;
    int i, count = 0;
    printf("\nEnter a string : ");
    gets(str);
    printf("\nEnter a character to find its frequency : ");
    ch = getchar();
    for (i = 0; str[i] != '\0'; i++)
    {
        if (ch == str[i])
            count++;
    }
    printf("\nFrequency of %c = %d", ch, count);
    return 0;
}

```

Test Run

```

Enter a string : I love programming in C language.
Enter a character to find its frequency : a
Frequency of a = 3

```

Example 4.9: स्ट्रिंगमध्ये vowels, consonants, digits, आणि white-spaces यांची संख्या मोजण्यासाठी प्रोग्राम लिहा.

Listing 4.18

```

/*
    Program to count the number of vowels, consonants, digits,
    and white-spaces in a string
*/
#include <stdio.h>
int main()
{
    char str[80];
    int vowels = 0, consonants = 0, digits = 0, spaces = 0;
    int i;
    printf("\nEnter a line of text : ");
    gets(str);
    for (i = 0; str[i] != '\0'; i++)

```

```

{
    if ( str[i] == 'a' || str[i] == 'A' || str[i] == 'e' ||
        str[i] == 'E' || str[i] == 'i' || str[i] == 'I' ||
        str[i] == 'o' || str[i] == 'O' || str[i] == 'O' ||
        str[i] == 'U' ) {
        vowels++;
    } else if ( (str[i] >='a' && str[i]<='z')
        || (str[i]>= 'A' && str[i] <= 'Z') ) {
        consonants++;
    } else if ( str[i] >= '0' && str[i] <= '9' ) {
        digits++;
    } else if ( str[i] == ' ' || str[i] == '\t' ) {
        spaces++;
    }
}

printf("\nVowels      = %d", vowels);
printf("\nConsonants   = %d", consonants);
printf("\nDigits       = %d", digits);
printf("\nWhite spaces = %d", spaces);
return 0;
}

```

Test Run

```

Enter a line of text : I love programming in C language.
Vowels      = 10
Consonants   = 17
Digits       = 0
White spaces = 5

```

Example 4.10: प्रोग्राम लिहा जो दिलेल्या अक्षरापासून सुरू असलेली स्ट्रिंग दर्शवितो.

Listing 4.19

```

/*
    Program to display strings which start with a given character
*/
#include <stdio.h>
int main()

```

```
{
    char str[20][31], ch;
    int i, n;
    printf( "\nEnter number of strings : " );
    scanf( "%d", &n );
    fflush(stdin);
    for ( i = 0; i < n; ++i )
        gets( str[i] );
    printf("\nEnter a character : ");
    ch = getchar();
    printf( "\nStrings that start with character : %c\n", ch );
    for ( i = 0; i < n; ++i ) {
        if ( ch == str[i][0] )
            puts(str[i]);
    }
    return 0;
}
```

Test Run

```
Enter number of strings : 10
Delhi
Dehradun
Hyderabad
Bangaluru
Dalhousie
Nagpur
Darjiling
Jaipur
Agra
Pune
Enter a character : D
Strings that start with character : D
Delhi
Dehradun
Dalhousie
Darjiling
```

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- अरे ही समान मूल्यांचे संग्रह दिलेल्या नावाखाली संग्रहित करण्यासाठी एक डेटा स्ट्रक्चर आहे
- contiguous मेमरी ठिकाणी संचयित केलेल्या अरेचे घटक.
- अरेचे घटक अरेच्या नावाने कंसात इंडेक्स नुसार ऍक्सेस केला जातो. प्रत्येक डायमेशनसाठी एक कंसा चा जोड वापरला जातो.
- अरे वापरण्यापूर्वी डिक्लेर करणे आवश्यक आहे. डिक्लेरेशन कंपायलरला अरेचे नाव, प्रत्येक घटकाचे प्रकार आणि प्रत्येक डायमेशनचे साईझ याबद्दल सांगते.
- डिक्लेरेशनच्या वेळी अरे इनिशियालाइज केला जाऊ शकतात.
- जेव्हा अरे अर्धवट इनिशियालाइज केला जातो, तेव्हा उर्वरित घटक शून्यावर सेट केले जातात.
- टू डायमेशनल अरे म्हणजे पंक्ती आणि स्तंभ असलेल्या सारणीचे (मॅट्रिक्स) प्रतिनिधित्व
- मल्टि डायमेशनल अरे म्हणजे टू डायमेशनल अरेचा विस्तार तीन, चार किंवा अधिक डायमेशनपर्यंत करणे.
- जसे की, C भाषा स्ट्रिंग डेटा प्रकारास समर्थन देत नाही परंतु स्ट्रिंगची कॅरेक्टरचा अरे म्हणून हाताळली जातात
- C भाषा नल-टर्मिनेटेड व्हेरिएबल लेन्थच्या स्ट्रिंगचा वापर करते.
- C भाषेमधील स्ट्रिंग कॉन्स्टन्ट म्हणजे दुहेरी अवतरणांमध्ये बंद केलेल्या कॅरेक्टरचा अनुक्रम
- एखादा स्ट्रिंग संचयित करण्यासाठी, आपल्याकडे स्ट्रिंग लेन्थच्या पेक्षा एक जास्त साईझ चा अरे आवश्यक आहे
- टू डायमेशनल कॅरेक्टरचा अरे स्ट्रिंगची यादी वापरून हाताळली जाते.
- स्ट्रिंग्स हाताळण्यासाठी फंक्शन्सची समृद्ध लायब्ररी आहे.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. अरे म्हणजे काय?
2. C मध्ये एकाच वेळी अरे डिक्लेर करणे आणि इनिशियालाइज करणे शक्य आहे काय? जर होय, कसे
3. अरेला नाव देण्याचे काय नियम आहेत?
4. अरेच्या पहिल्या घटकाची सबस्क्रिप्ट काय आहे?
5. 100 घटकांसह रिअल अरे नावाचा डेटा कसा घोषित केला जाईल? शेवटच्या घटकाचे सबस्क्रिप्ट काय असेल?
6. एकाच डिक्लेर च्या विधानात एकापेक्षा जास्त अरे डिक्लेरेशन करणे शक्य आहे काय?
7. खालील अरे डिक्लेरेशन बरोबर आहे? जर नसेल तर का?

```
int a(50);
```

8. जर डिक्लेरेशन विभाग हे असल्यास

```
#define ROWS 100
float height[ROWS];
```

9. अरेंमध्ये डेटा प्रविष्ट करण्यासाठी खालील विभाग योग्य आहे?

```
for ( i = 0; i <= ROWS; i++ )
    scanf ( "%f", height[i] );
```


10. जेव्हा फंक्शनच्या अर्ग्युमेण्ट म्हणून अरे पास केला जातो तेव्हा प्रत्यक्षात काय पास केले जाते?
11. जेव्हा संपूर्ण अरे एखाद्या कार्यासाठी अर्ग्युमेण्ट म्हणून पास केली जाते, तेव्हा फंक्शन अरेच्या घटकांच्या मूल्यांमध्ये बदल करू शकते. आपण फंक्शनला हे करण्यापासून रोखू इच्छिते, हे कार्य कसे पूर्ण केले जाऊ शकते.
12. खालील डिक्लेरेशन मध्ये काही चूक आहे काय? स्पष्ट करणे.

```
int a[3][] = { { 1, 2, 3 }, { 3, 2, 1 }, { 4, 5, 2 } };
```
13. C भाषेत स्ट्रिंग कशी संग्रहित केली जाते?
14. स्ट्रिंग नामक अरेसाठी कोणती डिक्लेरेशन आहे ज्यामध्ये “ C is just like sea “ हे संग्रहित करायचा आहे.
15. "A" आणि 'A' मध्ये काय फरक आहे?
16. पुढील डिक्लेरेशनचा विचार करा

```
char name[20];
```

 name योग्यरित्या संग्रहित केल्या जाणाऱ्या स्ट्रिंगची कमाल लेन्थ किती आहे?
17. स्ट्रिंगची जास्तीत जास्त लांबी किती आहे जी नावाने योग्यरित्या साठवली जाऊ शकते
18. पुढील डिक्लेरेशनचा विचार करा

```
char name[20];
```

 Is following the correct way of assigning the string "Rajan Mehta" to name?

एकाधिक निवड प्रश्न

1. टू डायमेंशनल अरेचे घटक त्यात साठवले जातात
 (a) column major order (b) row major order
 (c) random order (d) None
2. अरे डिक्लेरेशन मध्ये [] मधील मूल्य निर्दिष्ट करते
 (a) अरेचा आकार (b) सर्वात मोठा परवानगी असलेला सबस्क्रिप्ट मूल्य
 (c) दोन्ही (a) आणि (b) (d) वरीलपैकी काहीही नाही
3. “int a [10]” डिक्लेरेशन दिली खालीलपैकी कोणते चुकीचे आहे ते ओळखा?
 (a) a[-1] (b) a[0] (c) a[10] (d) ++a
4. जेव्हा आपण अरे वापरू?
 (a) जेव्हा आपल्याला कॉन्स्टन्ट ठेवण्याची आवश्यकता असते.
 (b) जेव्हा आम्हाला समान प्रकारच्या डेटा ठेवण्याची आवश्यकता असते.
 (c) जेव्हा आपल्याला भिन्न प्रकारच्या डेटा ठेवण्याची आवश्यकता असते.
 (d) जेव्हा आम्हाला स्वयंचलित मेमरी क्लीनअप कार्यक्षमता प्राप्त करण्याची आवश्यकता असते.
5. अरेचा वापर एक प्रोग्राम बनवितो_____.
 (a) सामान्य आणि समस्यांचा वर्ग हाताळण्यास सक्षम. (b) समजणे कठीण.
 (c) कॉम्पॅक्ट आणि कार्यक्षम (d) दोन्ही a आणि c
6. खालील एक्सप्रेशन मध्ये 5 च्या दरम्यान काय फरक आहे?

```
int num[5];
```

```
num[5]=10;
```

- (a) प्रथम अँरेचा आकार निर्दिष्ट करते, तर दुसरा एक स्वतंत्र घटक निर्दिष्ट करतो
- (b) प्रथम स्वतंत्र घटक निर्दिष्ट करते, दुसरे आकार निर्दिष्ट करते
- (c) दोन्ही अँरे आकार निर्दिष्ट करतात
- (d) वरीलपैकी कोणतेही नाही

7. अँरे ऍड्रेस 1200 वर सुरू झाल्यास प्रोग्रामचे आउटपुट काय असेल

```
void main()
{
    int a[5]={ 2,4,6,8,10 };
    printf("%u, %d ", a, sizeof(a) );
}
```

- (a) 1202, 10
- (b) 10, 1202
- (c) 10, 1200
- (d) 1200, 10

8. टू डायमेंशनल अँरे म्हणून म्हटले जाऊ शकते_____.

- (a) मॅट्रिक्स
- (b) वेक्टर
- (c) स्टॅक
- (d) queue

9. पुढील प्रोग्रामचे आउटपुट काय असेल?

```
void main()
{
    void fun(int x[]);
    int a[] = {1, 2, 3, 4, 5 };
    fun(a);
    printf("\n%d,%d", a[0], a[1]);
    printf("%d,%d,%d\n", a[2], a[3], a[4]);
}

void fun(int x[])
{
    x[2] = x[2] + 2;
    x[4] = x[4] + 4;
}
```

- (a) 5,6,7,8,9
- (b) 2,4,6,8,10
- (c) 3,4,5,6,7
- (d) 1, 2, 5, 4, 9

10. खालील प्रोग्रामचे आउटपुट कोणते असेल?

```
void main()
{
    int a[5] = { 1, 2 };
    printf("\n%d,%d,%d\n", a[2], a[3], a[4]);
}
```

- (a) 0, 0, 0
- (b) 2, 2, 2
- (c) 1, 1, 1
- (d) Garbage

11. जर आपण सी प्रोग्राममधील अरे एलिमेंटला व्हॅल्यू दिली तर त्याचे काय होईल ज्याचे सबस्क्रिप्ट अरेच्या साइजपेक्षा जास्त आहे?
 - (a) काहीही होणार नाही आणि त्या एलिमेंटला दिलेले मूल्य मिळेल.
 - (b) कंपाईलर एक वाक्यरचना त्रुटी ध्वजांकित करेल.
 - (c) अरेच्या शेवटच्या मेमरी स्थानानंतर लगेच मेमरी स्थान अधिलिखित केले जाईल आणि यामुळे सिस्टम क्रॅश होऊ शकते.
 - (d) नवीन मूल्य समाविष्ट करण्यासाठी अरेचा साइज आपोआप वाढविला जाईल.
12. डिक्लेरेशन `"char sample[80];"`, `sample` द्वारे योग्यरित्या प्रतिनिधित्व केले जाऊ शकते अशा स्ट्रिंगची लेन्थ किती आहे?
 - (a) 80
 - (b) 79
 - (c) 81
 - (d) None
13. C मधील स्ट्रिंग्स बदल खालीलपैकी कोणते विधान खरे आहे?
 - (a) प्रत्येक स्ट्रिंग नल कॅरेक्टर नुसार समाप्त केले पाहिजे ('must 0').
 - (b) स्ट्रिंग हा सी मधील एक प्राथमिक डेटा प्रकार आहे.
 - (c) असाइनमेंट ऑपरेटर '=' वापरून स्ट्रिंगला दुसऱ्या स्ट्रिंगला असाइन केले जाऊ शकते.
 - (d) वरील सर्व
14. पुढील प्रोग्रामचे आउटपुट काय असेल?


```
void main()
{
    printf(5+"Fascimile");
}
```

 - (a) Error
 - (b) Fascimile
 - (c) mile
 - (d) Fasci
15. खालीलपैकी कोणते स्ट्रिंग व्हॅल्यू "aeiou" सह `vowels` नावाचे `char` अरे तयार करण्यासाठी आणि इनिशियलाइज करण्यासाठी योग्य नाही?
 - (a) `char vowels[] = { 'a', 'e', 'i', 'o', 'u', '\0' };`
 - (b) `char vowels[] = { 'a', 'e', 'i', 'o', 'u', '\n' };`
 - (c) `char vowels[] = "aeiou";`
 - (d) None of the above
16. जर स्ट्रिंग `str1` आणि `str2` दोन समान असतील तर खालील कोड विभागातील मूल्य `y` असेल _____.


```
int y;
y = strcmp(str1, str2);
```

 - (a) 1
 - (b) -1
 - (c) 0
 - (d) None of the above
17. पुढील प्रोग्रामचे योग्य आउटपुट कोणते आहे?


```
#include <string.h>
void main() {
    char str[] = "Spider\0man\0";
```

```
printf("%d", strlen(str));
}
```

- (a) 6 (b) 10 (c) 11 (d) 13

18. पुढील प्रोग्रामचे योग्य आउटपुट कोणते आहे?

```
void main()
{
    char str[] = "Sales\0man";
    puts(str);
}
```

- (a) Sales (b) man (c) Salesman (d) Error

19. पुढील प्रोग्रामचे योग्य आउटपुट कोणते आहे?

```
int main()
{
    printf("%d", strcmp("ABC", "abc"));
    return 0;
}
```

- (a) 0 (b) 1 (c) -1 (d) Error

20. मल्टीवर्ड शब्द वाचण्यासाठी खालीलपैकी कोणते फंक्शन अधिक योग्य आहे

- (a) scanf() (b) gets()
(c) getchar() (d) getstring()

ANSWERS															
1.	(b)	2.	(a)	3.	(d)	4.	(b)	5.	(d)	6.	(a)	7.	(d)	8.	(a)
9.	(d)	10.	(a)	11.	(c)	12.	(b)	13.	(a)	14.	(c)	15.	(b)	16.	(c)
17.	(a)	18.	(a)	19.	(c)	20.	(b)								

प्रोग्रामिंग समस्या

- प्रोग्राम लिहा जो अरे स्वीकारतो, शेवटचा घटकासह पहिला घटक बदलतो, शेवटून दुसऱ्या घटकासह बदलतो, अशा प्रमाणे बाकीचे आणि शेवटी नवीन अरे प्रिंट करतो
- 2 आणि 10 च्या दरम्यान एन इन्टिजर चा संच वाचण्यासाठी प्रोग्राम लिहा आणि सेटमध्ये प्रत्येक इन्टिजर किती वेळा दिसेल याची गणना करा. वारंवारतेच्या क्रमाने इन्टिजर प्रिंट करा, ही संख्या प्रथम क्रमांकाची सर्वाधिक संख्या आहे.
- डिपार्टमेंट स्टोअर चेनमध्ये m (≤ 10) स्टोअर्स असतात आणि प्रत्येक स्टोअरमध्ये समान n (≤ 15) विभाग असतात. चेनची साप्ताहिक विक्री $m \times n$ अरे SALES (say) साठविली जाते. प्रोग्राम लिहा जे .
 - प्रत्येक स्टोअरच्या एकूण साप्ताहिक विक्रीचे प्रिंट. (Prints the total weekly sales of each store.)
 - प्रत्येक विभागाच्या एकूण साप्ताहिक विक्रीचे प्रिंट. (Prints the total weekly sales of each department.)
 - साखळीची एकूण साप्ताहिक विक्री छापते. (Prints the total weekly sales of the chain)

4. एक प्रोग्राम लिहा जो a अरे n च्या आकाराच्या kth च्या स्थानावर d एलिमेंट घालेल.
5. एक प्रोग्राम लिहा जो a अरे n च्या आकाराच्या kth च्या स्थानावर d च्या स्थानावरून एक घटक काढून टाकेल.
6. अरेमधून डुप्लिकेट घटक काढण्यासाठी प्रोग्राम लिहा.
7. ऑर्डर $m \times n$ एक मॅट्रिक्स A दिल्यास, जास्तीत जास्त घटकांची संख्या असलेल्या पंक्ती शोधण्यासाठी प्रोग्राम लिहा.
8. एक प्रोग्राम लिहा जो वापरकर्त्याला वाक्य प्रविष्ट करण्यास सांगेल आणि नंतर वाक्यातून शब्द विभक्त करतो आणि त्यास एका टेबलमध्ये ठेवतो.
9. एखादा प्रोग्राम लिहा जो वापरकर्त्याला शब्दांची यादी विचारेल आणि ती यादी उलट क्रमाने प्रिंट करते.
10. समजा तुम्ही स्पेलिंग केलेल्या अंकांच्या स्ट्रिंगमध्ये संख्या अनुवादित करता, प्रत्येक अंकासाठी एक शब्द, त्यानंतर एकच स्पेस. उदाहरणार्थ, संख्या 407 अंकी स्ट्रिंग बनते: "Four Zero Seven". सकारात्मक संख्या स्वीकारण्यासाठी प्रोग्राम लिहा आणि त्याची अंकांची स्ट्रिंग प्रिंट करा.
11. एक प्रोग्राम लिहा जो संख्या मध्ये रकम स्वीकारतो आणि शब्दात प्रिंट करतो. उदाहरणार्थ, 12500 च्या रकमेसाठी त्याने Twelve Thousand Five Hundred only स्ट्रिंग आउटपुट केली पाहिजे.
12. एक क्रमवारी न लावलेल्या अरेमध्ये तिसरा सर्वात मोठा घटक शोधण्यासाठी प्रोग्राम लिहा

प्रॅक्टिकल

1. तुम्हाला x नावाचे 1D अरे दिले आहेत, जे काही सर्वेक्षणाचा डेटा n (≤ 50) निरीक्षणासह साठवतात. डेटाचे सरासरी, भिन्नता आणि स्टँडर्ड विचलनाची गणना करण्यासाठी प्रोग्राम लिहा.

सरासरी, भिन्नता आणि स्टँडर्ड विचलनाची गणना करण्याचे सूत्र आहेत

$$\text{mean } (\bar{x}) = \frac{\sum_{i=1}^n x_i}{n}, \text{ variance} = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}, \text{ standard deviation} = \sqrt{\text{variance}}$$

Listing 4.20

```
/*
    Program to find mean, variance, and standard deviation
*/
#include <stdio.h>
#include <math.h>
int main()
{
    float x[50];
    int i, n;
    float mean, variance, std_deviation, sum, d;
    printf("\nEnter value for n : ");
    scanf("%d", &n);
    printf("\nEnter %d values\n", n);
    for ( i = 0; i < n; i++ ) {
```

```
        scanf("%f", &x[i]);
    }
    /* Compute the sum of all elements */
    sum = 0;
    for ( i = 0; i < n; i++ ) {
        sum = sum + x[i];
    }
    /* Compute mean (average) */
    mean = sum / n;
    /* Compute variance */
    sum = 0;
    for ( i = 0; i < n; i++ ) {
        d = x[i]-mean;
        sum = sum + d * d;
    }
    variance = sum / n;
    /* Compute standard deviation */
    std_deviation = sqrt(variance);
    /* print results of computations, rounded to 2 decimals */
    printf("Mean = %.2f\n", mean);
    printf("Variance = %.2f\n", variance);
    printf("Standard deviation = %.2f\n", std_deviation);
}
```

Test Run

```
Enter value for n : 10
Enter 10 values
12.5
10
15
25.75
30
22
16
19
24
11
Mean = 18.52
Variance = 41.06
Standard deviation = 6.41
```

2. मैट्रिक्सची बेरीज करण्यासाठी $A(m \times n)$ आणि $B(m \times n)$ प्रोग्राम लिहा.

Refer to Example 4.6.

3. मैट्रिक्सचा गुणाकार करण्यासाठी $A(m \times n)$ ने $B(p \times q)$ प्रोग्राम लिहा.

Refer to Example 4.7.

4. दिलेला शब्द पॅलिंड्रोम आहे की नाही हे तपासण्यासाठी प्रोग्राम लिहा.

जर शब्द दोन्ही टोकांपासून समान वाचला तर एक शब्द पालिंड्रोम आहे. उदाहरणे “*nitin*”, “*radar*”, “*madam*”, “*refer*”, etc.

शब्द पॅलिंड्रोम आहे की नाही हे तपासण्यासाठी, `strrev()` लायब्ररी फंक्शन वापरून स्ट्रिंग उलट करणे आणि नंतर मूळ स्ट्रिंगची उलट स्ट्रिंगशी तुलना करणे हा एक दृष्टीकोन आहे. जर दोन्ही सारखे असतील तर हा शब्द पॅलिंड्रोम आहे अन्यथा हा शब्द पॅलिंड्रोम नाही..

Listing 4.21

```
/*
    Program to check whether given word is palindrome or
    not,
    using library function strrev()
*/
#include<stdio.h>
#include<string.h>
int main()
{
    char str[30], temp[30];
    printf("\nEnter any word : ");
    gets(str);
    strcpy(temp, str);
    strrev(temp);
    printf("\nGiven word = %s\n", str);
    printf("\n%s after reversing = %s\n", str, temp);
    if ( strcmp(str, temp) == 0 )
        printf("\n%s is a palindrome word.\n", str);
    else
        printf("\n%s is not a palindrome word.\n", str);

    return 0;
}
```

Test Runs**First Run**

```
Enter any word : nitin
Given word = nitin
nitin after reversing = nitin
nitin is a palindrome word.
```

Second Run

```
Enter any word : never
Given word = never
never after reversing = reven
never is not a palindrome word.
```

दुसरा दृष्टिकोन असा आहे की आम्ही दोन इंडेक्स व्हेरिएबल्स घेतो, जसे की i आणि j , i पहिल्या अक्षराच्या अनुक्रमणिकेने आरंभ केले जाते आणि j शब्दाच्या शेवटच्या अक्षराच्या निर्देशांकासह प्रारंभ केले जाते.

$i < j$, पर्यंत पुनरावृत्ती करतो आणि प्रत्येक पुनरावृत्तीमध्ये आम्ही अनुक्रमणिका i आणि अनुक्रमणिका j मधील कॅरेक्टर ची प्रथम जुळत नसलेली तुलना करतो. जर एखादी जुळत सापडली तर शब्द पॅलिंड्रोम नाही.

पुढे, प्रत्येक पुनरावृत्तीमध्ये, अनुक्रमणिका i ची वाढ केली जाईल आणि अनुक्रमणिका j कमी केली जाईल.

जर कोणतीही जुळवाजुळव आढळली नाही आणि $i = j$ ही अट गाठली असेल, तर हे सूचित करेल की दिलेला शब्द एक पॅलिंड्रोम आहे.

Listing 4.22

```
/*
    Program to check whether given word is palindrome or
    not,
    without using any string manipulation function
*/
#include<stdio.h>
#include<string.h>
int main()
{
    char str[30];
    int i = 0, j, n = 0;
    printf("\nEnter any word : ");
    gets(str);
    /* to get length of word in variable 'n' */
    while ( str[n] != '\0' )
        n++;
    i = 0;          /* index of first character */
    j = n-1;        /* index of last character */
```



```

while ( i < j )
{
    if ( str[i] != str[j] ) {
        printf("\n%s is not a palindrome word.\n", str);
        return 0;
    }
    i++;
    j--;
}
printf("\n%s is a palindrome word.\n", str);
return 0;
}

```

Test Runs

First Run

```

Enter any word : madam
madam is a palindrome word.

```

Second Run

```

Enter any word : india
india is not a palindrome word.

```

आणखी माहिती

अरेचा विषय प्रोग्रामिंग भाषेचा एक महत्वाचा घटक आहे. असंख्य वास्तविक जीवनातील अडचणी आहेत ज्यांना डेटाचा मोठा संग्रह हाताळण्याची आवश्यकता आहे आणि संगणकावर प्रोग्राममध्ये डेटा संचयित करण्यासाठी अरे ही एक आदर्श डेटा स्ट्रक्चर आहेत.

शिक्षकाने अरेची संकल्पना, अरेची आवश्यकता आणि C भाषेत अरे हाताळण्याशी संबंधित विविध पैलू समजून घेणे अपेक्षित आहे.

वास्तविक जीवनातील परिस्थितीतून उदाहरणे घेऊन आणि त्यांचे निराकरण करण्यासाठी C प्रोग्राम तयार करून शिक्षकांनी अरेचा वापर दर्शविला पाहिजे.

संदर्भ आणि सूचविलेले वाचन

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi..
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

5

मूलभूत अल्गोरिदम

युनिट वैशिष्ट्ये

हे युनिट एखाद्या समीकरणाचे शोध, वर्गीकरण आणि उपाय शोधण्याशी संबंधित विषयांवर चर्चा करते. विविध शोध आणि वर्गीकरणाचे कार्य या युनिटमध्ये योग्य उदाहरणांसह स्पष्ट केले आहे. याव्यतिरिक्त, बाय सेक्शन पद्धतीचा वापर करून समीकरणाचे निराकरण शोधण्याची प्रक्रिया देखील स्पष्ट केली आहे.

तर्कशास्त्र

वास्तविक जीवनात अशा अनेक परिस्थिती आहेत जिथे आपल्याला विशिष्ट माहितीचा एक तुकडा शोधायचा असतो. जोपर्यंत आपण एखादा तुकडा किंवा ती माहिती शोधू शकत नाही जो पर्यंत प्रत्येक आयटमची तपासणी करण्याचा एकमेव मार्ग व्यवस्थापित केला नसल्यास तो सापडला नाही. तथापि, जर माहिती एका विशिष्ट क्रमाने आयोजित केली गेली असेल तर आपण कार्यक्षम अल्गोरिदम वापरून कोणतीही वस्तू खूप द्रुतपणे शोधू शकतो.

हे युनिट विद्यार्थ्यांना इच्छित क्रमाने माहितीची क्रमवारी लावण्यासाठी विविध तंत्रे, माहितीची दिलेली आयटम शोधण्यासाठी विविध तंत्रे समजण्यास मदत करते.

याव्यतिरिक्त, हे आपल्याला एखाद्या समीकरणाचे मूळ कसे शोधू शकते हे देखील स्पष्ट करते.

पूर्व-आवश्यकता

जरी, या कोर्ससाठी काही विशिष्ट आवश्यकता नाहीत. तथापि, खालील दिलेल्या मध्ये वाजवी ज्ञान हा एक अतिरिक्त फायदा होईल:

- कंडिशनल ब्रॅचिंग (Conditional branching)
- लूपिंग/ इटरेशन्स (Looping/Iteration)
- अरे (Arrays)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U5-O1:	विविध सर्चिंग अल्गोरिदम स्पष्ट आणि अंमलात आणणे.
U5-O2:	विविध सॉर्टिंग अल्गोरिदम स्पष्ट आणि अंमलात आणणे.
U5-O3:	एखाद्या समीकरणाची मुळे शोधण्याची पद्धत स्पष्ट करा आणि अंमलात आणणे.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 5 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U5-O1	3	3				3		
U5-O2	3	3				3		
U5-O3	3	3				3		

5.1 सर्चिंग अल्गोरिदम (SEARCHING ALGORITHMS)

अरे मध्ये दिलेल्या घटकाचे स्थान शोधण्याची प्रक्रिया म्हणजे शोध(सर्च). दिलेला घटक आढळल्यास शोध यशस्वी असल्याचे म्हटले जाते, म्हणजे घटक अरेमध्ये अस्तित्वात आहे; अन्यथा अयशस्वी.

शोध ऑपरेशनसाठी दोन पद्धती आहेत:

- लिनिअर सर्च (Linear search)
- बायनरी सर्च (Binary search)

एक निवडलेला अल्गोरिदम सामान्यतः अरे घटकांच्या संघटनेवर अवलंबून असतो. जर घटक यादृच्छिक क्रमाने असतील तर मग लिनिअर सर्च तंत्र वापरावे लागेल आणि अरे घटकांची क्रमवारी लावल्यास बायनरी सर्च तंत्र वापरणे अधिक श्रेयस्कर आहे. पुढील दोन भागात या दोन शोध तंत्रांचे वर्णन केले आहे.

5.1.1 लिनिअर सर्च (Linear Search)

अरे a बद्दल कोणतीही माहिती दिलेली नसल्यास, दिलेल्या घटकाचा शोध घेण्याचा एकमात्र मार्ग म्हणजे त्या घटकाची प्रत्येक घटकाशी तुलना करणे. आयटम शोधण्यासाठी अनुक्रमे फिरणाऱ्या या पद्धतीस लिनिअर सर्च किंवा सिकव्हेन्सशिअल सर्च (linear search or sequential search) म्हणतात.

Listing 5.1

```

/*
 *   Program to demonstrate the use of linear search technique
 *   to search a given element in an un-sorted array
 */
#include <stdio.h>
int main()
{
    int a[50], i, n, item, flag;
    printf("\nEnter size of array n(<=50) : " );
    scanf("%d", &n );
    printf("\nEnter %d elements of array\n", n);

```

```

for ( i = 0; i < n; i++ )
    scanf( "%d", &a[i] );
printf( "\nEnter element to search : " );
scanf( "%d", &item );
flag = 0;
for ( i = 0; i < n; i++ )
{
    if( item == a[i] )      /* match found */
    {
        flag = 1;
        break;
    }
}
if ( flag == 1 )
    printf( "\nElement found at index: %d\n", i );
else
    printf( "\nElement not found...\n" );
return 0;
}

```

Test Run - 1

```

Enter size of array n(<=50) : 10
Enter 10 elements of array
12 5 18 9 11 10 25 36 22 100
Enter element to search : 11
Element found at index: 4

```

Test Run - 2

```

Enter size of array n(<=50) : 10
Enter 10 elements of array
12 5 18 9 11 10 25 36 22 100
Enter element to search : 55
Element not found...

```

गुंतागुंत विश्लेषण (Complexity Analysis)

सर्वोत्तम शक्य परिस्थितीत, आयटम प्रथम स्थानावर येऊ शकते. या प्रकरणात, शोध ऑपरेशन केवळ एका तुलनेत यशस्वी समाप्त होते. तथापि, सर्वात वाईट परिस्थिती उद्भवते जेव्हा एखादी गोष्ट शेवटच्या स्थितीत असते किंवा अंरमधून गहाळ होते. मागील

प्रकरणात, शोध n तुलनाशी यशस्वीरित्या समाप्त होते. नंतरच्या प्रकरणात, शोध n तुलनांसह अपयशी ठरते. अशाप्रकारे, आपल्याला आढळले की सर्वात वाईट परिस्थितीत लिनिअर सर्च मध्ये $O(n)$ ऑपरेशन आवश्यक असते.

5.1.2 बायनरी सर्च (Binary Search)

समजा अरे ए चे घटक चढत्या क्रमाने सॉर्ट केले गेले आहेत (जर घटक संख्या असतील तर) किंवा शब्दकोष क्रमात (घटक स्ट्रिंग असल्यास). सर्वोत्तम शोध अल्गोरिदमला, बायनरी सर्च म्हणतात, दिलेला घटक शोधण्यासाठी वापरला जातो.

आपण हा दृष्टिकोन आपल्या दैनंदिन जीवनात वापरतो. उदाहरणार्थ, समजा संगणकाच्या शब्दकोषात आपल्याला मॉडेम या शब्दाचा अर्थ शोधायचा आहे. अर्थात, आपण पृष्ठाद्वारे पृष्ठ शोधत नाही. आपण अर्ध्यामध्ये (अंदाजे) शब्दकोष शोधत आहोत की कोणत्या अर्ध्या भागामध्ये हा शब्द शोधला जात आहे. त्यानंतरच्या शोधासाठी एक अर्धा सोडून दिला जातो आणि आपण दुसऱ्या अर्ध्यामध्ये शोधतो. आपण एकतर आवश्यक पद शोधत नाही तोपर्यंत ही प्रक्रिया चालू ठेवतो किंवा शब्दकोषामधून ती संज्ञा गहाळ होत नाही, जे शेवटी असे दर्शविते की आता फक्त एक शेवटचे पृष्ठ राहिले आहे.

Example 5.1: बायनरी शोध तंत्राचे कार्य स्पष्ट करण्यासाठी, 7 घटकांसह खालील क्रमवारी लावलेल्या अरे A चा विचार करा.

3, 10, 15, 20, 35, 40, 60

आणि आपल्याला घटक 15 शोधायचा आहे.

Solution:

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
Given Array a	3	10	15	20	35	40	60

सुरुवातीस, आपण $beg = 0$, $end = 6$ आणि मध्यम घटकाची अशी गणना करतो

(To start with, we take $beg = 0$, $end = 6$, and compute location of middle element as)

$$mid = (beg + end) / 2 = (0 + 6) / 2 = 3$$

Since $a[mid]$, i.e., $a[3] = 20$, and $beg \leq end$. आपण पुढील पुनरावृत्ती सुरू करतो.

As $a[mid] = 20 > 15$, therefore, we take $end = mid - 1 = 3 - 1 = 2$, beg अजूनही अपरिवर्तित राहते. (whereas beg remains unchanged). मध्यम घटकाची गणना करा.

$$mid = (beg + end) / 2 = (0 + 2) / 2 = 1$$

Since $a[mid]$, i.e., $a[1] = 10$, and $beg \leq end$. आपण पुढील पुनरावृत्ती सुरू करतो.

As $a[mid] = 10 < 15$, therefore, we take $beg = mid + 1 = 1 + 1 = 2$, end अजूनही अपरिवर्तित राहते. (whereas end remains unchanged.)

Since $beg = end$, मध्यम घटकाची गणना करा.

$$mid = (beg + end) / 2 = (2 + 2) / 2 = 2$$

Since $a[mid]$, i.e., $a[2] = 15$, शोध यशस्वी संपतो.

घटक निर्देशांक 2 वर आढळतो. (The element is found at index 2.)

Listing 5.2

```
/*
 * Program to demonstrate the use of binary search technique
 * to search a given element in a sorted array
 */
```

```
#include <stdio.h>
int main()
{
    int a[50], i, n, item, beg, end, mid, flag;
    printf("\nEnter size of array n(<=50) : " );
    scanf("%d", &n );
    printf("\nEnter %d elements of array in ascending order\n", n);
    for ( i = 0; i < n; i++ )
        scanf( "%d", &a[i] );
    printf( "\nEnter element to search : " );
    scanf( "%d", &item );
    flag = 0;
    beg = 0;
    end = n - 1;
    while ( beg < end )
    {
        mid = ( beg + end ) / 2;
        if( item == a[mid] ) {          /* match found */
            flag = 1;
            break;
        }
        if( item < a[mid] )
            end = mid - 1;
        else
            beg = mid + 1;
    }

    if ( flag == 1 )
        printf( "\nElement found at index: %d\n", mid );
    else
        printf( "\nElement not found...\n" );
    return 0;
}
```

Test Run - 1

```
Enter size of array n(<=50) : 10↵
Enter 10 elements of array in ascending order
5 9 10 11 12 18 25 36 44 100
Enter element to search : 44
Element found at index: 8
```

Test Run - 2

```
Enter size of array n(<=50) : 10
Enter 10 elements of array in ascending order
5 9 10 11 12 18 25 36 44 100
Enter element to search : 35
Element not found...
```

गुंतागुंत विश्लेषण (Complexity Analysis)

प्रत्येक पुनरावृत्तीमध्ये, शोध अरेच्या अर्ध्या भागापर्यंत कमी केला जातो. तर अरे मधील n घटकांकरिता $\log_2 n$ पुनरावृत्ती होईल. अशा प्रकारे, बायनरी शोधाची जटिलता $O(\log_2 n)$ आहे. अरेमध्ये नसतानाही घटकाची स्थिती विचारात न घेता ही जटिलता समान असेल.

5.2 सॉर्टिंग अल्गोरिदम (SORTING ALGORITHMS)

क्रमवारी(सॉर्टिंग) लावणे ही काही तार्किक क्रमाने घटकांची व्यवस्था करण्याची प्रक्रिया आहे. ही तार्किक क्रमवारी वर्णांक किंवा वर्णक्रमानुसार अक्षरी मूल्यांच्या बाबतीत चढत्या किंवा उतरत्या क्रमाने असू शकतात.

आपण जवळजवळ आपल्या रोजच्या कामांमध्ये रोज ही प्रक्रिया करतो. उदाहरणार्थ, आपण आपल्या क्लास नोट्स तारखेच्या वाढत्या क्रमांकावर व्यवस्थित करतो जेणेकरून नंतर त्यांचा संदर्भ घेता येईल. यासारख्या बऱ्याच परिस्थिती आहेत ज्यात आपण अशा गणन करू शकता.

या विभागात, आपण या सॉर्टिंग अल्गोरिदमची चर्चा करीत आहोत:

- बबल सॉर्ट (Bubble sort)
- इंसरशन सॉर्ट (Insertion sort)
- सिलेक्शन सॉर्ट (Selection sort)

आपण अरे a ज्याची साईझ n आहे चा विचार करूया ज्याचे घटक या सर्व अल्गोरिदमांच्या चर्चेसाठी `int` टाइप आहेत.

5.2.1 बबल सॉर्ट (Bubble Sort)

अरेची क्रमवारी लावण्यासाठी बबल सॉर्ट पद्धतीला $(n-1)$ पास आवश्यक आहे. प्रत्येक पासमध्ये प्रत्येक घटकाची $[i]$ तुलना $[i + 1]$ सह केली जाते, $i = 0$ ते $(n-k)$ साठी, जेथे k पास क्रमांक आहे आणि जर ते ऑर्डरच्या बाहेर नसतात, म्हणजे जर $[i] > a[i + 1]$, ते स्वॅप केले जातात. हे सर्वात मोठ्या घटकास कारणीभूत ठरू शकते किंवा बबल अप करेल.

पहिल्या पासच्या समाप्तीनंतर, अरेमधील सर्वात मोठा घटक $(n-1)$ व्या स्थानावर ठेवला जाईल आणि प्रत्येक पाठोपाठ पुढचा सर्वात मोठा घटक $(n-2)$, $(n-3)$, ..., 1 , वर ठेवला जाईल अनुक्रमे.

अधिक स्पष्टतेसाठी, खालील पायऱ्यांचे काळजीपूर्वक परीक्षण करा.

Pass 1:

- Step 1. If $a[0] > a[1]$ then swap $a[0]$ and $a[1]$.
 Step 2. If $a[1] > a[2]$ then swap $a[1]$ and $a[2]$.
 \vdots
 Step $n-1$. If $a[n-2] > a[n-1]$ then swap $a[n-2]$ and $a[n-1]$.

Pass 2:

- Step 1. If $a[0] > a[1]$ then swap $a[0]$ and $a[1]$.
 Step 2. If $a[1] > a[2]$ then swap $a[1]$ and $a[2]$.
 \vdots
 Step $n-2$. If $a[n-3] > a[n-2]$ then swap $a[n-3]$ and $a[n-2]$.
 \vdots

Pass k :

- Step 1. If $a[0] > a[1]$ then swap $a[0]$ and $a[1]$.
 Step 2. If $a[1] > a[2]$ then swap $a[1]$ and $a[2]$.
 \vdots
 Step $n-k$. If $a[n-k+1] > a[n-k]$ then swap $a[n-k+1]$ and $a[n-k]$.
 \vdots

Pass $n-1$:

- Step 1. If $a[0] > a[1]$ then swap $a[0]$ and $a[1]$.

($n-1$) पास झाल्यानंतर अरे चढत्या क्रमाने सॉर्ट केले जातील.

Example 5.2: बबल सॉर्ट पद्धतीचे कार्य स्पष्ट करण्यासाठी, पुढील अरे चढत्या क्रमाने क्रमवारी लावा.

12 40 3 2 15

Solution: लक्षात घ्या की दिलेल्या पासचे आउटपुट पुढील पाससाठी इनपुट बनते. क्रमवारी लावण्याची प्रक्रिया स्वयं स्पष्टीकरणात्मक आहे. प्रत्येक पासमध्ये, एक घटक, छायांकित म्हणून दर्शविलेला आहे, त्याच्या अंतिम स्थानापर्यंत पोचतो.

दिलेली अरे असा दर्शविली जाऊ शकतो.

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$
12	40	3	2	15

Pass-1:

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$
12	40	3	2	15

(a) तुलना करा $a[0]$ आणि $a[1]$. ज्याअर्थी $a[0] < a[1]$ ($12 < 40$), कोणतीही कारवाई केली जात नाही.

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$
12	40	3	2	15

(b) तुलना करा $a[1]$ आणि $a[2]$. ज्याअर्थी $a[1] > a[2]$ ($40 > 3$), हे घटक स्वॅप करा.

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$
12	3	40	2	15

(c) तुलना करा $a[2]$ आणि $a[3]$. ज्याअर्थी $a[2] > a[3]$ ($40 > 2$), हे घटक स्वॅप करा..

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$
12	3	2	40	15

(d) तुलना करा $a[3]$ आणि $a[4]$. ज्याअर्थी $a[3] > a[4]$ ($40 > 15$), हे घटक स्वॅप करा..

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$
12	3	2	15	40

अशा प्रकारे, प्रथम पास झाल्यानंतर 40 वर्षाचा सर्वात मोठा घटक अंतिम स्थानावर गेला आहे. तथापि, उर्वरित संख्या, ज्यांनी त्यांची स्थिती बदलली असेल त्यांची क्रमवारी लावणे बाकी आहे. छायांकित भाग क्रमवारी लावलेले घटक दर्शवितो.

उर्वरित पाससाठी, आवश्यक असल्यास केवळ तुलना आणि स्वॅप्स दर्शविल्या जातील.

Pass-2:

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
12	3	2	15	40	Swap
3	12	2	15	40	Swap
3	2	12	15	40	No swap
3	2	12	15	40	

दुसऱ्या पासनंतर, 15 क्रमांकाचा दुसरा सर्वात मोठा घटक अंतिम स्थानावर गेला आहे.

Pass-3:

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
3	2	12	15	40	Swap
2	3	12	15	40	No swap
2	3	12	15	40	

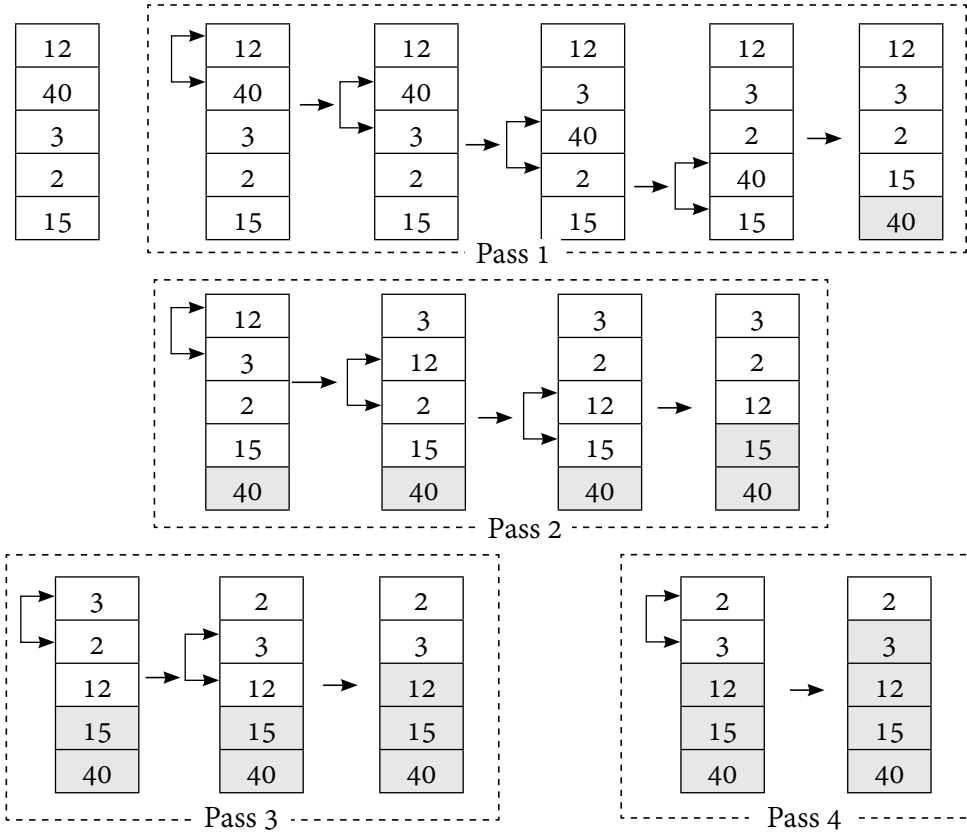
तिसऱ्या पासनंतर, तिसरा सर्वात मोठा घटक, 12, अंतिम स्थानावर गेला आहे.

Pass-4:

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
2	3	12	15	40	No swap
2	3	12	15	40	

चौथ्या पासनंतर, चौथा सर्वात मोठा घटक, 3, अंतिम स्थानावर गेला आहे. अशाप्रकारे अरे 4 पासमध्ये पूर्णपणे क्रमवारीत आहे.

वरील सॉर्टिंग प्रक्रिया खाली दर्शविल्यानुसार दृश्यमान देखील केली जाऊ शकते:



चित्र 5.1: बबल सॉर्ट पद्धतीचे उदाहरण

गुंतागुंत विश्लेषण(Complexity Analysis)

($n-1$) पास झाल्यानंतर अरे चढत्या क्रमाने सॉर्ट केले जातील. जसे आपण लक्षात घेतले असेल की पहिल्या पासला ($n-1$) तुलना आवश्यक आहे, दुसऱ्या पासला ($n-2$) आवश्यक आहे, ..., k th पास ($n-k$) आवश्यक आहे, आणि शेवटच्या पासला फक्त एक तुलना आवश्यक आहे. म्हणून, एकूण तुलना आहेत :

$$f(n) = (n-1) + (n-2) + (n-3) + \dots + (n-k) + \dots + 3 + 2 + 1 = n(n-1)/2 = O(n^2)$$

Listing 5.3

```

/*
 *   Program to demonstrate the use of Bubble sort method
 *   to sort a given array in ascending order
 */
#include <stdio.h>
int main()
{
    int a[50], i, j, n, temp;
    printf("\nEnter size of array n(<=50) : " );
    scanf("%d", &n );
    printf("\nEnter %d elements of array\n", n);
    for ( i = 0; i < n; i++ )
        scanf( "%d", &a[i] );
    for ( i = 0; i < n-1; i++ )
    {
        for ( j = 0; j < n-i-1; j++ )
        {
            if ( a[j] > a[j+1] ) {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    printf( "\nArray after sorting...\n\n" );
    for ( i = 0; i < n; i++ )
        printf( "%d ", a[i] );
    printf( "\n" );
    return 0;
}

```

Test Run

```

Enter size of array n(<=50) : 10
Enter 10 elements of array
5 12 10 15 22 18 2 -10 7 16
Array after sorting...
-10 2 5 7 10 12 15 16 18 22

```

5.2.2 सिलेक्शन सॉर्ट (Selection Sort)

अॅरेची क्रमवारी लावण्यासाठी सिलेक्शन सॉर्ट पद्धतीत $(n-1)$ पास आवश्यक आहे. पहिल्या पासमध्ये घटकांमधील सर्वात लहान घटक शोधा $a[0]$ पासून, $a[1], a[2], \dots, a[n-1]$ पर्यंत आणि पहिल्या घटकासह स्वॅप करा, म्हणजे, $a[0]$. दुसऱ्या पासमध्ये घटकांमधील सर्वात लहान घटक शोधा $a[1]$ पासून, $a[2], \dots, a[n-1]$ पर्यंत आणि $a[1]$ सह स्वॅप करा. आणि हे चालू ठेवा अधिक स्पष्टतेसाठी, खालील पायऱ्याचे काळजीपूर्वक परीक्षण करा:

Pass 1:

- संपूर्ण अॅरेमधील सर्वात लहान घटकाचे लोकेशन loc शोधा, i.e., $a[0], a[1], a[2], \dots, a[n-1]$.
- अदलाबदल $a[0]$ आणि $a[loc]$ करा. नंतर $a[0]$ क्षुल्लकपणे क्रमवारी लावली जाते.

Pass 2:

- सब अॅरेमधील सर्वात लहान घटकाचे लोकेशन loc शोधा, $a[1], a[2], a[3], \dots, a[n-1]$.
- अदलाबदल $a[1]$ आणि $a[loc]$ करा. नंतर घटकांची $a[0]$ आणि $a[1]$ क्रमवारी लावली जातात, $a[0] \leq a[1]$ पासून.
- \vdots

Pass k:

- सब अॅरेमधील सर्वात लहान घटकाचे लोकेशन loc शोधा, $a[k], a[k+1], a[k+2], \dots, a[n-1]$.
- अदलाबदल $a[k]$ आणि $a[loc]$ करा. मग घटक $a[0], a[1], a[2], \dots, a[k]$ क्रमवारी लावलेले आहेत.
- \vdots

Pass n-1:

- सर्वात लहान घटकाचे लोकेशन loc शोधा, $a[n-2], a[n-1]$.
 - अदलाबदल $a[n-2]$ आणि $a[loc]$ करा. मग घटक $a[0], a[1], a[2], \dots, a[n-1]$ क्रमवारी लावलेले आहेत.
- $(n-1)$ पास झाल्यानंतर अॅरे चढत्या क्रमाने सॉर्ट केले जातील.

Example 5.3: सिलेक्शन सॉर्ट पद्धतीचे कार्य स्पष्ट करण्यासाठी, 7 घटकांसह खालील अॅरे a चा विचार करा.

20, 35, 40, 100, 3, 10, 15

Solution: प्रत्येक पुनरावृत्ती i मध्ये आपल्याला अॅरेच्या क्रमांकाच्या भागातील सर्वात लहान घटकाचे लोकेशन loc आढळते. जर $loc \neq i$ तर i व loc मधील घटक स्वॅप केले जातील.

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
Given Array a	20	35	40	100	3	10	15

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$i = 0$
Pass 1:	20	35	40	100	3	10	15	$loc = 4$

खालील अॅरे मिळविण्यासाठी घटकांची अदलाबदल करा $a[0]$ आणि $a[4]$, म्हणजे, 20 आणि 3

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$i = 1$
Pass 2:	3	35	40	100	20	10	15	$loc = 5$

खालील अँरे मिळविण्यासाठी घटकांची अदलाबदल करा $a[1]$ आणि $a[5]$, म्हणजे, 35 आणि 10

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$i = 2$
Pass 3:	3	10	40	100	20	35	15	$loc = 6$

खालील अँरे मिळविण्यासाठी घटकांची अदलाबदल करा $a[2]$ आणि $a[6]$, म्हणजे, 40 आणि 15

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$i = 3$
Pass 4:	3	10	15	100	20	35	40	$loc = 4$

खालील अँरे मिळविण्यासाठी घटकांची अदलाबदल करा $a[3]$ आणि $a[4]$, म्हणजे, 100 आणि 20

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$i = 4$
Pass 5:	3	10	15	20	100	35	40	$loc = 5$

खालील अँरे मिळविण्यासाठी घटकांची अदलाबदल करा $a[4]$ आणि $a[5]$, म्हणजे, 100 आणि 35

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$i = 5$
Pass 6:	3	10	15	20	35	100	40	$loc = 6$

खालील अँरे मिळविण्यासाठी घटकांची अदलाबदल करा $a[5]$ आणि $a[6]$, म्हणजे, 100 आणि 40

	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
	3	10	15	20	35	40	100

चित्र 5.2: सिलेक्शन सॉर्ट पद्धतीचे उदाहरण

छायांकित भाग प्रत्येक पुनरावृत्तीनंतर क्रमवारी लावलेल्या अँरेचा भाग दर्शवितो. प्रत्येक पुनरावृत्तीमध्ये, क्रमवारी लावलेल्या भागाच्या उजवीकडे एक घटक जोडला जातो.

सिलेक्शन सॉर्ट अल्गोरिदम कार्यान्वित करण्यासाठी, घटकांमधील सर्वात लहान घटकाचे लोकेशन loc शोधणे आवश्यक आहे, $a[k-1]$, $a[k+1]$, $a[k+2]$, ..., $a[n-1]$, k th पास दरम्यान.

Listing 5.4

```
/*
 * Program to demonstrate the use of selection sort method
 * to sort a given array in ascending order
 */
#include <stdio.h>
int main()
{
    int a[50], i, j, n, temp, min, loc;
    printf("\nEnter size of array n(<=50) : ");
```

```
scanf("%d", &n );
printf("\nEnter %d elements of array\n", n);
for ( i = 0; i < n; i++ )
    scanf( "%d", &a[i] );
for ( i = 0; i < n-1; i++ )
{
    min = a[i];
    loc = i;
    for ( j = i+1; j < n; j++ )
    {
        if ( a[j] < min ) {
            min = a[j];
            loc = j;
        }
    }
    if ( loc != i ) {
        temp = a[i];
        a[i] = a[loc];
        a[loc] = temp;
    }
}
printf( "\nArray after sorting...\n\n" );
for ( i = 0; i < n; i++ )
    printf( "%d ", a[i] );
printf( "\n" );
return 0;
}
```

Test Run

```
Enter size of array n(<=50) : 6
Enter 6 elements of array
12 5 10 15 22 18
Array after sorting...
5 10 12 15 18 22
```

गुंतागुंत विश्लेषण (Complexity Analysis)

जसे आपण लक्षात घेतले असेल की पहिल्या पासला सर्वात लहान घटकाचे लोकेशन loc शोधण्यासाठी $(n-1)$ तुलना आवश्यक आहे, दुसऱ्या पासला $(n-2)$ आवश्यक आहे, ..., k th पास $(n-k)$ आवश्यक आहे, आणि शेवटच्या पासला फक्त एक तुलना आवश्यक आहे.

म्हणून एकूण तुलना आहेत :

$$\begin{aligned} f(n) &= (n-1) + (n-2) + (n-3) + \dots + (n-k) + \dots + 3 + 2 + 1 \\ &= n(n-1)/2 = O(n^2) \end{aligned}$$

5.2.3 इंसरशन सॉर्ट (Insertion Sort)

हा अल्गोरिदम ब्रिज पुल प्लेयर्समध्ये सर्वात लोकप्रिय आहे जेव्हा त्यांनी प्रथम त्यांची कार्डे क्रमवारी लावली.

या प्रक्रियेमध्ये, आम्ही एखादे विशिष्ट मूल्य उचलतो आणि नंतर त्यास क्रमवारी लावलेल्या उप यादीमध्ये योग्य ठिकाणी समाविष्ट करतो, म्हणजेच, k th आवर्ती दरम्यान घटक $a[k]$ ची क्रमवारी लावलेल्या उप अर्रेमध्ये त्याच्या योग्य ठिकाणी घातली जाते $a[1], a[2], a[3], \dots, a[k-1]$.

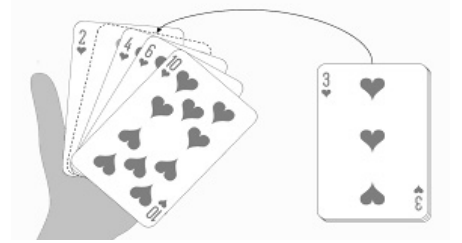
हे कार्य $a[k], a[k-1], a[k-2], a[k-3]$, आणि इतर घटकांची तुलना करून पूर्ण केले जाते आधी प्रथम घटक $a[j]$ जसे की $a[j] \leq a[k]$ आढळले. नंतर प्रत्येक घटक $a[k-1], a[k-2], \dots, a[j+1]$ एक स्थान वर हलविले जाते आणि नंतर अर्रे मध्ये $a[k]$ हे $(j+1)$ पोजीशन मध्ये समाविष्ट केले जाते.

अधिक स्पष्टतेसाठी, खालील पायऱ्याचे काळजीपूर्वक परीक्षण करा.

- Pass 1: $a[1]$ एकतर $a[0]$ च्या आधी किंवा नंतर घातला जातो जेणेकरून $a[0]$ and $a[1]$ क्रमवारी लावता येईल
- Pass 2: $a[2]$ $a[0]$ च्या आधी किंवा $a[0]$ आणि $a[1]$ किंवा $a[1]$ नंतर घातला जाईल जेणेकरून घटकांची क्रमवारी लावल्यास $a[0], a[1], a[2]$ होईल.
- Pass 3: $a[3]$ $a[0]$ च्या आधी किंवा $a[0]$ आणि $a[1]$ किंवा $a[1]$ आणि $a[2]$ किंवा $a[2]$ नंतर घातला आहे जेणेकरून घटक $a[0], a[1], a[2], a[3]$ क्रमवारी लावले आहेत
- ⋮
- Pass k : $a[k]$ क्रमवारी लावलेल्या उप अर्रे मध्ये योग्य ठिकाणी घातला आहे $a[0], a[1], a[2], \dots, a[k-1]$ जेणेकरून अंतर्भूत केल्यावर, घटक $a[0], a[1], a[2], \dots, a[k-1], a[k]$ सॉर्ट केले आहेत.
- ⋮
- Pass $n-1$: $a[n-1]$ क्रमवारी लावलेल्या उप अर्रे मध्ये योग्य ठिकाणी घातला आहे $a[0], a[1], a[2], \dots, a[n-2]$ जेणेकरून अंतर्भूत केल्यावर, घटक $a[0], a[1], a[2], \dots, a[n-1]$ सॉर्ट केले आहेत.
- $(n-1)$ पास झाल्यानंतर अर्रे चढत्या क्रमाने सॉर्ट केले जातील.

Example 5.4: इंसरशन सॉर्ट पद्धतीचे कार्य स्पष्ट करण्यासाठी, 7 घटकांसह खालील अर्रे a चा विचार करा

35, 20, 40, 100, 3, 10, 15



Solution: Given array a

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

35	20	40	100	3	10	15
----	----	----	-----	---	----	----

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

Pass 1:

35	20	40	100	3	10	15
----	----	----	-----	---	----	----

$a[1] < a[0]$, असल्याने, खालील अॅरे देण्यापूर्वी $a[0]$ च्या आधी घटक $a[1]$ घाला.

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

Pass 2:

20	35	40	100	3	10	15
----	----	----	-----	---	----	----

$a[2] > a[1]$, असल्याने कोणतीही क्रिया केली जात नाही.

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

Pass 3:

20	35	40	100	3	10	15
----	----	----	-----	---	----	----

Since $a[3] > a[2]$, पुन्हा कोणतीही कृती केली जात नाही.

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

Pass 4:

20	35	40	100	3	10	15
----	----	----	-----	---	----	----

$a[4]$ हे $a[3]$, $a[2]$, $a[1]$ तसेच $a[0]$ पेक्षा कमी असल्याने, म्हणून खालील अॅरे देऊन, $a[4]$ च्या आधी $a[0]$ घाला.

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

Pass 5:

3	20	35	40	100	10	15
---	----	----	----	-----	----	----

$a[5]$ हे $a[4]$, $a[3]$, $a[2]$ तसेच $a[1]$ पेक्षा कमी असल्याने, म्हणून खालील अॅरे देऊन, $a[5]$ च्या आधी $a[1]$ घाला.

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

Pass 6:

3	10	20	35	40	100	15
---	----	----	----	----	-----	----

$a[6]$ हे $a[5]$, $a[4]$, $a[3]$ तसेच $a[2]$ पेक्षा कमी असल्याने, म्हणून खालील अॅरे देऊन, $a[6]$ च्या आधी $a[2]$ घाला.

$a[0]$ $a[1]$ $a[2]$ $a[3]$ $a[4]$ $a[5]$ $a[6]$

3	10	15	20	35	40	100
---	----	----	----	----	----	-----

चित्र 5.3: इंसरशन सॉर्ट पद्धतीचे उदाहरण

Listing 5.5

```

/* Program to demonstrate the use of insertion sort method
 * to sort a given array in ascending order
 */
#include <stdio.h>
void main()
{
    int a[50], i, j, k, n, temp;
    printf("\nEnter size of array n(<=50) : " );
    scanf("%d", &n );
    printf("\nEnter %d elements of array\n", n);
    for ( i = 0; i < n; i++ )
        scanf( "%d", &a[i] );
    for ( k = 1; k < n; k++ ) {
        temp = a[k];
        j = k - 1;
        while ( ( temp < a[j] ) && ( j >= 0 ) ) {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = temp;
    }
    printf( "\nArray after sorting...\n\n" );
    for ( i = 0; i < n; i++ )
        printf( "%d ", a[i] );
    printf( "\n" );
}

```

Test Run

```

Enter size of array n(<=50) : 7
Enter 7 elements of array
5 12 10 15 22 18 16
Array after sorting...
5 10 12 15 16 18 22

```

5.3 एखाद्या समीकरणचे मूल शोधणे (FINDING ROOT OF AN EQUATION)

सामान्यतः समीकरणाची मुळे शोधण्यासाठी दोन प्रकारच्या पद्धती आहेत.

थेट पद्धती (Direct methods)

थेट पद्धती समीकरणाची मुळे एका मर्यादित संख्येने पायऱ्यामध्ये देतात. याव्यतिरिक्त, या पद्धती एकाच वेळी सर्व मुळे देण्यास सक्षम आहेत.

उदाहरणार्थ, चतुर्भुज समीकरणाची मुळे (roots of the quadratic equation)

$$ax^2 + bx + c = 0 \quad \text{where } a \neq 0$$

$$\text{are given by } x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

इटेरेटिव्ह पद्धत (Iterative methods)

चाचपणी आणि लुटी पद्धती म्हणून ओळखल्या जाणाऱ्या इटेरेटिव्ह पद्धती अनुक्रमे अंदाजे करण्याच्या कल्पनेवर आधारित आहेत. ते मुळाशी एक किंवा अधिक प्रारंभिक अंदाजासह प्रारंभ करतात आणि वाजवी अचूकतेसह समाधान प्राप्त होईपर्यंत पायऱ्याच्या निश्चित अनुक्रमांची पुनरावृत्ती करून अंदाजे क्रम मिळवतात. इटेरेटिव्ह पद्धती, सहसा, एका वेळी एक रूट देतात.

समीकरणे व्यक्तिचलितरित्या सोडवण्यासाठी इटेरेटिव्ह पद्धती अत्यंत अवजड आणि वेळ घेणाऱ्या असतात. तथापि, खालील कारणांमुळे ते संगणकावर वापरण्यासाठी सर्वात योग्य आहेत:

1. इटेरेटिव्ह पद्धती कॉम्प्यूटेशनल अल्गोरिदम म्हणून संक्षिप्तपणे व्यक्त केल्या जाऊ शकतात.
2. समान समस्यांचे वर्ग हाताळू शकणारे अल्गोरिदम तयार करणे शक्य आहे. उदाहरणार्थ, polynomial equation of degree n सोडविण्यासाठी अल्गोरिदम विकसित केला जाऊ शकतो.
3. थेट पद्धतींच्या तुलनेत इटेरेटिव्ह पद्धती Round-off लुटी नगण्य आहेत.

समीकरणाचे मूळ शोधण्यासाठी खालील लोकप्रिय इटेरेटिव्ह पद्धती आहेत.

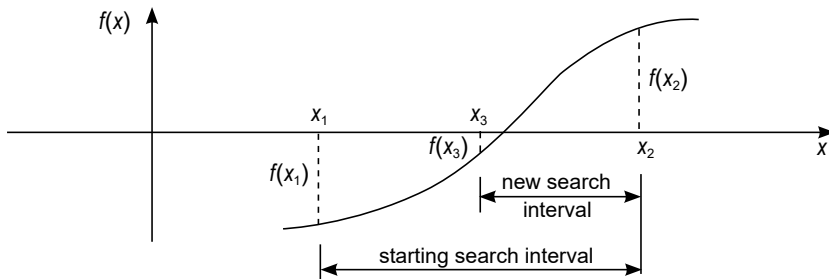
- Bisection Method
- Regula-falsi method
- Secant method
- Newton-Raphson method
- Method of Successive approximations

समीकरणाचे मूळ शोधण्यासाठी आपण बायसेकशन मेथड बदल शिकू या.

$$f(x) = 0$$

या विभागात

बायसेकशन पद्धत ही सर्वात सोपी पुनरावृत्ती करणारी पद्धत आहे. प्रारंभ करण्यासाठी, दोन प्रारंभिक अंदाजे, x_1 आणि x_2 असे म्हणा की $(x_1) \times f(x_2) < 0$, जे x_1 आणि x_2 दरम्यान रूट आहे याची खात्री करते. पुढील x -वॅल्यू, x_3 म्हणा, मध्यंतर म्हणून $[x_1, x_2]$ मोजले जाईल.



चित्र 5.4: बायसेकशन पद्धतीने रूट अंदाजे

तीन संभाव्यता उद्भवू शकतात:

- (i) जर $f(x_3) = 0$ असल्यास आपल्याकडे मूळ x_3 आहे.
- (ii) जर $f(x_1)$ आणि $f(x_3)$ विरुद्ध चिन्हाचे असल्यास, मूळ अंतराच्या (x_1, x_3) मध्ये असते. अशाप्रकारे x_2 ची जागा x_3 ने बदलली आणि नवीन मध्यांतर, जे सध्याच्या अंतराच्या निम्मे आहे, पुन्हा दुभाजक होईल.
- (iii) जर $f(x_1)$ आणि $f(x_3)$ समान चिन्हे असतील तर मूळ अंतराच्या (x_3, x_2) मध्ये असते. अशाप्रकारे x_1 चे स्थान बदलून x_3 केले जाईल आणि नवीन मध्यांतर पुन्हा दुभाजक होईल.

म्हणूनच या मध्यांतर दुभाजक प्रक्रियेची पुनरावृत्ती करून, आपण प्रत्येक नवीन पुनरावृत्तीमध्ये अर्धवट असलेल्या नवीन शोध मध्यांतर रूटला जोडत असतो.

जेव्हा शोध अंतर निर्धारित केलेल्या सहिष्णुतेपेक्षा लहान होतो (रूटमध्ये परवानगी असलेल्या त्रुटी) तेव्हा हे पुनरावृत्ती चक्र संपुष्टात आणले जाते. म्हणूनच, जर एप्सिलॉन आवश्यक रूटमध्ये निर्धारित सहनशीलता असेल तर पुनरावृत्ती चक्र संपुष्टात येते जेव्हा परिपूर्ण त्रुटी एप्सिलॉनपेक्षा कमी किंवा त्यास समान होते, म्हणजे,

$$|x_1 - x_2| \leq \epsilon$$

मूळच्या इच्छित अंदाजे म्हणून आपण शेवटच्या शोध मध्यांतरचा मध्य-बिंदू घेतो.

Listing 5.6

```
/*
 * Program to implement Bisection method to find one of the
 * root of equation x^3 - 4x - 9 = 0
 */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
double f( double x )
{
    return pow((double)x, (double)3.0)-4*x-9;
}
int main()
{
    float x1, x2, epsilon, x3;
    printf("Enter first point of the search interval : ");
    scanf("%f", &x1);
    printf("Enter second point of the search interval : ");
    scanf("%f", &x2);
    if ( ( f(x1) * f(x2) ) > 0 ) {
        printf("\nInitial approximations are unsuitable\n");
        return 1;
    }
}
```

```

printf("Enter prescribed tolerance : ");
scanf("%f", &epsilon);
do {
    x3 = (x1+x2)/2;
    if ( f(x1) * f(x3) < 0 )
        x2 = x3;
    else
        x1 = x3;
}
while( fabs((double)(x1-x2)) > epsilon);
printf("\nApproximate root = %8.4f\n", x3);
return 0;
}

```

Test Runs

Test Run 1

```

Enter first point of the search interval : 2.5
Enter second point of the search interval : 2.6
Initial approximations are unsuitable

```

Test Run 2

```

Enter first point of the search interval : 2.6
Enter second point of the search interval : 2.8
Enter prescribed tolerance : 0.0001
Approximate root = 2.7065

```

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- सर्चिंग अरेंमध्ये दिलेल्या घटकाचे स्थान शोधण्याची प्रक्रिया आहे.
- विविध शोध तंत्रांमध्ये लिनिअर सर्च आणि बायनरी सर्च समाविष्ट आहे.
- अरेंची क्रमवारी न लावली असल्यास, घटक शोधण्याची निवड केवळ लिनिअर सर्च आहे. तथापि, अरेंची क्रमवारी लावली असल्यास, बायनरी सर्च एक चांगली निवड आहे.
- सॉर्टिंग ही काही तार्किक क्रमाने घटकांची व्यवस्था करण्याची प्रक्रिया आहे.
- विविध सॉर्टिंग तंत्रांमध्ये बबल सॉर्ट, इंसरशन सॉर्ट आणि सिलेक्शन सॉर्ट समाविष्ट आहे.
- इतर पद्धतींपैकी, $f(x) = 0$. प्रकाराच्या polynomial समीकरणाच्या वेळी मूळ शोधण्याची सर्वात सोपी पद्धत म्हणजे बायसेकशन पद्धत.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. सर्चिंग काय आहे?
2. योग्य उदाहरणासह लिनिअर सर्च पद्धतीचे वर्णन करा.
3. योग्य उदाहरणासह बायनरी सर्च पद्धतीचे वर्णन करा..
4. सॉर्टिंग म्हणजे काय? सॉर्टिंग ची काय गरज आहे?
5. योग्य उदाहरणासह बबल सॉर्ट पद्धतीचे वर्णन करा.
6. योग्य उदाहरणासह इंसरशन सॉर्ट पद्धतीचे वर्णन करा.
7. योग्य उदाहरणासह सिलेक्शन बबल सॉर्ट पद्धतीचे वर्णन करा
8. समीकरणाचे मूळ शोधण्यासाठी बायसेकशन पद्धतीचे वर्णन करा.
9. 12, 7, 13, 9, 10, 77, 2, 8 प्रमाणे अरेचे घटक दिले आहेत. बबल सॉर्ट पद्धतीच्या पहिल्या पासनंतर घटकांची व्यवस्था काय असेल?
10. 11, 70, 13, 99, 5, 17, 21, 38 प्रमाणे अरेचे घटक दिले आहेत. सिलेक्शन सॉर्ट पद्धतीच्या तीन पास नंतर घटकांची व्यवस्था काय असेल?
11. 12, 7, 11, 92, 13, 71, 21, 38 प्रमाणे अरेचे घटक दिले आहेत. इंसरशन सॉर्ट पद्धतीच्या तीन पास नंतर घटकांची व्यवस्था काय असेल?
12. 12, 7, 11, 92, 13, 71, 21, 38 असे क्रमवारी न लावलेल्या (अनसॉर्ट) अरेचे घटक दिले आहेत. 71 आणि 100 घटक शोधण्यासाठी केलेल्या पायऱ्याचे वर्णन करा.
13. क्रमवारी लावलेल्या अरेचे घटक दिले गेले आहेत 82, 70, 61, 52, 43, 31, 24, 18. घटक 24 आणि 99 शोधण्यासाठी केलेल्या पायऱ्याचे वर्णन करा.

एकाधिक निवड प्रश्न

1. अरेमध्ये एकच आयटम शोधण्यासाठी लिनिअर सर्चचा सर्वात वाईट काळ कोणता आहे?
 - (a) Constant time
 - (b) Logarithmic time
 - (c) Linear time
 - (d) Quadratic time
2. अरेमध्ये एकच आयटम शोधण्यासाठी बायनरी सर्चचा सर्वात वाईट काळ कोणता आहे?
 - (a) Constant time
 - (b) Logarithmic time
 - (c) Linear time
 - (d) Quadratic time
3. अरेवर कोणती अतिरिक्त आवश्यकता ठेवली जाते, जेणेकरून बायनरी सर्च प्रविष्टी शोधण्यासाठी वापरला जाऊ शकतो?
 - (a) हिप अरे घटक तयार होणे आवश्यक आहे
 - (b) अरेमध्ये कमीतकमी 2 प्रविष्ट्या असणे आवश्यक आहे
 - (c) अरेची क्रमवारी लावणे आवश्यक आहे
 - (d) अरेचा आकार power of two असणे आवश्यक आहे

4. अरे सूचीचा वापर करून अनसॉर्ट केलेल्या अंमलबजावणीमध्ये आयटम शोधण्याचा उत्तम मार्ग आहे ____.
- (a) लिनिअर सर्च (b) बायनरी सर्च
(c) रँडम सर्च (d) डायरेक्ट सर्च
5. Big O संकेतक वापरून, बायनरी शोधासाठी आवश्यक तुलनांची संख्या आहे
- (a) $O(\log_2 n)$ (b) $O(n)$ (c) $O(n^2)$ (d) $O(n \log_2 n)$
6. n घटकांच्या सिलेक्शन सॉर्ट मध्ये, अल्गोरिदमच्या पूर्ण अंमलबजावणीमध्ये स्वॅप फंक्शनला किती वेळा कॉल केले जाते?
- (a) 1 (b) $n-1$ (c) $n \log_2 n$ (d) n^2
7. समजा 100 आयटमच्या सिलेक्शन सॉर्टने मुख्य लूपच्या 42 पुनरावृत्ती पूर्ण झाल्या आहेत. आता अंतिम घटकांमधील किती घटकांची हमी आहे?
- (a) 21 (b) 41 (c) 42 (d) 43
8. समजा आपण काही क्रमवारी लावणारे अल्गोरिदम वापरून चढत्या क्रमाने आठ इन्टिजर अरे सॉर्ट करत आहोत. अल्गोरिदमच्या मुख्य लूपच्या चार पुनरावृत्तीनंतर, अरे घटक येथे दर्शविल्याप्रमाणे क्रमित केले आहेत:
- 52 54 55 57 58 51 53 56
- कोणते विधान बरोबर आहे?
- (a) अल्गोरिदम एकतर सिलेक्शन सॉर्ट किंवा इंसरशन सॉर्ट असू शकते
(b) अल्गोरिदम कदाचित सिलेक्शन सॉर्ट असू शकेल परंतु तो इंसरशन सॉर्ट नाही
(c) अल्गोरिदम सिलेक्शन सॉर्ट नाही परंतु तो इंसरशन सॉर्ट असू शकतो
(d) अल्गोरिदम एकतर सिलेक्शन सॉर्ट किंवा इंसरशन सॉर्ट नाही
9. समजा आपण काही क्रमवारी लावणारे अल्गोरिदम वापरून चढत्या क्रमाने दहा इन्टिजर अरे सॉर्ट करत आहोत. अल्गोरिदमच्या मुख्य लूपच्या चार पुनरावृत्तीनंतर, अरे घटक येथे दर्शविल्याप्रमाणे क्रमित केले आहेत:
- 11 22 33 42 55 50 66 87 98 80
- कोणते विधान बरोबर आहे?
- (a) अल्गोरिदम एकतर सिलेक्शन सॉर्ट किंवा इंसरशन सॉर्ट असू शकते
(b) अल्गोरिदम कदाचित सिलेक्शन सॉर्ट असू शकेल परंतु तो इंसरशन सॉर्ट नाही
(c) अल्गोरिदम सिलेक्शन सॉर्ट नाही परंतु तो इंसरशन सॉर्ट असू शकतो
(d) अल्गोरिदम एकतर सिलेक्शन सॉर्ट किंवा इंसरशन सॉर्ट नाही
10. त्याची कार्यप्रदर्शन सुधारण्यासाठी खालीलपैकी कोणते अल्गोरिदम सुधारित केले जाऊ शकते?
- (a) सिलेक्शन सॉर्ट (b) बबल सॉर्ट
(c) इंसरशन सॉर्ट (d) वरीलपैकी काहीही नाही

ANSWERS									
1.	(c)	2.	(b)	3.	(c)	4.	(a)	5.	(a)
6.	(b)	7.	(c)	8.	(c)	9.	(b)	10.	(b)

प्रोग्रामिंग समस्या

1. दिलेले घटक उतरत्या क्रमाने क्रमवारी लावले जातात तेव्हा बायनरी सर्च वापरून घटक शोधण्यासाठी कोणता प्रोग्राम आहे.
2. तुम्हाला तुमच्या वर्गातील विद्यार्थ्यांची नावे दिली जातात. आपल्या आवडीचे क्रमवारी लावणारे अल्गोरिदम वापरून शब्दकोष क्रमाने नावे क्रमवारी लावण्यासाठी कोणता प्रोग्राम आहे.
3. आपल्या नावाची अक्षरे उलट शब्दकोष क्रमाने क्रमबद्ध करण्यासाठी एक प्रोग्राम लिहा.
4. बबल सॉर्ट पद्धत वापरून उतरत्या क्रमाने संख्यांची यादी क्रमवारी लावण्यासाठी प्रोग्राम लिहा. अल्गोरिदममध्ये क्षमता समाविष्ट करा की जर सर्व टप्पे संपण्यापूर्वी यादीची क्रमवारी लावली गेली तर अल्गोरिदम थांबला पाहिजे.

प्रॅक्टिकल

प्रयोगशाळेचा भाग म्हणून शोधण्यासाठी आणि क्रमवारी लावण्यासाठी वरील प्रोग्राम व्यतिरिक्त, संख्यात्मक पद्धती सोडविण्यासाठी प्रयोगशाळेत असे प्रोग्रॅम्स आहेत ज्यात संख्यात्मक भिन्नता आणि एकत्रीकरण शोधणे समाविष्ट आहे.

या प्रोग्रॅम्सचे पुढे वर्णन केले आहे.

1. संख्यात्मक भिन्नता (Numerical Differentiation)

विज्ञान आणि अभियांत्रिकीमधील बऱ्याच समस्यांमध्ये विविध प्रकारच्या कार्ये यांच्या भिन्नतेचा वापर समाविष्ट असतो. एखादे कार्य गणिताच्या स्वरूपात व्यक्त केले असल्यास, त्याचे व्युत्पन्न विश्लेषणात्मक पद्धतीद्वारे सहज मिळवता येते.

परंतु बऱ्याच घटनांमध्ये स्वतंत्र व्हेरिएबल x ची मूल्ये आणि अवलंबून व्हेरिएबल y ची संबंधित मूल्ये सारणीच्या स्वरूपात उपलब्ध आहेत.

100 मीटर धावण्याच्या शर्यतीत धावपटूंनी कशाप्रकारे अंतर पार केले त्या परिस्थितीचा विचार करा:

Time (Secs.) :	0	1	2	3	4	5	6	7	8	9	10
Distance (Mts.) :	0	2.5	10	20	30.5	50	54	65.5	77.2	88.5	100

येथे बरेच प्रश्न विचारले जाऊ शकतात, उदाहरणार्थ:

1. 5 सेकंदांनंतर ॲथलीटची गती किती होती?
2. टेपजवळ येताच ॲथलीटचा वेग किती होता?
3. 88 सेकंदांनंतर ॲथलीटचे प्रवेग काय होते?

सामान्यतः वेग आणि प्रवेग विश्लेषणात्मक पद्धतीने मोजले जातात. समजा y नंतर ॲथलीटने x सेकंदात प्रवास केलेले अंतर दर्शवते

$$\text{Speed} = \frac{dy}{dx} \quad \text{Acceleration} = \frac{d^2y}{dx^2}$$

परंतु y आणि x मधील गणितीय संबंध वरील उदाहरणात माहित नाही. म्हणून, विश्लेषणात्मक पद्धती वापरणे शक्य नाही. तथापि, आम्ही मूल्यांच्या मूल्यांकनासाठी टेबलमध्ये दिलेला डेटा वापरू शकतो, अंदाजे $\frac{dy}{dx}$ and $\frac{d^2y}{dx^2}$. तंत्र, जे अशा गणना करते, संख्यात्मक भिन्नता म्हणून ओळखले जाते.

टॅबुलेटेड केलेल्या बिंदूपैकी एकावर प्रथम व्युत्पत्तीचे संख्यात्मक मूल्य प्राप्त करण्याचे सामान्य सूत्र आहे.

$$\left. \frac{dy}{dx} \right|_{x=x_k} = \frac{1}{h} \sum_{j=1}^{n-k} (-1)^{j+1} \frac{d_{kj}}{j}$$

जिथे d_{kj} प्रतिनिधित्व Δ_k^j (jth order forward difference at tabulated point $x = x_k$) करते आणि n पॉइंट्स टेब्युलेट केलेल्या फंक्शनसाठी ऑर्डरच्या मॅट्रिक्स D द्वारे प्रतिनिधित्व केलेल्या फॉरवर्ड डिफरन्स टेबलचे घटक $(n-1) \times (n-1)$ आहे.

Table 5.1: Forward difference table

i	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
1	x_1	y_1	$\Delta y_1 = y_2 - y_1$	$\Delta^2 y_1 = \Delta y_2 - \Delta y_1$	$\Delta^3 y_1 = \Delta^2 y_2 - \Delta^2 y_1$
2	x_2	y_2	$\Delta y_2 = y_3 - y_2$	$\Delta^2 y_2 = \Delta y_3 - \Delta y_2$	
3	x_3	y_3	$\Delta y_3 = y_4 - y_3$		
4	x_4	y_4			

वरील टेबलावरून आपण पाहू शकतो की फंक्शनसाठी फॉरवर्ड डिफरंट टेबल सारख्या चार समान टॅब्युलेटेड केलेल्या 3×3 आकाराचे मॅट्रिक्स D अंतराचे बिंदू दर्शवू शकतो. सर्वसाधारणपणे n समान अंतराच्या बिंदूवर टेबलेटेड फंक्शनसाठी फॉरवर्ड डिफरंट टेबल $(n-1) \times (n-1)$ आकाराच्या मॅट्रिक्सद्वारे दर्शविले जाऊ शकते जेथे i th पॉइंट () वर j th ऑर्डर फॉरवर्ड फ्रंट दर्शविला जातो मॅट्रिक्स D चा घटक d_{ij} . लक्षात ठेवा $i = 1, 2, 3, \dots$,

$n-1$, पंक्तीसाठी 1 ते 1 ($n - i$) स्तंभातील घटक interest आहे.

Example 5.5: खाली सारणी दिली आहे.

x :	0.50	0.75	1.00	1.25	1.50
$y = f(x)$:	0.13	0.42	1.00	1.95	2.35

Find $f'(0.75)$.

Solution: दिलेल्या डेटासाठी अग्रेषित फरक टेबल आहे

i	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$
1	0.50	0.13	0.29	0.29	0.08	-1.00
2	0.75	0.42	0.58	0.37	-0.92	
3	1.00	1.00	0.95	-0.55		
4	1.25	1.95	0.40			
5	1.50	2.35				

$h = 0.25$ असल्याने आणि व्युत्पन्न $x = 0.75$ (म्हणजेच, $k = 2$) वर इच्छित आहे. तिसऱ्या मुदतीपर्यंत सूल विस्तारित करत आहोत.

$$\begin{aligned}
 \left. \frac{dy}{dx} \right|_{x=0.75} &= \frac{1}{h} \left[d_{21} - \frac{d_{22}}{2} + \frac{d_{23}}{3} \right] \\
 &= \frac{1}{0.25} \left[0.58 - \frac{0.37}{2} + \frac{-0.92}{3} \right] = \frac{1}{0.25} [0.58 - 0.185 - 0.307] \\
 &= 0.352
 \end{aligned}$$

Listing 5.7

```

/*
 * Program to compute first derivative of the tabulated function
 */
#include<stdio.h>
int main()
{
    float x[10], y[10], d[10][10], a, sum, derivative, h, term;
    int i, j, k, n, sign;
    printf("Enter number of table points n(<=10) : ");
    scanf("%d", &n);
    printf("Enter interval size : ");
    scanf("%f", &h);
    printf("Enter %d pair of values as x,y\n", n);
    for ( i = 1; i <= n; i++ )
        scanf("%f,%f", &x[i], &y[i]);
    printf("Enter tabulated point where to find derivative : ");
    scanf("%f", &a);
    if ( (a < x[1]) || (a > x[n]) )
    {
        printf("\nValue lies outside range\n");
        return 1;
    }
    i = 1;
    while ( a != x[i] )
        i++;
    k = i;
    for ( j = 1; j <= (n-1); j++ )
    {
        for ( i = 1; i <= (n-j); i++ )
        {
            if ( j == 1 )
                d[i][j] = y[i+1] - y[i];
            else
                d[i][j] = d[i+1][j-1] - d[i][j-1];
        }
    }
    sum = 0.0;
    sign = 1;
    for ( j = 1; j <= (n-k); j++ )

```

```

{
    term = sign * d[k][j] / j;
    sum = sum + term;
    sign = - sign;
}
derivative = (1.0/h) * sum;
printf("\nValue of derivative = %.3f\n", derivative);
return 0;
}

```

Test Run

```

Enter number of table points n(<=10) : 5
Enter interval size : 0.25
Enter 5 pair of values as x,y
0.5,0.13
0.75,0.42
1.0,1.0
1.25,1.95
1.5,2.35
Enter tabulated point where to find derivative : 0.75
Value of derivative = 0.352

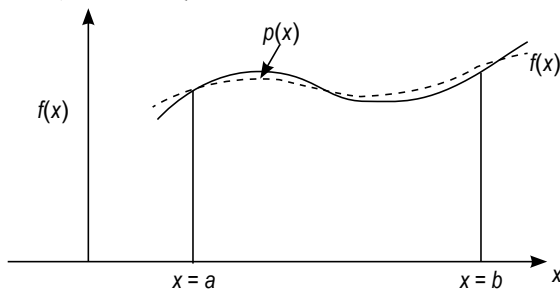
```

2. संख्यात्मक एकत्रीकरण (Numerical Integration)

न्यूमेरिकल इंटीग्रेशन फंक्शनच्या न्यूमेरिकल व्हॅल्यूजच्या सेटमधून निश्चित इंटीग्रलच्या व्हॅल्यूची गणना करण्याची प्रक्रिया आहे. जर एखाद्या कार्यास गणितीय अभिव्यक्ती म्हणून परिभाषित केले गेले असेल तर त्याचे अविभाज्य सहसा कॅल्क्युलसच्या तंत्राद्वारे निश्चित केले जाते, अशा उपायांना closed form समाधान म्हणतात.

बहुतेकदा, टेबल्सच्या रूपात उपलब्ध असलेली फंक्शन आणि व्हेरिएबल मधील गणितीय संबंध ज्ञात नसतात.

अशा सर्व परिस्थितींमध्ये, अविभाज्य मूल्ये संख्यात्मक तंत्राद्वारे मिळविली जाऊ शकतात ज्याचे लक्ष्य निश्चित अविभाजनांच्या अंदाजे मूल्यांकनासाठी प्रभावी प्रक्रिया प्रदान करणे आहे.

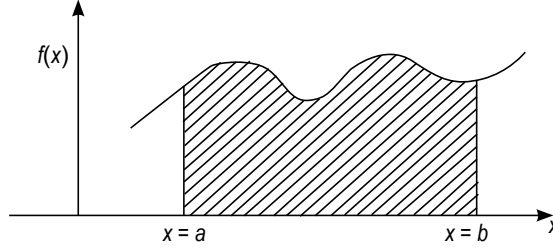


चित्र 5.5: फलन $f(x)$ साठी बहुपद $p(x)$

सराव मध्ये, फंक्शन $f(x)$ साठी मूल्यांचा सेट, डेटाची खालील सारणी आहे.

x :	a	x_1	x_2	x_3	...	b
$f(x)$:	$f(a)$	$f(x_1)$	$f(x_2)$	$f(x_3)$...	$f(b)$

इंटिग्रल च्या व्हॅल्यू मोजण्यासाठी वापरला जातो.



चित्र 5.6: छायांकित क्षेत्राद्वारे दर्शविलेले निश्चित अविभाज्य

फंक्शनचा निश्चित अविभाज्य म्हणजे $x = a$ आणि $x = b$ च्या मर्यादेच्या दरम्यान वक्र $y = f(x)$ अंतर्गत क्षेत्रफळ, फंक्शनचे अविभाज्य मोजण्याची समस्या कमी केल्याने छायांकित क्षेत्र शोधण्याची समस्या कमी होते. चित्र 5.6 मध्ये दर्शविले आहे.

संख्यात्मक एकत्रीकरण शोधण्यासाठी वापरल्या जाणाऱ्या लोकप्रिय पद्धतींमध्ये हे समाविष्ट आहे:

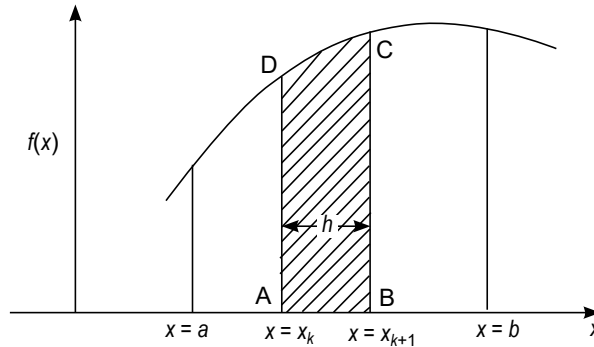
- Trapezoidal rule
- Simpson's 1/3rd rule
- Simpson's 3/8th Rule

फंक्शनचे संख्यात्मक एकत्रीकरण शोधण्यासाठी आम्ही ट्रॅपेझॉइडल नियम एक उदाहरण म्हणून विचार करू

ट्रॅपेझॉइडल नियम (Trapezoidal Rule)

ट्रॅपेझॉइडल नियम वक्र खाली असलेल्या क्षेत्राच्या जवळपास एक समान रंदीचे ट्रॅपेझॉइड तयार करण्यासाठी वक्र वर लागोपाठ बिंदू जोडतो आणि नंतर वक्र अंतर्गत अंदाजे क्षेत्र मिळविण्यासाठी या ट्रॅपेझॉइड्स अंतर्गत क्षेत्रांची बेरीज करतो.

$f(x)$ फंक्शनचा विचार करा, ज्याचा $x = a$ आणि $x = b$ मधील आलेख चित्र 5.7 मध्ये दर्शविला गेला आहे. वक्र अंतर्गत क्षेत्राचे अंदाजे अंतर प्रत्येक $[a, b]$ च्या प्रत्येक h रंदीच्या n च्या पट्ट्यामध्ये विभाजित करून आणि छायांकित क्षेत्राद्वारे दर्शविल्या जाणाऱ्या ट्रॅपेझॉइडच्या सहाय्याने प्रत्येक पट्टीचे क्षेत्रफळ मिळते.



चित्र 5.7: ट्रॅपेझॉइडल नियमानुसार क्षेत्राचा अंदाज

ट्रॅपेझॉइड नियमाचा फॉर्म्युला हा आहे.

$$I = \frac{h}{2} [y_1 + 2y_2 + 2y_3 + 2y_4 + \dots + 2y_n + y_{n+1}]$$

समजा $f(x)$ हे फंक्शन खाली असे दिले गेले आहे.

x	x_1	x_1+h	x_1+2h	...	x_1+nh
$y = f(x)$	y_1	y_2	y_3	...	y_{n+1}

Example 5.6: $f(x)$ हे फंक्शन खालीलप्रमाणे दिले आहे

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	1	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0

$x = 0$ आणि $x = 1.0$ दरम्यान $f(x)$ च्या अविभाज्य गणना करा.

Solution: Given $h = 0.1$ and $n = 10$

ट्रॅपेझॉइड नियमाचा फॉर्म्युला हा आहे.

$$I = \frac{h}{2} [y_1 + 2(y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 + y_9 + y_{10}) + y_{11}]$$

Substituting the values of y_i 's and h , we get

$$I = \frac{0.1}{2} [1 + 2 \times (1.2 + 1.4 + 1.6 + 1.8 + 2.0 + 2.2 + 2.4 + 2.6 + 2.8) + 3.0] = 2.00$$

Listing 5.8

```
/* Program to implement Trapezoidal rule for tabulated function */
#include<stdio.h>
int main()
{
    int i, n;
    float x[20], y[20], h, sum, integral;
    printf("Enter number of intervals n(<=20) : ");
    scanf("%d", &n);
    printf("Enter size of interval : ");
    scanf("%f", &h);
    printf("Enter %d pair of values as x,y \n", n+1);
    for ( i = 1; i <= (n+1); i++ )
        scanf("%f,%f", &x[i], &y[i]);
    sum = ( y[1] + y[n+1] ) / 2;
    for ( i = 2; i <= n; i++ )
        sum = sum + y[i];
    integral = h * sum;
    printf("\nValue of the integral = %7.2f\n", integral);
    return 0;
}
```

Test Run

```

Enter number of intervals n(<=20) : 10
Enter size of interval : 0.1
Enter 11 pair of values as x,y
0,1
0.1,1.2
0.2,1.4
0.3,1.6
0.4,2.0
0.5,2.2
0.6,2.4
0.7,2.6
0.8,2.8
0.9,3.0
1.0,1.2
Value of the integral = 2.00

```

आणखी माहिती

शिक्षकांनी या पद्धती उदाहरणांच्या मदतीने स्पष्ट केल्या पाहिजेत आणि स्वतःच सोडविण्यास सक्षम असावे अशी अपेक्षा आहे. एकदा त्यांना चर्चा केलेल्या विविध पद्धती समजल्यानंतर त्यांनी स्वतः प्रोग्राम तयार करण्यास प्रोत्साहित केले पाहिजे.

संदर्भ म्हणून प्रोग्राम विकसित करण्यासाठी विद्यार्थ्यांना मार्गदर्शन करा. पुस्तकात प्रोग्राम देण्यामागील हेतू म्हणजे चमच्याने आहार देण्यास प्रोत्साहित करणे नव्हे तर चांगली प्रोग्रामिंग शैली दर्शविणे होय.

संदर्भ आणि सूचविलेले वाचन

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi..
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

6

फंक्शन्स

युनिट वैशिष्ट्ये

हे युनिट फंक्शन्सशी संबंधित विषयांवर चर्चा करते. फंक्शन्सचा वापर प्रोग्रामरला जटिल समस्यांना लहान आकाराच्या स्वतंत्र उप समस्यांमध्ये विभाजित करण्यास अनुमती देतो ज्या स्वतंत्रपणे सोडवता येतात आणि दिलेल्या समस्यांचे निराकरण करण्यासाठी त्यांचे निराकरण संश्लेषित केले जाऊ शकते. हे युनिट विविध पैलूंची कार्ये स्पष्ट करते आणि त्यांचा वापर योग्य उदाहरणांसह प्रदर्शित करते.

तर्कशास्त्र

वास्तविक जीवनातील बऱ्याच परिस्थितींमध्ये आपल्याला अशा समस्यांना सामोरे जावे लागू शकते ज्यांचे आकार आणि गुंतागुंत जास्त आहे. या समस्यांचे एकाच शॉटमध्ये निराकरण करणे खूप कंटाळवाणे आणि त्रुटी असू शकते. अशा समस्यांचे निराकरण करण्याचा उत्तम मार्ग म्हणजे त्यांना एकमेकांपेक्षा स्वतंत्र असलेल्या उप समस्यांमध्ये विभाजित करणे म्हणजे सहजतेने निराकरण केले जाऊ शकते. एकदा प्रत्येक उप समस्येचे निराकरण झाल्यानंतर संपूर्ण समस्येचे निराकरण करण्यासाठी या उप समस्यांचे निराकरण एकत्रित केले जाऊ शकते. त्याचप्रमाणे, एक जटिल प्रोग्राम स्वतंत्र असलेल्या कार्यांमध्ये विभागला जाऊ शकतो. ही कार्ये एकमेकांपासून स्वतंत्रपणे विकसित केली जाऊ शकतात आणि नंतर संपूर्ण प्रोग्राम करण्यासाठी एकत्रित केली जाऊ शकतात. हे युनिट विद्यार्थ्याला फंक्शन्सशी संबंधित विविध पैलू समजण्यास मदत करेल.

पूर्व-आवश्यकता

जरी, या कोर्ससाठी काही विशिष्ट आवश्यकता नाहीत. तथापि, खालील दिलेल्या मध्ये वाजवी ज्ञान हा एक अतिरिक्त फायदा होईल:

- कंडिशनल ब्रॅचिंग (Conditional branching)
- ऑरे आणि स्ट्रिंग्स (Arrays and strings)
- लूपिंग/ इटरेशन्स (Looping/Iteration)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U6-O1:	फंक्शन्सची उपयुक्तता समजावून सांगा.
U6-O2:	विविध प्रकारची फंक्शन्स समजावून सांगा.
U6-O3:	लोकल डेटा आणि ग्लोबल डेटा संकल्पना स्पष्ट करा.
U6-O4:	फंक्शन्स डिक्लेअरिंग, डिफाईनिंग आणि कॉल करण्याशी संबंधित सर्व बाबी स्पष्ट करा.
U6-O5:	फंक्शनला वितर्क करण्याचे वेगवेगळे मार्ग समजावून सांगा.
U6-O6:	वास्तविक जीवनातील अडचणी सोडविण्यासाठी मॉड्यूलर प्रोग्राम विकसित करा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 6 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U6-O1	1				1			
U6-O2					1			
U6-O3					1			
U6-O4			3		2			
U6-O5			2		2			
U6-O6					3			

6.1 ओळख (INTRODUCTION)

आपण आतापर्यंत चर्चा केलेले प्रोग्राम्स खूप सोपे आणि सरळ होते. ते खूप प्रयत्न न करता समजल्या जाणाऱ्या समस्यांचे निराकरण करू शकले. जसजसे आपण मोठ्या आणि अधिक जटिल समस्यांकडे जात आहोत आणि म्हणूनच आपण या प्रोग्राम्सकडे जाताना प्रोग्राम्स अधिक प्राथमिक भागांपर्यंत कमी न करता, आपण समजू शकता की अशा समस्यांचे सर्व पैलू समजणे शक्य नाही.

या युनिटमध्ये आपण फंक्शनशी संबंधित विविध बाबी शिकू या.

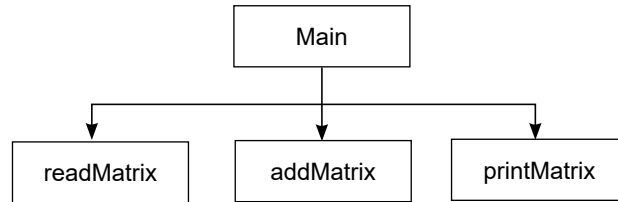
6.2 फंक्शन म्हणजे काय? (WHAT IS A FUNCTION?)

फंक्शन एक्झिक्युशनचे स्वतंत्र एकक असते जे प्रोग्रामच्या चौकटीत काही विशिष्ट कार्ये करते.

हे त्या दृष्टीने स्वतंत्र आहे की त्याचे कार्ये व्हेरिएबल्स लोकल डिक्लेअर करण्यासाठी स्वतःच्या डिक्लेरेशन आहेत आणि कार्ये पूर्ण करण्यासाठी कार्य परिभाषित करणाऱ्या निर्देशांचे स्वतःचे संच आहेत.

प्रत्येक C प्रोग्राम एक किंवा अधिक फंक्शन्सचा बनलेला असतो. जर हे फक्त एका फंक्शनचे बनलेले असेल तर ते main फंक्शन असेल. प्रोग्रामचे एक्झिक्युशन नेहमीच main फंक्शन पासून होते. तथापि, प्रोग्राममध्ये फंक्शनच्या संख्येवर कोणतेही बंधन नाही. आवश्यक फंक्शनची संख्या निश्चित करण्यासाठी घटक कार्य करत असलेल्या समस्यांचा आकार आणि जटिलता असेल.

जेव्हा आपण C प्रोग्राम चालवितो, ऑपरेटिंग सिस्टम main फंक्शनला कॉल करते आणि नंतर नियंत्रण main फंक्शनवर जाते. main फंक्शन कोणत्याही फंक्शनला कॉल करू शकते आणि main व्यतिरिक्त एखादे फंक्शन दुसऱ्या फंक्शनला कॉल करू शकते.



चित्र 6.1: मल्टीफंक्शन प्रोग्रामची श्रेणीबद्ध संस्था

चित्र 6.1, मध्ये दर्शविल्याप्रमाणे, प्रोग्राममध्ये main फंक्शन सह चार फंक्शन्स आहेत, प्रोग्रामचे उद्दीष्ट मॅट्रिक्स एड करण्याचा आहे. दिलेल्या मॅट्रिक्सचे इनपुट करण्यासाठी readMatrix, फंक्शन आणि $A_{m \times n}$ आणि $B_{m \times n}$ सारख्या दोन मॅट्रिक्सचे इनपुट करण्यासाठी main फंक्शनद्वारे दोनदा कॉल केला जाईल.

नंतर फंक्शन addMatrice कॉल केले जाईल जे $A_{m \times n}$ आणि $B_{m \times n}$ मॅट्रिक्सची ऍडिशन करेल आणि परिणामी आणखी एक $C_{m \times n}$ मॅट्रिक्स main फंक्शनला परत करेल.

अखेरीस, main फंक्शन printMatrix फंक्शनला कॉल करेल $C_{m \times n}$ मॅट्रिक्सचे आउटपुट करण्यासाठी.

6.3 फंक्शन वापरण्याचे फायदे (ADVANTAGES OF USING FUNCTIONS)

फंक्शन्सच्या वापराशी संबंधित बरेच फायदे आहेत. त्यातील काही अधिक महत्वाचे फायदे आहेत :

1. फंक्शन्सचा वापर मोठ्या आणि जटिल समस्यांना लहान, साध्या आणि व्यवस्थापित भागांमध्ये विभागून व्यवस्थापित करण्यास अनुमती देतो.
2. फंक्शन्सचा वापर कोडचा पुनर्वापर करण्याचा एक मार्ग प्रदान करतो जो प्रोग्रामपेक्षा एकापेक्षा जास्त वेळा आवश्यक असतो.
3. उदाहरण म्हणून, असे समजा की प्रोग्रामला पाच वेगवेगळ्या भागांमधील सरासरी संख्येची गणना करणे आवश्यक आहे. प्रत्येक वेळी डेटा भिन्न असतो. आपण सरासरी पाच वेळा मोजण्यासाठी कोड लिहू शकतो, परंतु यासाठी बरेच प्रयत्न करावे लागतील. फंक्शन म्हणून एकदा कोड लिहिणे आणि त्यास डेटाचे वेगवेगळे सेट देऊन सरासरी मोजण्यासाठी पाच वेळा कॉल करणे खूप सोपे आहे.
4. फंक्शन्सचा वापर डेटाचे संरक्षण करू शकतो. हे लोकल डेटा संकल्पना वापरून केले जाते. लोकल डेटामध्ये फंक्शनमध्ये वर्णन केलेला डेटा असतो. हा डेटा केवळ फंक्शनला उपलब्ध आहे आणि तोही फंक्शन एक्झिक्युट होत असताना.
5. प्रोग्रामसहित विविध फंक्शन समांतर पद्धतीने विकसित आणि चाचणी केली जाऊ शकतात आणि यामुळे प्रोग्रामचा एकूण विकास वेळ कमी होतो.

6.4 फंक्शनचे प्रकार (TYPE OF FUNCTIONS)

पुढील फंक्शनच्या दोन श्रेणी आहेत:

- **लायब्ररी फंक्शन्स (Library functions)** – ही फंक्शन्स जी पूर्व-लिखित, संकलित केलेली आहेत आणि त्यांचा मशीन कोड सिस्टम लायब्ररी फाइल्स मध्ये उपलब्ध आहे. आपल्या प्रोग्राममध्ये संदर्भित लायब्ररी फंक्शन्सचा मशीन कोड लिंकइंग साधण्याच्या दरम्यान लिंकर द्वारे जोडला जातो.

लक्षात घ्या की लायब्ररीची फंक्शन्स C भाषेचा भाग नाहीत; ते युझरच्या सोयीसाठी C कंपाइलर विकसित करणाऱ्या विक्रेत्यांद्वारे प्रदान केले गेले आहेत. त्यांचा वापर करण्यासाठी आपल्याला योग्य हेडर फाइल समाविष्ट करणे आवश्यक आहे.

- **युझर- डिफाईंड फंक्शन्स (User-defined functions)** – हे फंक्शन्स युझरने हातात घेतलेल्या प्रोग्रामच्या आवश्यकता पूर्ण करण्यासाठी विकसित केल्या आहेत. जर ते आधीपासूनच विकसित झाले असतील तर आपण त्यांचा नवीन कोडमध्ये त्यांचा सौस कोड वापरू आणि अशा प्रकारे बराच वेळ आणि मेहनत वाचवू शकतो.

या युनिटमध्ये आमची चर्चा प्रामुख्याने युझर- डिफाईंड फंक्शन्सवर आहे.

6.5 लोकल डेटा आणि ग्लोबल डेटाची संकल्पना (CONCEPT OF LOCAL DATA & GLOBAL DATA)

फंक्शनच्या मुख्य भागामध्ये डिक्लेअर औपचारिक अर्ग्युमेण्ट आणि व्हेरिएबल हे फंक्शनच्या बाहेर अज्ञात आहेत. व्हिझिबिलिटी हा कोणत्या फंक्शनने व्हेरिएबलला मान्यता दिली आणि कोणत्या व्हेरिएबलला दिली नाही हे निर्धारित करणारा घटक. लोकल व्हेरिएबल हे ज्या फंक्शनमध्ये डिक्लेअर केले आहे त्यास दृश्यमान (व्हिझिबल) असेल, परंतु इतरांना नाही.

याउलट, ग्लोबल व्हेरिएबल सुरुवातीला डिक्लेअर केले जाते, म्हणजेच main फंक्शन पूर्वी. हे सर्व फंक्शन्ससाठी दृश्यमान(व्हिझिबल)आहे आणि प्रोग्राम समाप्त होईपर्यंत अस्तित्वात आहे.

6.6 युझर- डिफाईंड फंक्शन्स (USER-DEFINED FUNCTIONS)

जरी सर्व C कंपाईलर्स उपलब्ध ग्रंथालयातील फंक्शन्स समृद्ध करतात, तरीही वास्तविकतेच्या समस्येवर सॉफ्टवेअर सोल्यूशन प्रदान करणार्या प्रोग्रामच्या आवश्यकतेनुसार, आवश्यकतेनुसार विशिष्ट फंक्शन्स तयार करण्याची आवश्यकता आहे.

पुढील आपण चर्चा करणार युझर- डिफाईंड फंक्शन्सच्या सर्व बाबींवर व्यवहार करण्यासाठी C भाषा सर्व वैशिष्ट्ये प्रदान करते यावर आपण पुढील चर्चा करणार आहोत.

C मधील युझर- डिफाईंड फंक्शन्स कार्ये खालील पाच श्रेणींमध्ये येतात:

- Functions that takes no argument(s) and return no value.
- Functions that takes argument(s) but return no value.
- Functions that takes no argument(s) but return a value.
- Functions that takes argument(s) and also return a value.
- Functions that return multiple values

6.7 डिक्लेअरिंग आणि डिफाईनिंग फंक्शन्स (DECLARING AND DEFINING FUNCTIONS)

C मधील प्रत्येक व्हेरिएबल प्रमाणे, प्रत्येक फंक्शन देखील डिक्लेअर आणि डिफाईंड आवश्यक आहे.

6.7.1 डिक्लेअरिंग फंक्शन (Declaring a Function)

फंक्शन डिक्लेरेशन मध्ये केवळ फंक्शन हेडर असते; यात कोणताही कोड नाही. फंक्शन हेडरमध्ये तीन भाग असतात: रिटर्न प्रकार, फंक्शनचे नाव आणि औपचारिक अर्ग्युमेण्ट सूची. अर्धविराम फंक्शन हेडरला अनुसरण करते.

```
returnType functionName(formal argument list);
```

चित्र 6.2: फंक्शन डिक्लेरेशन सिंटॅक्स

फंक्शन डिक्लेरेशन फंक्शनचे संपूर्ण चित्र देते ज्याचे नंतर वर्णन करणे आवश्यक आहे.

फंक्शन डिक्लेरेशनला फंक्शन प्रोटोटाइप असेही म्हणतात.

जर फंक्शनमध्ये कोणतेही अर्ग्युमेण्ट नसतील तर आपण कंसात शून्य लिहितो. फंक्शनमध्ये दोन किंवा अधिक अर्ग्युमेण्ट असल्यास, प्रत्येक स्वल्पविरामाने विभक्त केला जाईल.

खालील फंक्शन डिक्लेरेशन

```
int target(int a, int b, int c);
```

कंपाईलरला सांगते की फंक्शनचा रिटर्न टाईप इंट आहे, फंक्शनचे नाव largest आहे आणि त्यात तीन अर्ग्युमेण्ट आहेत, सर्व इंट प्रकारचे. अर्ग्युमेण्टची नावे महत्त्वपूर्ण नाहीत.

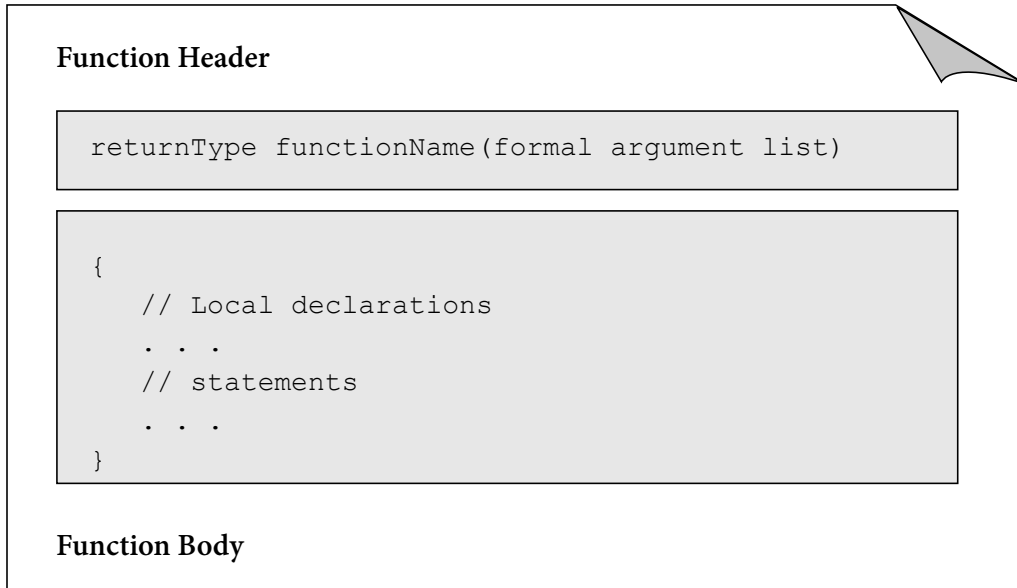
जरी खालील फंक्शन डिक्लेरेशन

```
int target(int, int, int);
```

हे तितकेच चांगले आहे आणि पूर्वीच्या डिक्लेरेशन प्रमाणे हेतू साध्य करते.

6.7.2 डिफाईनिंग फंक्शन (DEFINING A FUNCTION)

फंक्शन डेफिनिशन मध्ये दोन भाग असतात; फंक्शन हेडर आणि फंक्शन बॉडी.



चित्र 6.3: फंक्शन डेफिनिशन सिंटॅक्स

फंक्शन हेडर (Function Header)

फंक्शन हेडरमध्ये तीन भाग असतात: रिटर्न प्रकार, फंक्शनचे नाव आणि औपचारिक अर्ग्युमेंट सूची(list). फंक्शन हेडरच्या शेवटी सेमीकोलन वापरला जात नाही.

जेव्हा काहीही परत करायचे नसते तेव्हा रिटर्न टाईप व्हॉइड(void) वापरला जातो.

औपचारिक अर्ग्युमेंट सूची व्हेरिएबल घोषित करते जी कॉलिंग फंक्शनमधून डेटा प्राप्त करेल आणि त्यात व्हेरिएबल असू शकते ज्यायोगे डेटा कॉलिंग फंक्शनमध्ये परत पाठविला जाऊ शकतो.



The *function prototype* and *function header* must match, except semicolon.

फंक्शन बॉडी (Function Body)

फंक्शन बॉडीमध्ये लोकल डिक्लरेशन आणि फंक्शन पूर्ण करण्याचे कार्य परिभाषित केलेल्या सूचना असतात. बॉडी स्थानिक डिक्लरेशनसह प्रारंभ होते जे निर्देशानंतर फंक्शनद्वारे आवश्यक बदल निर्दिष्ट करतात.

सोप्या कार्ये करण्यासाठी डिक्लेअर आणि डिफाइन काही फंक्शन ची उदाहरणे बघूया.

Example 6.1: *maximum* नावाचे फंक्शन डिक्लेअर आणि डिफाइन करा जे तीन प्रकारचे अर्ग्युमेंट int घेते आणि त्यातील सर्वात मोठा रिटर्न करते.

डिक्लरेशन:

```
int maximum(int x, int y, int z);
```

डेफिनिशन:

```
int maximum(int x, int y, int z)
{
    if ( ( x > y ) && ( x > z ) )
        return x;
    else if ( y > z )
        return y;
    else
        return z;
}
```

वरील फंक्शनचे लॉजिक असे लिहिले जाऊ शकते.

```
int maximum(int x, int y, int z)
{
    int big;
    big = x;
    if ( y > big )
        big = y;
    if ( z > big )
        big = z;
    return big;
}
```

Example 6.2: *isEven* नावाचे फंक्शन डिक्लेअर आणि डिफाइन करा जो टाईप *int* चा सिंगल अर्ग्युमेंट *n* घेते आणि जर *n* इव्हन असेल तर 1 रिटर्न करा अन्यथा रिटर्न 0.

डिक्लरेशन:

```
int isEven(int n);
```

डेफिनिशन:

```
int isEven(int n)
{
    if ( n % 2 == 0 )
        return 1;
    else
        return 0;
}
```

वरील फंक्शनचे लॉजिक असे लिहिले जाऊ शकते.

```
int isEven(int n)
{
    return ( n % 2 ? 1 : 0 );
}
```

Example 6.3: *sumOfDigits* नावाचे फंक्शन डिक्लेअर आणि डिफाइन करा जो टाइप *int* चा सिंगल अर्ग्युमेंट *n* घेते आणि *n* च्या अंकांची बेरीज रिटर्न करा.

डिक्लेरेशन:

```
int sumOfDigits(int n);
```

डेफिनिशन:

```
int sumOfDigits(int n)
{
    int i, s = 0, d ;
    while ( n > 0 )
    {
        d = n % 10;
        s = s + d;
        n = n / 10;
    }
    return s;
}
```

6.8 कॉलिंग फंक्शन (CALLING A FUNCTION)

लायब्ररी फंक्शन्स प्रमाणेच फंक्शन मधून युजर-डिफाईन्ड फंक्शन्सला फक्त त्याच्या नावाने कॉल केले जाते, ज्यामध्ये कंसात असलेल्या वास्तविक अर्ग्युमेंटसह, काही असल्यास.

तथापि, कंसात कार्य करण्यासाठी काही वास्तविक अर्ग्युमेंट नसले तरीही पॅरंथेसिसने फंक्शनचे नाव अनुसरण केले पाहिजे.

वास्तविक अर्ग्युमेंट, जर काही असतील तर, number, type आणि order औपचारिक अर्ग्युमेंटसह क्रमवारीत असणे आवश्यक आहे.

जेव्हा फंक्शनचे नाव समोर येते तेव्हा नियंत्रण कॉल केलेल्या फंक्शनमध्ये हस्तांतरित केले जाते. औपचारिक अर्ग्युमेंट वास्तविक अर्ग्युमेंट द्वारे बदलले जातात आणि फंक्शनची एक्झिक्युशन केले जाते. जेव्हा रिटर्न स्टेटमेंट कार्यान्वित होते किंवा शेवटचे विधान त्याची एक्झिक्युशन पूर्ण करते, तेव्हा नियंत्रण परत कॉलिंग फंक्शनमध्ये हस्तांतरित केले जाते.

जर फंक्शन मूल्य परत करत नसेल तर फंक्शन कॉल एक स्वतंत्र सूचना म्हणून दिसून येईल.

जर फंक्शन व्हॅल्यू परत करत असेल तर फंक्शन कॉल असाईनमेंट स्टेटमेंटमध्ये, अरीथमेटिक एक्स्प्रेसन किंवा कंडिशनल स्टेटमेंट किंवा printf() फंक्शनमध्ये देखील दिसू शकेल.

उदाहरणार्थ 6.1 to 6.3 मध्ये परिभाषित फंक्शन्स खालील संभाव्य मार्गांनी कॉल केल्या जाऊ शकतात.

```
big = maximum(a, b, c);
```

```
if ( isEven(n) == 1 )
    printf( "\n%d is an Even number.\n" );
else
    printf( "\n%d is an Odd number.\n" );
```

```
printf( "\nSum of digits of %d = %d\n", n, sumOfDigits(n) );
```

6.9 रिटर्न स्टेटमेंट (THE RETURN STATEMENT)

रिटर्न स्टेटमेंटचा सिंटॅक्स आहे:

```
return;                or                return (exp);
```

जेथे `exp` एक कॉन्स्टन्ट, व्हेरिएबल किंवा एक्सप्रेशन असू शकते. `exp` आसपास कंसांचा वापर पर्यायी आहे. प्रथम फॉर्म रिटर्न टाईप व्हाईड म्हणून डिफाईड फंक्शन्ससह वापरला जातो.

रिटर्न स्टेटमेंट दोन उद्दीष्टे देते:

- रिटर्न स्टेटमेंटचा फंक्शनचे एक्झिक्युशन थांबवते आणि फंक्शनमधून कॉलिंग फंक्शनमध्ये नियंत्रण हस्तांतरित करते.
- जे काही रिटर्न स्टेटमेंटचे अनुसरण करीत आहे ते कॉलिंग फंक्शनला व्हॅल्यू म्हणून परत केले जाते.

रिटर्न स्टेटमेंट हे फंक्शनचे अंतिम विधान असू नये. हे फंक्शनमध्ये कुठेही वापरले जाऊ शकते. हे एक्झिक्युट होताच, नियंत्रण कॉलिंग फंक्शनमध्ये परत येईल. पुढे, फंक्शनमध्ये कितीही रिटर्न स्टेटमेंट असू शकतात.



There is key limitation of `return` statement — it can return only one value. If you want your function to return two or more values to the calling function, you have to use other means.

6.10 पाससिंग अर्ग्युमेंट्स टू फंक्शन (PASSING ARGUMENTS TO A FUNCTION)

फंक्शनमध्ये डेटा पाठविण्यासाठी वापरलेली यंत्रणा अर्ग्युमेंट यादीद्वारे असते, जिथे वैयक्तिक अर्ग्युमेंटना वास्तविक अर्ग्युमेंट म्हणतात. हे अर्ग्युमेंट फंक्शनच्या नावा नंतर कंसात बंद केलेले आहेत. वास्तविक अर्ग्युमेंट `number`, `type` आणि `order` फंक्शनच्या

डेफिनिशनमध्ये निर्दिष्ट औपचारिक अर्ग्युमेंट्सह क्रमवारीत असणे आवश्यक आहे. वास्तविक अर्ग्युमेंट `constants`, `variables`, `array names`, किंवा `expressions` असू शकतात.

फंक्शनकडे अर्ग्युमेंट पाठविण्यासाठी दोन पद्धती आहेत:

- मूल्यानुसार कॉल करा (Call by value)
- संदर्भाने कॉल करा (Call by reference)

आपण एक एक करून याचे वर्णन करूया.

6.10.1 मूल्यानुसार कॉल करा (Call by Value)

या दृष्टिकोनातून फंक्शन कॉलमध्ये वास्तविक अर्ग्युमेंट वापरले जातात. वास्तविक अर्ग्युमेंट एक `variable`, `constant`, किंवा `expression` असू शकतो.

जेव्हा फंक्शन कॉल केले जाते तेव्हा वास्तविक अर्ग्युमेंटची मूल्ये संबंधित औपचारिक अर्ग्युमेंटशी बदलली जातात आणि नंतर नियंत्रण कार्यामध्ये हस्तांतरित केले जाते.

फंक्शनचे लोकल व्हेरिएबल्स तयार केले जातात आणि नंतर त्या टास्कला फंक्शन म्हणून डिफाइन करणारे स्टेटमेंट्स कार्यान्वित केले जातात. जर कॉल केलेले फंक्शन मूल्य परत करायचे असेल तर ते रिटर्न स्टेटमेंटद्वारे परत केले जाईल.

मूल्य यंत्रणेद्वारे कॉल वापरून अर्ग्युमेंट पास करण्याबद्दल खालील मुद्दे लक्षात घेणे आवश्यक आहे:

1. वास्तविक अर्ग्युमेंट्स can be constants, variables, किंवा expressions असू शकतात.
2. जेव्हा कॉलिंग फंक्शनमधून कॉल फंक्शनवर नियंत्रण हस्तांतरित केले जाते, तेव्हा लोकल व्हेरिएबल्ससाठी मेमरी ऑलोकेट

केली जाते आणि फंक्शन बॉडीमधील स्टेटमेंट्स कार्यान्वित केली जातात.

2. कॉल केलेल्या फंक्शनची एक्झिक्युशन पूर्ण होताच, त्याच्या लोकल व्हेरिएबल्ससाठी दिलेली मेमरी डी - ऑलोकेट केली जाते आणि शेवटी नियंत्रण कॉलिंग फंक्शनमध्ये परत हस्तांतरित केले जाते.
3. औपचारिक अर्ग्युमेंट करण्यात आलेल्या कोणत्याही बदलाचा वास्तविक अर्ग्युमेंटवर परिणाम होणार नाही, कारण फंक्शन फक्त अर्ग्युमेंटची स्थानिक प्रत वापरत असेल.

खालील फंक्शन डेफिनिशन मूल्याद्वारे वितर्क पास करण्याची यंत्रणा स्पष्ट करते.

```
void swap( int a, int b )
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

वरील फंक्शन असे कॉलड केले जाते

```
swap(x, y);
```

जिथे x आणि y कॉलिंग फंक्शनमधील वास्तविक अर्ग्युमेंट आहेत.

Listing 6.1

```
/*
    Program to illustrate the passing of arguments by value.
    It calls a function swap() that swaps/interchanges
    values of arguments.
*/
#include <stdio.h>
void swap(int a, int b); /* function prototype */
int main()
{
    int x, y;
    printf("\nEnter value for x : ");
    scanf("%d", &x);
    printf("\nEnter value for y : ");
    scanf("%d", &y);
    printf("\nBefore calling swap function\n");
    printf("\nValue of x = %d, Value of y = %d\n", x, y);
    swap(x,y); /* function call */
}
```

```

    printf("\nAfter returning from swap function\n");
    printf("\nValue of x = %d, Value of y = %d\n", x, y);
    return 0;
}

void swap( int a, int b )
{
    int temp;
    printf("\nValues received from the main function\n");
    printf("\nValue of a = %d, Value of b = %d\n", a, b);
    temp = a;
    a = b;
    b = temp;
    printf("\nValues of local copy after swapping\n");
    printf("\nValue of a = %d, Value of b = %d\n", a, b);
}

```

Test Run

```

Enter value for x : 10
Enter value for y : 12
Before calling swap function
Value of x = 10, Value of y = 12
Values received from the main function
Value of a = 10, Value of b = 12
Values of local copy after swapping
Value of a = 12, Value of b = 10
After returning from swap function
Value of x = 10, Value of y = 12

```

तुम्ही पाहिले असेल की औपचारिक अर्ग्युमेण्ट करण्यात आलेल्या कोणत्याही बदलाचा वास्तविक अर्ग्युमेण्टवर परिणाम होत नाही.

6.10.2 संदर्भानुसार कॉल करा (CALL BY REFERENCE)

या दृष्टिकोनात, वास्तविक अर्ग्युमेण्टचे ॲड्रेस (पत्ता) फंक्शन कॉलमध्ये वापरले जातात. वास्तविक अर्ग्युमेण्ट केवळ व्हेरिएबल असू शकतात.

औपचारिक अर्ग्युमेण्ट वास्तविक अर्ग्युमेण्टच्या डेटा प्रकारांशी जुळणारे पॉइंटर्स म्हणून डिक्लेअर केले जातात.

जेव्हा फंक्शन कॉलड केले जाते तेव्हा वास्तविक अर्ग्युमेण्टचे ॲड्रेस (पत्ते) संबंधित औपचारिक अर्ग्युमेण्टकडे प्रतिस्थापित केले जातात जे पॉइंटर्स असतात आणि नंतर नियंत्रण फंक्शन कडे हस्तांतरित केले जाते.

फंक्शनचे लोकल व्हेरिएबल्स तयार केले जातात आणि त्या नंतर टास्कला फंक्शन म्हणून डिफाइन करणारे स्टेटमेंट्स कार्यान्वित केले जातात.

जर कॉल केलेले फंक्शनला मूल्य परत करायचे असेल तर ते रिटर्न स्टेटमेंटद्वारे परत केले जाईल.

संदर्भ यंत्रणेद्वारे कॉल वापरून अर्ग्युमेंट पास करण्याबद्दल खालील मुद्दे लक्षात घेणे आवश्यक आहे:

1. वास्तविक अर्ग्युमेंट केवळ व्हेरिएबल असू शकतात..
2. जेव्हा कॉलिंग फंक्शनमधून कॉल्ड फंक्शनवर नियंत्रण हस्तांतरित केले जाते, तेव्हा लोकल व्हेरिएबल्ससाठी ऑलोकेट केली जाते आणि फंक्शन बॉडीमधील स्टेटमेंट्स कार्यान्वित केली जातात.
3. कॉल केलेल्या फंक्शनची एक्झिक्युशन पूर्ण होताच, त्याच्या लोकल व्हेरिएबल्ससाठी दिलेली मेमरी डी - ऑलोकेट केली जाते आणि शेवटी नियंत्रण कॉलिंग फंक्शनमध्ये परत हस्तांतरित केले जाते.
4. औपचारिक अर्ग्युमेंट करण्यात आलेल्या कोणत्याही बदलाचा त्वरित परिणाम वास्तविक अर्ग्युमेंट वर होईल, कारण फंक्शन पॉईंटर्सद्वारे वास्तविक अर्ग्युमेंटवर कार्य करेल.

संदर्भ (अॅड्रेस) द्वारे अर्ग्युमेंट पास करण्याची कार्यपद्धती खालील कार्ये दर्शविते.

```
void swap( int a, int b )
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

वरील फंक्शन असे कॉल्ड केले जाते.

```
swap(x, y);
```

जिथे x आणि y कॉलिंग फंक्शनमधील वास्तविक अर्ग्युमेंट आहेत.

Listing 6.2

```
/*
    Program to illustrate the passing of arguments by reference.
    It calls a function swap() that interchanges values of arguments.
*/

#include <stdio.h>
void swap(int *a, int *b); /* function prototype */

int main()
{
    int x, y;
    printf("\nEnter value for x : ");
```



```

scanf("%d", &x);
printf("\nEnter value for y : ");
scanf("%d", &y);
printf("\nBefore calling swap function\n");
printf("\nValue of x = %d, Value of y = %d\n", x, y);
swap(&x,&y); /* function call */
printf("\nAfter returning from swap function\n");
printf("\nValue of x = %d, Value of y = %d\n", x, y);
return 0;
}
void swap( int *a, int *b )
{
    int temp;
    printf("\nValues received from the main function\n");
    printf("\nValue of *a = %d, Value of *b = %d\n", *a, *b);
    temp = *a;
    *a = *b;
    *b = temp;
    printf("\nValues of local copy after swapping\n");
    printf("\nValue of *a = %d, Value of *b = %d\n", *a, *b);
}

```

Test Run

```

Enter value for x : 10
Enter value for y : 12
Before calling swap function
Value of x = 10, Value of y = 12
Values received from the main function
Value of *a = 10, Value of *b = 12
Values of local copy after swapping
Value of *a = 12, Value of *b = 10
After returning from swap function
Value of x = 12, Value of y = 10

```

आपण पाहिले असेल की औपचारिक अर्ग्युमेण्ट मधील कोणताही बदल वास्तविक अर्ग्युमेण्ट दिसून येतो.

संदर्भानुसार कॉलचा वापर करून अर्ग्युमेण्टचे आणखी एक उदाहरण पुढील प्रोग्राम आहे.

Listing 6.3

```

/* Program that calls function example() to find the average of
two numbers, largest & smallest of these numbers. The
average is returned via return statement while largest and
smallest numbers are returned via formal arguments

```

```
*/
#include <stdio.h>
/* function prototype */
float example(float x, float y, float *big, float *small);
int main()
{
    float x, y, avg, larger, smaller;
    printf("\nEnter value for x : ");
    scanf("%f", &x);
    printf("\nEnter value for y : ");
    scanf("%f", &y);
    avg = example(x,y,&larger,&smaller); /* function call */
    printf("\nAverage of %.2f and %.2f is %.2f\n", x, y, avg);
    printf("\nLarger number is %.2f\n", larger);
    printf("\nSmaller number is %.2f\n", smaller);
    return 0;
}
/*
    Function to compute the average of two numbers and find the
    largest and smallest of these numbers
*/
float example(float x, float y, float *big, float *small)
{
    float average;
    average = ( x + y ) / 2;
    if ( x > y ) {
        *big = x;
        *small = y;
    } else {
        *big = y;
        *small = x;
    }
    return average;
}
```

Test Run

```
Enter value for x : 10.5
Enter value for y : 12.5
Average of 10.50 and 12.50 is 11.50
Larger number is 12.50
Smaller number is 10.50
```

6.10.3 मूल्यानुसार कॉल आणि संदर्भ द्वारे कॉल यांच्यात तुलना (Comparison between Call by Value and Call by Reference)

Table 6.1 प्रोटोटाइप, डेफिनिशन आणि फंक्शन कॉल संबंधित भिन्नतेचे मुख्य मुद्दे अधोरेखित करते.

Table 6.1 मूल्यानुसार कॉल आणि संदर्भ द्वारे कॉल यांच्यात तुलना -1

Call by Value	Call by Reference
Function Prototype: void swap(int a, int b);	Function Prototype: void swap(int *a, int *b);
Function Definition: void swap(int a, int b) { int temp; temp = a; a = b; b = temp; }	Function Definition: void swap(int *a, int *b) { int temp; temp = *a; *a = *b; *b = temp; }
Function Call: swap(x, y);	Function Call: swap(&x, &y);

Table 6.2 वास्तविक अर्ग्युमेण्टचे स्वरूप, औपचारिक अर्ग्युमेण्टचे स्वरूप आणि वास्तविक अर्ग्युमेण्टवर औपचारिक अर्ग्युमेण्टतील कोणत्याही बदलांचा / बदलाचा होणारा परिणाम या संदर्भातील मतभेदांचे मुख्य मुद्दे अधोरेखित करतात.

Table 6.2 मूल्यानुसार कॉल आणि संदर्भ द्वारे कॉल यांच्यात तुलना -2

Call by Value	Call by Reference
Actual arguments can be constants, variables, or expressions.	Actual arguments can only be variables.
Formal arguments are ordinary variables.	Formal arguments are pointer variables.
Values of actual arguments are substituted in formal arguments.	Addresses of the actual arguments are substituted in formal arguments.
Any change made to formal arguments will have no effect on actual arguments, since the function will only be using the local copy of the arguments.	Any change made to pointer variables will have immediate effect on actual arguments, since the function will be working on actual arguments through address.

6.10.4 अर्ग्युमेण्ट म्हणून वन डायमेन्शनल अरे पास करणे (Passing One-Dimensional Array as Argument)

लक्षात घ्या की अरेच्या सर्व घटकांची व्हॅल्यू फंक्शनकडे पाठविण्याऐवजी केवळ अरेचा अॅड्रेस पुरविला गेला आहे. तुम्हाला माहिती आहे त्याप्रमाणे C मध्ये अरेचे नाव base address आहे, म्हणजे अरेच्या पहिल्या घटकाचा अॅड्रेस.

```
/* main function */
#include <stdio.h>
void fun( int x[], int m );
void main()
{
    int a[10], n;
    /* other local declarations */

    func(a, n); /* function call */
    /* other statements */
}
```

```
/* function definition */
void fun( int x[], int m )
{
    /* local declarations */
    /* other statements */
}
```

चित्र 6.4: अर्ग्युमेण्ट म्हणून वन डायमेन्शनल अरे पास करणे

कारण अरेचे नाव खरं तर त्याचा अॅड्रेस आहे, म्हणून अरेचे नाव पुरवणे म्हणजे फंक्शनला कॉलिंग फंक्शनमध्ये अरेचा संदर्भ घेऊ देते.

Listing 6.4

```
/* Program to demonstrate the passing of an array as an argument */
#include <stdio.h>
int largest( int x[], int n );      /* function prototype */
int main(void)
{
    int a[10]={12,15,20,17,25,50,11,10,8,13};
    int i;
    printf( "Largest element of array = %d\n", largest(a,10));
    return 0;
}
/* function that returns largest element of the array */
int largest( const int x[], int n )
{
    int big, i;
    big = x[0];
```

```

    for ( i = 1; i < n; i++ ) {
        if ( x[i] > big )
            big = x[i];
    }
    return big;
}

```

Test Run

Largest element of array = 50

6.10.5 अर्ग्युमेण्ट म्हणून टू डायमेन्शनल अरे पास करणे (Passing Two-Dimensional Array as Arguments)

जेव्हा आपण फंक्शनमध्ये टू डायमेन्शनल अरे पास करतो तेव्हा आपण अरे चे नाव जसे आपण वन डायमेन्शनल अरे मध्ये वापरतो. कॉल केलेल्या फंक्शन हेडरमधील अर्ग्युमेण्ट यादीमधील औपचारिक अर्ग्युमेण्ट अरेचे टू डायमेन्शन असल्याचे दर्शविले जाणे आवश्यक आहे. हे टू डायमेन्शनसाठी एक कंसात दोन संच समाविष्ट केले जाते. प्रथम कंसातील जोडी रिक्त असू शकते परंतु दुसऱ्या जोडीच्या कंसात आपल्याला वास्तविक अरेच्या संबंधित डायमेन्शनच्या आकारमान समान आकार निर्दिष्ट करणे आवश्यक आहे. फंक्शन कॉल दरम्यान, अरेचा फक्त बेस अॅड्रेस पुरविला जातो.

```

/* main function */
#include <stdio.h>
void fun(int x[][4], int m, int n);
void main()
{
    int a[4][4];
    /* other local declarations */
    func(a,m,n); /* function call */
    /* other statements */
}

```

```

/* function definition */
void fun(int x[][4], int m, int n)
{
    /* local declarations */
    /* other statement */
}

```

चित्र 6.5: अर्ग्युमेण्ट म्हणून टू डायमेन्शनल अरे पास करणे

Listing 6.5

```

/*
    Program to add matrix A(mxn) and matrix B(mxn).
    This program uses function to read, add and display matrices.
*/
#include<stdio.h>
#define ROWS 10
#define COLS 10
/* function declarations */

```

```
void readMatrix(int a[][COLS], int m, int n);
void printMatrix(int a[][COLS], int m, int n);
void addMatrices(int a[][COLS], int b[][COLS], int c[][COLS],
                 int m, int n);

int main()
{
    int a[ROWS][COLS], b[ROWS][COLS], c[ROWS][COLS];
    int n, m;
    char ch;
    printf( "Enter size of matrices as mxn: " );
    scanf( "%d%c%d", &m, &ch, &n);
    readMatrix(a,m,n);          /* input matrix A(mxn) */
    readMatrix(b,m,n);          /* input matrix B(mxn) */
    addMatrices(a,b,c,m,n);     /* add matrix A(mxn) & matrix B(mxn) */
    printf( "\nSum A+B is\n\n" );
    printMatrix(c,m,n);         /* output matrix C(mxn) */
    return 0;
}

/* function that reads a matrix A(mxn) */
void readMatrix(int a[][COLS], int m, int n)
{
    int i, j;
    printf("\nEnter %d elements of matrix A row-wise\n", m*n );
    for ( i = 0 ; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
            scanf( "%d", &a[i][j] );
        }
    }
}

/* function that displays a matrix 'a' of order mxn */
void printMatrix(int a[][COLS], int m, int n)
{
    int i, j;
    for ( i = 0 ; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
            printf( "%4d", a[i][j] );
        }
        printf ( "\n" );
    }
}
```

```

/*
    function that adds A(mxn) & b(mxn), and
    stores the sum in matrix c(mxn)
*/
void addMatrices(int a[][COLS], int b[][COLS], int c[][COLS],
                  int m, int n)
{
    int i, j;
    for ( i = 0 ; i < m; i++ )
    {
        for ( j = 0 ; j < n; j++ )
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}

```

Test Run

```

Enter size of matrices as mxn : 3x3
Enter 9 elements of matrix A row-wise
1 3 2
4 5 1
6 5 8
Enter 9 elements of matrix A row-wise
7 3 5
2 7 3
4 2 1
Sum A+B is
  8   6   7
  6  12   4
 10   7   9

```

6.10.6 अर्ग्युमेण्ट म्हणून स्ट्रिंग फंक्शनला पास करणे (Passing String as Arguments)

अॅरिसाठी सबस्क्रिप्ट नोटेशन वापरून फंक्शनच्या अर्ग्युमेण्ट म्हणून स्ट्रिंग पुरविली जाऊ शकते. फंक्शनच्या डेफिनिशन मध्ये औपचारिक पॅरामीटर अक्षराच्या अॅर म्हणून डिक्लेअर केले जाते. पुढील प्रोग्राम फंक्शनला स्ट्रिंग पुरवणे स्पष्ट करते.

Listing 6.6

```

/* Program to illustrates passing of a string to a function */
#include<stdio.h>
#include<string.h>
void fun( char temp[] );    /* function prototype */
int main()
{
    char str[] = "Sample string";
    fun(str);
    return 0;
}
void fun( char temp[] )
{
    printf( "String passed to fun : " );
    puts( temp );
    printf( "and its length is : %d\n", strlen(temp) );
}

```

Test Run

```

String passed to fun : Sample string
and its length is : 13

```

स्पष्टीकरणात्मक उदाहरणे

फंक्शनची आणखी काही उदाहरणे विचारत घेऊ या.

Example 6.4: सकारात्मक इन्टिजर संख्येचा क्रमगुणित शोधण्यासाठी फंक्शन लिहा, int factorial (int n) म्हणा. फॅक्टोरियल फंक्शनची ही व्याख्या वापरून, Binomial coefficient म्हणून परिभाषित करण्यासाठी प्रोग्राम लिहा

The Binomial coefficient nC_r defined as

$${}^nC_r = \frac{n!}{r!(n-r)!}$$

Listing 6.7

```

/* Program to compute Binomial coefficient */
#include <stdio.h>
int factorial(int n);    /* function prototype */
int main()

```



```
{
    int ncr, n, r;
    printf( "\nEnter value of n: " );
    scanf( "%d", &n );
    printf( "\nEnter value of r: " );
    scanf( "%d", &r );
    ncr = factorial(n)/(factorial(r)*factorial(n-r));
    printf( "\nValue of ncr = %d\n" , ncr);
    return 0;
}
/* Function to compute factorial of a +ve integer number */
int factorial(int n)
{
    int prod = 1, i;
    for ( i = 1; i <= n; i++ )
        prod = prod * i;
    return prod;
}
```

Test Run

```
Enter value of n: 5
Enter value of r: 3
Value of ncr = 10
```

हा प्रोग्राम अर्ग्युमेण्ट n , r , $(n-r)$ सह तीनदा फॅक्टोरियल फंक्शन कॉल करतो आणि नंतर या व्हॅल्यूजचा वापर करून binomial coefficient मोजतो.

Example 6.5: एखादा फंक्शन लिहा, म्हणजे *int computeHCF(int m, int n)* म्हणा, जे m & n चा HCF रिटर्न करेल. या डेफिनिशनचा वापर करून m आणि n दिलेल्या दोन positive integers चा HCF गणना करण्यासाठी एक संपूर्ण प्रोग्राम लिहा.

Listing 6.8

```
/*
    Program to compute HCF of two given positive integers (m and n)
*/
#include <stdio.h>
int computeHCF( int m, int n );    /* function prototype */
int main()
{
    int m, n, hcf;
    printf( "\nEnter value of n: " );
    scanf( "%d", &n );
```

```

    printf( "\nEnter value of m: " );
    scanf( "%d", &m );
    printf( "\nEnter value of n: " );
    scanf( "%d", &n );
    hcf = computeHCF(m,n);
    printf( "\nValue of HCF = %d\n" , hcf);
    return 0;
}
/* Function to compute HCF of two positive integer numbers */
int computeHCF( int m, int n )
{
    int r;
    while ( 1 )
    {
        r = m % n;
        if ( r == 0 )
            return n;
        m = n;
        n = r;
    }
}

```

Test Run

```

Enter value of m: 125
Enter value of n: 35
Value of HCF = 5

```

Examples 6.6: एखादा फंक्शन लिहा, म्हणजे int smallestDigit (int n),) म्हणा, जे n मधील सर्वात लहान अंक मिळवते. प्रोग्राममध्ये हे फंक्शन त्याचा वापर दर्शविण्यासाठी वापरा.

Listing 6.9

```

/*
    Program to find smallest digit in a positive integer number
*/
#include <stdio.h>
int smallestDigit(int n); /* function prototype */

int main()
{
    int n;
    printf( "\nEnter positive integer number : " );
    scanf( "%d", &n );

```

```

        printf( "\nSmallest digit in %d is %d\n", n,
smallestDigit(n));
        return 0;
    }
    /* function that returns smallest digit in a positive integer number
    */
    int smallestDigit(int n)
    {
        int sd = 9;
        int d;
        while ( n > 0 )
        {
            d = n % 10;
            if ( d < sd )
                sd = d;
            n = n / 10;
        }
        return sd;
    }

```

Test Run

```

Enter positive integer number : 23145
Smallest digit in 23145 is 1

```

Examples 6.7: एखादा फंक्शन लिहा, *int isPrime(int n)*, म्हणा, जर *n* हा प्राइम नंबर असेल तर व्हॅल्यू 1 परत करेल अन्यथा 0 मिळेल. दिलेली नॅचरल नंबर प्राइम नंबर आहे की नाही याची चाचणी घेण्यासाठी प्रोग्राममध्ये हे फंक्शन वापरा.

Listing 6.10

```

/*
    Program to test whether the given natural number is
    prime number or not.
*/
#include<stdio.h>
#include<math.h>
int isPrime(int n); /* function prototype */
int main()
{
    int n;
    printf( "\nEnter a positive integer number: " );
    scanf("%d", &n);
    if ( isPrime(n) == 1 )
        printf( "\n%d is a prime number.\n", n );
    else
        printf( "\n%d is not a prime number.\n", n );
    return 0;
}

```

```

/*
    definition of function that tests whether the given positive
    integer number 'n' is prime number
*/
int isPrime(int n)
{
    int k, m;
    if ( ( n > 2 ) && ( ( n % 2 ) == 0 ) ) {
        return 0;
    }
    m = sqrt( n );
    for ( k = 3; k <= m; k += 2 )
    {
        if ( n % k == 0 )
        {
            return 0;
        }
    }
    return 1;
}

```

Test Runs

First Run

Enter a positive integer number: 43
43 is a prime number.

Second Run

Enter a positive integer number: 92
92 is not a prime number.

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- ❑ सॉफ्टवेअरच्या आकारामुळे आणि जटिलतेमुळे उद्भवणाऱ्या बऱ्याच समस्या मॉड्यूलर डिझाइन पध्दतीचा वापर करून कार्यक्षमतेने हाताळल्या जाऊ शकतात.
- ❑ प्रत्येक फंक्शन स्वतः मध्ये पूर्ण आणि स्वतंत्र आहे..
- ❑ प्रत्येक फंक्शन जटिल समस्येचे एक पैलू हाताळते..
- ❑ रिटर्न स्टेटमेंट वापरून फंक्शनमधून मूल्य परत मिळू शकते.
- ❑ फंक्शन कोणत्याही प्रकारचे मूल्य परत देऊ शकते. यात पॉइंटर्स देखील समाविष्ट आहेत.
- ❑ मूल्यांकनाद्वारे किंवा संदर्भ पध्दतीद्वारे कॉलद्वारे अर्ग्युमेण्ट पास केले जाऊ शकतात..

- व्हेरिएबल डिक्लरेशन प्रमाणे फंक्शन प्रोटोटाइप ही एक डिक्लरेशन असते जी रिटर्न type, name आणि type of formal arguments करते.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. युझर- डिफाईड फंक्शन आणि लायब्ररी फंक्शनमध्ये काय फरक आहे?
2. फंक्शनचे एक्झिक्युशन कधी बंद होते?
3. एखाद्या फंक्शनला किती अर्ग्युमेण्ट पास करता येतील?
4. औपचारिक अर्ग्युमेण्ट आणि वास्तविक अर्ग्युमेण्ट यांच्यातील संबंधांचे काय नियम आहेत? ?
5. फंक्शन कसे सुरू केले जाते?
6. एखादा फंक्शन किती वेळा कॉल केला जाऊ शकतो?
7. खालील फंक्शन डेफिनिशन मध्ये काय चुकीचे आहे?

```
testFunction( int k ) {
    float temp = 5.25;
    temp = k / 2.0;
    return temp;
}
```

8. खालील फंक्शन डेफिनिशन मध्ये लुटी पहा, काही असल्यास

```
void fun( int x, int y ) {
    int z;
    /* some statement */
    return z;
}
```

9. खालील फंक्शन डेफिनिशन मध्ये लुटी पहा, काही असल्यास

```
void fun( int x, y ) {
    int z;
    /* some statement */
    return;
}
```

10. खालील फंक्शन डेफिनिशन मध्ये लुटी पहा, काही असल्यास

```
int fun1( int x, int y ) {
    int z;
    /* some statement */
    int fun2( int t ) {
        return (t-2);
    }
}
```

```

    }
    /* some more statements */
    return z;
}

```

11. पुढील प्रोग्रामचे आउटपुट काय असेल?

```

void main() {
    float r = 5.0, c;
    float func( float t ); /* function prototype */
    c = func( r );
    printf( "\nValue returned by function" );
    printf( " = %d\n", c );
}

float func( float t ) {
    float temp;
    temp = t * t / 2;
    return 5.0;
}

```

12. खालील फंक्शन डेफिनिशन मध्ये त्रुटी पहा, काही असल्यास

- (a) `int diff(int x, y);` (b) `void fun(void, void);`
 (c) `float diff(float, float);` (d) `int test(float x, int y)`

प्रोग्रामिंग समस्या

1. फंक्शन लिहा, `int sumOfDigits(int number)` म्हणा, जो positive integer संख्येच्या अंकांची बेरीज रिटर्न करतो.
2. फंक्शन लिहा, `float absValue(float value)` म्हणा, जो असंख्य प्रकारच्या प्लोटचे *absolute* मूल्य रिटर्न करतो.
3. फंक्शन लिहा, `int round(float x)` म्हणा, जे rounded value x चे nearest integer मूल्य रिटर्न करते.
4. फंक्शन लिहा, `int sumOfN(int n, int m)` म्हणा, जे mth integer, सह प्रारंभ होणाऱ्या n integer ची बेरीज रिटर्न करते. i.e., $m + (m + 1) + (m + 2) + \dots + (m + n - 1)$.
5. फंक्शन लिहा, म्हणजे `int isPrime(int n)` म्हणा, जर संख्या n प्राइम नंबर तर 1 रिटर्न करा अन्यथा 0.
6. पुढे किंवा मागे वाचताना एखादी संख्या समान असते तर ती पॅलिंड्रोम असते. उदाहरणार्थ, संख्या 1991, 1001 आणि 1221 पॅलिंड्रोम आहे. फंक्शन लिहा, `int isPalindrome(unsigned int k)` म्हणा, जर k पॅलिंड्रोम असेल तर व्हॅल्यू 1 रिटर्न करा अन्यथा व्हॅल्यू 0.
7. IJK एक positive integer number जर $I < J < K$. योग्य क्रमवार असल्याचे म्हटले जाते. उदाहरणार्थ, 138 क्रमांकास सुव्यवस्थित म्हटले जाते कारण संख्या (1, 3, 8) मधील अंक डावीकडून उजवीकडे वाढतात, म्हणजे $1 < 3 < 8$.

- क्रमांक 365 योग्य प्रकारे क्रमवारीत नाही कारण 6 पेक्षा 5 मोठे आहे. फंक्शन लिहा, `int isWellOrdered(unsigned int k)` म्हणा, जर `k` जर योग्य क्रमांकाची संख्या असेल तर मूल्य 1 परत करेल अन्यथा 0.
8. फंक्शन लिहा, `int largestDigit(int x)` म्हणा, जो `x` मधील सर्वात मोठा अंक परत करेल.
9. फंक्शन लिहा, `int unitDigit(int n)` म्हणा, जे अर्ग्युमेण्ट `n` मध्ये प्रतिनिधित्व केलेल्या संख्येचा अंक परत करेल.
10. फंक्शन लिहा, `int isLeapYear(int year)` म्हणा, जर अर्ग्युमेण्ट वर्ष लीप वर्षाचे प्रतिनिधित्व करत असेल तर मूल्य 1 परत करेल अन्यथा 0.
11. फंक्शन लिहा, `int isValidDate(int dd, int mm, int yyyy)` म्हणा, जर तारीख वैध असेल तर मूल्य 1 परत करेल अन्यथा 0.
12. फंक्शन लिहा, `int isTrianglePossible(int a, int b, int c)` म्हणा, जर `a`, `b` आणि `c` वापरून त्रिकोण पास केला जाऊ शकतो तर त्याचे मूल्य 1 परत करेल अन्यथा 0.
13. फंक्शन लिहा, `int countDigits(int n)` म्हणा, जे `n` मधील अंकांची संख्या मिळवते, म्हणजेच `n` संख्येचा आकार.

एकाधिक निवड प्रश्न

- फंक्शन्स बदल खालीलपैकी कोणते विधान चुकीचे आहे?
 - प्रोग्राम युनिटमध्ये एकापेक्षा जास्त फंक्शनला परवानगी आहे.
 - फंक्शन दुसऱ्या फंक्शनला कॉल करू शकते.
 - फंक्शन स्वतःला कॉल करू शकतो.
 - औपचारिक अर्ग्युमेण्ट यादीमध्ये कॉन्स्टन्ट स्थिर घटक उपस्थित होऊ शकतात.
- खालीलपैकी कोणते फंक्शन वापरण्यास वैध कारण नाही?
 - समान कोडची पुनरावृत्ती करण्यापेक्षा ते कमी मेमरी वापरतात.
 - ते विविध प्रोग्राम उपक्रम स्वतंत्र ठेवतात.
 - ते वेगाने रन होते.
 - ते प्रोग्रामच्या इतर भागांमधून व्हेरिएबल्स सुरक्षित ठेवतात
- फंक्शनचा डीफॉल्ट रिटर्न प्रकार म्हणजे काय?

(a) oid	(b) int
(c) float	(d) char
- प्रोग्राम एक्झिक्युशन ____ पासून सुरू होते.

(a) <code>main()</code> फंक्शन	(b) प्रथम डिफाइंड केलेले फंक्शन
(c) अंतिम डिफाइंड केलेले फंक्शन	(d) कंपाईलरवर अवलंबून आहे
- C भाषा अर्ग्युमेण्ट्स पास करण्यास अनुमती देते

(a) only call by value	(b) only call by reference
(c) both a & b	(d) depends on compiler
- पुढील प्रोग्रामचा विचार करा


```
swap(int i, int j)
{
    i = i + j;
```

```

        j = i - j;
        i = i - j;
    }
    void main()
    {
        int i = 5, j = 10;
        swap(i, j);
        printf( "\n%d, %d", i, j );
    }

```

जेव्हा आपण प्रोग्राम कंपाईल आणि रन करण्याचा प्रयत्न करतो तेव्हा काय होईल?

- प्रोग्राम कंपाईल करण्यात अयशस्वी होईल कारण main () आणि swap () फंक्शन्समध्ये समान व्हेरिएबल नावे वापरली जाऊ शकत नाहीत.
- प्रोग्राम कंपाईल करण्यात अयशस्वी होईल कारण स्वॅप फंक्शनचा रिटर्न प्रकार निर्दिष्ट केलेला नाही.
- प्रोग्राम देणारे आउटपुट 5,10 असे कंपाईल व एक्झिक्युट करेल.
- प्रोग्राम 10 आणि 5 म्हणून देण्याचे आउटपुट कंपाईल व एक्झिक्युट करेल.

7. योग्य विधान ओळखा

- प्रोग्राम मध्ये फंक्शन एकापेक्षा जास्त वेळा डिफाईंड केले जाऊ शकते.
- क फंक्शन दुसऱ्या फंक्शन डेफिनेशनमध्ये डिफाईंड केले जाऊ शकत नाही.
- वर्ग फंक्शन्स समान फाईलमध्ये असणे आवश्यक आहे.
- main फंक्शनमध्ये त्यांना कॉल केल्यानुसार क्रमाने फंक्शन दिसले पाहिजे.

8. खालील फंक्शन डेफिनेशन विचारात घ्या आणि योग्य विधान ओळखा

```

myFunction( a + b, c, d, 5 )
int a, b, c, d;
{
    int sum;
    sum = a + b + c + d + 5;
    return sum;
}

```

- फंक्शन कंपाईल करण्यात अयशस्वी होईल कारण औपचारिक अर्ग्युमेण्ट म्हणून एक्स्प्रेसनला परवानगी नाही.
- फंक्शन कंपाईल करण्यात कार्य अयशस्वी होईल कारण औपचारिक अर्ग्युमेण्ट म्हणून कॉन्स्टन्ट परवानगी नाही.
- फंक्शन कंपाईल करण्यात कार्य अयशस्वी होईल कारण फंक्शन चा रिटर्न प्रकार वगळला आहे.
- दोन्ही (a) आणि (b).

9. फंक्शन हेडरचा पुढीलपैकी कोणता भाग आहे?

- | | |
|------------------------|-------------------|
| (a) फंक्शन नाव | (b) रिटर्न प्रकार |
| (c) अर्ग्युमेण्ट लिस्ट | (d) वरील सर्व |

10. खालील पैकी कोणते पूर्ण फंक्शन आहे?

- (a) `int fun();`
- (b) `int fun(int x) { return x+1; }`
- (c) `void fun(int) { print("Hello"); }`
- (d) `void fun(x) { print("hello"); }`

ANSWERS																			
1.	(d)	2.	(c)	3.	(b)	4.	(a)	5.	(c)	6.	(c)	7.	(b)	8.	(d)	9.	(d)	10.	(b)

प्रॅक्टिकल

1. सर्वात मोठे घटक, सर्वात लहान घटक आणि n (≤ 50) घटकासह खाली असलेल्या युझर- डिफाईंड फंक्शन्सचा वापर करून a अरेच्या घटकांची सरासरी शोधण्यासाठी प्रोग्राम लिहा

- (a) अरेचा सर्वात मोठा घटक शोधण्यासाठी फंक्शन, `int findLargest(int a[], int n)`
- (b) रेचा सर्वात लहान घटक शोधण्यासाठी फंक्शन, `int findSmallest(int a[], int n)`
- (c) अरेच्या घटकांची सरासरी शोधण्यासाठी फंक्शन, `float findAverage(int a[], int n)`

Listing 6.11

```

/*
    Program to find the largest element, smallest element,
    and average of elements of array using functions
*/
#include<stdio.h>
/* function prototypes */
int findLargest(int a[], int n);
int findSmallest(int a[], int n);
float findAverage(int a[], int n);
int main()
{
    int a[50];
    int i, n;
    printf("\nEnter size of array n(<=50) : ");
    scanf("%d", &n);
    printf("\nEnter %d elements of array\n\n", n);
    for ( i = 0; i < n; i++ )
    {
        scanf("%d", &a[i]);
    }
}

```

```
printf("\nSmallest element    = %d", findSmallest(a,n));
printf("\nLargest element     = %d", findLargest(a,n));
printf("\nAverage of elements = %.2f", findAverage(a,n));
return 0;
}

int findLargest(int a[], int n)
{
    int i, max;
    max = a[0];
    for ( i = 1; i < n; i++ ) {
        if ( a[i] > max )
            max = a[i];
    }
    return max;
}

int findSmallest(int a[], int n)
{
    int i, min;
    min = a[0];
    for ( i = 1; i < n; i++ ) {
        if ( a[i] < min )
            min = a[i];
    }
    return min;
}

float findAverage(int a[], int n)
{
    int i, sum;
    float avg;
    sum = 0;
    for ( i = 0; i < n; i++ ) {
        sum = sum + a[i];
    }
    avg = (float)sum/n;
    return avg;
}
```

Test Run

```

Enter size of array n(<=50) : 10
Enter 10 elements of array
25 20 40 32 10 15 45 50 30 24
Smallest element      = 10
Largest element       = 50
Average of elements   = 29.10

```

2. फंक्शन लिहा, म्हणजे *int isPrime (int n)* म्हणा, जर संख्या *n* प्राइम नंबर तर 1 रिटर्न करा अन्यथात 0. रिकर्सनचा वापर करून प्रथम *n* प्राइम नंबर छापण्यासाठी प्रोग्राममध्ये हे फंक्शन वापरा.

Listing 6.12

```

/*
   Program to print first 'm' prime numbers using a function
*/
#include<stdio.h>
#include<math.h>
int isPrime(int n); /* function prototype */
int main()
{
    int m, num = 2, count = 0;
    printf( "\nEnter value for m : " );
    scanf("%d", &m);
    printf( "\nFirst %d prime number are\n\n", m );
    while ( count < m )
    {
        if ( isPrime(num) == 1 ) {
            count++;
            printf( "%d  ", num );
        }
        num++;
    }
    printf( "\n" );
    return 0;
}

```

```

/*
    definition of function that tests whether the given
    positive integer number 'n' is prime number
*/
int isPrime(int n)
{
    int k, m;
    if ( ( n > 2 ) && ( ( n % 2 ) == 0 ) )
    {
        return 0;
    }

    m = sqrt( n );
    for ( k = 3; k <= m; k += 2 )
    {
        if ( n % k == 0 )
        {
            return 0;
        }
    }
    return 1;
}

```

Test Run

```

Enter value for m : 10
First 10 prime number are
2 3 5 7 11 13 17 19 23 29

```

3. फंक्शन लिहा, int product(int a, int b) म्हणा, जे दोन a आणि b संख्येचे product मिळवते. दिलेल्या संख्येचे product शोधण्यासाठी प्रोग्राममध्ये हे फंक्शन वापरा.

Listing 6.13

```

*/
    Program to find product of two numbers using function
*/
#include <stdio.h>
/* function prototype */
int product(int a, int b);
int main()
{

```

```

    int m, n, result;
    printf("\nEnter first number : ");
    scanf("%d", &m);
    printf("\nEnter second number : ");
    scanf("%d", &n);
    result = product(m, n);
    printf("\nProduct of %d and %d = %d\n", m, n, result);
    return 0;
}
/* definition of functio that returns product of two numbers
*/
int product(int a, int b)
{
    int temp = 0;
    while (b != 0)
    {
        temp += a;
        b--;
    }
    return temp;
}

```

Test Run

```

Enter first number : 15
Enter second number : 12
Product of 15 and 12 = 180

```

आणखी माहिती

आपल्याला माहिती आहे की फंक्शन वापरणे कोडमध्ये सुलभ, डीबग करणे सोपे आणि सुधारित करणे सोपे असलेल्या मोठ्या आणि जटिल प्रोग्रामचे निराकरण करण्यासाठी प्रोग्राम तयार करण्यास सक्षम करते.

शिक्षकांनी विद्यार्थ्यांमध्ये फंक्शन्स वापरण्याचे फायदे आणि फंक्शन्सच्या विकासाशी संबंधित विविध पैलूंबद्दल समज विकसित करणे अपेक्षित आहे.

वास्तविक जीवनातील परिस्थितीतून उदाहरणे घेऊन आणि त्यांचे निराकरण करण्यासाठी C प्रोग्राम तयार करून शिक्षकांनी फंक्शनचा वापर दर्शविली पाहिजेत.

संदर्भ आणि सूचविलेले वाचन

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

7

रिकरशन

युनिट वैशिष्ट्ये

हे युनिट पुनरावृत्तीशी संबंधित विषयांवर चर्चा करते. रिकर्शन ही गणित आणि संगणक शास्त्रातील महत्वाच्या संकल्पनांपैकी एक आहे आणि वास्तविक जीवनातील अनेक समस्या सोडवण्यासाठी त्याचा मोठ्या प्रमाणावर वापर केला जातो. हे युनिट रिकर्शन चे विविध पैलू स्पष्ट करते आणि योग्य उदाहरणांसह त्यांचा वापर दर्शवते.

तर्कशास्त्र

वास्तविक जीवनातील परिस्थितींमध्ये बऱ्याच समस्यांचे पुनरावृत्ती समाधान तसेच रिकर्सिव्ह सोल्यूशन असू शकते. मग कोणास प्राधान्य द्यायचे हा महत्वाचा प्रश्न उद्भवतो. आपण रिकरशन वापरण्याचे मुख्य कारण म्हणजे अल्गोरिदमला बहुतेक लोकांना सहज समजल्या जाणाऱ्या शब्दांमध्ये सोपे करणे. येथे हे लक्षात घेणे महत्वाचे आहे की पुनरावृत्तीचा उद्देश (त्याचा फायदा करण्याऐवजी) आपला कोड वाचणे आणि तर्क करणे सोपे करणे हा आहे. तथापि, हे लक्षात घेणे आवश्यक आहे की पुनरावृत्ती ही एक कार्यप्रणाली म्हणून आपण आपल्या कोडचा अनुकूलित करण्यासाठी वापरू शकणारी यंत्रणा नाही - जर काही असेल तर; पुनरावृत्तीच्या लिहिलेल्या समतुल्य कार्याच्या तुलनेत याचा कार्यक्षमतेवर विपरीत परिणाम होऊ शकतो. लक्षात ठेवण्यासाठी, आपण असे म्हणू शकतो “रिकर्सिव्ह फंक्शन्स डेव्हलपरसाठी सुवाच्यता अनुकूल करतात; पुनरावृत्ती कार्ये संगणकासाठी कार्यप्रदर्शन अनुकूलित करतात.” जेव्हा आम्ही वास्तविक जीवनातील परिदृश्यासारखा एखादा कोड लिहिण्याचा प्रयत्न करतो तेव्हा रिकर्सनची उपयुक्तता लक्षात येते आणि कौतुक होऊ शकते. हे युनिट विद्यार्थ्यांना रिकर्सिव्ह फंक्शन्सचा वापर करून पुनरावृत्ती आणि C मध्ये त्याच्या अंमलबजावणीशी संबंधित विविध बाबी समजण्यास मदत करेल.

पूर्व-आवश्यकता

जरी, या कोर्ससाठी काही विशिष्ट आवश्यकता नाहीत. तथापि, खालील दिलेल्या मध्ये वाजवी ज्ञान हा एक अतिरिक्त फायदा होईल:

- कंडिशनल ब्रँचिंग (Condition branching)
- युझर- डिफाईंड फंक्शन (User-defined functions)
- अरे (Arrays)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U7-O1:	रिकर्सन संकल्पना स्पष्ट करा.
U7-O2:	बेस केस आणि रिकर्सिव्ह स्टेप ही संकल्पना समजावून सांगा.
U7-O3:	निवडलेल्या रिकर्सिव्ह अडचणींसाठी रिकर्सिव्ह फंक्शन्स डिफाईंड करणे आणि वापरणे दाखवा.

U7-O4:	क्विक सॉर्ट अल्गोरिदम स्पष्ट करा आणि अंमलात आणा.
U7-O5:	मर्ज सॉर्ट अल्गोरिदम स्पष्ट करा आणि अंमलात आणा
U7-O6:	वास्तविक जीवनातील अडचणी सोडवण्यासाठी रिकर्सिव्ह फंक्शनचा वापर करून मॉड्यूलर प्रोग्राम विकसित करा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 7 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U7-O1	1							
U7-O2	1							
U7-O3					2			
U7-O4							3	
U7-O5							3	
U8-O6							3	

7.1 ओळख (INTRODUCTION)

रिकर्सन ही प्रोग्राम्सच्या विकासाची एक शक्तिशाली संकल्पना आहे ज्यात एखाद्या समस्येचे निराकरण करण्यासाठी एखाद्या कार्यात स्वतःचा संदर्भ घेण्याची क्षमता असते. रिकर्सन नावाचे हे नियंत्रण तंत्र संगणक विज्ञानाची एक महत्वाची संकल्पना आहे आणि अशा अनेक समस्यांसाठी सोयीस्कर आहे ज्यांचे पुनरावृत्ती करणे, जसे की, व्हाइल, फॉर लूप – व्हाइल लूप्स वापरून निराकरण करणे कठीण आहे.

अनेक वास्तविक जीवनातील अडचणी सोडवण्यासाठी रिकर्सनचा मोठ्या प्रमाणात वापर केला जातो.

रिकर्सिव्ह फंक्शन्स जवळजवळ सर्व आधुनिक उच्च-स्तरीय प्रोग्रामिंग भाषांमध्ये थेट C/C++/C#/Java/Python मध्ये लागू केली जाऊ शकतात.

हे युनिट रिकर्सिव्ह फंक्शन्सची ओळख करून देते आणि विविध उदाहरणांद्वारे त्यांचा उपयोगिता स्पष्ट करते.

7.2 रिकर्सिव्ह फंक्शन्स (RECURSIVE FUNCTIONS)

रिकर्सिव्ह फंक्शन म्हणजे एक असे फंक्शन ज्याची डेफिनिशन स्वतःवर आधारित असते, म्हणजेच, जो स्वतःला कॉल करते.

रिकर्सिव्ह फंक्शन बेस केस आणि रिकर्सिव्ह स्टेपच्या संदर्भात डिफाइंड केले जाते.

बेस केस (Base Case): फंक्शनला दिलेल्या इनपुटसह परिणामांची त्वरित गणना केली जाते, म्हणजेच, तेथे अग्युमेण्ट (स) चे मूल्य आहे, ज्यासाठी फंक्शन स्वतःला कॉल करीत नाही.

दुसऱ्या शब्दांत, बेस केस म्हणजे “सोप्या” संभाव्य समस्येचे निराकरण.

1. संख्यांच्या यादीमध्ये जास्तीत जास्त मूल्य शोधण्याचे उदाहरण विचारात घ्या. यादीमधील कमाल मूल्य एकतर पहिली संख्या किंवा उर्वरित संख्यांमधील सर्वात मोठी असते.

समस्येचा मूळ मुद्दा असा आहे की यादीमध्ये फक्त एक नंबर होता आणि डेफिनिशन नुसार, जर तेथे फक्त एक संख्या असेल तर ती सर्वात मोठी आहे.

- रिकर्सिव्ह स्टेप (Recursive Step): फंक्शनला एक किंवा अधिक रिकर्सिव्ह कॉलच्या मदतीने निकाल मोजला जातो, परंतु अर्ग्युमेण्ट (स) साईझ मध्ये काही प्रमाणात घट झाली, म्हणजे बेस केसच्या जवळ.

संख्यांच्या यादीत जास्तीत जास्त मूल्य शोधण्याच्या वरील उदाहरणात, रिकर्सिव्ह पायरी म्हणजे उर्वरित संख्येच्या यादीमध्ये जास्तीत जास्त मूल्य शोधणे, म्हणजे, कमी आकाराची यादी (आधीच्या आकारापेक्षा 1 ने कमी).

रिकर्सिव्ह फंक्शन्सची काही ज्ञात उदाहरणे घेऊ या.

7.2.1 फॅक्टोरियल फंक्शन (Factorial Function)

$n!$ म्हणून लिहिलेल्या सकारात्मक इन्टिजर संख्येचे फॅक्टोरियल हे 1 ते n पर्यंतच्या इन्टिजरचे उत्पादन (प्रॉडक्ट) आहे:

$$n! = 1 \times 2 \times 3 \times \dots \times (n-2) \times (n-1) \times n \quad \text{where as} \quad 0! = 1$$

अशा प्रकारे, फॅक्टोरियल फंक्शनच्या पुनरावृत्ती आवृत्तीचे डेफिनिशन म्हणून लिहिले जाऊ शकते:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ \prod_{i=1}^n i & \text{if } n > 0 \end{cases}$$

वरील डेफिनिशन पासून, आपल्याकडे आहे:

$$0! = 1$$

$$1! = 1$$

$$2! = 1 \times 2 = 2$$

$$3! = 1 \times 2 \times 3 = 6$$

$$4! = 1 \times 2 \times 3 \times 4 = 24$$

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120 \quad \text{and so on.}$$

ते पहा

$$4! = 4 \times 3! = 4 \times 6 = 24 \quad \text{and} \quad 5! = 5 \times 4! = 5 \times 24 = 120$$

आणि प्रत्येक सकारात्मक इन्टिजर n साठी हे खरे आहे; म्हणजे

$$n! = n \times (n-1)!$$

अशा प्रकारे, फॅक्टोरियल फंक्शन म्हणून डिफाइंड केले जाऊ शकते.

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$n!$ ची ही डेफिनिशन रिकर्सिव्ह आहे, कारण जेव्हा तो वापरतो तेव्हा तो स्वतःचा संदर्भ घेतो $(n-1)!$.

Listing 7.1

```
/*
    Program to print the factorials of first 'n' natural numbers
    using recursive function
*/
```

```

#include <stdio.h>
int factorial( int n );      /* function prototype */
int main()
{
    int i, n;
    printf( "\nEnter value of n: " );
    scanf( "%d", &n );
    for ( i = 1; i <= n; i++ ) {
        printf( "\n%d! = %d", i, factorial(i) );
    }
    return 0;
}
/*
    Recursive function to compute factorial of a number
*/
int factorial( int n )
{
    if ( n == 0 )
        return 1;
    else
        return ( n * factorial( n - 1 ) );
}

```

Test Run

```

Enter value of n: 6
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720

```

7.2.2 फिबोनाची नंबरस (Fibonacci Numbers)

एक अतिशय महत्वाचा क्रम, फिबोनाची अनुक्रम, सहसा F_0, F_1, \dots, F_n , दर्शविला जातो, खालीलप्रमाणे आहे

0, 1, 1, 2, 3, 5, 8, 13, ...

म्हणजेच $F_0 = 0$ आणि $F_1 = 1$ आणि त्यानंतरचे पद हे आधीच्या दोन पदांची बेरीज आहेत उदाहरणार्थ, वरील अनुक्रमातील पुढील पद आहे.

$$8 + 13 = 21$$

फिबोनाची नंबरससाठी औपचारिक रिकर्सिव्ह डेफिनिशन F_n आहे :

$$fib(n) = \begin{cases} n & \text{if } n \leq 1 \\ fib(n-1) + fib(n-2) & \text{if } n > 1 \end{cases}$$

लक्षात घ्या की फिबोनाची नंबरसची रिकर्सिव्ह डेफिनिशन फॅक्टोरियल फंक्शनच्या रिकर्सिव्ह डेफिनिशनपेक्षा भिन्न आहे कारण ती स्वतःला दोनदा संदर्भित करते.

Listing 7.2

```
/*
    Program to print first 'n' terms of Fibonacci sequence
    using recursive functions
*/
#include <stdio.h>
int fib( int n );          /* function prototype */
int main()
{
    int i, m;
    printf( "\nEnter value of m: " );
    scanf( "%d", &m );
    for ( i = 0; i < m; i++ )
    {
        printf( "%d ", fib(i) );
    }
    return 0;
}

/*
    Recursive function to compute Fibonacci number 'n'
*/
int fib( int n )
{
    if ( n <= 1 )
        return n;
    else
        return ( fib(n-1) + fib(n-2) );
}
```

Test Run

```
Enter value of m: 8
0  1  1  2  3  5  8  13
```

7.2.3 अकर्मन फंक्शन (Ackermann Function)

M आणि n च्या सर्व नॉन-नेगेटिव्ह, अकर्मन फंक्शनची रिकर्सिव्ह डेफिनिशन केली आहे.

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } n = 0 \\ A(m-1, A(m,n-1)) & \text{Otherwise} \end{cases}$$

Listing 7.3

```
/*
    Program to compute Ackermann function
*/
#include <stdio.h>
int ackermann(int m, int n); /* function prototype */
int main()
{
    int m, n;
    printf( "\nEnter value of m : " );
    scanf( "%d", &m );
    printf( "\nEnter value of n : " );
    scanf( "%d", &n );
    printf( "\nA(%d,%d) = %d\n", m, n, ackermann(m,n) );
    return 0;
}
/* function definition to compute ackermann function A(m,n) */
int ackermann(int m, int n)
{
    if ( m == 0 )
        return (n+1);
    else if ( n == 0 )
        return ackermann(m-1, 1);
    else
        return ackermann(m-1, ackermann(m,n-1));
}
```

Test Run

```
Enter value of m : 1
Enter value of n : 3
A(1,3) = 5
```

7.3 क्विक सॉर्ट अल्गोरिदम (QUICK SORT ALGORITHM)

क्विकसॉर्ट एक क्रमवारी लावणारा अल्गोरिदम आहे जो विभाजन आणि ताब्यात घेणे (divide and conquers) या कल्पनेचा वापर करतो. या अल्गोरिदमला पिव्होट नावाचा घटक सापडतो, जो अर्रेला दोन भागांमध्ये अशा प्रकारे विभाजित करतो की डावीकडील सब अर्रेमधील घटक कमी असतात आणि उजव्या सब अर्रेमधील घटक विभाजन घटकापेक्षा जास्त असतात. मग या दोन सब अर्रेस स्वतंत्रपणे क्रमवारी लावल्या जातात. ही प्रक्रिया बेस केससह निसर्गात रिकर्सिव्ह आहे -अर्रे मधील घटकांची संख्या 1 पेक्षा जास्त नाही.

समजा व्हेरिएबल start प्रारंभ आणि end अर्रेच्या पहिल्या आणि शेवटच्या घटकाची अनुक्रमणिका दर्शवित असेल तर, क्विकॉर्ट म्हणून रिकर्सिव्हली डिफाईंड केले जाऊ असे शकते:

```
If ( start < end ) then
    Partition the array into two halves
    Quicksort the left half
    Quicksort the right half
Endif
```

पिव्होट घटक निवडण्याच्या मार्गावर अवलंबून, क्विकॉर्ट अल्गोरिदमची काही भिन्नता खालीलप्रमाणे आहेत:

- पिव्होट घटक म्हणून पहिला घटक (First element as the pivot)
- पिव्होट घटक म्हणून शेवटचा घटक (Last element as the pivot)
- पिव्होट घटक म्हणून रँडम घटक (Random element as the pivot)
- पिव्होट घटक म्हणून मध्यगा (Median as the pivot)

शेवटचा घटक पिव्होट याचे स्पष्टीकरण बघूया.

सुरू करण्यासाठी, आपण सेट केले:

```
pIndex = start
pivot = a[end]
```

आता, आपण इंडेक्स व्हेरिएबल i चा वापर करून $start$ पासून $(end-1)$ पुनरावृत्ती करतो आणि प्रत्येक पुनरावृत्तीमध्ये पुढील गोष्टी करतो:

```
If ( a[i] < pivot ) then
    Swap a[i] and a[pIndex]
    Increment pIndex
Endif
```

आणि शेवटी

```
swap a[pIndex] and a[end]
```

एकत्र ठेवून, खालील फंक्शन विभाजनाचे कार्य पूर्ण करते.

```
int partition(int a[], int start, int end)
{
    int i, temp;
    int pIndex = start;
    int pivot = a[end];
    for( i = start; i < end; i++ )
```

```

{
    if ( a[i] < pivot )
    {
        temp = a[i];
        a[i] = a[pIndex];
        a[pIndex] = temp;
        pIndex++;
    }
}
temp = a[end];
a[end] = a[pIndex];
a[pIndex] = temp;
return pIndex;
}

```

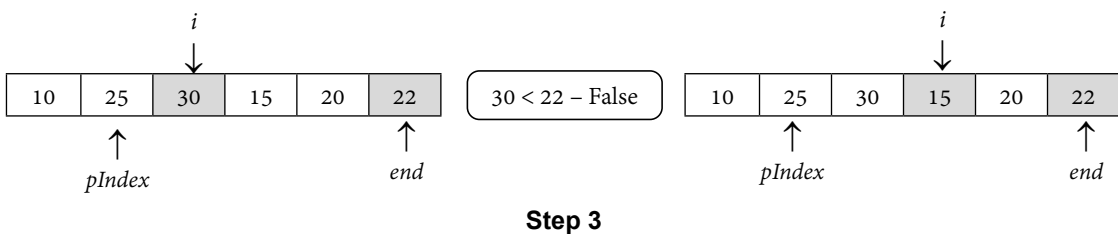
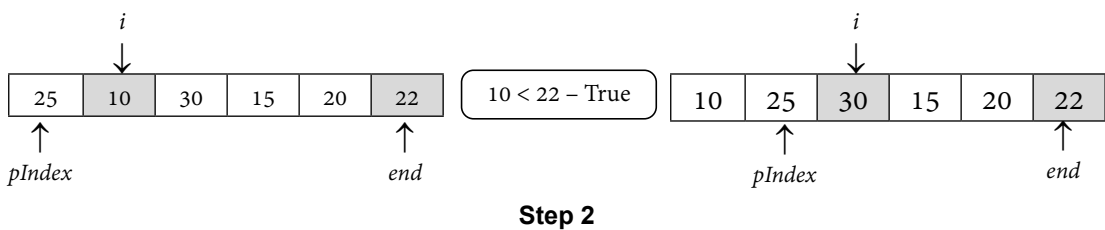
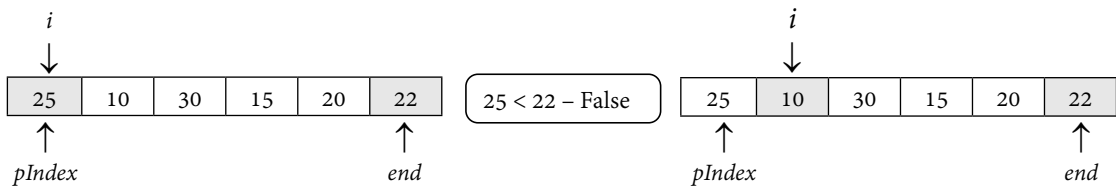
Example 7.1: विभाजन प्रक्रिया स्पष्ट करण्यासाठी, खालील अ‍ॅरचा विचार करा

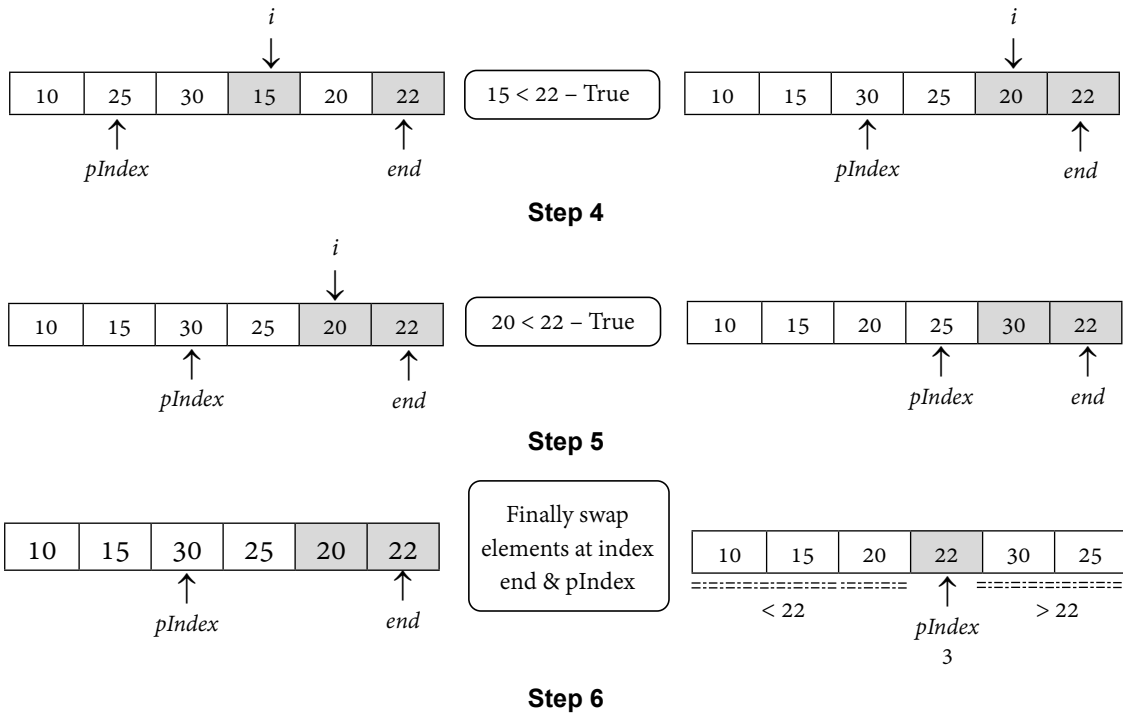
25 10 30 15 20 28

Solution: दिलेल्या अ‍ॅरमध्ये, start = 0, end = 5

येथे शेवटचा घटक (22) पिव्होट आहे.

सुरुवात करण्यासाठी, आपण घेतो $pIndex = start$, $i = start$





चित्र 7.1: अरेच्या विभाजनाचे उदाहरण

पिव्होट घटक 22 त्याच्या अंतिम स्थितीत ठेवला आहे आणि ते अरेला दोन उप अरे मध्ये विभाजित करते

10 15 15 and 30 25

तुम्ही बघू शकता की डाव्या सब अरे मधील घटक 22 पेक्षा लहान आहेत आणि उजव्या सब अरे मधील घटक 22 पेक्षा मोठे आहेत.

याचा अर्थ असा की घटक 22 योग्य प्रकारे त्याच्या अंतिम स्थितीत ठेवला आहे आणि उर्वरित घटक दोन उप अरेमध्ये विभाजित केले आहेत जेथे डावीकडील अरेमधील घटक त्यापेक्षा कमी आहेत आणि उजव्या सब अरेमधील घटक त्यापेक्षा जास्त आहेत.

वरील विभाजन पायरी 2 किंवा अधिक घटक असलेल्या प्रत्येक उप अरेसह पुनरावृत्ती होते. कारण आपण एकाच वेळी एका सब अरेवर काम करू शकतो, तर दुसऱ्या सब अरेचा मागोवा ठेवण्यात आपण सक्षम असले पाहिजे. हे कार्य एकतर स्टॅक (पुनरावृत्ती अंमलबजावणी) किंवा सुस्पष्टपणे (रिकर्सिव अंमलबजावणी) वापरून पूर्ण केले जाते. दोन्ही अंमलबजावणी या विभागात देण्यात आल्या आहेत.

Listing 7.4

```
/*
    Program to sort an array of integers in ascending order using
    Quick sort method
*/
#include<stdio.h>
/* function prototypes */
```

```

void quickSortRecursive( int a[], int lb, int ub );
int partition(int a[], int start, int end);
int main()
{
    int i, n, a[20];
    printf( "\nEnter size of array n(<=20) : " );
    scanf( "%d", &n );
    printf( "\nEnter %d integer elements of array\n\n", n );
    for ( i = 0; i < n; i++ )
        scanf( "%d", &a[i] );
    quickSortRecursive( a, 0, n-1 );
    printf( "\n\nSorted list of elements\n\n" );
    for ( i = 0; i < n; i++ )
        printf( "%d ", a[i] );
    printf( "\n" );
} /*---- end of main function ----*/
void quickSortRecursive( int a[], int lb, int ub )
{
    int pIndex;
    if ( lb < ub )
    {
        pIndex = partition( a, lb, ub);
        quickSortRecursive( a, lb, pIndex-1 );
        quickSortRecursive( a, pIndex+1, ub );
    }
}
int partition(int a[], int start, int end)
{
    int i, temp;
    int pIndex = start;
    int pivot  = a[end];
    for(i = start; i < end; i++)
    {
        if (a[i] < pivot)
        {
            temp = a[i];
            a[i] = a[pIndex];
            a[pIndex] = temp;
            pIndex++;
        }
    }
    temp = a[end];
    a[end] = a[pIndex];
    a[pIndex] = temp;

    return pIndex;
}

```


Test Run

```
Enter size of array n(<=20) : 6
Enter 6 integer elements of array
25 10 30 15 20 28
Sorted list of elements
10 15 20 25 28 30
```

7.4 मर्ज सॉर्ट अल्गोरिदम (MERGE SORT ALGORITHM)

मर्ज सॉर्ट ही आणखी एक सॉर्टिंग अल्गोरिदम आहे जी विभाजन आणि ताब्यात घेणे (divide and conquers) या कल्पनेचा वापर करते. हा अल्गोरिदम अरेला दोन भागांमध्ये विभागतो, त्यास स्वतंत्रपणे क्रमवारी लावतो आणि नंतर त्यांना मर्ज करतो.

ही प्रक्रिया रिकर्सिव्ह आहे, बेस केससह - अरे मधील घटकांची संख्या फक्त 1 आहे.

समजा व्हेरिएबल `beg` आणि `end` अनुक्रमे अरेच्या पहिल्या आणि शेवटच्या घटकाची अनुक्रमणिका दर्शवित असेल, तर मर्ज सॉर्ट म्हणून रिकर्सिव्हली डिफाइंड असे केले जाऊ शकते.

```
If ( beg < end ) then
    Divide the list into two halves
    Mergesort the left half
    Mergesort the right half
    Merge the two-sorted halves into one sorted list
Endif
```

Example 7.2: मर्ज सॉर्ट पध्दतीचे कार्य स्पष्ट करण्यासाठी, 7 घटकांसह खालील अरेचा विचार करा

33, 26, 35, 29, 18, 10, 24

Solution: मर्ज सॉर्टची पहिली पायरी म्हणजे अरेला दोन उप अरेमध्ये विभागणे. अशा प्रकारे आपण अरे मध्ये विभागू

33, 26, 35, 29 and 18, 10, 24

आणि प्रथम डाव्या सब अरेचा विचार करा. हे पुन्हा दोन उप अरेमध्ये विभागले गेले आहे

33, 26 and 35, 29

या प्रत्येक उप अरेसाठी, आपण पुन्हा तीच पद्धत लागू करतो, त्या प्रत्येकास एका घटकाच्या उप अरेमध्ये विभागतो. एक आकाराचे उप अरे अर्थातच वर्गीकरण आवश्यक नाही. शेवटी, आपण क्रमवारी लावलेले अरे मिळविण्यासाठी सब अरे मर्ज करण्यास सुरवात करतो.

क्रमवारी लावलेले अरे 26, 33 देण्यासाठी उप अरे 33 आणि 26 मर्ज करतात आणि उप अरे 35 आणि 29 मर्ज झाल्यास 29, 35 क्रमवारी लावते.

पुढील पायरीत, आकार चारची क्रमवारी लावलेली अरे मिळविण्यासाठी आपण आकार दोनचे हे दोन क्रमवारी लावलेले उप अरे मर्ज करतो

26, 29, 33, 35

आता दिलेल्या अरेचा डावा अर्धा क्रमवारी लावला आहे, तर आपण उजव्या अर्ध्या भागावर तसेच समान पायऱ्या करतो. प्रथम आपण त्यास दोन उप अरे मध्ये विभागले

18, 10 and 24

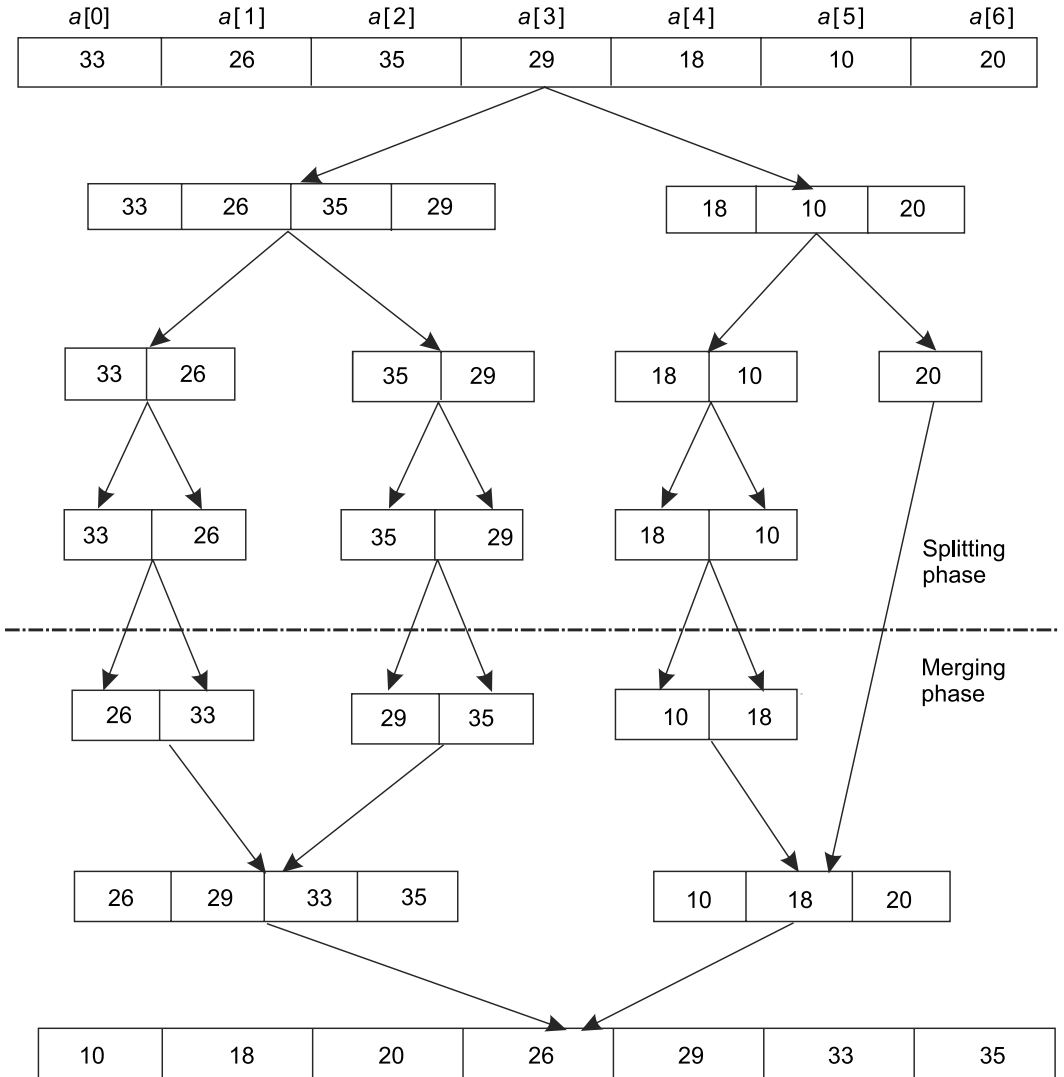
यापैकी प्रथम आकार एकच्या दोन उप अंशमध्ये विभागले गेले आहेत, जे 10, 18 देण्यासाठी मर्ज केले आहेत. दुसऱ्या सब अंश, 24, चा एक आकार आहे, म्हणून त्यास क्रमवारी लावण्याची गरज नाही. सॉर्ट केलेले अंश देण्यासाठी हे मर्ज केले गेले आहे.

10, 18, 24

शेवटी, आकार चार आणि तीन चे क्रमवारी लावलेले उप अंश देण्यासाठी मर्ज केले गेले

10, 18, 24, 26, 29, 33, 35

वरील सॉर्टिंग प्रक्रिया देखील चित्र 7.2. दर्शविल्यानुसार दृश्यमान केली जाऊ शकते.



चित्र 7.2: मर्ज सॉर्टच्या सलग पायऱ्यांचे स्पष्टीकरण

Listing 7.5

```

/*
    Program to sort an array of integers in ascending order using
    merge sort method
*/
#include<stdio.h>
/*----- function prototype -----*/
void mergeSortMethod(int a[], int beg, int end );
void mergingSortedSubArrays(int a[],int lb, int lr,int rb, int  rr);
int main()
{
    int i, n, a[20];
    printf( "\nEnter size of array n(<=20) : " );
    scanf( "%d", &n );
    printf( "\nEnter %d integer elements of array\n\n", n );
    for ( i = 0; i < n; i++ )
        scanf( "%d", &a[i] );
    mergeSortMethod( a, 0, n-1 );
    printf( "\n\nSorted list of elements\n\n" );
    for ( i = 0; i < n; i++ )
    {
        printf( "%d ", a[i] );
    }
    printf( "\n" );
} /*----- end of main function -----*/
void mergeSortMethod( int a[], int beg, int end )
{
    int mid;
    if ( beg < end )
    {
        mid = ( beg + end ) / 2;
        mergeSortMethod( a, beg, mid );
        mergeSortMethod( a, mid+1, end );
        mergingSortedSubArrays( a, beg, mid, mid+1, end );
    }
} void mergingSortedSubArrays(int a[], int lb, int lr, int rb, int rr)
{
    int na, nb, nc, k, c[MAX];

```

```

na = lb;
nb = rb;
nc = lb;
while ( ( na <= lr ) && ( nb <= rr ) )
{
    if ( a[na] < a[nb] )
        c[nc] = a[na++];
    else
        c[nc] = a[nb++];
    nc++;
}
if ( na > lr ) {
    while ( nb <= rr )
        c[nc++] = a[nb++];
} else {
    while ( na <= lr )
        c[nc++] = a[na++];
}
for ( k = lb; k <= rr; k++ )
    a[k] = c[k];
}

```

Test Run

```

Enter size of array n(<=20) : 7
Enter 7 integer elements of array
33 26 35 29 18 10 20
Sorted list of elements
10 18 20 26 29 33 35

```

7.5 रिकरशन, इट्रेशन किंवा? (RECURSION, ITERATION OR ...?)

जेव्हा समस्येचे निराकरण करण्याचे अनेक मार्ग आहेत, तेव्हा उद्भवणारा स्पष्ट प्रश्न “कोणाची निवड करावी?”

विशिष्ट मार्ग निवडण्यासाठी पूर्व-परिभाषित नियमांचा कोणताही सेट नाही; तथापि, आपण पुढील मुद्द्यांचा विचार केला पाहिजे:

- प्रक्रिया वेळ घेतला (Processing time taken.)
- संगणक मेमरी वापरली (Computer memory used.)
- प्रोग्राम विकसित करण्यासाठी लागणारा वेळ (Time taken to develop the program.)
- प्रोग्राम डीबग करण्यासाठी घेतलेला वेळ.(Time taken to debug the program.)
- प्रोग्राम सांभाळण्याची वेळ.(Time to maintain the program.)

सामान्यीकरण म्हणून रिकर्सिव्ह सोल्यूशन्स 3, 4 आणि 5 च्या श्रेणींमध्ये चांगले काम करतात. कारण रिकर्सिव्ह सोल्यूशन्स सोप्या आणि लहान असतात कारण नंतर प्रोग्राम्स सुधारित करण्यासाठी लागणारा वेळ कमी नसलेल्या सोल्यूशनसाठी आवश्यक वेळेपेक्षा कमी असतो. समान समस्या, जर ते अस्तित्वात असेल तर.

दुसरीकडे, सर्वसाधारणपणे, रिकर्सिव्ह सोल्यूशन्स त्यांच्या प्रक्रियेचा वेळ वापरण्यासाठी आणि त्यांना आवश्यक असलेल्या कॉम्प्युटर मेमरीच्या प्रमाणात चांगले कार्य करत नाहीत.

Table 7.1 रिकरशन आणि इट्रेशन तुलनेचे मुख्य मुद्दे हायलाइट करते.

Table 7.1: तुलना: रिकरशन आणि इट्रेशन

Criteria	Recursion	Iteration
Mode of Implementation	Using function call(s) to itself	Using loops
State	Defined by the argument value(s) stored in stack	Defined by the value of the control variable
Progression	Function state converges towards the base case	Value of control variable moves towards the final value
Termination	Base case is reached	Control variable satisfies the condition
No Termination State	Infinite recursive calls may occur due to some mistake in specifying the base case, and as a result the function keeps calling itself, which may lead system to crash.	Infinite loop due to mistake in assignment, increment, or terminating condition, and will result in program execute endlessly
Code size	Tends to be very small	Tends to be large
Execution	Execution is slower	Execution is faster

स्पष्टीकरणात्मक उदाहरणे

Example 7.3: रिकरशनचा वापर करून प्रथम एन नैसर्गिक संख्यांची बेरीज शोधण्यासाठी एक प्रोग्राम लिहा.

प्रथम एन नैसर्गिक संख्येची बेरीज शोधण्यासाठी फंक्शन, findSum(n) म्हणा, म्हणून रिकर्सिव्हली डिफाईंड असे केले जाऊ शकते.

$$\text{findSum}(n) = \begin{cases} 1 & \text{if } n = 1 \\ n + \text{findSum}(n-1) & \text{if } n > 1 \end{cases}$$

Listing 7.6

```
/*
    Program to find the sum of first 'n' natural numbers
    using recursion
*/
#include<stdio.h>
int findSum(int n); /* function prototype */
int main()
{
    int n, sum;
    printf("\nEnter value for n : ");
    scanf("%d", &n);
    sum = findSum(n);
```

```

printf("\nSum of first %d natural numbers = %d\n\n", n, sum);
return (0);
}
/*
    definition of recursive function that returns
    sum of first 'n' natural numbers
*/
int findSum(int n)
{
    if ( n == 1 )
        return 1;
    else
        return ( n + findSum(n-1) );
}

```

Test Run

```

Enter value for n : 10
Sum of first 10 natural numbers = 55

```

Example 7.4: रिकरशनचा वापर करून सकारात्मक घातांकीय (positive exponential) power (x^n) मोजण्यासाठी प्रोग्राम लिहा.

सकारात्मक घातांकीय(positive exponential) power (x^n) रिकर्सिव्हली डिफाईंड असे केले जाऊ शकते.

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x \times x^{n-1} & \text{if } n > 0 \end{cases}$$

Listing 7.7

```

/*
    Program to find the value of the positive exponential power
    function using recursion
*/
#include<stdio.h>
float power(float x, int n); /* function prototype */
int main()
{
    int n;
    float x;
    printf("\nEnter value for x : ");
    scanf("%f", &x);
    printf("\nEnter value for n : ");
    scanf("%d", &n);
}

```

```

printf("\nValue of power function = %.2f\n\n", power(n));
return (0);
}
/*
    definition of recursive function that returns value
    positive exponential power function (x^n)
*/
float power(float x, int n)
{
    if ( n == 0 )
        return 1;
    else
        return ( x * power(x,n-1) );
}

```

Test Run

```

Enter value for x : 5.25
Enter value for n : 2
Value of power function = 27.56

```

Example 7.5: रिकरशनचा वापर करून नैसर्गिक संख्येच्या m आणि n चा सर्वात सामान्य विभाजक (जीसीडी) शोधण्यासाठी प्रोग्राम लिहा.

GCD फंक्शन रिकर्सिव्हली डिफाईंड असे केले जाऊ शकते.

$$\text{gcd}(m,n) = \begin{cases} n & \text{if } m \% n = 0 \\ \text{gcd}(n, m \% n) & \text{if } m \% n \neq 0 \end{cases}$$

Listing 7.8

```

/*
    Program to compute GCD of natural numbers m and n
    using recursion
*/
int gcd( int m, int n );    /* function prototype */
void main()
{
    int m, n;
    printf( "\nEnter value of m : " );
    scanf( "%d", &m );
    printf( "\nEnter value of n : " );

```

```

scanf( "%d", &n );
printf( "\nValue of GCD(%d,%d) = %d\n", m, n, gcd(mn,n) );
}
/*
    definition of recursive function that returns GCD of
    natural numbers 'm' and 'n'
*/
int gcd( int m, int n )
{
    if ( m % n == 0 )
        return n;
    else
        return gcd( n, m % n );
}

```

Test Run

```

Enter value of m: 35
Enter value of n: 125
Value of GCD(35,125) = 5

```

Example 7.6: दिलेली दशांश क्रमांक (decimal number) n त्याच्या समतुल्य बायनरी क्रमांकात (equivalent binary number) रूपांतरित करण्यासाठी रिकरशनचा वापर करून प्रोग्राम लिहा.

Listing 7.9

```

/*
    Program to convert decimal number to equivalent binary number
    function using recursion
*/
#include<stdio.h>
void convert(int n); /* function prototype */
int main()
{
    int n;
    printf("\nEnter decimal number : ");
    scanf("%d", &n);
    printf("\nBinary equivalent of %d = ",n);
    convert(n);
    printf("\n");
    return (0);
}
/*

```



```

definition of recursive function that convert
decimal number to binary
*/
void convert(int n)
{
    if ( n > 0 )
    {
        convert(n/2);
        printf("%d", n%2);
    }
}

```

Test Run

```

Enter decimal number : 105
Binary equivalent of 105 = 1101001

```

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- ❑ रिकरशन ही प्रोग्राम्सच्या विकासाची एक शक्तिशाली संकल्पना आहे.
- ❑ एक फंक्शन जे स्वतःला कॉल करते त्याला रिकर्सिव्ह फंक्शन म्हणून ओळखले जाते.
- ❑ रिकर्सिव्ह फंक्शन्स थेट C मध्ये लागू केले जाऊ शकतात.
- ❑ रिकर्सिव्ह फंक्शन्सच्या लोकप्रिय उदाहरणांमध्ये फॅक्टोरियल आणि फिबोनाची क्रमांक समाविष्ट आहेत.
- ❑ क्लिक सॉर्ट एक क्रमवारी लावणारा अल्गोरिदम आहे जो विभाजन आणि ताब्यात घेणे (divide and conquers) या संकल्पनेचा वापर करतो. त्याला पिव्होट नावाचा घटक सापडतो, ज्यामुळे अरेला दोन भागांमध्ये अशा प्रकारे विभाजीत केले जाते की डावीकडील अरेमधील घटक कमी असतात आणि उजव्या सब अरेमधील घटक विभाजन घटकापेक्षा जास्त असतात. मग या दोन सब अरे स्वतंत्रपणे क्रमवारी लावल्या जातात.
- ❑ मर्ज सॉर्ट ही आणखी एक सॉर्टिंग अल्गोरिदम आहे जी विभाजन आणि ताब्यात घेणे (divide and conquers) या कल्पनेचा वापर करते. हा अल्गोरिदम अरेला दोन भागांमध्ये विभागतो, त्यास स्वतंत्रपणे क्रमवारी लावतो आणि नंतर त्यांना मर्ज करतो.
- ❑ रिकर्सिव्ह सोल्यूशन्स सामान्यतः प्रोग्राम विकसित आणि डीबग करण्यासाठी कमी वेळ घेतात आणि इटरेटिव्ह सोल्यूशनपेक्षा देखरेखीसाठी सोपी असतात.
- ❑ दुसऱ्या बाजूला, रिकर्सिव्ह सोल्यूशन्स प्रक्रियेस जास्त वेळ देतात आणि इटरेटिव्ह सोल्यूशन्सपेक्षा मेमरीचा जास्त वापर करतात.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

- रिकरशन म्हणजे काय?
- कधी रिकर्सिव्ह फंक्शन चांगल्या प्रकारे डिफाईंड केले जाते?
- बेस केस म्हणजे काय?
- रिकर्सिव्ह फंक्शन इटरेटिव फंक्शनशी कशा तुलना करता?
- रिकर्सिव्ह फंक्शनला इटरेटिव फंक्शनमध्ये रूपांतरित करणे शक्य आहे का? जर होय, तर एक उदाहरण द्या.
- इटरेटिव फंक्शनला रिकर्सिव्ह फंक्शनमध्ये रूपांतर करणे शक्य आहे का? जर होय, तर एक उदाहरण द्या.
- क्विक सॉर्ट अल्गोरिदमचे काम थोडक्यात सांगा.
- मर्ज सॉर्ट अल्गोरिदमचे काम थोडक्यात सांगा.
- रिकर्सिव्ह इटरेटिवची तुलना कशी करते?

एकाधिक निवड प्रश्न

- रिकरशन वापरून पुढीलपैकी कोणत्या समस्यांचे निराकरण करता येणार नाही?
 - Factorial of a number
 - n th fibonacci number
 - Length of a string
 - Problems without base case
- रिकरशन, ज्या स्थितीसाठी फंक्शन स्वतः कॉल करणे थांबवेल त्याची अट आहे _____.
 - बेस्ट केस
 - वस्ट केस
 - बेस केस
 - अशी कोणतीही स्थिती नाही
- खालीलपैकी कोणते विधान सत्य आहे?
 - रिकरशन नेहमीच ईट्रेशनपेक्षा चांगली असते
 - रिकरशन ईट्रेशनच्या तुलनेत अधिक मेमरी वापरते
 - रिकरशन ईट्रेशनच्या तुलनेत कमी मेमरी वापरते
 - रिकरशन पेक्षा ईट्रेशन नेहमीच चांगले आणि सोपे असते
- खाली कोड कार्यान्वित झाल्यावर काय होईल?

```
void fun()
{
    fun();
}
int main()
{
    fun();
    return 0;
}
```

- (a) कोड यशस्वीरित्या कार्यान्वित होईल आणि कोणतेही आउटपुट व्युत्पन्न केले जाणार नाही
- (b) कोड यशस्वीरित्या कार्यान्वित होईल आणि यादृच्छिक आउटपुट व्युत्पन्न केले जाईल
- (c) कोड एक कंपाईल टाइम बूटी दर्शवेल
- (d) कोड थोड्या काळासाठी चालेल आणि जेव्हा स्टॅक ओव्हरफ्लो होईल तेव्हा थांबेल

5. खालील कोडचे आउटपुट काय आहे?

```
void fun(int n)
{
    if(n == 0)
        return;
    printf("%d ", n);
    fun(n-1);
}

int main()
{
    fun(10);
    return 0;
}
```

- (a) 10 (b) 1 (c) 10 9 8 ... 1 0 (d) 10 9 8 ... 1

6. खालील कोडसाठी बेस केस काय आहे?

```
void fun(int n)
{
    if (n == 0)
        return;
    printf("%d ", n);
    fun(n-1);
}

int main()
{
    fun(10);
    return 0;
}
```

- (a) return (b) printf("%d ", n) (c) if(n == 0) (d) fun(n-1)

7. खालील कोडचे आउटपुट काय आहे?

```
#include<stdio.h>
int main()
{
    printf("Hello");
}
```

```
main();  
return 0;
```

```
}
```

(a) Hello is printed once

(b) Hello infinite number of times

(c) Hello is not printed at all

(d) 0 is returned

8. खालील कोडचे आउटपुट काय आहे?

```
fun(int x)  
{  
    int b;  
    if(x==1)  
        return 1;  
    else  
        b=x*fun(x-1);  
    return b;  
}
```

```
int main()  
{  
    int n;  
    n=fun(4);  
    printf("%d",n);  
    return 0;  
}
```

(a) 24

(b) 4

(c) 12

(d) 10

9. खालील कोडचे आउटपुट काय आहे?

```
int fun(int x)  
{  
    if(x==2)  
        return 2;  
    else  
    {  
        printf("+");  
        fun(x-1);  
    }  
}
```

```
int main()  
{
```

```

    int n,i;
    n=fun(6);
    printf("%d",n);
    return 0;
}

```

(a) ++++2

(b) +++++2

(c) ++++++

(d) 2

10. खालील C कोड कार्यान्वित केल्यावर किती वेळा 'a' प्रिंट होईल?

```

int fun(int b)
{
    if(b==0)
        return 0;
    else
    {
        printf("a");
        fun(b--);
    }
}

int main()
{
    int a;
    a=fun(10);
    printf("%d",a);
    return 0;
}

```

(a) 9 times

(b) 10 times

(c) 0 times

(d) Infinite number of times

11. खालील कोडचे आउटपुट काय आहे?

```

int f(int n)
{
    if(n>0)
        return(n+f(n-2));
}

main()
{
    int n=10;
    int f(int n);
    printf("%d",f(n));
}

```

(a) 10

(b) 80

(c) 30

(d) Error

12. $X = 4$ आणि $y = 5$ असल्यास खालील स्यूडोकोडचे आउटपुट शोधा:

```
int fun(int x, int y)
{
    if(x > 1)
        fun(x - 2, y + 2);
    printf("%d", y);
}
```

- (a) 56 (b) 765 (c) 975 (d) 579

13. $n = 2$ साठी खालील स्यूडोकोडचे आउटपुट काय असेल?

```
int fun ( int n)
{
    if ( n == 4 )
        return n;
    else
        return 2*fun(n+1);
}
```

- (a) 2 (b) 16 (c) 8 (d) 4

14. इनपुट $n = 134$ साठी खालील स्यूडोकोडचे आउटपुट किती असेल?

```
int fun (int n)
{
    static int a = 0;
    if ( n > 0 ) {
        a = a + 1;
        fun(n/10);
    }
    else
        return a;
}
```

- (a) 8 (b) 3 (c) 2 (d) 431

15. खालील कोडचे आउटपुट काय आहे?

```
int f( int x )
{
    if ( x <= 0 )
        return 1;
    return f(x-1) + x;
}

int main()
{
```

```
printf("%d", f(5) );
return 0;
}
```

- (a) 12 (b) 16 (c) 15 (d) 11

16. N = 25 साठी खालील फंक्शन काय प्रिंट करते?

```
void fun(int n)
{
    if (n == 0)
        return;
    printf("%d", n%2);
    fun(n/2);
}
```

- (a) 11001 (b) 11111 (c) 00000 (d) 10011

17. खालील C रिकर्सिव फंक्शन fun(x, y) लक्षात घ्या. fun(4, 3) चे मूल्य काय आहे?

```
int fun(int x, int y)
{
    if (x == 0)
        return y;
    return fun(x - 1, x + y);
}
```

- (a) 13 (b) 12 (c) 10 (d) 9

18. n = 25 साठी खालील फंक्शन काय प्रिंट करते?

```
void fun(int n)
{
    if (n == 0)
        return;
    fun(n/2);
    printf("%d", n%2);
}
```

- (a) 11001 (b) 11111 (c) 00000 (d) 10011

19. योग्य आउटपुट निवडा.

```
int rec(int num)
{
    return (num) ? num%10 + rec(num/10):0;
}
int main()
{
```

```

        printf("%d", rec(4567));
        return 0;
    }

```

- (a) 4 (b) 12 (c) 22 (d) 21

20. योग्य आउटपुट निवडा.

```

int doSomething(int a, int b)
{
    if (b==1)
        return a;
    else
        return a + doSomething(a,b-1);
}
int main()
{
    int k;
    k = doSomething(2,3);
    printf("%d", k);
    return 0;
}

```

- (a) 4 (b) 6 (c) 5 (d) 7

ANSWERS									
1.	(d)	2.	(c)	3.	(b)	4.	(d)	5.	(d)
6.	(c)	7.	(b)	8.	(a)	9.	(a)	10.	(d)
11.	(c)	12.	(c)	13.	(b)	14.	(b)	15.	(b)
16.	(d)	17.	(a)	18.	(a)	19.	(c)	20.	(b)

प्रोग्रामिंग समस्या

- एक रिकर्सिव्ह फंक्शन लिहा जे उलट क्रमाने प्रथम एन नैसर्गिक क्रमांकाचे मुद्रण करते. उदाहरणार्थ, $n = 10$ असल्यास आउटपुट 10 9 8 7 6 5 4 3 2 1 असावे.
- अरेचा सर्वात मोठा घटक शोधण्यासाठी रिकर्सिव्ह फंक्शन लिहा.
- अरेच्या घटकांचा क्रम उलट करण्यासाठी रिकर्सिव्ह फंक्शन लिहा.
- दोन नैसर्गिक संख्येचा सर्वात मोठा सामान्य विभाजक शोधण्यासाठी रिकर्सिव्ह फंक्शन लिहा.
- दोन नैसर्गिक संख्यांचे product शोधण्यासाठी रिकर्सिव्ह फंक्शन लिहा.
- नैसर्गिक संख्येचे फॅक्टोरियल शोधण्यासाठी रिकर्सिव्ह फंक्शन लिहा.
- नैसर्गिक संख्येच्या अंकांची बेरीज शोधण्यासाठी रिकर्सिव्ह फंक्शन लिहा.
- दशांश संख्येस (Decimal number) बायनरी क्रमांकावर रूपांतरित करण्यासाठी रिकर्सिव्ह फंक्शन लिहा.

प्रॅक्टिकल

- रिकर्सनचा वापर करून एक नैसर्गिक संख्या n ची फॅक्टोरियल शोधण्यासाठी प्रोग्राम लिहा.
Refer to Listing 7.1
- रिकर्सनचा वापर करून n ची फिबोनाची नंबर शोधण्यासाठी प्रोग्राम लिहा.
Refer to Listing 7.2
- रिकर्सनचा वापर करून दशांश संख्येला (Decimal number) त्याच्या समकक्ष बायनरी क्रमांकात रूपांतरित करण्यासाठी प्रोग्राम लिहा.
Refer to Listing 7.9

आणखी माहिती

रिकरशन हा विषय समस्या निराकरण करण्याचा आणखी एक महत्त्वाचा विषय आहे आणि बऱ्याच वास्तविक जीवनातील अडचणी सोडविण्यासाठी मोठ्या प्रमाणात वापरला जातो.

शिक्षकांनी रिकरशनच्या संकल्पनांबद्दल समजून घेणे आणि विद्यार्थ्यांच्या सहभागासह रिकर्सिव्ह फंक्शनचा वापर करून समस्येचे निराकरण करणे अपेक्षित आहे.

संदर्भ आणि सूचविलेले वाचन

- R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
- E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
- Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
- Byron Gottfried, Programming with C, Schaum's Outlines.
- https://onlinecourses.nptel.ac.in/noc21_cs01/preview
- <https://ocw.mit.edu/courses/intro-programming/>
- <https://www.programiz.com/c-programming>
- <https://www.javatpoint.com/c-programming-language-tutorial>

8

स्ट्रक्चर्स

युनिट वैशिष्ट्ये

हे युनिट स्ट्रक्चर्सशी संबंधित विषयांवर चर्चा करते. संरचनेचा वापर एखाद्या घटकाशी संबंधित डेटा साठवण्यासाठी केला जातो, जिथे एखादी संस्था कर्मचारी, ग्राहक, वाहन किंवा पुस्तक असू शकते. हे युनिट स्ट्रक्चर्सचे विविध पैलू स्पष्ट करते आणि त्यांचा वापर योग्य उदाहरणांसह दाखवते.

तर्कशास्त्र

वास्तविक डेटा अडचणी उद्भवू शकतात जिथे आपल्याला डेटा संकलनास सामोरे जावे लागू शकते जिथे वैयक्तिक डेटा आयटम वेगवेगळ्या डेटा प्रकारचे असू शकतात. उदाहरणार्थ, शैक्षणिक संस्थेत विद्यार्थ्यांची माहिती संग्रहित करण्याची आवश्यकता आहे ज्यात रोल नंबर, नाव, वडिलांचे नाव, आईचे नाव, जन्मतारीख, ईमेल-आयडी, मोबाइल नंबर, पत्ता, प्रत्येक सेमेस्टरचे प्रत्येक विषयाच्या गुणांचा तपशील असू शकतो, फीचे तपशील इ. हे डेटा आयटम भिन्न डेटा प्रकारांशी संबंधित आहेत जसे की इन्टिजर, लॉन्ग इन्टिजर, स्ट्रिंग, प्लोट इ. म्हणून, आम्हाला या प्रकारचा डेटा संग्रह एकल युनिट म्हणून ठेवण्यासाठी एक यंत्रणा आवश्यक आहे आणि या प्रश्नाचे उत्तर म्हणजे स्ट्रक्चर. हे युनिट विद्यार्थ्यांना संरचनेचे विविध पैलू समजून घेण्यास आणि वास्तविक जीवनातील अडचणी सोडविण्यासाठी स्ट्रक्चर्सचा वापर करून प्रोग्राम विकसित करण्यास मदत करेल.

पूर्व-आवश्यकता

- मूलभूत डेटा प्रकार (Basic data types)
- कंडिशनल ब्रँचिंग (Conditional Branching)
- लूपिंग (Looping)
- अरे आणि स्ट्रिंग्स (Arrays and strings)
- फंक्शन्स (Functions)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U8-O1:	स्ट्रक्चरची संकल्पना समजावून सांगा
U8-O2:	स्ट्रक्चर डिफाईनिंगचे वेगवेगळे मार्ग सांगा
U8-O3:	स्ट्रक्चर व्हेरिएबल्स डिक्लेअर आणि इनिशियलाइज करा
U8-O4:	स्ट्रक्चर्सचा अरे डिक्लेअर आणि इनिशियलाइज करा
U8-O5:	स्ट्रक्चर्सचा उपयोग करून वास्तविक जीवनातील समस्यांसाठी प्रोग्राम विकसित करा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 8 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U8-O1						1		
U8-O2						2		
U8-O3						2		
U8-O4						2		
U8-O5						3		

8.1 ओळख (INTRODUCTION)

आपण एखाद्या स्ट्रक्चरला जवळून संबंधित वस्तूचा अर्रे म्हणून विचार करू शकता. तथापि, अर्रेच्या विपरीत, स्ट्रक्चर ही संबंधित डेटा आयटमचा संग्रह आहे जी वेगवेगळ्या डेटा प्रकारांशी संबंधित असू शकतात. एखादी स्ट्रक्चर बनविणाऱ्या डेटा आयटम कन्टिग्युअस (contiguous) मेमरी स्थान व्यापतात, त्याच प्रकारे अर्रेचे घटक कन्टिग्युअस (contiguous) मेमरी ठिकाणी संग्रहित केले जातात.

जटिल डेटा अधिक अर्थपूर्ण मार्गाने आयोजित करण्यासाठी स्ट्रक्चर्सचा वापर केला जाऊ शकतो.

समजा आपल्याला कर्मचार्यांची माहिती एखाद्या संस्थेत संचयित करायची आहे. या माहितीमध्ये कर्मचार्यांचे नाव (कॅरेक्टर अर्रे), डिपार्टमेंट कोड (इन्टिजर क्रमांक), पगार (फ्लोटिंग पॉइंट नंबर) आणि इतर समाविष्ट असू शकते. आणि आपला प्रोग्राम त्यांच्याशी अर्रेच्या घटकांप्रमाणे व्यवहार करावा अशी आपली इच्छा आहे

एक संभाव्य दृष्टीकोन म्हणजे अनेक अर्रे वापरणे $\frac{3}{4}$ नावांसाठी कॅरेक्टर अर्रे, विभाग कोडसाठी इन्टिजर अर्रे, पगारासाठी एक फ्लोटिंग-पॉइंट अर्रे आणि पुढे. आणि सामान्य अनुक्रमणिका वापरून आपण एखाद्या विशिष्ट कर्मचा-याची माहिती मिळवू शकतो. तरीही हा दृष्टीकोन आपल्या एका उदाहरणामधील कर्मचारी, एकाच घटकाशी संबंधित डेटा आयटमवर व्यवहार करतो ही वस्तुस्थिती अस्पष्ट करते.

या प्रकारच्या समस्येचे निराकरण करण्यासाठी, C भाषा एक विशेष डेटा प्रकार प्रदान करते: स्ट्रक्चर. आधीच नमूद केल्याप्रमाणे, आपण एका युनिटमध्ये एकत्रित केलेल्या भिन्न डेटा प्रकारांच्या डेटा आयटमचा गट म्हणून एखाद्या संरचनेचा विचार करू शकता. आपल्या उदाहरणात, संरचनेत कर्मचार्यांचे नाव, विभाग क्रमांक, पगार इत्यादींचा समावेश असेल.

स्ट्रक्चरची आणखी काही उदाहरणे येथे आहेत

Name of Structure	Probable Constituent Data Items
Date	day, month, year
Time	hours, minutes, seconds
Book	title, author, publisher, price
Address	name, house number, locality, city, state, pincode
Student	roll number, name, father's name, grade
Customer	name, address, mobile no
Inventory	item code, description, quantity, unit price
Employee	employee id, name, address, mobile no

या युनिटमध्ये आपण स्ट्रक्चरच्या संबंधित विविध पैलूबद्दल शिकू या.

8.2 डिफाईनिंग स्ट्रक्चर (DEFINING A STRUCTURE)

C भाषेमध्ये स्ट्रक्चर डिफाइन करण्याचे दोन मार्ग आहेत: टॅग्ड स्ट्रक्चर आणि टाइप- डिफाईंड स्ट्रक्चर.

टॅग स्ट्रक्चर (Tagged Structure)

स्ट्रक्चर डिफाइन करण्याचा पहिला मार्ग म्हणजे टॅग स्ट्रक्चर वापरणे. टॅग स्ट्रक्चर डिक्लेअर करण्यासाठी सिंटॅक्स आहे:

```
struct STUDENT
{
    . . .
    Field list
    . . .
};
```

(a) Format

```
struct STUDENT
{
    int rollNo;
    char name[20];
    char fname[20];
    char grade;
};
```

(b) Example

चित्र 8.1: डिफाईनिंग टॅग स्ट्रक्चर

टॅग स्ट्रक्चर कीवर्ड struct पासून प्रारंभ होते. डेफिनिशन मधील पुढील घटक टॅग आहे. टॅग हा संरचनेचा आयडेंटिफायर आहे, आणि तो व्हेरिएबल, फंक्शन्सचे अर्ग्युमेण्ट आणि फंक्शन्ससाठी रिटर्न प्रकार डिक्लेअर करण्यासाठी वापरला जाईल.

क्लोजिंग कंसांनंतर जर आपण सेमिकोलोन देऊन स्ट्रक्चर डेफिनिशन पूर्ण केली तर कोणतेही व्हेरिएबल्स डिक्लेअर होणार नाहीत. या प्रकरणात, स्ट्रक्चर फक्त संबद्ध संचय(associated storage) नसलेली एक प्रकारची टेम्पलेट आहे.

टाइप- डिफाईंड स्ट्रक्चर (Type-defined Structure)

typedef, टाइप डेफिनेशन वापरून स्ट्रक्चरची व्याख्या करण्याचा अधिक शक्तिशाली मार्ग आहे. टाइप- डिफाईंड स्ट्रक्चर डिक्लेअर करण्यासाठी सिंटॅक्स आहे:

```
typedef struct
{
    . . .
    Field list
    . . .
} TYPE;
```

(a) Format

```
typedef struct
{
    int rollNo;
    char name[20];
    char fname[20];
    char grade;
} STUDENT;
```

(b) Example

चित्र 8.2: डिक्लेअरिंग टाइप-डिफाईंड स्ट्रक्चर

टाइप- डिफाईंड स्ट्रक्चर खालील बाबींमध्ये टॅग स्ट्रक्चरपेक्षा भिन्न आहे:

- *struct* कीवर्डच्या आधी *typedef* कीवर्ड वापरला जातो.

- टॅग स्ट्रक्चर बाबतीत struct कीवर्ड नंतर आयडेंटिफायर अनिवार्य नाही.
- क्लोजरिंग कंसांनंतर आणि सेमीकोलनपूर्वी आयडेंटिफायर आवश्यक आहे.
- आपण नंतर पाहू, व्हेरिएबल डिक्लेअर हे टॅग स्ट्रक्चर डिक्लेरेशन केले जाऊ शकते, परंतु ते टाइप-डिफाईन्ड स्ट्रक्चर्ससह असू शकत नाही.



A type definition, *typedef*, give a name to a data type by creating a new type that can be used anywhere a type is permitted. The syntax for type definition is

```
typedef dataType IDENTIFIER;
```

where *dataType* is either built-in data type or user-defined data type, and *IDENTIFIER*, usually in uppercase, is the new and convenient name for the *dataType*. The *typedef* keyword tells the compiler to recognize the *IDENTIFIER* as synonymous of *dataType*.

स्ट्रक्चर डेफिनिशनची आणखी काही उदाहरणे खाली देत आहेत:

```

struct Date
{
    int day;
    int month;
    int year;
};

struct Time
{
    int hours;
    int minutes;
    int seconds;
};

struct Book
{
    char title[25];
    char author[25];
    char publisher[15];
    float price;
};

struct Customer
{
    char name[25];
    char address[80];
    long mobile_no;
};

struct Inventory
{
    char item_code[25];
    char desc[25];
    int qty;
    float unit_price;
};

struct Address
{
    char name[25];
    int house_no;
    char locality[25];
    char city[15];
    char state[15];
    long pincode;
};

```

8.3 स्ट्रक्चर व्हेरिएबल्स डिक्लेअर करणे (DECLARING STRUCTURE VARIABLES)

एकदा स्ट्रक्चर डिफाइंड झाल्यावर आपण व्हेरिएबल्स डिक्लेअर करू शकतोसहसा, डेफिनिशन प्रोग्रामच्या ग्लोबल भागा मध्ये ठेवली जाते, म्हणजेच, `main ()` फंक्शनपूर्वी, जेणेकरून ती प्रोग्राममधील सर्व फंक्शन्ससाठी दृश्यमान असेल. दुसरीकडे, व्हेरिएबल्स सामान्यतः फंक्शन्सच्या लोकल भागात किंवा फंक्शन हेडर्स मधील अर्ग्युमेंट यादीमध्ये डिक्लेअर केले जातात. पुढील विधाने स्ट्रक्चर व्हेरिएबल्स डिक्लेअर करण्याचे मार्ग दर्शवितात.

टॅग स्ट्रक्चर साठी, व्हेरिएबल असे डिक्लेअर केले जाते.

```
struct STUDENT aStudent;
```

टाइप-डिफाईन्ड स्ट्रक्चरसाठी व्हेरिएबल असे डिक्लेअर केले जाते.

```
STUDENT bStudent;
```

दोन्ही डिक्लरेशनची तुलना करून, आपल्याला व्हेरिएबलच्या डिक्लरेशनमध्ये टाइप-डिफाईन्ड स्ट्रक्चर अधिक सुलभ असल्याचे आढळेल कारण आपल्याला struct कीवर्ड वापरण्याची आवश्यकता नाही. प्रोग्राममधील बऱ्याच ठिकाणी व्हेरिएबल डिक्लेअर करणे आवश्यक असल्यास हा बराच वेळ आणि प्रयत्न-बचत असू शकते.

पुढे लक्षात घ्या की टॅग स्ट्रक्चरच्या बाबतीत, व्हेरिएबलचे डिक्लरेशन खाली दर्शविल्यानुसार त्याच्या डेफिनिशन सह एकल केली जाऊ शकते.

```
struct EMPLOYEE
{
    int    code;
    char   name[25];
    char   department[15];
    float  salary;
} aEmployee, bEmployee;
```

सहसा, खालील कारणांसाठी व्हेरिएबल्स डिक्लेअर करण्याचा हा दृष्टीकोन वापरण्याची शिफारस केलेली नाही.

स्ट्रक्चर डेफिनिशन ग्लोबल भागात ठेवली गेली आहे आणि त्यामुळे व्हेरिएबल देखील ग्लोबल होतील.

जर आपल्याला हा दृष्टिकोन वापरून लोकल व्हेरिएबल डिक्लेअर करायचे असतील तर आपल्याला स्ट्रक्चर डेफिनिशन लोकल भागात ठेवावी लागेल, ज्यामुळे स्ट्रक्चर डेफिनिशन इतर फंक्शन मध्ये अदृश्य होईल.

8.4 इनिशियालाइजिंग स्ट्रक्चर (INITIALIZING STRUCTURES)

व्हेरिएबल्स आणि अरे प्रमाणेच स्ट्रक्चर व्हेरिएबल्स देखील डिक्लरेशनच्या वेळी इनिशियालाइज करता येतात. वापरलेले स्वरूप अरे इनिशियालाइज करण्यासाठी वापरले जाण्यासारखेच आहे.

पुढील विभाग हे स्पष्ट करतो

```
typedef struct
{
    int    rollno;
    char   name[25];
    int    age;
    float  height;
} STUDENT;

STUDENT aStudent = { 1000, "Surbhi", 18, 5.6 };
```

8.5 स्ट्रक्चर एलिमेंट्स ऍक्सेसिंग (प्रवेश करत आहे) (ACCESSING STRUCTURE ELEMENTS)

‘.’ डॉट ऑपरेटरचा वापर करून वैयक्तिक स्ट्रक्चर एलिमेंट्सच्या स्ट्रक्चर व्हेरिएबल्स मध्ये ऍक्सेस करणे शक्य आहे. या डॉट ऑपरेटरला मेम्बरशिप ऑपरेटर देखील म्हटले जाते.

स्ट्रक्चर एलिमेंट्स ऍक्सेसिंग सिंटॅक्स खालीलप्रमाणे आहे.

```
sname.vname
```

जेथे *sname* हे स्ट्रक्चर व्हेरिएबलचे नाव आहे, आणि *vame* हे डेटा फील्डचे नाव आहे पुढील विधानांचा विचार करा

```
struct Student
{
    int rollno;
    char name[25];
    char fname[25];
    char grade;
};

Student aStudent;;
```

स्ट्रक्चर व्हेरिएबल *aStudent* वैयक्तिक एलिमेंट्स खाली दर्शविल्यानुसार ऍक्सेस केला जाईल:

<i>aStudent.rollno</i>	// reference to element rollno
<i>aStudent.name</i>	// reference to element name
<i>aStudent.fname</i>	// reference to element fname
<i>aStudent.grade</i>	// reference to element grade

8.6 ऍक्सेसिंग स्ट्रक्चर्स (ASSIGNING OF STRUCTURES)

स्ट्रक्चर ही एक अशी स्वतंत्र असते संस्था जी संपूर्णपणे मानली जाऊ शकते. संपूर्ण स्ट्रक्चरवर फक्त एक ऑपरेशन केले जाऊ शकते असाईनमेंट ऑपरेटरचा वापर करून स्ट्रक्चर त्याच प्रकारच्या दुसऱ्या स्ट्रक्चर मध्ये असाईन् / कॉपी करणे शक्य आहे.

पुन्हा खालील विधानांचा विचार करा

```
typedef struct
{
    int rollno;
    char name[25];
    int age;
    float height;
} STUDENT;

STUDENT aStudent, bStudent;
```

खालील असाईनमेंट स्टेटमेंट

```
aStudent = bStudent;
```

स्ट्रक्चर *bStudent* ची सामग्री (contents) *aStudent* ला असाईन करते.

अरेच्या तुलनेत हे स्ट्रक्चर्सचे वैशिष्ट्य आहे, जेथे एका अरेला त्याच प्रकारच्या दुसऱ्या अरेमध्ये कॉपी करण्यासाठी एलिमेंट-बाय-एलिमेंट कॉपी करण्यासाठी लूप सेट करावा लागेल.

जेव्हा आपण त्याबद्दल विचार करता तेव्हा ही खरोखर एक आश्चर्यकारक क्षमता आहे. जेव्हा आपण एखादे स्ट्रक्चर दुसऱ्यास स्ट्रक्चरला असाईन् करता तेव्हा स्ट्रक्चर मधील सर्व मूल्ये संबंधित स्ट्रक्चर एलिमेंट दिली जातात. अरेसाठी अशा प्रकारे सोप्या

असाइनमेंट स्टेटमेंट्सचा वापर केला जाऊ शकत नाही. म्हणजेच, एक अर्रे दुसऱ्यास असाईन् करण्यासाठी आपल्याला एलिमेंट-बाय-एलिमेंट असाईन करणे आवश्यक आहे.

8.7 स्ट्रक्चर्सचे वाचन / लेखन (READING/WRITING STRUCTURES)

आपण साधारण व्हेरिएबल्स प्रमाणेच स्ट्रक्चर एलिमेंटमधून डेटा वाचू आणि लिहू शकतो.

उदाहरणार्थ, *aStudent* स्ट्रक्चरचे एलिमेंट, आपण कीबोर्ड वरून खालील स्टेटमेंट्स वापरून *aStudent* स्ट्रक्चर मधील डेटा, परस्पर वाचू शकतो.

```
printf("\nEnter roll number of the student : ");
scanf("%d", &aStudent.rollno);
printf("\nEnter name of the student : ");
scanf("%d", aStudent.name);
printf("\nEnter age of the student : ");
scanf("%d", &aStudent.age);
printf("\nEnter height of the student : ");
scanf("%d", &aStudent.height);
```

लक्षात घ्या की *scanf()* फंक्शनमध्ये एक्सप्रेशन

```
&aStudent.rollno
```

कंपाईलरद्वारे अर्थ लावला आहे

```
&(aStudent.rollno)
```

अँड्रेसऑफ ऑपरेटर (&) च्या तुलनेत थेट निवड ऑपरेटरची उच्चता असल्यामुळे आणि नेमके हेच आवश्यक आहे.

Listing 8.1

```
/*
   Program to demonstrate input/output of structures
*/
#include<stdio.h>
#include<string.h>
struct EMPLOYEE
{
    int code;
    char name[25];
    char department[15];
    float salary;
};
int main()
{
    struct EMPLOYEE aEmployee;
```



```

printf("Enter employee's code: " );
scanf( "%d", &aEmployee.code );
fflush(stdin);
printf("Enter employee's name: " );
gets( aEmployee.name );
printf("Enter employee's department: " );
gets( aEmployee.department );
printf("Enter employee's salary: " );
scanf("%f", &aEmployee.salary );
printf("\n\nParticulars of employee as" );
printf(" entered by user\n" );
printf("\nEmployee's code: %d", aEmployee.code );
printf("\nEmployee's name: %s", aEmployee.name );
printf("\nEmployee's department: %s",aEmployee.department);
printf("\nEmployee's salary: %.2f\n", aEmployee.salary );
return 0;
}

```

Test Run

```

Enter employee's code: 826
Enter employee's name: Rajesh Kumar
Enter employee's department: Computer Science
Enter employee's salary: 20000

Particulars of employee as entered by user

Employee's code: 826
Employee's name: Rajesh Kumar
Employee's department: Computer Science
Employee's salary: 20000.00

```

8.8 स्ट्रक्चरचा अरे (ARRAYS OF STRUCTURES)

बऱ्याच व्यावहारिक परिस्थितीत आपल्याला स्ट्रक्चरचा अरे तयार करण्याची आवश्यकता असते. उदाहरणार्थ, एका वर्गातील विद्यार्थ्यांच्या गटासह कार्य करण्यासाठी आपल्याला विद्यार्थ्यांचा अरे तयार करण्याची आवश्यकता आहे. विद्यार्थ्यांचा डेटा स्ट्रक्चर्सच्या अरे मध्ये ठेवून, आपण विद्यार्थ्यांच्या डेटावर जलद आणि कार्यक्षमतेने प्रक्रिया करू शकतो.

विद्यार्थ्यांचा डेटा संग्रहित करण्यासाठी स्ट्रक्चरचा अरे असा डिक्लेअर केला जाईल :

```
STUDENT students[50];
```

8.8.1 स्ट्रक्चरचा अ‍ॅर्रे एलिमेंट्स ऍक्सेसिंग (Accessing Elements of Array of Structures)

वैयक्तिक स्ट्रक्चर एलिमेंट्सच्या स्ट्रक्चर्सचा अ‍ॅर्रे व्हेरिएबलच्या नावाचा संदर्भ घेता येतो, त्यानंतर इंडेक्स, डायरेक्ट सिलेक्शन ऑपरेटर आणि इच्छित स्ट्रक्चर एलिमेंट्ससह समाप्त होते.

आपल्या वर्गाच्या विद्यार्थ्यांच्या उदाहरणामध्ये, इथ विद्यार्थ्यांच्या नावावर ऍक्सेस केला जाऊ शकतो:

```
students[i].name;
```

8.8.2 इनिशियालाइजिंग स्ट्रक्चरचा अ‍ॅर्रे (Initializing Array of Structures)

इतर कोणत्याही अ‍ॅर्रे प्रमाणे स्ट्रक्चर्सचे अ‍ॅर्रे देखील इनिशियालाइज केले जाऊ शकतात. अ‍ॅर्रे स्ट्रक्चर्सचे प्रत्येक घटक वेगळ्या स्ट्रक्चर असल्यामुळे दर्शविल्याप्रमाणे आपल्याला प्रत्येक स्ट्रक्चर्सचे घटक वेगळ्या ब्रेसिसमध्ये समाविष्ट करणे आवश्यक आहे.

```
STUDENT students[50] = { { 1000, "Monika", {56,76,85,69}, 'A' },
                          { 1001, "Ram", {50,66,70,60}, 'B' },
                          :
                          { 1049, "Raju", {65,62,68,59}, 'B' },
                          };
```

पुढील प्रोग्राम स्ट्रक्चर्सच्या अ‍ॅर्रेचे इनपुट, प्रोसेसिंग आणि आउटपुटचे वर्णन करते.

Listing 8.2

```
/*
   Program to illustrate processing of array of structures by
   storing list of students in memory, and computes their grades
*/
#include <stdio.h>
typedef struct {
    int rollno;
    char sname[25];
    int marks[4];
    char grade;
} STUDENT;
int main()
{
    STUDENT students[50];
    int totalMarks, i, j, n;
    printf( "\n\nEnter number of students in a class : " );
    scanf( "%d", &n );
    for ( i = 0; i < n; i++ )
    {
        printf("\nParticulars of student #%d\n\n", i+1);
        printf( "Enter students' roll number : " );
        scanf("%d", &students[i].rollno );
        printf( "\nEnter students' name : " );
        gets( students[i].sname );
```

```

        printf( "\nEnter students marks in four subjects\n\n" );
        for ( j = 0; j < 4; j++ )
            scanf("%d", &students[i].marks[j] );
    }
    for ( i = 0; i < n; i++ )
    {
        totalMarks = 0;
        for ( j = 0; j < 4; j++ )
            totalMarks += students[i].marks[j];
        if ( totalMarks > 75 )
            students[i].grade = 'A';
        else if ( totalMarks > 60 )
            students[i].grade = 'B';
        else if ( totalMarks > 50 )
            students[i].grade = 'C';
        else if ( totalMarks > 40 )
            students[i].grade = 'D';
        else
            students[i].grade = 'F';
    }
    printf("\nPerformance of students\n\n");
    printf("%-10s", "Roll No.");
    printf("%-30s", "Name");
    printf("%6s\n", "Grade");
    for ( i = 0; i < n; i++ )
    {
        printf("%-10d", students[i].rollno);
        printf("%-30s", students[i].sname);
        printf("%4c\n", students[i].grade);
    }
    return 0;
}

```

Test Run

```

Enter number of students in a class : 3
Particulars of student #1
Enter students' roll number : 1000
Enter students' name : Ram Kumar
Enter students marks in four subjects
98 99 89 88
Particulars of student #2
Enter students' roll number : 1001
Enter students' name : Mohit Kumar
Enter students marks in four subjects
65 55 45 60

```

```

Particulars of student #3
Enter students' roll number : 1002
Enter students' name : Jaspreet Singh
Enter students marks in four subjects
41 44 45 50
Performance of students

```

SNo.	Student's Name	Grade
1	Sonu	A
2	bhollu	C
3	jaspreet	D

8.9 फंक्शनला स्ट्रक्चर पास करणे (PASSING STRUCTURE TO A FUNCTION)

संपूर्णपणे उपयुक्त होण्यासाठी, आपण अर्ग्युमेण्ट म्हणून फंक्शन्सकडे स्ट्रक्चर्स पास करण्यास आणि त्यांना फंक्शन्समधून परत करण्यात सक्षम असणे आवश्यक आहे.

```

typedef struct {
    int day;
    int month;
    int year;
} DATE;
void printDate(DATE);
void main()
{
    DATE aDate = { 10, 6, 2008 };
    int x;
    printDate(aDate);
    /* more statements */
}

```

```

/* function definition */
void printDate(DATE tDate )
{
    /* local declarations */
    /* other statements */
}

```

चित्र 8.3: फंक्शनला स्ट्रक्चर पास करणे

Listing 8.3

```

/*
    Program to illustrate the passing of a structure to a function
*/
#include <stdio.h>
typedef struct
{
    int day;
    int month;
    int year;
} DATE;

```

```

void printDate( DATE aDate );          /* function declaration */
void main()
{
    DATE tDate = { 10, 6, 2008 }; /* initialize a structure */
    printDate( tDate );             /* function call */
}
void printDate( DATE aDate )
{
    printf( "\nDate in format dd/mm/yyyy: %02d/%02d/%2d\n",
            aDate.day, aDate.month, aDate.year );
}

```

Test Run

Date in format dd/mm/yyyy: 10/06/2008

8.10 फंक्शन रिटर्नइंग स्ट्रक्चर (FUNCTION RETURNING A STRUCTURE)

फंक्शन स्ट्रक्चर प्रकारची व्हॅल्यू रिटर्न स्टेटमेंटद्वारे खाली दाखविल्याप्रमाणे परत करू शकते.

```

typedef struct
{
    int day;
    int month;
    int year;
} DATE;
DATE getDate(void);
void main()
{
    DATE aDate;
    int x;
    aDate = getDate();
    /* more statement */
}

```

```

/* function definition */
DATE getDate(void)
{
    DATE tDate;
    /* local declarations */
    /* more statements */
    return tDate;
}

```

चित्र 8.4: फंक्शन रिटर्नइंग स्ट्रक्चर

Listing 8.4

```

/*
    Program to illustrate function returning a structure
*/
#include <stdio.h>
typedef struct

```

```

{
    int day;
    int month;
    int year;
} DATE;
DATE getDate( void );          /* function declaration */
void main()
{
    DATE tDate;
    printf( "Enter date in format dd/mm/yyyy : " );
    tDate = getDate();
    printf( "\nDate entered by you: %02d/%02d/%02d\n",
            tDate.day, tDate.month, tDate.year );
}
DATE getDate( void )
{
    DATE xDate;
    scanf( "%d/%d/%d", &xDate.day, &xDate.month, &xDate.year );
    return xDate;
}

```

Test Run

```

Enter date in format dd/mm/yyyy: 10/6/2008
Date entered by you: 10/06/2008

```

स्पष्टीकरणात्मक उदाहरणे**Example 8.1: पुढील डिक्लरेशन**

```

struct STUDENT
{
    int   rollNo;
    char  name[31];
    int   marks;
};

```

एखाद्या विद्यार्थ्याच्या माहितीचे प्रतिनिधित्व करते.

या स्ट्रक्चरचा वापर करून, विद्यार्थ्यांची यादी अरेच्या रूपात, त्याच्या अर्ग्युमेण्टच्या रूपात घेणारी फंक्शन लिहा आणि विद्यार्थ्यांनी मिळवलेल्या गुणांच्या उतरत्या क्रमाने त्यांचे क्रमवारी लावा.

वरील स्ट्रक्चर आणि फंक्शन वापरून, एक प्रोग्राम लिहा जो n विद्यार्थ्यांचा डेटा वाचतो आणि विद्यार्थ्यांद्वारे सुरक्षित असलेल्या गुणांच्या उतरत्या क्रमाने तो प्रदर्शित करतो.

Listing 8.5

```

/*
    Program to the given information of students in
    descending order of the marks secured by them
*/
#include <stdio.h>
typedef struct
{
    int  rollno;
    char name[20];
    int  marks;
} STUDENT;

int main()
{
    STUDENT st[50], temp;
    int i, j, n;
    printf( "\nEnter number of students : " );
    scanf( "%d", &n );
    for ( i = 0; i < n; i++ )
    {
        printf("\nParticulars of student #%d\n\n", i+1);
        printf( "Enter students' roll number : " );
        scanf("%d", &st[i].rollno );
        fflush(stdin);
        printf( "Enter students' name : " );
        gets(st[i].name );
        printf( "Enter students' marks : " );
        scanf("%d", &st[i].marks );
    }
    for ( i = 1; i < n; i++ )
    {
        for ( j = 0; j < n-i; j++ )
        {
            if ( st[j].marks < st[j+1].marks )
            {
                temp = st[j];
                st[j] = st[j+1];

```

```
                st[j+1] = temp;
            }
        }
    }
    printf("\nStudent's Information after sorting\n\n");
    printf("%-10s", "Roll No.");
    printf("%-20s", "Name");
    printf("%6s\n\n", "Marks");
    for ( i = 0; i < n; i++ )
    {
        printf("%-10d", st[i].rollno);
        printf("%-20s", st[i].name);
        printf("%4d\n", st[i].marks);
    }
    return 0;
}
```

Test Run

```
Enter number of students : 5
Particulars of student #1
Enter students' roll number : 1000
Enter students' name : Ravi
Enter students' marks : 80
Particulars of student #2
Enter students' roll number : 1001
Enter students' name : Shankar
Enter students' marks : 79

Particulars of student #3

Enter students' roll number : 1002
Enter students' name : Ankit
Enter students' marks : 65

Particulars of student #4

Enter students' roll number : 1003
Enter students' name : Rupinder
Enter students' marks : 75
```


Particulars of student #5

Enter students' roll number : 1004

Enter students' name : Mehak

Enter students' marks : 88

Student's Information after sorting

Roll No.	Name	Marks
1004	Mehak	88
1000	Ravi	80
1001	Shankar	79
1003	Rupinder	75
1002	Ankita	65

Example 8.2: स्ट्रक्चर वापरून इंच- फीट (inch-feet) सिस्टीममध्ये दिलेली दोन अंतराची बेरीज करण्यासाठी प्रोग्राम लिहा.

Listing 8.6

```
/*
    Program to add two distances given in inch-feet system
    using structures and functions
*/

#include <stdio.h>

typedef struct
{
    int feets;
    int inches;
} Distance;

/* function prototypes */
Distance getInput(void);
Distance addDistances(Distance d1, Distance d2);
void printDistance(Distance d);
int main()
{
    Distance d1, d2, d3;
    printf("\nEnter the first distance in feets and inches\n");
    d1 = getInput();
    printf("\nEnter the second distance in feets and inches\n");
    d2 = getInput();
    d3 = addDistances(d1, d2);
    printf("\nSum of given distances\n\n");
```

```
        printDistance(d3);
        return 0;
    }
    Distance getInput()
    {
        Distance temp;
        printf("\n\tEnter feets : ");
        scanf("%d", &(temp.feets));
        printf("\n\tEnter inches : ");
        scanf("%d", &(temp.inches));
        return temp;
    }
    Distance addDistances(Distance d1, Distance d2)
    {
        Distance temp;
        temp.feets = d1.feets + d2.feets;
        temp.inches = d1.inches + d2.inches;
        if ( temp.inches >= 12 )
        {
            temp.feets += 1;
            temp.inches -= 12;
        }
        return temp;
    }
    void printDistance(Distance d)
    {
        printf("\n\tFeets = %d", d.feets);
        printf("\n\tInches = %d", d.inches);
    }
}
```

Test Run

```
Enter the first distance in feets and inches
Enter feets : 4
Enter inches : 8

Enter the second distance in feets and inches
Enter feets : 5
Enter inches : 7

Sum of given distances
Feets = 10
Inches = 3
```

Example 8.3: जटिल संख्यांचे मॉडेल तयार करण्यासाठी स्ट्रक्चर वापरा. ही

स्ट्रक्चर वापरून पुढील कार्ये पूर्ण करण्यासाठी फंक्शन लिहा :

- ❑ एक फंक्शन जे कीबोर्डवरून वाचलेल्या स्ट्रक्चरला परत करते.
- ❑ असे फंक्शन जे मूल्य यंत्रणेद्वारे (call by value) दोन स्ट्रक्चर घेतात व त्याचे अर्ग्युमेण्ट म्हणून त्यांची बेरीज परत करतात.
- ❑ एक फंक्शन जे मूल्य यंत्रणेद्वारे (call by value) एक स्ट्रक्चर घेते आणि त्याला प्रिंट करते.

त्यांचे कार्य दर्शविण्यासाठी प्रोग्राममध्ये ही फंक्शन्स वापरा.

Listing 8.7

```
/*
   Program to demonstrate operations on complex number using
   structure & functions
*/
#include <stdio.h>
typedef struct
{
    float real;
    float imag;
} COMPLEX;
COMPLEX input(void);          /* function declarations */
void output(COMPLEX);
COMPLEX add(COMPLEX, COMPLEX);
COMPLEX multiply(COMPLEX, COMPLEX);
int main()
{
    COMPLEX n1, n2, n3, n4;
    printf("\nProgram to add two complex numbers\n\n");
    printf("\nEnter the first complex number\n");
    n1 = input();
    printf("\nEnter the second complex number\n");
    n2 = input();
    n3 = add(n1, n2);
    printf("\nSum of the given complex numbers\n\n");
    output(n3);
    n4 = multiply(n1, n2);
    printf("\nProduct of the given complex numbers\n\n");
    output(n4);
    return 0;
}
COMPLEX input(void)
{
    COMPLEX t;
```

```
printf(«\nEnter real part : «);
scanf(«%f», &t.real);
printf(«Enter imaginary part : «);
scanf(«%f», &t.imag);
return t;
}
void output(COMPLEX t)
{
    printf(«Real part : %.2f», t.real);
    printf(«\nImaginary part : %.2f», t.imag);
}
COMPLEX add(COMPLEX a, COMPLEX b)
{
    COMPLEX t;
    t.real = a.real + b.real;
    t.imag = a.imag + b.imag;
    return t;
}
COMPLEX multiply(COMPLEX a, COMPLEX b)
{
    COMPLEX t;
    t.real = a.real * b.real - a.imag * b.imag;
    t.imag = a.real * b.imag + a.imag * b.real;
    return t;
}
```

Test Run

```
Program to add two complex numbers
Enter the first complex number
Enter real part : 1.0
Enter imaginary part : 2.5
Enter the second complex number
Enter real part : 2.5
Enter imaginary part : -1.5
Sum of the given complex numbers
Real part : 3.50
Imaginary part : 1.00
Product of given complex numbers
Real part : 6.25
Imaginary part : 4.75
```

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- स्ट्रक्चर संबंधित घटकांचा(element) संग्रह आहे, ज्याचे नाव एकच असू शकते.
- स्ट्रक्चरच्या प्रत्येक घटकास फील्ड असे म्हणतात .
- स्ट्रक्चर डिक्लेअर करण्याचे दोन मार्ग आहेत: टॅग स्ट्रक्चर आणि टाइप- डिफाइंड स्ट्रक्चर.
- जेव्हा एखादे स्ट्रक्चर डिक्लेअर आणि डिफाइंड केले जाते तेव्हा त्याची सुरुवात केली जाऊ शकते. स्वल्पविरामाने विभक्त केलेल्या मूल्यांची यादी कंसात जोडलेली आहे.
- आपण थेट सिलेक्शन, डॉट ऑपरेटर (.) चा वापर करून स्ट्रक्चरच्या फील्डला ऍक्सेस करू शकतो.
- अरे आणि स्ट्रिंगच्या विपरीत असाईनमेंट ऑपरेटरचा वापर करून स्ट्रक्चर दुसऱ्या स्ट्रक्चरला असाइन करू शकतो.
- फंक्शनच्या अर्ग्युमेंट म्हणून स्ट्रक्चर पास केली जाऊ शकते.
- फंक्शन स्ट्रक्चर रिटर्न करू शकतो.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. स्ट्रक्चर वापरण्याचे मुख्य कारण काय आहे?
2. अरेपेक्षा स्ट्रक्चर कसे वेगळे आहे?
3. स्ट्रक्चर टॅग म्हणजे काय आणि त्याचा हेतू काय आहे?
4. स्ट्रक्चर निश्चित करण्याचे वेगवेगळे मार्ग काय आहेत? प्रत्येक बाबतीत उदाहरण द्या.
5. खालील गोष्टींमध्ये काही चूक आहे काय?

```
struct Time
{
    int hrs;
    int mts;
    int secs;
}
time t1;
```

6. पुढील गोष्टींमध्ये काय चूक आहे काही असल्यास,?

```
struct Time
{
    int hrs =10;
    int mts = 20;
    int secs = 35;
} t1, t2, t3;
```

7. खालील लुटी, काही असल्यास त्या ओळखा

```
struct
{
    int hrs;
    int mts;
    int secs;
} time;
time t1, t2, t3;
```

8. पुढील प्रोग्रामचे आउटपुट द्या:

```
#include <stdio.h>
struct MyBox {
    int Length, Breadth, Height;
};
void Dimension( MyBox M )
{
    printf("\n%d x %d x %d\n",M.Length,M.Breadth,M.Height);
}
void main()
{
    MyBox B1 = { 12, 20, 8 }, B2, B3;
    ++B1.Height;
    Dimension(B1);
    B3 = B1;
    ++B3.Length;
    B3.Breadth++;
    Dimension(B3);
    B2 = B3;
    B2.Height += 5;
    B2.Length--;
    Dimension(B2);
}
```

एकाधिक निवड प्रश्न

1. स्ट्रक्चर हे

(a) स्केलर डेटा प्रकार

(c) Primitive डेटा प्रकार

(b) Derived डेटा प्रकार

(d) युझर- डिफाईंड डेटा प्रकार

2. खालीलपैकी कोणते विधान स्ट्रक्चर बहुल खरे आहे?
 - (a) स्ट्रक्चरचे घटक भिन्न प्रकारचे असणे आवश्यक आहे.
 - (b) सर्व प्रकरणांमध्ये स्ट्रक्चर टॅग अनिवार्य आहे.
 - (c) स्ट्रक्चर प्रकाराचे व्हेरिएबल घोषित करताना, आपल्याला टॅगच्या नावापूर्वी कीवर्ड struct चा उपसर्ग आवश्यक आहे.
 - (d) स्टार (*) ऑपरेटरचा वापर करून स्ट्रक्चर घटकावर प्रवेश केला जातो.
3. स्ट्रक्चर म्हणजे डेटा प्रकार
 - (a) प्रत्येक घटकात समान प्रकार असणे आवश्यक आहे.
 - (b) घटक भिन्न प्रकारचे असू शकतात.
 - (c) प्रत्येक घटक पॉइंटर प्रकारचा असावा.
 - (d) वरीलपैकी कोणतेही नाही
4. पुढील डिक्लरेशनचा विचार करा

```
struct ex {
    char cVvar;
    int iVar;
    long lVar;
};
```

Ex वर कॉल केलेल्या sizeof ऑपरेटरद्वारे कोणते मूल्य दिले जाईल?

- (a) 4 (b) 7 (c) 1 (d) 6
5. स्ट्रक्चर डेटा प्रकार तयार करण्यासाठी वापरलेला कीवर्ड आहे
 - (a) structure (b) structr (c) struct (d) struc
6. स्ट्रक्चर मेंबरला ऍक्सेस करण्यासाठी वापरलेला ऑपरेटर आहे
 - (a) * (b) . (c) & (d) []
7. विधानाचा विचार करा: एखाद्या स्ट्रक्चरचा आकार नेहमीच त्याच्या मेंबरच्या बरोबरीचा असतो.
 - (a) वैध (b) म्हणू शकत नाही
 - (c) वैध (d) कंपाइलरवर अवलंबून
8. विधानाचा विचार करा: एखाद्या स्ट्रक्चरचा आकार त्याच्या सदस्यांच्या बेरजेइतकाच असतो.

```
#include<stdio.h>
struct st
{
    int x;
    static int y;
};
int main()
{
```

```
printf("%d", sizeof(struct st));
return 0;
}
```

(a) 4

(b) 8

(c) Runtime error

(d) Compiler time error

9. पुढीलपैकी कोणता ऑपरेटर स्ट्रक्चर व्हेरिएबल्सवर लागू केला जाऊ शकतो?

(a) ==

(b) =

(c) >

(d) +

10. "s.b = 10" सारख्या कोडची उपस्थिती दर्शवते

(a) ordinary variable name

(b) double data type

(c) structure

(d) syntax error

11. पुढील प्रोग्रामचे आउटपुट द्या:

```
struct Test
{
    char str[20];
};
void main()
{
    struct Test st1, st2;
    strcpy(st1.str, "C Quiz");
    st2 = st1;
    st1.str[0] = 'S';
    cout << st2.str;
}
```

(a) C Quiz

(b) S Quiz

(c) Runtime error

(d) Compile time error

12. जर तुम्ही फंक्शनला स्ट्रक्चर व्हेरिएबल पास केले तर काय केले जाईल?

(a) स्ट्रक्चर व्हेरिएबलचा संदर्भ

(b) स्ट्रक्चर व्हेरिएबलची कॉपी

(c) स्ट्रक्चर व्हेरिएबलचा प्रारंभ ऍड्रेस

(d) स्ट्रक्चर व्हेरिएबलचा शेवटचा ऍड्रेस

13. खालील C कोडच्या आउटपुटवर टिप्पणी द्या.

```
struct temp {
    int a;
    int b;
    int c;
};
main()
```



```
{
    struct temp p[] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
}
```

- (a) आकार 3 च्या स्ट्रक्चर ची अरे तयार करते (b) आकार 9 च्या स्ट्रक्चर ची अरे तयार करते
 (c) स्ट्रक्चर सदस्यांना बेकायदेशीर असाइनमेंट (d) मल्टि डायमेंशनल अरेची अवैध डिक्लेरेशन
14. पुढीलपैकी कोणती स्ट्रक्चर सदस्य असू शकत नाही?
 (a) अरे (b) स्ट्रक्चर
 (c) स्ट्रिंग्स (d) फंक्शन
15. पुढीलपैकी कोणती योग्यरित्या डिफाइंड struct आहे?
 (a) struct {int a;} (b) struct a_struct {int a;};
 (c) struct a_struct int a; (d) struct a_struct {int a;}

ANSWERS									
1.	(d)	2.	(c)	3.	(b)	4.	(b)	5.	(c)
6.	(b)	7.	(a)	8.	(d)	9.	(b)	10.	(c)
11.	(a)	12.	(b)	13.	(a)	14.	(d)	15.	(b)

प्रोग्रामिंग समस्या

- आवारातील, फीट आणि इंच मध्ये लांबी साठवण्यासाठी distance नावाच्या स्ट्रक्चरची रचना करा. या स्ट्रक्चरचा वापर करून, एक प्रोग्राम लिहा जो युझर कडून या स्ट्रक्चर द्वारे प्रतिनिधित्व केल्यानुसार दोन मोजमाप स्वीकारतो आणि या मोजमापांमधील परिपूर्ण फरक दर्शवितो.
- विद्यार्थ्यांविषयी डेटा साठवण्यासाठी student नावाची स्ट्रक्चरची अशी रचना करा ज्यात खालील घटक आहेत - rollno of type int, name an array of type char of size 20, college an array of type char of size 40, and score of type float. समजा 100 पेक्षा जास्त विद्यार्थी नाहीत. विद्यार्थ्यांविषयी डेटा इनपुट करण्यासाठी प्रोग्राम लिहा आणि विद्यार्थ्यांच्या गुणवत्तेनुसार संग्रहित डेटा आउटपुट करा.

- पुढील डिक्लेरेशनचा विचार करा:

```
struct EMPLOYEE
{
    int code;
    char name[31];
    float salary;
};
```

एखाद्या संस्थेमध्ये n (≤ 50) कर्मचारी आहेत. एक प्रोग्राम लिहा ज्यामध्ये कर्मचाऱ्यांच्या अधिक पगाराची माहिती मिळेल.

- तास, मिनिटे आणि सेकंद अशी तीन फील्ड असलेल्या 12 तास प्रणालीमध्ये Time मॉडेल करण्यासाठी एक स्ट्रक्चर तयार करा. ElapsedTime नावाचे फंक्शन लिहा जे दोन अर्ग्युमेंट घेते, प्रारंभ वेळ आणि समाप्ती वेळ. फंक्शनने दोन अर्ग्युमेंट मधील वेळ व्यतीत केलेली वेळ स्ट्रक्चर परत करावी. वरील फंक्शनची चाचणी घेण्यासाठी एक प्रोग्राम लिहा.

5. A कार्टेशियन सिस्टममधील विमानातील बिंदूचे निर्देशांक x आणि y यांनी दर्शविले जाऊ शकतात. खाली दर्शविल्याप्रमाणे प्लेनमधील पॉइंटचे प्रतिनिधित्व करण्यासाठी आम्ही एक स्ट्रक्चर तयार करू शकतो:

```
struct POINT
{
    int x;
    int y;
};
```

एक फंक्शन लिहा जे बिंदूचे प्रतिनिधित्व करणारी रचना स्वीकारते आणि बिंदू स्थित असलेल्या चतुष्पाद(quadrant) दर्शविणारा integer मूल्य 1, 2, 3 किंवा 4 मिळवते. वरील फंक्शनची चाचणी घेण्यासाठी एक प्रोग्राम लिहा.

6. देशाचे नाव, राजधानी नाव आणि दरडोई उत्पन्न अशी तीन सदस्य असलेली स्ट्रक्चर तयार करा. या स्ट्रक्चर चा वापर करून, दरडोई उत्पन्नाच्या घटत्या क्रमवारीत देशांच्या राजधानीसह त्यांची यादी करण्यासाठी प्रोग्राम लिहा.

प्रॅक्टिकल

1. वेळेचे प्रतिनिधित्व करण्यासाठी स्ट्रक्चर वापरून डिजिटल घड्याळ (24 तासांच्या स्वरूपात) नकल करण्यासाठी प्रोग्राम लिहा.

Listing 8.8

```
#include <stdio.h>
/* typedefed structure to represent clock time */
typedef struct {
    int hh;
    int mm;
    int ss;
} CLOCK;
int main()
{
    CLOCK myClock = { 12, 40, 55 };
    int i;
    printf("\nPress any key to stop the clock\n");
    while ( !kbhit() )
    {
        /* increment clock */
        (myClock.ss)++;
        if ( myClock.ss == 60 ) {
            myClock.ss = 0;
            (myClock.mm)++;
```

```

        if ( myClock.mm == 60 ) {
            myClock.mm = 0;
            (myClock.hh)++;
            if ( myClock.hh == 24 ) {
                myClock.hh = 0;
            }
        }
    }
    /* show clock */
    printf("%02d:%02d:%02d\n",
           myClock.hh, myClock.mm, myClock.ss);
}
return 0;
}

```

Test Run

```

Press any key to stop the clock
12:40:56
12:40:57
12:40:58
12:40:59
12:41:00
12:41:01
12:41:02

```

- वेळेचे प्रतिनिधित्व करण्यासाठी स्ट्रक्चर वापरून start वेळ आणि end वेळ (१२ तासांच्या स्वरूपात) मधील फरक शोधण्यासाठी एखादा प्रोग्राम लिहा.

Listing 8.9

```

#include <stdio.h>
typedef struct {
    int hh;
    int mm;
    int ss;
} TIME;

/* function prototype */
TIME differenceTime( TIME t1, TIME t2);
int main()
{
    TIME startTime, stopTime, diffTime;

```

```

printf("Enter the start time in format hh mm ss : ");
scanf("%d %d %d", &startTime.hh, &startTime.mm, &startTime.ss);
printf("Enter the stop time in format hh:mm:ss : ");
scanf("%d %d %d", &stopTime.hh, &stopTime.mm, &stopTime.ss);
/* functiona call */
diffTime = differenceTime(startTime, stopTime);
printf("\nTime Difference: %d:%d:%d\n", diffTime.hh,
        diffTime.mm, diffTime.ss);
return 0;
}
/* function that return difference between time periods */
TIME differenceTime( TIME t1, TIME t2)
{
    TIME temp;
    if ( t1.ss > t2.ss ) {
        --t2.mm;
        t2.ss += 60;
    }
    temp.ss = t2.ss - t1.ss;
    if ( t1.mm > t2.mm ) {
        --t2.hh;
        t2.mm += 60;
    }
    temp.mm = t2.mm - t1.mm;
    temp.hh = t2.hh - t1.hh;
    return temp;
}

```

Test Run

```

Enter the start time in format: hh mm ss : 5 20 30
Enter the stop time in format: hh mm ss : 7 15 10
Time Difference: 1:54:40

```

आणखी माहिती

स्ट्रक्चर एक युझर- डिफाईंड प्रकार आहे. बऱ्याच समस्या आहेत जिथे मूलभूत प्रकारांद्वारे मिळविलेले मूलभूत प्रकार आणि अरे वास्तविक जीवनातील समस्यांचे मॉडेल तयार करण्यास सक्षम नाहीत. अशा परिस्थितीत आपल्याला एक नवीन डेटा प्रकार तयार करण्याची आवश्यकता आहे, ज्यास युझर- डिफाईंड प्रकार म्हणतात, जी C मधील एक स्ट्रक्चर आहे.

वास्तविक जीवनातल्या बहुतेक वास्तविक जीवनातील ॲप्लिकेशनमध्ये, स्ट्रक्चरचा वापर हा एकमेव मार्ग आहे. हे वास्तविक जीवनातील समस्यांचे निराकरण करण्यासाठी संरचनेचे महत्त्व दर्शविते.

शिक्षकाकडून स्ट्रक्चरची संकल्पना समजून घेण्याची आणि वास्तविक जीवनातून योग्य उदाहरणे घेऊन स्ट्रक्चरचा वापर दर्शविण्याची अपेक्षा आहे.

संदर्भ आणि सूचविलेले वाचन

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

9

पॉइंटर्स

युनिट वैशिष्ट्ये

हे युनिट पॉइंटर्सशी संबंधित विषयांवर चर्चा करते. पॉइंटर्स सी भाषेच्या महत्त्वाच्या विषयांपैकी एक प्रतिनिधित्व करतात. पॉइंटर्सच्या विवेकपूर्ण वापराने सी भाषेची वास्तविक शक्ती वापरली जाऊ शकते. हे युनिट पॉइंटर्सचे विविध पैलू स्पष्ट करते आणि योग्य उदाहरणांसह त्यांचा वापर दर्शवते.

तर्कशास्त्र

सामान्य परिस्थितीत, प्रत्येक सी प्रोग्रामला डेटा संग्रहित करण्यासाठी आणि प्रोग्रामच्या सूचनांसाठी मेमरी वाटप केली जाते. दोन्ही साठवणुकीसाठी आवश्यक असलेल्या मेमरीची मात्रा कॅम्पयलेशन च्या वेळी ठरवली जाते, आणि ती निश्चित असते आणि प्रोग्रामच्या एक्झिक्युशनच्या दरम्यान बदलली जाऊ शकत नाही. व्यावहारिकरित्या, आपण प्रोग्राम कार्यान्वित करताना प्रत्येक वेळी सूचनांसाठी आवश्यक असलेली मेमरी निश्चित केली जाते, तथापि, डेटाचे आकार एका समस्येमधून दुसऱ्या समस्येत बदलू शकते. डेटाचे वेगवेगळे आकार, डेटाच्या डायनॅमिक प्रवृत्तीचा हा मुद्दा फक्त प्रोग्रामच्या डेटा आयटमच्या डिक्लरेशन मध्ये बदल करून आणि प्रोग्राम पुन्हा तयार करून सोडवला जाऊ शकतो. जर आपणास सौर्स कोडमध्ये प्रवेश नसेल तर हा दृष्टिकोन कार्य करणार नाही. पॉइंटर्सद्वारे हा मुद्दा हाताळण्यासाठी एक चांगला पर्यायी पर्याय प्रदान केला जातो, कारण पॉइंटर्स वापरून, एक्झिक्युशनच्या वेळी वास्तविक आवश्यकतेनुसार डेटा आयटमसाठी मेमरी वाटप केली जाऊ शकते. याव्यतिरिक्त, पॉइंटर्सचा वापर खालील परिस्थितीत फायदे देते:

- फंक्शनसाठी कार्यक्षमतेने अर्ग्युमेंट्स पास करणे.
- कॉल केलेले कार्य अर्ग्युमेंट्सद्वारे कॉलिंग फंक्शन्समध्ये एकापेक्षा जास्त मूल्य रिटर्न करणे.
- प्रोग्राम स्थापित केलेल्या मेमरीच्या कोणत्याही मेमरी स्थान तसेच कोणत्याही डिव्हाइस कनेक्ट केलेला जो प्रवेशयोग्य आहे प्रवेश करू शकतो.
- वास्तविक जीवनातील समस्यांचे मॉडेल करण्यासाठी जटिल डेटा स्ट्रक्चर तयार करणे.

थोडक्यात, आपण असे म्हणू शकतो की पॉइंटर्स C भाषेची सर्वात वेगळी आणि रोमांचक वैशिष्ट्ये आहेत. हे भाषेला सामर्थ्य आणि लवचिकता प्रदान करते. हे युनिट विद्यार्थ्याला पॉइंटर्सशी संबंधित विविध पैलू समजून घेण्यास आणि पॉइंटर्सचा उपयोग करून वास्तविक जीवनातील समस्या कार्यक्षमतेने सोडविण्यासाठी प्रोग्राम विकसित करण्यास मदत करेल.

पूर्व-आवश्यकता

- मूलभूत डेटा प्रकार (Basic data types)
- ऑपरेटर (Operators)
- अरे आणि स्ट्रिंग्स (Arrays and strings)

- फंक्शन्स (Functions)
- स्ट्रक्चर्स (Structures)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U9-O1:	पॉईंट्सच्या जगाचे आकलन करा, C भाषेचे अधोगती वैशिष्ट्यांपैकी एक.
U9-O2:	पॉईंट्स डिक्लेअ आणि इनिशियालाइज करा.
U9-O3:	पॉईटरचा संदर्भ न घेता डेटामध्ये प्रवेश करा.
U9-O4:	पॉईंट्सवर परवानगी असलेल्या ऑपरेशन्सचे स्पष्टीकरण द्या.
U9-O5:	फंक्शनला अर्ग्युमेण्ट देताना पॉईंट्सची उपयुक्तता समजावून सांगा.
U9-O6:	डायनॅमिक मेमरी लोकेशनची संकल्पना समजावून सांगा.
U9-O7:	विविध समस्या सोडविण्यासाठी पॉईंट्स वापरून कार्यक्षम प्रोग्राम लिहा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 9 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U9-O1			1					
U9-O2			1					
U9-O3			1					
U9-O4						2		
U9-O5					2			
U9-O6						2		
U9-O7						3		

9.1 ओळख (INTRODUCTION)

पॉईंट्स C भाषेची एक महत्त्वपूर्ण वैशिष्ट्य आहे. पॉईंट्स, बहुतेक लोक, C मधील सर्वात कठीण विषयांपैकी एक मानले जातात. आम्ही तुम्हाला अगदी स्पष्टपणे सांगतो, हे तथ्यपेक्षा एक मिथक आहे. आम्ही हे सांगण्यास प्राधान्य देतो की पॉईंट्स C भाषेतील सर्वात नाजूक पैलूंपैकी एक आहेत आणि आपल्याला माहिती आहे; नाजूक गोष्टींसाठी सावध हाताळणी आवश्यक आहे. म्हणून, पॉईंट्सना काळजीपूर्वक हाताळणी आवश्यक आहे.

पॉईंट्स वापरण्याची अनेक कारणे आहेत; त्यापैकी काही दिलेली आहेत:

- पॉईटर फंक्शन किंवा प्रोग्रामच्या पूर्वावलोकनाबाहेरील व्हेरिएबल (अधिक चांगले आपण मेमरी लोकेशन म्हणतो) वर प्रवेश करण्यासाठी फंक्शन किंवा प्रोग्रामला अनुमती देतो. पॉईटर वापरून, आपला प्रोग्राम संगणकाच्या मेमरीमधील कोणत्याही मेमरी स्थानावर प्रवेश करू शकतो.

- रिटर्न स्टेटमेंट वापरल्यामुळे, फंक्शन फक्त कॉलिंग फंक्शनमध्ये एकच मूल्य परत करू शकते, पॉइंटर्स फंक्शनला कॉलिंग फंक्शनमध्ये प्रवेश करण्यायोग्य मेमरी ठिकाणी लिहून फंक्शनला एकापेक्षा जास्त मूल्य परत देतात.
- पॉइंटर्स मुळे अरे आणि स्ट्रक्चर्स वापरणे अधिक कार्यक्षम पद्धतीने हाताळले जाऊ शकते.
- पॉइंटर्सशिवाय, जोडलेल्या lists, trees आणि graphs यासारखी जटिल डेटा स्ट्रक्चर तयार करणे अशक्य होईल.
- मेमरीबद्दल ऑपरेटिंग सिस्टमशी संवाद साधणे. उदाहरणार्थ, ऑपरेटर नवीन पॉइंटरचा वापर करून फ्री मेमरी ब्लॉकचे स्थान परत करते आणि ऑपरेटर डिलीट ऑपरेटिंग सिस्टमला पॉइंटरद्वारे दर्शविलेले मेमरी ब्लॉक परत करते.

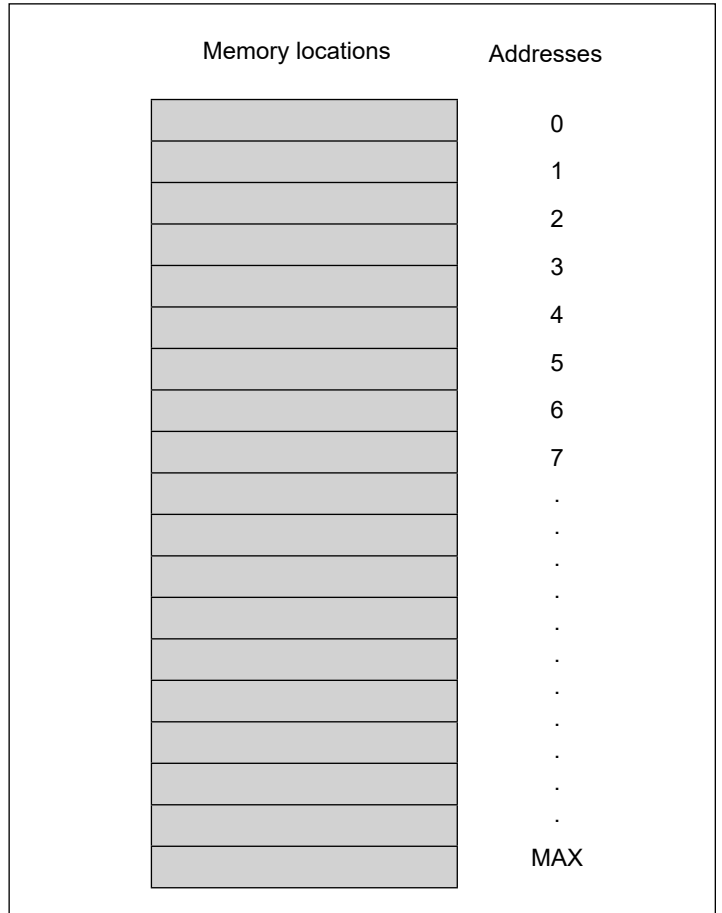
एकूणच, आपण असे म्हणू शकतो की C भाषेची वास्तविक शक्ती पॉइंटर्सच्या न्याय्य वापरामध्ये आहे. म्हणूनच, सी च्या प्रत्येक वाचकाने पॉइंटर्स वापरण्याची कला शिकणे आणि त्यावर प्रभुत्व असणे आवश्यक आहे.

या युनिटमध्ये आपण सी भाषेच्या पॉइंटर्सशी संबंधित विविध पैलूंबद्दल शिकू या.

9.2 पॉइंटर्सची कल्पना (IDEA OF POINTERS)

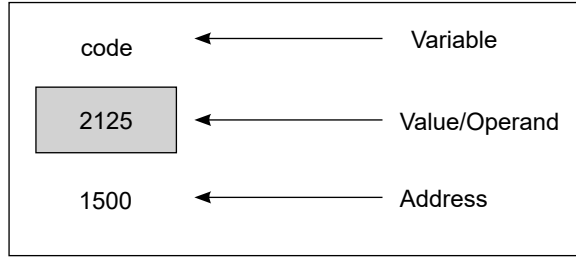
कॉम्प्यूटरची मेमरी बाइट्स किंवा शब्दांच्या लिनिअर संग्रह म्हणून आयोजित केली जाते. इंटेल प्रोसेसरसाठी मेमरी बाइट-ऑर्गनाइज्ड असते तर बय्याच इंटेल नसलेल्या प्रोसेसरसाठी मेमरी वर्ड-ऑर्गनाइज्ड असते. बाइट-ऑर्गनाइज्ड मेमरीमध्ये, प्रत्येक मेमरी स्थान एक बाइट असते, तर शब्द-संयोजित असताना, प्रत्येक मेमरी स्थान एक शब्द असते. प्रत्येक मेमरी स्थानास स्थान क्रमांक किंवा ऍड्रेस नावाचा एक अनोखा क्रमांक नियुक्त केला जातो. आकृती 9.1 मेमरी संस्थेची योजनाबद्ध योजना दर्शविते.

आम्हाला माहित आहे की एक बाइट 8 बिट्सच्या बरोबरीचा आहे, तर एक शब्द दोन किंवा अधिक बाइटचा बनलेला असू शकतो. एक शब्द एकल (सिंगल) ऑपरेशनमध्ये प्रक्रिया करू शकणाऱ्या ऑपरेशन्सच्या आकाराच्या दृष्टीने प्रोसेसरची क्षमता मोजतो (म्हणजे बेरीज, वजाबाकी, गुणाकार, विभाजन, तुलना इ.). जेव्हा आम्ही असे म्हणतो की प्रोसेसर 16-बिट प्रोसेसर आहे, तेव्हा त्याचा शब्द आकार 16-बिट (2 बाइट) असतो आणि तो एकल(सिंगल) ऑपरेशन्समध्ये 16-बिट ऑपरेंड्सवर प्रक्रिया करू शकतो. मोठ्या आकाराच्या ऑपरेंडसाठी, त्यास अधिक ऑपरेशन्सची आवश्यकता आहे.



चित्र 9.1: मेमरी संघटना

एक मेमरी सेल एकतर बाइट / शब्द किंवा अधिक संबद्ध (कन्टिग्युअस) बाइट्स / शब्दांपैकी एक आहे. प्रत्येक सेल दिलेल्या वेळी एक मूल्य ठेवू शकतो. सेलचा ऍड्रेस नेहमी प्रथम बाइट / शब्दाचा ऍड्रेस असतो.

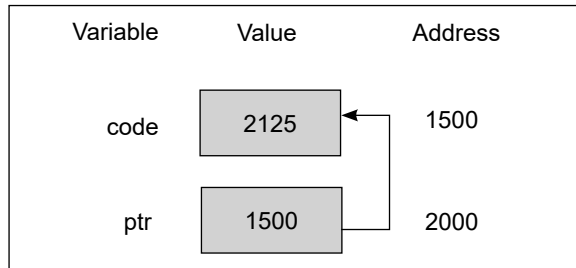


चित्र 9.2: मेमरीमधील व्हेरिएबलचे प्रतिनिधित्व

जेव्हा आपण एखादा प्रोग्राम चालविण्यासाठी कमांड देता तेव्हा ऑपरेटिंग सिस्टम प्रथम संगणक (मुख्य) मेमरीमधून आवश्यक आकाराचे एक विनामूल्य ब्लॉक शोधते आणि नंतर प्रोग्राम त्या ब्लॉकमध्ये लोड करते. त्या ब्लॉकमध्येही सहसा प्रोग्रामचा कोड एका भागामध्ये आणि प्रोग्रामचा डेटा ब्लॉकच्या दुसऱ्या भागात लोड केला जातो. प्रत्येक डेटा ऑपरेंड सेलमध्ये संग्रहित केला जातो आणि सिस्टम प्रत्येक पत्ते या पत्त्यांसह जोडते. हे चित्र 9.2 मध्ये स्पष्ट केले आहे.

डेटा ऑपरेंडमध्ये प्रवेश करण्यासाठी आपण एकतर व्हेरिएबलचे नाव वापरतो किंवा मेमरी सेलचा ऍड्रेस वापरतो. हे ऍड्रेस फक्त सकारात्मक इन्टिजर संख्या असल्यामुळे इतर कोणत्याही व्हेरिएबलप्रमाणे ते मेमरीमध्ये संग्रहित काही व्हेरिएबलसना देखील दिले जाऊ शकतात. ऍड्रेस असणारे असे व्हेरिएबल्स पॉइंटर म्हणून ओळखले जातात. म्हणून पॉइंटर हे व्हेरिएबलशिवाय काहीही नसते ज्यामध्ये मेमरीमधील दुसऱ्या व्हेरिएबलचा ऍड्रेस असतो.

पॉइंटर व्हेरिएबल असल्याने त्याचे मूल्य मेमरीमध्ये दुसऱ्या प ऍड्रेसवरही साठवले जाते. समजा आपण व्हेरिएबल कोडचा ऍड्रेस व्हेरिएबल ptr ला नेमला आहे. व्हेरिएबल ptr आणि कोडमधील दुवा चित्र 9.3 मध्ये दाखवल्यानुसार व्हिज्युअलाइज्ड बनविला जाऊ शकतो.



चित्र 9.3: व्हेरिएबल म्हणून पॉइंटर

व्हेरिएबल ptr ची व्हॅल्यू व्हेरिएबल कोडचा ऍड्रेस असल्यामुळे व्हेरिएबल ptr ची व्हॅल्यू वापरून आपण व्हेरिएबल कोडचे मूल्य बघू शकतो. म्हणून आपण असे म्हणतो की व्हेरिएबल ptr व्हेरिएबल कोड कडे निर्देशित करते, आणि म्हणून ptr ला नेम पॉइंटर मिळेल.

9.3 ऍक्सेसिंग व्हेरिएबलचा पत्ता (ACCESSING ADDRESS OF A VARIABLE)

व्हेरिएबलचा वास्तविक ऍड्रेस सिस्टम-आधारित असतो. कॅम्पयलेशन आणि लिंकिंग दरम्यान, ऍड्रेस काही बेस ऍड्रेसशी संबंधित असतात असे मानले जाते, सहसा 0. जेव्हा ऑपरेटिंग सिस्टम प्रोग्राम विनामूल्य ब्लॉकमध्ये लोड करते, तेव्हा विनामूल्य सर्व ब्लॉकच्या

प्रथम मेमरी स्थानाच्या ऍड्रेसशी संबंधित सर्व ऍड्रेसचे रूपांतर होते. म्हणून आपल्याला व्हेरिएबलचा ऍड्रेस माहित नाही. म्हणून, एक प्रश्न उद्भवतो - आपण व्हेरिएबलचा ऍड्रेस कसा ठरवू शकतो? हे ऑपरेटरच्या address of(&) मदतीने केले जाते.

पुढील विधानाचा विचार करा

```
ptr = &code;
```

हे व्हेरिएबल ptr ला 1500 नंबर (व्हेरिएबल कोडचा ऍड्रेस) नियुक्त करेल.

पुढील निर्बंध पाळले पाहिजेत:

1. ऑपरेटरचा ऍड्रेस केवळ व्हेरिएबल किंवा अर्रे घटकांसह वापरला जाऊ शकतो. *addressof* ऑपरेटरचा बेकायदेशीर वापर खालीलप्रमाणे आहेत::

(a) &1200 (pointing at constant expression)

(b) &(a/b) (pointing at expression)

2. पॉइंटर व्हेरिएबलची सुरुवात फक्त ऑपरेटरच्या ऍड्रेस च्या उपयोगाने किंवा दुसऱ्या सुसंगत पॉइंटर व्हेरिएबलचे मूल्य देऊन केली जाऊ शकते. पॉइंटर व्हेरिएबल ptr सुरू करण्याचा खालील मार्ग बेकायदेशीर आहे:

```
ptr = 1500;
```

जरी, क्रमांक 1500 हा व्हेरिएबल कोडचा ऍड्रेस असेल.

9.4 पॉइंटर डिक्लेअर करा (DECLARING A POINTER)

पॉइंटर व्हेरिएबल डिक्लरेशन करणे हे स्टॅंडर्ड व्हेरिएबलच्या घोषणेसारखेच आहे परंतु पॉइंटर व्हेरिएबलचे नाव * चिन्हासह उपसर्ग केलेले आहे.

पॉइंटर व्हेरिएबल डिक्लेअर करण्यासाठी सिंटॅक्स हा आहे.

```
datatype *pointerVariableName
```

उदाहरणार्थ, विधान

```
int *ptr;
```

पॉइंटर व्हेरिएबल ptr डिक्लेअर करा जो कोणत्याही इन्टिजर व्हेरिएबलचा ऍड्रेस संचयित करण्यासाठी वापरला जाऊ शकतो.

9.5 पॉइंटरला ऍड्रेस असाइनिंग (ASSIGNING ADDRESS TO A POINTER)

एकदा पॉइंटर घोषित झाल्यानंतर, त्यास व्हेरिएबलचा ऍड्रेस असावा. स्टॅंडर्ड व्हेरिएबल्स प्रमाणे पॉइंटर व्हेरिएबल एकदा डिक्लेअर केल्या नंतर गारबेज(garbage) मूल्य घेईल. म्हणूनच उपयोग करण्यापूर्वी पॉइंटर व्हेरिएबलला व्हेरिएबलचा ऍड्रेस असावा.

पॉइंटर व्हेरिएबलला ऍड्रेस असाइन करण्यासाठी सिंटॅक्स

```
pointerVariableName = &variableName;
```

उदाहरणार्थ, विधान

```
int a, *ptr;
```

a एक सामान्य इन्टिजर व्हेरिएबल डिक्लेअर करते आणि ptr हे इन्टिजर पॉइंटर व्हेरिएबल आहे.

पॉइंटर व्हेरिएबल ptr चा व्हेरिएबलचा ऍड्रेस असाइन करण्यासाठी आपण खालील स्टेटमेंट वापरू

```
ptr = &a;
```

येथे आपण असे म्हणतो की पॉइंटर व्हेरिएबल ptr हे व्हेरिएबल a कडे निर्देशित करते.

9.6 पॉइंटर व्हेरिएबल वापरून ऍक्सेसइंग व्हेरिएबल (ACCESSING VARIABLE USING A POINTER VARIABLE)

एकदा पॉइंटर व्हेरिएबलला व्हेरिएबलचा अॅड्रेस दिल्यावर प्रश्न राहतो - पॉइंटर व्हेरिएबलचा वापर करून व्हेरिएबलच्या व्हॅल्यूला कसे ऍक्सेस करायचा. हे पॉइंटर व्हेरिएबल सह * प्रिफिक्स करून केले जाते.

पॉइंटर व्हेरिएबल वापरून व्हेरिएबलला ऍक्सेस करण्यासाठी सिंटॅक्स

```
*pointerVariableName
```

Listing 9.1

```
/*
    Program to illustrate the use of pointer variable
*/
#include<stdio.h>
int main()
{
    int *ptr, x = 10;
    ptr = &x;
    printf(«Value of variable x = %d\n», x );
    printf(«Value of variable pointed to by ptr = %d\n», *ptr );
    printf(«Address of variable x = %u\n», ptr );
    return 0;
}
```

Test Run on Turbo C++ Compiler

```
Value of variable x = 10
Value of variable pointed to by ptr = 10
Address of variable x = 65524
```

9.7 पॉइंटर अरीथमेटिक (POINTER ARITHMETIC)

पॉइंटर अरीथमेटिक करण्याच्या दृष्टिकोनात C भाषा खूपच सुसंगत आहे. पॉइंटर्स, अॅरे आणि पॉइंटर अरीथमेटिक एकत्र करणे ही C भाषेची एक शक्ती आहे. या विभागात आपण पॉइंटर्सवर परवानगी असलेल्या विविध ऑपरेशन्सचे थोडक्यात वर्णन करू या.

पॉइंटर्सवर पुढील ऑपरेशन्सची परवानगी आहे.

1. पॉइंटर व्हेरिएबलमध्ये नंबरची (बेरीज) ऍडिशन (.Addition of a number to a pointer variable.)

समजा p हा एक पॉइंटर व्हेरिएबल आहे जो इन्टिजर प्रकारच्या घटकाकडे निर्देशित करतो, तर स्टेटमेंट

```
p++;      or      ++p;
```

p चे मूल्य 2 च्या घटकाद्वारे इन्क्रीमेंट होते, जेणेकरून ते पुढील स्थानाकडे निर्देश करते ज्यामध्ये इन्टिजर प्रकाराचे आणखी मूल्य असते. हा इन्क्रीमेंट चा घटक वर्णासाठी 1, लॉन्ग इन्टिजर आणि फ्लोटसाठी 4 आणि लॉन्ग फ्लोटसाठी 8 असेल. विधान

```
p += i;
```

जिथे i एकतर एक सकारात्मक इन्टिजर कॉन्स्टन्ट किंवा सकारात्मक मूल्य असणारा इन्टिजर व्हेरिएबल असतो, इन्क्रीमेंट p जसे की तो सध्या ज्या स्थानाकडे निर्देश करीत आहे त्या पलीकडे ith स्थानाकडे निर्देश करते.

2. पॉइंटर व्हेरिएबलमधून नंबरची वजाबाकी. (Subtraction of a number from a pointer variable.)

समजा p हा एक पॉइंटर व्हेरिएबल आहे जो इन्टिजर प्रकारच्या घटकाकडे निर्देशित करतो, तर स्टेटमेंट

$p--;$ or $--p;$

p चे मूल्य 2 च्या घटकासह डिक््रीमेंट करते, जेणेकरून ते सध्याच्या स्थानाच्या आधीच्या स्थानाकडे निर्देश करेल. विधान

$p -= i;$

3. जिथे i एकतर सकारात्मक इन्टिजर कॉन्स्टन्ट किंवा सकारात्मक मूल्य असणारा इन्टिजर व्हेरिएबल असतो, p डिक््रीमेंट होते ज्यामुळे ते सध्या ज्या स्थानाकडे निर्देश करीत आहे त्या स्थानापूर्वी ith स्थान दर्शविते.

पॉइंटर व्हेरिएबलमधून दुसऱ्या पॉइंटर व्हेरिएबलची वजाबाकी (Subtraction of one pointer variable from another.)

एका पॉइंटर व्हेरिएबलला दुसऱ्या पॉइंटर व्हेरिएबलतून वजा करता येते जर समान डेटा

प्रकार असेल. दोन्हीतला फरक संबंधित घटकांना विभक्त करणारी बाइट्सची संख्या दर्शवितो.

या अरीथमेटिक ऑपरेशन्स व्यतिरिक्त, पॉइंटर व्हेरिएबल्सची तुलना एकमेकांशी केली जाऊ शकते, बशर्ते ते सुसंगत असतील, म्हणजेच समान डेटा प्रकारांच्या व्हेरिएबल्सकडे निर्देश करतात.

पॉइंटर्सवर पुढील अरीथमेटिक क्रियांना परवानगी नाही.

- दोन पॉइंटर व्हेरिएबल्सची बेरीज.
- संख्येद्वारे पॉइंटर व्हेरिएबलचा गुणाकार.
- एका संख्येद्वारे पॉइंटर व्हेरिएबलचा भागाकार.

9.8 फंक्शन अर्ग्युमेंट म्हणून पॉइंटर्स (POINTERS AS FUNCTION ARGUMENTS)

6 व्या अध्यायात आम्ही नमूद केले होते की फंक्शनचे अर्ग्युमेंट एकतर मूल्यद्वारे कॉलद्वारे (call by value) किंवा पत्ता संदर्भानुसार कॉल (call by reference) केला जाऊ शकतो. जेव्हा आम्ही व्हॅल्यू मॅकेनिझमद्वारे कॉल वापरतो, तेव्हा फंक्शनचे औपचारिक अर्ग्युमेंट सिम्पल व्हेरिएबल म्हणून डिक्लेअर केले जातात. एक्झिक्युशनच्या वेळी वास्तविक अर्ग्युमेंटची मूल्ये औपचारिक अर्ग्युमेंटमध्ये कॉपी केली जातात (औपचारिक अर्ग्युमेंट कार्य करण्यासाठी स्थानिक असतात), आणि फंक्शनची एक्झिक्युशन सुरू होते.

तथापि, जेव्हा आम्ही संदर्भ यंत्रणेद्वारे कॉल वापरतो तेव्हा औपचारिक अर्ग्युमेंट पॉइंटर्स म्हणून डिक्लेअर केले जातात. एक्झिक्युशनच्या वेळी, वास्तविक अर्ग्युमेंटचे ऍड्रेस औपचारिक अर्ग्युमेंटमध्ये कॉपी केले जातात आणि फंक्शनची एक्झिक्युशन सुरू होते.

Listing 9.2

```
/*
    Program to illustrate pointers as function arguments
*/
#include<stdio.h>
/* function prototype */
```

```

void swap( int *x, int *y )
int main()
{
    int a = 10, b = 20;
    /* function prototype */
    void interchange( int *, int *);
    printf( "\nValue of a = %d and b = %d", a, b );
    printf( " before function call\n" );
    /* function call */
    swap ( &a, &b);
    printf( "\nValue of a = %d and b = %d", a, b );
    printf( " after function call\n" );
    return 0;
}
/* function definition */
void swap( int *x, int *y )
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

```

Test Run on Turbo C++ Compiler

```

Value of a = 10 and b = 20 before function call
Value of a = 20 and b = 10 after function call

```

9.9 पॉइंटर्स आणि स्ट्रक्चर्स (POINTERS AND STRUCTURES)

जसे आपण इतर डेटा प्रकारांसह पॉइंटर्स वापरू शकतो, तसेच स्ट्रक्चर व्हेरिएबल्स साठी पॉइंटर्स देखील वापरू शकतो. पुढील डिक्लरेशन लक्षात घ्या

```

typedef struct
{
    int code;
    char name[20];
    int dept_code;
    float salary;
} EMPLOYEE;
EMPLOYEE emp, *ptr;

```

या डिक्लरेशन युझर- डिफाइंड डेटा प्रकार तयार करतात आणि त्यास EMPLOYEE असे नाव देतात आणि employee साठी *emp* व्हेरिएबल आणि *ptr* पॉइंटर व्हेरिएबल म्हणून डिक्लेअर करतात.

खालील विधान

```
ptr = &emp;
```

पॉइंटर व्हेरिएबल *ptr* ला व्हेरिएबल *emp* चा ऍड्रेस असाइन करण्यासाठी वापरला जाऊ शकतो. आता आपण पॉइंटर व्हेरिएबल *ptr* चा वापर करून स्ट्रक्चर व्हेरिएबल *emp* च्या सदस्यांपर्यंत पोहोचू शकतो.

एरो ऑपरेटर ‘->’ वापरून स्ट्रक्चरच्या घटकांना ऍक्सेस केला जातो जो hyphen ‘-’ आणि greater-than ‘>’ या दोन कॅरेक्टर जोडतो.

एरो ऑपरेटरच्या सहाय्याने स्ट्रक्चरच्या सदस्यांपर्यंत पोहोचण्याचा सिंटॅक्स:

```
ptr->vname
```

जेथे *ptr* हे पॉइंटर व्हेरिएबल स्ट्रक्चरला निर्देशित करते आणि *vname* स्ट्रक्चरचा घटक असतात.

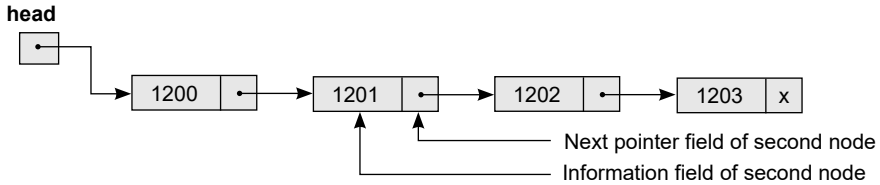
एरो ऑपरेटर वापरून पॉइंटर व्हेरिएबल *ptr* द्वारे निर्देशित प्रकारच्या employee स्ट्रक्चरतील घटक म्हणून ऍक्सेस करता येतो

```
ptr->code      ptr->name      ptr->dept_code      ptr->salary
```

9.9.1 स्वतः ची संदर्भ देणारे स्ट्रक्चर (Self-referential Structures)

एक स्वतः ची संदर्भित स्ट्रक्चर अशी एक स्ट्रक्चर आहे ज्यामध्ये कमीतकमी एक घटक समाविष्ट असतो जो त्याच्या स्वतः च्या प्रकाराचा निर्देशक असतो. या स्ट्रक्चर जटिल डेटा स्ट्रक्चर तयार करण्यासाठी त्यांचा ॲप्लिकेशन्स आढळतात जसे की linked lists, trees आणि graphs.

सेल्फ रेफरेन्शल स्ट्रक्चरचा असाच एक ॲप्लिकेशन खालीलप्रमाणे आहे.



चित्र 9.4: नोड्स 4 सह पूर्णांक मूल्यांची Linear linked list

मेमरीमध्ये वरील *linked lists* प्रतिनिधित्व करण्यासाठी आम्हाला पुढील डिक्लरेशन ची आवश्यकता आहे

```
struct NODE
{
    int info;
    struct NODE *next;
};
struct NODE *head;
```

9.10 डायनेमिक मेमरी अलोकेशन (DYNAMIC MEMORY ALLOCATION)

इन्स्ट्रक्शन्साठी मेमरी आवश्यकता नेहमी निश्चित केल्या जातात. तथापि, अशी परिस्थिती असू शकते जेव्हा डेटासाठी अचूक मेमरी आवश्यकता आधीपासूनच माहित नसते. डेटा आवश्यकता एका प्रोग्राम एक्झिक्युशन पासून दुसऱ्या प्रोग्राम एक्झिक्युशनमध्ये भिन्न असू शकतात, म्हणजे डेटासाठी मेमरी आवश्यकता डायनेमिक असतात.

अशा परिस्थितीत जेव्हा एखाद्या विशिष्ट डेटा आयटम (स) साठी आधी मेमरीची अमाऊंट माहित नसते, तेव्हा एक्जीक्यूशनच्या वेळी मेमरीचे वाटप करणे नेहमीच चांगले असते, म्हणजे जेव्हा प्रोग्राम चालू असेल. अशा प्रकारे एक्जीक्यूशनवेळी मेमरी अलोकेशन डायनामिक मेमरी अलोकेशन म्हटले जाते.

प्रत्येक प्रोग्राममध्ये अनऑलॉकटेड मेमरीचा एक पूल प्रदान केला जातो जो एक्जीक्यूशन दरम्यान वापरू शकतो. विनाअनुदानित मेमरीचा हा पूल विनामूल्य स्टोअर(free store) म्हणून ओळखला जातो. म्हणूनच जेव्हा जेव्हा प्रोग्रामद्वारे आवश्यक आकार (अमाऊंट) ची मेमरी आवश्यक असते तेव्हा ती विनामूल्य स्टोअरमधून घेतली जाते. पूर्वी वाटप केलेल्या मेमरीची पुढील आवश्यकता नसते तेव्हा ती विनामूल्य स्टोअरमध्ये परत केली जाते.

C भाषा लायब्ररी फंक्शन्स प्रदान करते, ज्यास डायनामिक मेमरी मॅनेजमेंट फंक्शन्स म्हणतात, एक्जीक्यूशनच्या वेळी मेमरीचे वाटप(अॅलोकेशन) आणि डी-वाटप(डी-अॅलोकेशन) करणे म्हणजेच डायनेमिक.

Table 9.1: मेमरी व्यवस्थापन फंक्शन्स

Function Name	Description
malloc	Allocates memory from free store.
free	Deallocates a previously allocated.

अॅलोकेशन करून, याचा अर्थ असा आहे की प्रोग्रामच्या एक्झिक्युशन दरम्यानही प्रोग्रामला आवश्यक तितकी मेमरी प्रोग्राम प्राप्त करू शकतो.

डी-अॅलोकेशनद्वारे, याचा अर्थ असा आहे की डायनेमिकली मिळवलेली मेमरी आपल्या प्रोग्रामच्या एक्झिक्युशन दरम्यान कोणत्याही वेळी सोडली जाऊ शकते.

हे लक्षात घेणे महत्वाचे आहे की डायनेमिकली मेमरीचे वाटप केले गेले आहे, आपला प्रोग्राम एक्झिक्युशन होण्यापूर्वी त्याची वाटप न करणे आवश्यक आहे. अन्यथा, जरी तुमचा प्रोग्राम संपला तरी, डायनेमिकली वाटप केलेली मेमरी कधीही स्वयंचलितपणे सोडली जात नाही आणि ऑपरेटिंग सिस्टमच्या दृष्टिकोनातून मेमरी अजूनही वापरात आहे.

म्हणूनच, जर तुम्ही तोच प्रोग्राम अनेक वेळा रन केला, तर बरेच युझर्स एकाच वेळी तुमचा प्रोग्राम वापरत असतील, ऑपरेटिंग सिस्टमची मेमरी संपू शकते.

9.10.1 मेमरी वाटप (Allocating Memory)

malloc () फंक्शन एक अर्ग्युमेण्ट घेते जो बाइटमधील ब्लॉकचा आकार निर्दिष्ट करते. हे फंक्शन हिप वरून आवश्यक आकाराची मेमरी वाटप (allocating) करते आणि पॉइंटर यशस्वीरित्या वाटप केलेल्या मेमरीला किंवा पॉइंटरमध्ये अयशस्वी झाल्यास NULL पॉइंटर (0) परत करते. अयशस्वी झाल्यास, प्रोग्रामने सुरक्षित बाहेर पडावे.

परत केलेले पॉइंटर हे टाइप void असते ज्यास इच्छित प्रकारच्या पॉइंटरमध्ये कास्ट करणे आवश्यक आहे. वाटप केलेली मेमरी अन-इनिशिएलिज्ड ठेवली आहे, म्हणजेच, सर्व ठिकाणी गारबेजचे मूल्य मिळते.

malloc() फंक्शनच्या वापरासाठी सेंटेंक्स आहे.

```
datatype *ptr
ptr = (datatype *) malloc( n * sizeof (datatype) );
```

जिथे n घटकांची संख्या असते.

उदाहरणार्थ,

```
int *ptr
ptr = (int *) malloc( 10 * sizeof(int) );
if ( ptr == NULL )
{
    printf("\nMemory allocation failed\n");
    return 1;
}
```

इंटचा आकार 16-बिट सी कंपाईलरसाठी 2 बाइट आणि 16-बिट सी कंपाईलरसाठी 4-बाइट असल्यामुळे, हे विधान 20/40 बाइट मेमरीचे ब्लॉक वाटप करेल. आणि पॉईंटर ptr ने वाटप केलेल्या मेमरीतील प्रथम बाइटचा ऍड्रेस ठेवला आहे.

9.10.2 डी-वाटप मेमरी (De-allocating Memory)

The *free()* फंक्शन मेमरीची डायनेमिक वाटप केलेले ब्लॉक डी-वाटप करते. *free()* फंक्शनच्या वापरासाठी सिंटॅक्स आहे

```
free(ptr);
```

हे एक अर्ग्युमेण्ट घेते जे वाटप केलेल्या ब्लॉकसाठी पॉईंटर निर्दिष्ट करते. हे लक्षात घेणे महत्वाचे आहे की केवळ वाटप केलेला ब्लॉक डी-वाटप केलेला आहे, पॉईंटर व्हेरिएबल हटविला(not deleted) नाही.

Listing 9.3

```
/*
    Program to illustrate the allocation of memory dynamically
    for one-dimensional array at execution time.
    Program takes an array of arbitrary size as input and finds the
    Largest element of the array
*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int *a, n, i, max;
    printf( "\nEnter size of 1D array : " );
    scanf( "%d", &n );
    /* dynamically allocate memory for array */
    a = (int *) malloc(n*sizeof(int));
    if ( a == NULL )
    {
        printf("\nMemory allocation failed.\n");
        return 1;
    }
}
```



```

printf( "\nEnter %d elements of array\n", n );
for ( i = 0 ; i < n ; i++ )
    scanf( "%d", &a[i] );
max = a[0];
for ( i = 1; i < n; i++ )
{
    if ( a[i] > max )
        max = a[i];
}
printf( "\nLargest element of array = %d\n", max );
/* de-allocate memory */
free(a);
return 0;
}

```

Test Run

```

Enter size of 1D array : 10
Enter 10 elements of array
10 35 12 9 45 11 50 20 44 32
Largest element of array = 50

```

स्पष्टीकरणात्मक उदाहरणे

Example 9.1: पॉइंटर्स वापरून अरेचे घटक प्रिंट करण्यासाठी प्रोग्राम लिहा.

Listing 9.4

```

#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a[5] = { 1, 2, 3, 4, 5 };
    int i, *ptr;
    ptr = a;
    printf("\nElements of array\n\n");
    for ( i = 0; i < 5; i++ )
        printf("%d ", *(ptr+i));
    printf("\n\n");
    return 0;
}

```

Test Run

```
Elements of array
1  2  3  4  5
```

Example 9.2: पॉइंटर वापरून फंक्शनला 1D अरे पुरवणे दर्शविण्यासाठी प्रोग्राम लिहा.

Listing 9.5

```
#include<stdio.h>
#include<stdlib.h>
void fun(int *, int); /* function prototype */
int main()
{
    int a[10] = { 5, 2, 1, 7, 9, 10, 4, 6, 8, 3 };
    /* call to function */
    fun(a,10);
    return 0;
}
/* function definition */
void fun(int *p, int n)
{
    int i;
    printf("\nElements of array\n\n");
    for ( i = 0; i < n; i++ )
        printf("%d  ", p[i]);
    printf("\n\n");
}
```

Test Run

```
Elements of array
5 2 1 7 9 10 4 6 8 3
```

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- ❑ पॉइंटर एक व्हेरिएबल आहे ज्यामध्ये ऑपरेंडचा ऍड्रेस असतो.
- ❑ पॉइंटर्स अनेक फायदे प्रदान करतात जसे की फंक्शनमधून एकापेक्षा जास्त मूल्य परत करणे, गुंतागुंतीची डेटा स्ट्रक्चर्स बनविणे, H/W शी संप्रेषण करणे इ.

- ❑ पॉइंटर्सवरील बेरीज ऑपरेशनला परवानगी आहे. वजाबाकी ऑपरेशनला प्रतिबंधित अर्थाने परवानगी आहे की त्यास सकारात्मक फरक देणे आवश्यक आहे. गुणाकार आणि भागाकार ऑपरेशनला परवानगी नाही.
- ❑ पॉइंटर्सचा उपयोग ऍड्रेसद्वारे अर्ग्युमेण्ट पास करण्यासाठी केला जातो, ज्याद्वारे संदर्भ म्हणून कॉल(call by reference) म्हणून ओळखल्या जाणाऱ्या यंत्रणा.
- ❑ स्वतःची संदर्भ देणारी (self-referential) स्ट्रक्चर अशी स्ट्रक्चर असते ज्यात स्वतःसाठी निर्देशक म्हणून कमीतकमी एक घटक समाविष्ट असतो.
- ❑ मेमरीचे वाटप दोन प्रकारे केले जाऊ शकते - स्टॅटिकली आणि डायनॅमिकली.
- ❑ स्टॅटिक मेमरी अलोकेशन हे कॅम्पयलेशन वेळी केले जाते.
- ❑ डायनॅमिक मेमरी अलोकेशन हे एक्झिक्युशन वेळी केले जाते.
- ❑ प्रत्येक प्रोग्राम अनावश्यक मेमरीचा एक पूल (तलाव) प्रदान करतो जो एक्झिक्युशन दरम्यान वापरू शकतो. विनाअनुदानित मेमरीचा हा पूल हिप किंवा विनामूल्य स्टोअर(free store) म्हणून ओळखला जातो.
- ❑ एक्झिक्युशनच्या वेळी मेमरी व्यवस्थापित करण्याची प्रक्रिया डायनॅमिक मेमरी मॅनेजमेंट म्हणून ओळखली जाते.
- ❑ `malloc ()` चा वापर डायनॅमिकली मेमरी वाटप करण्यासाठी केला जातो.
- ❑ `free()` हे `malloc ()` फंक्शनद्वारे पूर्वी वाटप केलेली मेमरी डी-वाटप करण्यासाठी वापरला जातो
- ❑ आरंभिक पॉइंटरला डॅंगलिंग / वाइल्ड पॉइंटर(dangling/wild pointer) असे संबोधले जाते कारण ते स्मृतीत कोठेही पॉइंटिंग समाप्त करू शकते.
- ❑ जेव्हा शून्य पत्त्यावर लिहिण्याचा प्रयत्न केला जाईल (ज्यास नल पॉइंटर देखील म्हणतात) किंवा सिस्टम ध्वजांकित करेल आणि प्रोग्राम समाप्त झाल्यावर “Null pointer assignment”
- ❑ मेमरी गळती ही एक प्रकारची परिस्थिती असते जेव्हा मेमरीला कॉल केलेल्या फंक्शनमध्ये वाटप केले जाते परंतु ते वापरल्यानंतर त्या फंक्शनमध्ये ते कमी होत नाही. परिणामी, जेव्हा म्हणतात फंक्शन त्याची एक्झिक्युशन पूर्ण करते, तेव्हा पॉइंटर व्हेरिएबल नष्ट होते आणि त्या वाटप केलेल्या मेमरी ब्लॉकवर पोहोचण्याचे कोणतेही साधन नसते.
- ❑ वाटप अयशस्वी ही अशी परिस्थिती असते जेव्हा सिस्टम विनामूल्य मेमरीचा आवश्यक ब्लॉक शोधण्यात सक्षम नसते. सिस्टम ही परिस्थिती एक नल पॉइंटर परत करून सूचित करते.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. व्हेरिएबलचा ऍड्रेस आपण कसा मिळवू शकतो?
2. पॉइंटर व्हेरिएबल म्हणजे काय?
3. पॉइंटर व्हेरिएबल कसे डिक्लेअर केले जाते?
4. पॉइंटर व्हेरिएबलने निर्देशित केलेल्या मूल्यांमध्ये कसे प्रवेश करता येईल?
5. पॉइंटर्सचे विविध प्रकार काय आहेत?
6. पॉइंटर्सवर कोणती ऑपरेशन्स परवानगी आहेत?
7. पॉइंटर्सवर परवानगी नसलेली ऑपरेशन्स कोणती आहेत?

एकाधिक निवड प्रश्न

- ___ ऑपरेटर वापरून व्हेरिएबलचा ऍड्रेस मिळू शकतो.
(a) * (b) & (c) ? (d) ->
- पुढील पैकी कोणते ऑपरेटर इंडिरेक्शन (डीरेफरेन्स) ऑपरेटर म्हणून ओळखला जातो?
(a) & (b) << (c) ^ (d) *
- खालील पैकी कोणते ऑपरेटर addressof म्हणून ओळखला जातो?
(a) & (b) * (c) -> (d) **
- पॉइंटर व्हेरिएबल्स ptr1, ptr2 ची योग्य डिक्लरेशन ओळखा.
(a) int ptr1, *ptr2; (b) int *ptr1, ptr2;
(c) int ptr1, ptr2; (d) int *ptr1, *ptr2;
- ऑपरेटर फक्त पॉइंटर्ससह वापरले जातात
(a) * and / (b) & and * (c) & and ^ (d) * and +
- अॅड्रेसऑफ ऑपरेटरचा ऑपरेंड _____ एक असू शकतो.
(a) कॉन्स्टन्ट (b) अर्रे घटक (c) एक्सप्रेशन (d) काहीही नाही
- पॉइंटर व्हेरिएबल्सला असाइन केले जाऊ शकतात
(a) हेक्साडेसिमल नोटेशनमध्ये दर्शविलेले मूल्य (b) ऑक्टल नोटेशनमध्ये दर्शविलेले मूल्य
(c) दुसऱ्या व्हेरिएबलचा ऍड्रेस (d) बायनरी नोटेशनमध्ये दर्शविलेले ऍड्रेसचे मूल्य
- ऑपरेंड ऑफ इंडिरेक्शन (डीरेफरेन्स) ऑपरेटर आहे
(a) पॉइंटर व्हेरिएबल (b) सामान्य व्हेरिएबल
(c) इन्टिजर कॉन्स्टन्ट (d) वरीलपैकी काहीही नाही
- पॉइंटर्ससह न वापरलेले ऑपरेटर ओळखा
(a) -> (b) & (c) * (d) >>
- जेव्हा अॅड्रेसऑफ ऑपरेटर (आणि) व्हेरिएबलला प्रिफिक्स केले गेले, तर ते मिळते
(a) व्हेरिएबलचे मूल्य (b) व्हेरिएबलचा डेटा प्रकार
(c) व्हेरिएबलचा ऍड्रेस (d) वरीलपैकी कोणताही नाही
- पॉइंटर्सवर पुढीलपैकी कोणत्या ऑपरेशनची परवानगी नाही?
(a) इन्क्रिमेंट पॉइंटर व्हेरिएबल (b) पॉइंटर व्हेरिएबल आणि संख्या यांची बेरीज
(c) पॉइंटर व्हेरिएबलला संख्येचा भागाकार (d) दोन पॉइंटर व्हेरिएबल्समधील फरक
- खालील विभाग एक्झिक्युट केल्या नंतर x चे मूल्य किती आहे?
int x = 5, *p;
p = &x;
*p = 7;
(a) 5 (b) 7 (c) Undefined (d) None

13. खालील विभागाचे आउटपुट किती असेल?

```
int a[5] = {1,2,3,4};
int *p = a;
printf("%d",*(p+2));
```

- (a) 4 (b) 0 (c) 3 (d) 2

14. पुढील प्रोग्राम विभागाचा विचार करा:

```
int *p, a[] = { 1, 2, 3, 4, 5 };
p = &a[2];
printf("%d", p[-1]);
```

वरील प्रोग्राम विभागाविषयी खालीलपैकी कोणते बरोबर आहे?

- (a) कंपाइलर टाइम लुटी कारण p एक अर्रे नाही
(b) आउटपुट 2 असेल
(c) कंपाइलर टाइम लुटी कारण सबस्क्रिप्ट नकारात्मक असू शकत नाही
(d) वरीलपैकी कोणतेही नाही

15. पुढील प्रोग्राम विभागाचा विचार करा:

```
const int i = 5;
int *p;
i = 20;
p = &i;
printf("\n%d, %d\n", *p, i);
```

वरील प्रोग्राम विभागाविषयी खालीलपैकी कोणते बरोबर आहे?

- (a) Output will be 20, 20 (b) Output will be 5, 20
(c) Output will be 20, 5 (d) कंपाइल वेळ लुटी कारण i चे मूल्य बदलू शकत नाही.

ANSWERS															
1.	(b)	2.	(d)	3.	(a)	4.	(d)	5.	(b)	6.	(b)	7.	(c)	8.	(a)
9.	(d)	10.	(c)	11.	(c)	12.	(b)	13.	(c)	14.	(b)	15.	(d)		

प्रोग्रामिंग समस्या

- पॉइंटर्स वापरून एक अर्रे दुसऱ्यावर कॉपी करण्यासाठी प्रोग्राम लिहा.
- पॉइंटर्स वापरून अर्रे उलट करण्यासाठी प्रोग्राम लिहा.
- पॉइंटर वापरून स्ट्रिंगची लेंथ शोधण्यासाठी प्रोग्राम लिहा.
- पॉइंटर्स वापरून स्ट्रिंगचा उलटा (रिव्हर्स) शोधण्यासाठी प्रोग्राम लिहा.
- पॉइंटर्स वापरून दोन स्ट्रिंगची तुलना करण्यासाठी प्रोग्राम लिहा.
- एक फंक्शन लिहा जे सर्वात लहान घटक, सर्वात मोठा घटक आणि पॉइंटर्सचा वापर करून दिलेल्या अर्रेच्या घटकांची सरासरी मिळवते. या फंक्शनच्या वापराचे प्रदर्शन करा.
- फ्लोटिंग पॉइंट नंबर प्राप्त करणारे फंक्शन लिहा आणि पॉइंटर्सचा वापर करून त्याचा अविभाज्य भाग आणि अपूर्णांक भाग परत करा. या फंक्शनच्या वापराचे प्रदर्शन करा.

8. अर्रेमध्ये डुप्लिकेट घटक आहेत किंवा पॉइंटर्स वापरत नाहीत हे शोधण्यासाठी प्रोग्राम लिहा.
9. पॉइंटर्स वापरून अर्रेमधून डुप्लिकेट घटक काढण्यासाठी प्रोग्राम लिहा.
10. पॉइंटर वापरून रचनेचे हाताळणी दर्शविण्यासाठी प्रोग्राम लिहा.
11. ज्याचा प्रोटोटाइप खाली दिलेला आहे replace() फंक्शन लिहा

```
int replace(char *str, char c1, char c2);
```

जे कॅरेक्टर c1 च्या प्रत्येक घटकास c2 अक्षरासह पुनर्स्थित करते आणि केलेल्या एकूण बदली परत करते. वरील फंक्शनची चाचणी घेण्यासाठी एक प्रोग्राम लिहा.

12. खालील स्ट्रक्चर डेफिनिशनचा विचार करा:

```
struct BOX
{
    char make[25];
    float length, breadth, height;
};
```

एक फंक्शन लिहा जो BOX स्ट्रक्चरचा ऍड्रेस घेईल आणि त्याचे व्हॉल्यूम परत करेल.

13. पुढील डिक्लरेशनचा विचार करा:

```
struct EMPLOYEE
{
    int code;
    char name[31];
    float salary;
};
```

एखाद्या संस्थेमध्ये n (£50) कर्मचारी आहेत हे दिले. एखाद्या कर्मचार्याच्या डेटामध्ये प्रवेश करण्यासाठी पॉइंटर नोटेशनचा वापर करून एक प्रोग्राम लिहा जो कर्मचार्यांच्या अधिक पगाराच्या तपशिलाचे आउटपुट देतो.

14. पुढील डिक्लरेशनचा विचार करा:

```
char *names[]={ "Vimal", "Amit", "Anuj", "Rohit", "Abhijit" };
```

पॉइंटर्स वापरून खालील आउटपुट तयार करण्यासाठी प्रोग्राम लिहा.

```
char *names[]={ "tijihbA", "tihoR", "junA", "timA", "lamiV" };
```

15. एक फंक्शन substr() लिहा ज्याचा प्रोटोटाइप खाली दिलेला आहे

```
char *substr(char *str1, char *str2);
```

जे दुसऱ्या स्ट्रिंगच्या घटनेसाठी प्रथम स्ट्रिंग स्कॅन करते आणि जिथे दुसरी स्ट्रिंग सुरू होते त्या पहिल्या स्ट्रिंगमधील घटकाला पॉइंटर परत करते. तथापि, जर पहिल्या स्ट्रिंगमध्ये दुसरी स्ट्रिंग येत नसेल तर फंक्शन NULL परत करेल. वरील फंक्शनची चाचणी घेण्यासाठी एक प्रोग्राम लिहा.

प्रॅक्टिकल

1. मर्यादित संख्येच्या ऑपरेशन्ससह linear linked lists लागू करण्यासाठी प्रोग्राम लिहा

Listing 9.6

```

/*
   Program to demonstrate the use of pointers and structures
   by implementing a linear linked list, where the insertion
   and deletions operations are limited to beginning of the l
   inked list, for simplicity.
   Program used separate function to perform each operation.
*/
#include <stdio.h>
#include <stdlib.h>
typedef struct nodeType
{
    int info;
    struct nodeType *next;
} NODE;
/*----- function prototypes -----*/
void traverse(NODE *);
void search(NODE *, int);
NODE *insert(NODE * int);
NODE *delete(NODE *);
int main()
{
    NODE *head = NULL;
    int choice, element, after;
    while ( 1 )
    {
        printf( "\n\n          Options available \n" );
        printf( "+++++ \n\n" );
        printf( " 1.  Insert\n" );
        printf( " 2.  Delete\n" );
        printf( " 3.  Search\n" );
        printf( " 4.  Traverse\n" );
        printf( " 5.  Exit\n\n" );
        printf( "Enter your choice ( 1-5 ) : " );
        scanf( "%d", &choice );
        switch ( choice )
        {

```

```
        case 1 : printf( "\nEnter element : " );
                  scanf( "%d", &element );
                  head = insert( head, element );
                  break;
        case 2 : head = delete(head);
                  break;
        case 3 : printf( "\nEnter element to search : " );
                  scanf( "%d", &element );
                  search( element );
                  break;
        case 4 : if ( head == NULL )
                      printf( "\nList is empty ... " );
                  else
                      traverse(head);
                  printf("\nPress any key to continue...");
                  getch();
                  break;
        case 5 :
                  printf("\nProgram terminated on success\n");
                  return 0;
    }
}

}

} /****** end of main function *****/

/*
function to traverse and print elements of the linked list
*/
void traverse(NODE *head)
{
    NODE *ptr = head;
    printf("\n\nLinked list\n\n");
    printf("\thead" );
    while ( ptr != NULL )
    {
        printf( " -> %d", ptr->info );
        ptr = ptr->next;
    }
}
```



```
/*
    function to search a given element in the linked list
*/
void search(NODE *head, int item)
{
    NODE *ptr = head;
    while ( ptr != NULL )
    {
        if ( item == ptr->info )
        {
            printf("\nElement %d found in linked list\n", item);
            return;
        }
        ptr = ptr->next;
    }
    printf("\nElement %d not found in linked list\n", item);
}
/*
    function to insert node in the beginning of the linked list
*/
NODE *insert(NODE *head, int item )
{
    NODE *ptr;
    ptr = ( NODE * ) malloc( sizeof( NODE ) );
    ptr->info = item;
    ptr->next = head;
    head = ptr;
    return ptr;
}
/*
    function to delete node from the beginning of the linked list
*/
NODE *delete(NODE *head)
{
    NODE *ptr;
    if ( head == NULL ) {
        printf("\nList is empty ... ");
        return;
    }
}
```

```

    printf("\nElement %d deleted from list\n", head->info);
    ptr = head;
    head = head->next;
    free(ptr);
    return head;
}

```

Test Run

```

Options available
+++++
1. Insert
2. Delete
3. Search
4. Traverse
5. Exit
Enter your choice ( 1-5 ) : 1
Enter element : 1

Options available
+++++
1. Insert
2. Delete
3. Search
4. Traverse
5. Exit
Enter your choice ( 1-5 ) : 1
Enter element : 15

Options available
+++++
1. Insert
2. Delete
3. Search
4. Traverse
5. Exit
Enter your choice ( 1-5 ) : 1
Enter element : 10

Options available
+++++
1. Insert
2. Delete
3. Search

```

```

    4. Traverse
    5. Exit
Enter your choice ( 1-5 ) : 1
Enter element : 25

    Options available
+++++
    1. Insert
    2. Delete
    3. Search
    4. Traverse
    5. Exit
Enter your choice ( 1-5 ) : 4
Linked list
    head -> 25 -> 10 -> 15 -> 1
Press any key to continue ...

    Options available
+++++
    1. Insert
    2. Delete
    3. Search
    4. Traverse
    5. Exit
Enter your choice ( 1-5 ) : 3
Enter element to search : 15
Element 15 found in linked list

    Options available
+++++
    1. Insert
    2. Delete
    3. Search
    4. Traverse
    5. Exit
Enter your choice ( 1-5 ) : 2
Element 25 deleted from list

    Options available
+++++
    1. Insert
    2. Delete
    3. Search

```

```

4.  Traverse
5.  Exit
Enter your choice ( 1-5 ) : 4
Linked list
    head -> 10 -> 15 -> 1
Press any key to continue ...

    Options available
+++++
1.  Insert
2.  Delete
3.  Search
4.  Traverse
5.  Exit
Enter your choice ( 1-5 ) : 5
Program terminated on success\n");

```

आणखी माहिती

पॉइंटर्स C भाषेचे सर्वात वेगळे आणि रोमांचक वैशिष्ट्य आहेत. हे भाषेला सामर्थ्य आणि लवचिकता प्रदान करते. सुरुवातीस, पॉइंटर्सचा वापर थोडा गोंधळात टाकणारा आणि गुंतागुंतीचा दिसू शकतो परंतु एकदा ही संकल्पना समजल्यानंतर आपण C भाषेसह बरेच काही करू शकाल.

शिक्षकांनी पॉइंटर्सच्या संकल्पना आणि C भाषेत पॉइंटर्स हाताळण्याशी संबंधित विविध पैलू समजून घेणे अपेक्षित आहे.

शिक्षकांनी देखील योग्य उदाहरणे देऊन पॉइंटर्सचा वापर दर्शविला पाहिजे आणि विद्यार्थ्यांच्या सहभागासह त्यांचे निराकरण करण्यासाठी C प्रोग्राम तयार केले पाहिजेत.

संदर्भ आणि सूचविलेले वाचन

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi..
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

10

फाईल हँडलिंग

युनिट वैशिष्ट्ये

हे युनिट फाईल्सशी संबंधित विषयांवर चर्चा करते. फायली डेटाच्या दीर्घकालीन संचयनासाठी एक साधन प्रदान करतात. काही कमांड वापरून फाइल फाईलमधून वाचली जाऊ शकते किंवा फाईलमध्ये लिहीली जाऊ शकते. हे युनिट फाईल्सचे विविध पैलू स्पष्ट करते आणि त्यांचा वापर योग्य उदाहरणांसह दाखवते.

तर्कशास्त्र

वास्तविक जीवनातील समस्यांमध्ये आपल्याला पुढीलपैकी कोणत्याही परिस्थितीचा सामना करावा लागतो:

- इनपुट डेटाचा व्हॉल्युम प्रचंड आहे.
- समान डेटावर बऱ्याच वेळा प्रक्रिया करणे आवश्यक आहे.
- एका प्रोग्रामचे आउटपुट दुसऱ्या प्रोग्रामसाठी इनपुट म्हणून वापरावे लागेल.
- डेटा मशीनद्वारे उत्पन्न केला जाऊ शकतो, जो मानवी वाचनीय नाही.

या सर्व परिस्थितीत, प्रोग्राम एक्झिक्युशन दरम्यान कीबोर्डवरून डेटा प्रविष्ट करणे आणि संगणकाच्या स्क्रीनवर निकाल देण्याची पारंपारिक पद्धत जवळजवळ अव्यवहार्य आहे. चांगला उपाय म्हणजे इनपुट डेटा एखाद्या फाईलमध्ये संग्रहित केला जाऊ शकतो, ज्याला डेटा फाईल म्हटले जाते आणि नंतर त्या फाईलमधील डेटा वाचण्यासाठी प्रोग्राम निर्देशित केला जाऊ शकतो. त्याचप्रमाणे प्रोग्रामचे आउटपुट डेटा फाईलवर लिहिण्याचेही प्रोग्रामला निर्देश दिले जाऊ शकतात. फायलींबरोबर काम करणाऱ्या या सर्व ऑपरेशन्सना फाइल हँडलिंग असे संबोधले जाते. हे युनिट विद्यार्थ्यांना फायलींचे विविध पैलू समजण्यास आणि वास्तविक जीवनातील अडचणी सोडविण्यासाठी फायलींचा वापर करून प्रोग्राम विकसित करण्यास मदत करेल.

पूर्व-आवश्यकता

- स्टॅंडर्ड इनपुट / आउटपुट (Standard Input/output)
- अनफॉर्मॅटेड आणि फॉर्मॅटेड इनपुट / आउटपुट (Unformatted and formatted input/output)
- प्रिडिफाइन्ड फाइल्स / स्टीम्स (Predefined files/streams)

युनिट निकाल

युनिट पूर्ण झाल्यावर विद्यार्थी खाली दिलेल्या मध्ये सक्षम होतील

U10-O1:

डेटा फाईलची संकल्पना आणि त्याची उपयुक्तता समजावून सांगा.

U10-O2:	फाईल्स उघडणे व बंद (opening and closing) करणे दाखवा.
U10-O3:	फायलीवर वाचन आणि लेखन (read and write) ऑपरेशन्स करा.
U10-O4:	फाईल हाताळणीसाठी कार्यक्षम प्रोग्राम लिहा.

MAPPING OF UNIT WISE LEARNING OUTCOMES WITH THE COURSE OUTCOMES

Unit 10 Outcomes	EXPECTED MAPPING WITH COURSE OUTCOMES (1- Weak Correlation; 2- Medium correlation; 3- Strong Correlation)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U10-O1	1							
U10-O2			1					
U10-O3			1					
U10-O4						2		

10.1 ओळख (INTRODUCTION)

फाईल म्हणजे संबंधित डेटाचा संग्रह. फाईलचा मुख्य उद्देश डेटाची नोंद ठेवणे आहे. संगणकाची मेमरी अस्थिर असल्याने (म्हणजे संगणक बंद होताना मेमरी सामग्री नष्ट होते), आपल्याला नंतरच्या वापरासाठी डेटा संग्रहित करण्याची आवश्यकता आहे. याव्यतिरिक्त, दिलेल्या मेमरीमध्ये संपूर्णपणे रहाण्यासाठी डेटाचे प्रमाण महत्त्वपूर्ण ठरू शकते. म्हणूनच, उर्वरित डेटा फाईलमध्येच असताना आपल्या प्रोग्राममध्ये डेटाचा एक भाग वाचण्याची आणि लिहिण्याची (read and write) क्षमता असणे आवश्यक आहे.

प्रोग्राम्सद्वारे वापरल्या जाणाऱ्या फाइल्स सामान्यतया हार्ड डिस्कवर साठवल्या जातात. जेव्हा प्रोग्राम वाचतो तेव्हा डेटा फाईलमधून मेमरीमध्ये हलविला जातो; जेव्हा ते लिहितात तेव्हा ते मेमरीमधून फाइलकडे जाते. हा डेटा हलविला गेलेला डेटा बफर म्हणून ओळखला जाणारा विशिष्ट कार्य क्षेत्र वापरतो, म्हणजे, बफर एक तात्पुरता स्टोरेज क्षेत्र आहे जो मेमरीमध्ये जातांना किंवा येतांना डेटा ठेवतो.

C भाषेमधील प्रोग्राम विविध प्रकारे फायली वाचू आणि लिहू शकतो. यातील प्रत्येक मार्ग या युनिटमध्ये सरळ स्पष्टपणे वर्णन केले आहेत आणि चांगल्या प्रकारे डिझाइन केलेल्या प्रोग्रामिंगच्या उदाहरणासह सचित आहेत

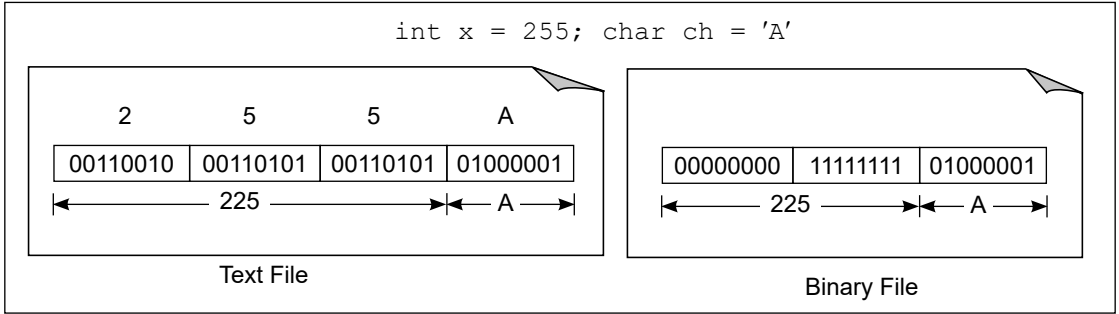
10.2 फायलीचे प्रकार (TYPES OF FILES)

C मध्ये दोन प्रकारचे फाइल्स आहेत - टेक्स्ट फाइल आणि बायनरी फाइल.

- **टेक्स्ट फाईल (Text File)** - टेक्स्ट फाईल अल्फाबेट्स, अंक आणि इतर विशिष्ट चिन्हे मध्ये त्यांची ASCII मूल्ये संचयित करून डेटा साठवते आणि ती मानवी-वाचनीय स्वरूपात आहे. जेव्हा आपण या फायली उघडता तेव्हा आपण फाईलमधील सर्व सामग्री साध्या मजकूराच्या रूपात पाहू शकता. आपण सामग्री सहजपणे एडिट किंवा डिलीट करू शकता. या फायली देखरेखीसाठी कमीतकमी मेहनत घेतात, सहज वाचता येण्यासारख्या असतात, कमीतकमी सुरक्षा प्रदान करतात आणि मोठी साठवण जागा (storage space) घेतात.
- **बायनरी फाईल (Binary File)** - बायटर्सच्या अनुक्रम म्हणून बायनरी फाईल स्टोअर डेटा जो मानवी वाचनीय स्वरूपात नाही. ते केवळ प्रोग्राम वापरून तयार केले जाऊ शकतात. ते मोठ्या प्रमाणात डेटा ठेवू शकतात, सहज वाचनीय नाहीत आणि मजकूर फायलीपेक्षा अधिक चांगली सुरक्षा प्रदान करतात.

Table 10.1: टेक्स्ट फाईल आणि बायनरी फाईल मधील फरक

Text File	Binary File
Data is stored using human-readable characters.	Data is stored in the same format as it is stored in memory.
Each line of data ends with a newline character.	There is no newline character.
There is a unique character called end-of-file (EOF) at the end of the file.	There is an end-of-file marker at the end of the file.



चित्र 10.1: फायलीमध्ये डेटा साठवण्याचे उदाहरण

10.3 फायलीवर प्रक्रिया करण्याच्या पायऱ्या (STEPS IN PROCESSING A FILE)

फाईलच्या प्रक्रियेमध्ये पुढील पायऱ्यांचा समावेश आहे

- फाईल उघडणे (Opening a file) :** फाईल प्रक्रियेची पहिली पायरी म्हणजे फाईल योग्य मोडमध्ये उघडणे. आपण आपल्या प्रोग्राममधील एखाद्या फाईलमधून वाचू इच्छित असाल तर फाईल वाचन / इनपुट मोडमध्ये उघडली पाहिजे. त्याचप्रमाणे, जर तुम्हाला तुमच्या प्रोग्रामचा आऊटपुट फाईलवर लिहायचा असेल तर फाईल राइट / आऊटपुट मोडमध्ये उघडली पाहिजे. तथापि, जेव्हा आपण वाचू इच्छित असाल आणि त्याच फाईलवर लिहा; तर, फाईल रिड-राइट मोडमध्ये उघडली जाणे आवश्यक आहे.
- फाईलमधून वाचणे किंवा लिहिणे (Reading from or writing onto a file) :** एकदा फाईल यशस्वीरित्या उघडली की डेटा फायलीवरून वाचला जाऊ शकतो किंवा फायलीवर विविध मार्गांनी लिहिला जाऊ शकतो.
- फाईल बंद करणे (Closing the file) :** फाईल प्रक्रियेची ही अंतिम पायरी आहे. एकदा आम्ही फायलीचे वाचन आणि लेखन पूर्ण केले की प्रत्येक फाईल बंद केली जाणे आवश्यक आहे.

10.3.1 फाईल उघडणे (Opening a File)

प्रोग्राम एखाद्या फाईलवर लिहिण्यापूर्वी किंवा फाईलमधून वाचण्यापूर्वी, प्रोग्रामने त्यास उघडणे आवश्यक आहे. फाईल उघडणे प्रोग्राम आणि ऑपरेटिंग सिस्टम दरम्यान समजून घेते. हे ऑपरेटिंग सिस्टमला फाईलचे नाव आणि फाईल उघडण्याच्या मोडसह उपलब्ध करते, म्हणजे वाचणे, लिहिणे किंवा जोडण्यासाठी (reading, writing, or appending).

फाईल आणि प्रोग्राम दरम्यान संप्रेषण क्षेत्रे सेट केली जातात. या क्षेत्रांपैकी एक फाईल प्रकारची स्ट्रक्चर आहे, जी हेडर फाईल `stdio.h` मध्ये डिक्लेअर केली जाते जी फाईलविषयी माहिती ठेवते.

आम्हाला पुढील डिक्लरेशनचा वापर करून प्रत्येक फाईलसाठी एक फाईल पॉइंटर डिक्लेअर करण्याची आवश्यकता आहे.

```
FILE *fp;
```

पुढे, आपण *fopen()* फंक्शनचा वापर करून फाईल उघडू शकतो.

```
fp = fopen( "filename", "mode" );
```

fopen() फंक्शन *filename* नावाची फाईल दिलेल्या मोडमध्ये उघडण्यासाठी विनंती करते. विनंती मंजूर झाल्यास ते फाईल स्ट्रक्चरला पॉइंटर परत करते; अन्यथा NULL पॉइंटर परत करते.

म्हणून एखादा प्रोग्राम *fopen()* फंक्शनद्वारे फाईल यशस्वीरित्या उघडला आहे की नाही ते तपासण्यासाठी परत केलेली व्हॅल्यू तपासू शकतो. म्हणूनच, फाईल उघडण्यासाठी जबाबदार असलेल्या प्रोग्राम सेगमेंटमध्ये लिहणे नेहमीच चांगले.

```
fp = fopen( "filename", "mode" );
if ( fp == NULL )
{
    printf( "\nUnable to open file.\n" );
    return 1;
}
```

उघडलेल्या प्रत्येक फाईलची पॉइंटर असलेली स्वतः ची FILE स्ट्रक्चर असते. कोणताही प्रोग्राम अनेक प्रकारच्या फायली उघडू शकतो.

Table 10.2: फाईल उघडण्याचे मोड्स

Mode	Description
<i>r</i>	Open a text file for reading. The file must already exist.
<i>w</i>	Open a text file for writing. If the file already exists, its contents are overwritten. If it does not exist, it will be created.
<i>a</i>	Open a text file for append. Data will be added to the end of the file. If the file does not exist, it will be created.
<i>r+</i>	Open a text file for both reading and writing. The file must already exist.
<i>w+</i>	Open a text file for both reading and writing. If the file exists, its contents are overwritten. If it does not exist, it will be created.
<i>a+</i>	Open a text file for both reading and appending. If the file does not exist, it will be created.
<i>rb</i>	Open a binary file for reading. The file must already exist.
<i>wb</i>	Open a binary file for writing. If the file already exists, its contents are overwritten. If it does not exist, it will be created.
<i>ab</i>	Open a binary file for append. Data will be added to the end of the file. If the file does not exist, it will be created.
<i>rb+</i>	Open a binary file for both reading and writing. The file must already exist.
<i>wb+</i>	Open a binary file for both reading and writing. If the file exists, its contents are overwritten. If it does not exist, it will be created.
<i>ab+</i>	Open a binary file for both reading and appending. If the file does not exist, it will be created.

10.3.2 फाईल बंद करणे (Closing a File)

जेव्हा एखादा प्रोग्राम फाईलचे वाचन / लेखन(reading/writing) समाप्त करतो, तेव्हा तो बंद करणे आवश्यक आहे. हे कार्य `fclose()` लायब्ररी फंक्शन वापरून केले आहे, ज्याचा सिंटॅक्स हा आहे.

```
fclose( fp );
```

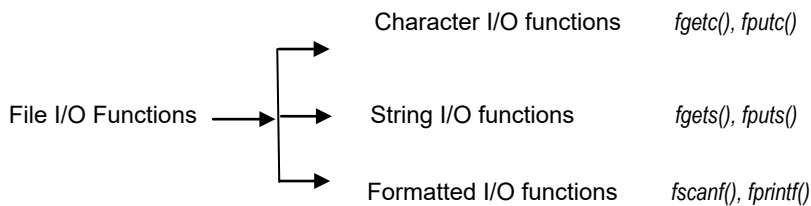
जेथे `fp` फाईल बंद केली जावी अशा फाईलशी संबंधित एक पॉइंटर आहे.

फाईल बंद केल्याने स्पष्टपणे दोन परिणाम होतात. ते हे आहेत

- बफरमध्ये उरलेला कोणताही डेटा फाईलवर लिहिला जातो. हे लक्षात घेतले जाऊ शकते की वापरलेला बफर प्रोग्रामरसाठी अदृश्य आहे.
- विशिष्ट फाईलद्वारे वापरलेल्या संप्रेषण क्षेत्रास फ्रेश करते जेणेकरून ते इतर फाईल्ससाठी उपलब्ध असतील. या भागात FILE स्ट्रक्चर आणि स्वतः बफरचा समावेश आहे.

10.3.3 टेक्स्ट फायलींचे वाचन आणि लेखन (Reading and Writing of Text Files)

टेक्स्ट फाईलवरून डेटा वाचण्याचे आणि लिहिण्याचे विविध मार्ग आहेत. ते हे आहेत



10.3.3.1 कॅरेक्टर I / O फंक्शन्स वापरून वाचन आणि लेखन (Reading and Writing using Character I/O Functions)

कॅरेक्टर I / O फंक्शन्स वापरून, डेटा एका वेळी एक कॅरेक्टर वाचला किंवा लिहिला जाऊ शकतो. हे फंक्शन्स `putchar()` आणि `getchar()` कीबोर्ड प्रमाणेच आहे. स्क्रीनवर डेटा लिहा आणि स्क्रीन वरून डेटा वाचा ह्या साठी.

फाईलवर लिहिणे (Writing to a file)

एकदा प्रोग्रामने एखाद्या विशिष्ट फाईलशी उघडण्यासाठी संप्रेषणाची ओळ स्थापित केली की ती त्यास लिहू शकते. एका वेळेस एक कॅरेक्टर लिहिणारे फंक्शनचा सिंटॅक्स हा आहे.

```
fputc( ch, fp );
```

जेथे `ch` हे कॅरेक्टर व्हेरिएबल किंवा कॉन्स्टन्ट आहे आणि `fp` एक फाईल पॉइंटर आहे.

खाली दिलेली लिस्टिंग प्रोग्राममधील उदाहरणे दाखवते जी कीबोर्डमधील टेक्स्टची ओळ स्वीकारते, एकावेळी एक कॅरेक्टर, आणि `file1.dat` नावाच्या डिस्क फाइलमध्ये लिहिते.

Listing 10.1

```

/*
   Program to demonstrate writing of one character at a time to file
*/

```

```
#include<stdio.h>
int main()
{
    FILE *fp;
    char ch;
    fp = fopen( "file1.dat", "w" );
    if ( fp == NULL ) {
        printf( "\nUnable to open file1.dat\n" );
        return 1;
    }
    printf( "\nType a line of text, when finished" );
    printf( ", when finished hit Enter key\n" );
    while ( ( ch = getche() ) != '\r' )
        fputc ( ch, fp );
    fclose( fp );
    return 0;
}
```

जेव्हा वरील प्रोग्राम एक्झिक्युट केला जातो तेव्हा तो युझरला टेक्स्टची ओळ टाईप करण्यास प्रॉम्प्ट करतो. आपण पूर्ण केल्यावर, आपण प्रोग्राम समाप्त करण्यासाठी एंटर की (↵) दाबा.

Test Run

```
C is a powerful procedural language. It provides both low-level as well
as high-level features. ↵
```

फाईलमधून वाचत आहे (Reading from a file)

जर प्रोग्राम एखाद्या फाईलवर लिहू शकत असेल तर तो फाईलमधून वाचण्यात देखील सक्षम असावा. फंक्शनचा सिंटॅक्स जे एका वेळी एक कॅरेक्टर वाचते आणि परत करते

```
ch = fgetc( fp );
```

जेथे *ch* हे कॅरेक्टर व्हेरिएबल आहे आणि *fp* एक फाईल पॉइंटर आहे.

The *fgetc ()* फंक्शन फाईलमधून वाचलेले कॅरेक्टर किंवा फाईलच्या शेवटी पोहोचले असेल तर फाईल-एंड-फाईल (EOF) कॅरेक्टर परत करते.

पुढील लिस्टिंग प्रोग्राममधील उदाहरण दाखवते जी फाईलमधील एक कॅरेक्टर वाचते आणि स्क्रीनवर लिहिते.

Listing 10.2

```
/*
    Program to demonstrate reading of character at a time from a file
*/
#include<stdio.h>
int main()
```

```

{
    FILE *fp;
    char ch;
    fp = fopen( "file1.dat", "r" );
    if ( fp == NULL ) {
        printf( "\nUnable to open file1.dat\n" );
        return 1;
    }
    printf( "\nContents of file are:\n\n" );
    while ( ( ch = fgetc (fp) ) != EOF )
        putchar( ch );
    fclose( fp );
    return 0;
}

```

जेव्हा वरील प्रोग्राम एक्झिक्युट केला जातो, तेव्हा तो फाईल file1.dat, एका वेळी एक कॅरेक्टर वाचतो, आणि त्या फाईलच्या समाप्तीपर्यंत (EOF) आढळत नाही तोपर्यंत स्क्रीनवर लिहितो.

Test Run

Contents of the file are:

C is a powerful procedural language. It provides both low-level as well as high-level features.

10.3.3.2 स्ट्रिंग I / O फंक्शन्स वापरून वाचन आणि लेखन (Reading and Writing using String I/O functions)

स्ट्रिंग I / O फंक्शन्सचा वापर करून, कॅरेक्टर स्ट्रिंगच्या रूपात डेटा वाचला किंवा लिहिला जाऊ शकतो. कॅरेक्टर स्ट्रिंगचे वाचन आणि लेखन वैयक्तिक कॅरेक्टर इतकेच सोपे आहे.

फाईलवर लिहिणे (Writing to a file)

फंक्शनचा सिंटॅक्स जो एका वेळी कॅरेक्टरची स्ट्रिंग लिहितो.

```
fputs( str, fp );
```

जिथे *str* कॅरेक्टरचा अर्रे किंवा स्ट्रिंग कॉन्स्टन्ट असतो आणि *fp* फाईल पॉइंटर असते.

पुढील लिस्टिंग प्रोग्रामिंगचे उदाहरण दर्शविते जी कीबोर्डवरील स्ट्रिंगची मालिका(series) स्वीकारते आणि डिस्क फाइलवर लिहिते.

Listing 10.3

```

/*
    Program to demonstrate writing of strings to a file
*/
#include<stdio.h>
#include<string.h>
int main()

```

```

{
    FILE *fp;
    char str[81];
    fptr = fopen( "file2.dat", "w" );
    if ( fp == NULL ) {
        printf( "\nUnable to open file2.dat\n" );
        return 1;
    }
    printf( "\nEnter a set of strings, press just" );
    printf( " Enter key to finish\n" );
    while ( strlen( gets( str ) ) > 0 )
    {
        fputs( str, fp );
        fputs( "\n", fp );
    }
    fclose( fp );
    return 0;
}

```

Test Run

```

C is a powerful procedural language. ↵
It is a middle-level language. ↵
All Programs are tested using Turbo C Compiler. ↵
↵

```

जेव्हा वरील प्रोग्राम एक्झिक्युट केला जातो तेव्हा तो एंटर की दाबून समाप्त केलेल्या स्ट्रिंगचा एक संच स्वीकारतो. आपण पूर्ण केल्यावर, सुरुवातीस काहीही न देता एंटर की दाबा, जे प्रोग्राम समाप्त करण्यासाठी 0 लेंथची स्ट्रिंग, म्हणजेच नल स्ट्रिंग म्हणून घेतले जाते.

वरील प्रोग्राममध्ये, आपण स्ट्रिंग स्टोअर करण्यासाठी कॅरेक्टरची अॅरे सेट केली आहेत. `fputs()` फंक्शन नंतर अॅरेची सामग्री डिस्कवर लिहिते. स्ट्रिंगच्या शेवटी `fputs()` फंक्शन आपोआप न्यूलाईन कॅरेक्टर जोडत नसल्यामुळे फाईल मधून स्ट्रिंग पुन्हा वाचणे आपण सुस्पष्टपणे केले पाहिजे.

फाईलमधून वाचणे (Reading from a file)

फाईलमधील स्ट्रिंग वाचणारे फंक्शनचा सिंटॅक्स हा आहे.

```
fgets( str, n, fp );
```

जिथे `str` ही कॅरेक्टरचा अॅरे असते आणि जिथे स्ट्रिंग संग्रहित केला जातो तिथे ऍड्रेस निर्दिष्ट करते, `n` हे इनपुट स्ट्रिंगची कमाल लेंथ आहे आणि `fp` फाईल पॉइंटर आहे.

`fgets()` फंक्शन जेव्हा फाईलच्या शेवटी-ईओएफ वाचते तेव्हा एक NULL मूल्य मिळवते.

खाली दिलेली लिस्टिंग प्रोग्रामचे उदाहरण दाखवते जी फाईलमधून एकावेळी एक स्ट्रिंग वाचते आणि स्क्रीनवर लिहिते.

Listing 10.4

```

/*
    Program to demonstrate reading of strings from a file
*/
#include<stdio.h>
#include<string.h>
int main()
{
    FILE *fp;
    char str[81];
    fp = fopen( "file2.dat", "r" );
    if ( fp == NULL ) {
        printf( "\nUnable to open file2.dat\n" );
        return 1;
    }
    printf( "\nContents of file are:\n\n" );
    while ( fgets( str, 80, fp ) != NULL ) {
        puts( str );
    }
    fclose( fp );
    return 0;
}

```

जेव्हा वरील प्रोग्राम एक्झिक्युट होतो, तेव्हा तो एकावेळी file3.dat मधील एक स्ट्रिंगची सामग्री वाचतो आणि EOF येण्यापर्यंत स्क्रीनवर लिहितो.

Test Run

```

Contents of the file are:

C is a powerful procedural language.
It is a middle-level language.
All Programs are tested using Turbo C Compiler.

```

10.3.3.3 फॉर्मॅटेड I / O फंक्शन्स वापरून वाचन आणि लेखन (Reading and Writing using Formatted I/O functions)

आतापर्यंत आपण कॅरेक्टर , स्ट्रिंग आणि इन्टिजर संख्या वाचणे आणि लिहिणे यावर विचार केला आहे. वास्तविक संख्या काय? आणि मिश्रित प्रकारच्या डेटाचे काय? उदाहरणार्थ, समजा आम्हाला त्याचे नाव (एक स्ट्रिंग), कोड नंबर (इन्टिजर संख्या) आणि उंची (एक नैसर्गिक संख्या) असलेल्या एजंटबद्दल माहिती संग्रहित करायची आहे. एजंट्सच्या दिलेल्या सूचीसाठी आम्हाला डेटा फाईल तयार करायची आहे. हे फॉर्मॅटेड I / O फंक्शन्स वापरून करता येते.

फाईलवर लिहिणे (Writing to a file)

फाईलवर फॉर्मेटेड डेटा लिहिणाऱ्या फंक्शनचा सिंटॅक्स आहे

```
fprintf( fp, "format-string" , ditems );
```

जिथे *fp* फाईल पॉइंटर असते आणि फाईलवर ditems लिहिल्या जाणाऱ्या व्हेरिएबल्सची यादी असते. *fprintf()* हे *printf()* फंक्शनसारखेच आहे; परक इतकाच आहे की *printf()* फंक्शन फॉर्मेटेड डेटा स्क्रीनवर लिहितो.

Listing 10.5

```
/*
   Program to demonstrate writing of formatted data to a file
*/
#include<stdio.h>
int main()
{
    FILE *fp;
    char yes_no;
    char name[41];
    int code;
    float height;
    fp = fopen( "file3.dat", "w" );
    if ( fp == NULL ) {
        printf( "\nUnable to open file3.dat\n" );
        return 1;
    }
    while(1)
    {
        printf( "\nEnter name, code number, height" );
        scanf( "%s,%d,%f", name, &code, &height );
        fprintf( fp,"%s,%d,%f", name, code, height );
        printf( "Any more input y/n?: " );
        yes_no = tolower( getche() );
        fflush ( stdin );
        if ( yes_no == 'n' )
            break;
    }
    fclose( fp );
    return 0;
}
```

Test Run

```

Enter name, code number, height
Geetu, 10, 160.25
Any more input y/n?: y
Enter name, code number, height
Shivani, 11, 158.5
Any more input y/n?: y
Enter name, code number, height
Vijay, 12, 165.5
Any more input y/n?: y
Enter name, code number, height
Gurpreet, 13, 166.25
Any more input y/n?: n

```

वरील माहिती आता *file3.dat* नावाच्या फाईलमध्ये आहे. जर आपण त्याकडे TYPE कमांडचा वापर करून पहाण्याचा प्रयत्न केला तर डेटामध्ये *newlines* नसल्यामुळे संपूर्ण आउटपुट त्याच लाईनवर जाईल. आउटपुट अधिक सोयीस्करपणे फॉर्मेट करण्यासाठी, फॉर्मेट इनपुट वापरून वरील फाईल वाचण्यासाठी आपण प्रोग्राम लिहू शकतो.

फाईलमधून वाचणे (Reading from a file)

फाईलमध्ये फॉर्मॅटेड डेटा वाचणाऱ्या फंक्शनचा सिंटॅक्स आहे

```
fscanf( fp, "format-string" , ditems );
```

जेथे *fptr* फाईल पॉइंटर असते आणि *ditems* म्हणजे ऍड्रेसची यादी असते जिथे फाईलमधून वाचलेली व्हॅल्यूज संग्रह करायची असतात.

Listing 10.6

```

/*
   Program to demonstrate writing of formatted from a file
*/
#include<stdio.h>
int main()
{
    FILE *fp;
    char yes_no, char name[41];
    int code;
    float height;
    fp = fopen( "file3.dat", "r" );

```

```

if ( fp == NULL ) {
    printf( "\nUnable to open file3.dat\n" );
    return 1;
}
printf( "\nContents of file are:\n\n" );
while( fscanf(fp,"%s,%d,%f", name, &code, &height) != EOF ) {
    printf( "%s %d %f\n", name, code, height );
}
fclose( fp );
return 0;
}

```

Test Run

```

Contents of file are:
Geetu 10 160.25
Shivani 11 158.5
Vijay 12 165.5
Gurpreet 13 166.25

```

10.3.4 बायनरी फाइल्सचे वाचन आणि लेखन (Reading and Writing of Binary Files)

C भाषा रेकॉर्ड I/O प्रदान करते, ज्यास कधीकधी ब्लॉक I/O म्हणतात, बायनरी फायलींवर डेटा वाचण्यासाठी आणि लिहिण्यासाठी कार्य करते.

रेकॉर्ड I/O डिस्क फायलींवर बायनरी स्वरूपात क्रमांक लिहितो जेणेकरून इन्टिजर दोन बाइट्समध्ये संग्रहित होईल; लॉन्ग इन्टिजर चार बाइट्समध्ये, सिंगल - प्रिसिजन फ्लोटिंग-पॉइंट्स चार बाइट्समध्ये आणि डबल-प्रिसिजन फ्लोटिंग-पॉइंट्स आठ बाइट्समध्ये संग्रहित केले जातात - हेच स्वरूप मेमरीत नुमेरिक डेटा संग्रहित करण्यासाठी वापरले जाते.

रेकॉर्ड I/O एकाच वेळी डेटा वाचण्यास आणि लिहिण्यास परवानगी देतो; सिंगल कॅरेक्टर किंवा स्ट्रिंग किंवा काही मूल्यांमध्ये मर्यादित नाही. अॅरे, स्ट्रक्चर्स, स्ट्रक्चर्सचे अॅरे युनिट म्हणून वाचू आणि लिहिले जाऊ शकतात.

फाईलवर लिहिणे (Writing to a file)

फंक्शनचा सिंटॅक्स जो एका वेळी डेटा ब्लॉक लिहितो

```
fwrite( ptr, m, n, fp );
```

जेथे *ptr* हा अॅरेचा किंवा लिखित असलेल्या स्ट्रक्चरचा ऍड्रेस असेल तर, *m* हे अॅरे किंवा स्ट्रक्चरचा आकार असेल तर, *n* अशा अॅरे किंवा स्ट्रक्चर लिहिण्यासाठीची संख्या आहे आणि *fp* हे लिहिण्यासाठी बायनरी मोडमधील ओपन पॉइंटर.

ब्लॉक लिहिल्यानंतर, *fwrite()* फंक्शन प्रत्यक्षात लिहिलेल्या डेटा आयटमची संख्या परत करते. विनंती केलेल्या लेखी आयटमची संख्या कमी असल्यास, याचा अर्थ असा की काही लुटी आली आहे.

अरे लिहिणे (Writing Arrays)

समजा आपल्याकडे *file4.dat* नावाच्या फाईलमध्ये 10 घटक असलेले इंटिजर अरे मध्ये संग्रहित करायचे आहेत.

Listing 10.7

```
/*
    Program to demonstrate writing of an entire array to a file
*/

#include <stdio.h>
int main()
{
    FILE *fp;
    int i, a[10];
    fp = fopen( "file4.dat", "wb" );
    if ( fp == NULL ) {
        printf( "\nUnable to open file4.dat\n" );
        return 1;
    }
    printf( "\nEnter ten values\n" );
    for ( i = 0; i <= 10; i++ )
        scanf( "%d", &a[i] );
    fwrite(a, sizeof(a), 1, fp); /* write entire array to file */
    fclose( fp );
    return 0;
}
```

एक्झिक्युट केल्यावर, हा प्रोग्राम युझरला दहा इन्टिजर मूल्य प्रविष्ट करण्यास प्रॉम्प्ट करतो आणि नंतर *file4.dat* नावाच्या डिस्क फाईलवर संपूर्ण अरे लिहितो.

स्ट्रक्चर लिहिणे (Writing Structures)

समजा आम्हाला एजंट नावाची स्ट्रक्चर लिहायचे आहे ज्याचे घटक – नाव (जास्तीत जास्त 40 कॅरेक्टरस), कोड (जास्तीत जास्त 5 कॅरेक्टरस) आणि उंची (रिअल संख्या) *file5.dat* नावाच्या डिस्क फाईलमध्ये लिहायची आहे.

Listing 10.8

```
/*
    Program to demonstrate writing of an entire structure to a file
*/
#include<stdio.h>
struct
```

```

{
    char name[41];
    char code[6];
    float height;
} agent;
int main()
{
    FILE *fp;
    char yes_no, numstr[40];
    fp = fopen( "file5.dat", "wb" );
    if ( fp == NULL ) {
        printf( "\nUnable to open file5.dat\n" );
        return 1;
    }
    while ( 1 ) {
        printf( "\nEnter name : " );
        gets( agent.name );
        printf( "\nEnter code number : " );
        gets( agent.code );
        printf( "\nEnter height : " );
        gets( numstr );
        agent.height = atof ( numstr );
        fwrite( &agent, sizeof(agent), 1, fp );
        printf( "Any more input y/n?: " );
        yes_no = tolower( getche() );
        fflush( stdin );
        if ( yes_no == 'n' )
            break;
    }
    fclose( fp );
    return 0;
}

```

एन्ड्रिक्सक्युट केल्यावर, हा प्रोग्राम युझरला एजंटची तपशील प्रविष्ट करण्यास सांगेल, त्यांना एका स्ट्रक्चरमध्ये साठवतो, आणि नंतर हा स्ट्रक्चर सिंगल राइट ऑपरेशनमध्ये *file5.dat* नावाच्या डिस्क फाइलवर लिहितो.

स्ट्रक्चर व्हेरिएबल ऐवजी एजंट *n* घटकांसह स्ट्रक्चर्सचा अर्रे असतो आणि आपल्याला सिंगल राइट ऑपरेशनमध्ये संपूर्ण अर्रे लिहायच्या आहेत.

हे कार्य *fwrite()* फंक्शन लिहून पूर्ण केले जाऊ शकते

```
fwrite( agent, sizeof(agent[0]), n, fp );
```

वरील प्रोग्राम संख्यात्मक डेटाचे इनपुट करण्याचा आणखी एक मार्ग दर्शवितो. संख्यात्मक डेटा देखील एक स्ट्रिंग म्हणून वाचू शकतो आणि नंतर योग्य मूल्यावर रूपांतरित करू शकतो. लायब्ररी फंक्शन `atoi()` इन्टिजर स्ट्रिंगला इन्टिजर मूल्यामध्ये रूपांतरित करते, तर फंक्शन `atof()` रिअल स्ट्रिंगला रिअल संख्येमध्ये रूपांतरित करते. त्यांचे समकक्ष म्हणजे `itoa()` आणि `fcvt()` आहेत, जे इन्टिजर मूल्य आणि रिअल मूल्य योग्य स्ट्रिंगमध्ये रूपांतरित करतात. ही फंक्शन्स `stdlib.h` हेडर फाईल्स मध्ये डिफाईंड केली आहेत. प्रश्न उद्भवू शकतो - ही रूपांतरणे का आवश्यक आहेत?

याची दोन कारणे आहेत.

- कधीकधी आपल्याला स्ट्रिंगसह संख्यात्मक मूल्य एकल करण्याची आवश्यकता असू शकते आणि हे केवळ तेव्हाच शक्य होईल जेव्हा संख्यात्मक मूल्य स्ट्रिंगमध्ये रूपांतरित केले जाईल.
- C सिस्टम `scanf()` फंक्शनसह रिअल नंबरच्या इनपुटशी विसंगत आहे. काहीवेळा तो एक त्रुटी संदेश देतो - फ्लोटिंग पॉइंट स्वरूप जोडलेला नाही.

वरील प्रोग्राममध्ये वापरलेले आणखी एक फंक्शन म्हणजे `fflush()` फंक्शन. हे फंक्शन अनावश्यक डेटा फ्लश करण्यासाठी वापरला जातो जो कीबोर्ड बफरमध्ये सोडला जाऊ शकतो. आपण कदाचित असे पाहिले असेल की काही वेळा काही इनपुट फंक्शनच्या कॉलला प्रतिसाद म्हणून सिस्टम विशिष्ट इनपुटसाठी थांबली नाही. हे कीबोर्ड बफरमध्ये उरलेल्या काही अवांछित डेटामुळे आहे. `fflush()` फंक्शन वापरून पुढचे इनपुट घेण्यापूर्वी आपण ते फ्लश करू या.

फाईलमधून वाचणे (Reading from a file)

फाईलमधील ब्लॉक वाचणाऱ्या फंक्शनचा सिंटॅक्स

```
fread( ptr, m, n, fp );
```

जेथे `ptr` हा अरेचा किंवा स्ट्रक्चरचा प्‌ट्रेंस आहे जेथे ब्लॉक वाचल्यानंतर संग्रहित केला जाईल, `m` अरेचा आकार किंवा वाचण्याजोगी स्ट्रक्चरचा आकार आहे, `n` वाचण्याजोगी अशा अरे किंवा स्ट्रक्चर्सची संख्या आहे, आणि `fp` वाचनासाठी बायनरी मोडमध्ये उघडलेल्या फाईलचे फाईल पॉइंटर आहे.

`fread()` फंक्शन वाचलेल्या वास्तविक डेटा आयटमची संख्या परत करते. पुढील प्रोग्राममध्ये या फंक्शनचा उपयोग स्पष्ट केला आहे.

Listing 10.9

```
/*
   Program to demonstrate reading of entire structure from a file
*/
#include<stdio.h>
struct
{
    char name[41];
    char code[6];
    float height;
} agent;
int main()
{
    FILE *fp;
```

```

int record_no = 0;
fp = fopen( "file5.dat", "rb" );
if ( fp == NULL )
{
    printf( "\nUnable to open file5.dat\n" );
    return 1;
}
while ( fread(&agent,sizeof(agent),1,fp) > 0 )
{
    printf( "\nRecord #%d\n", record_no );
    printf( "\nName : %s", agent.name );
    printf( "\nCode number : %s", agent.code );
    printf( "\nHeight : %.2f", agent.height );
    record_no++;
    printf("\n\nPress any key to see next record..." );
    getch();
}
fclose( fp );
return 0;
}

```

वरील प्रोग्राम *listing 10.8* तयार केलेली डेटा फाईल वाचतो हे एका वेळी एक स्ट्रक्चर वाचते आणि ती स्क्रीनवर प्रदर्शित करते. जेव्हा *fread()* फंक्शन 0 व्हॅल्यू देईल तेव्हा व्हाइल लूप संपेल, म्हणजे ब्लॉक वाचू शकत नाही. हे फाईलचा शेवट दर्शवते.

10.4 फाईल पोजिशनिंग फंक्शन्स (FILE POSITIONING FUNCTIONS)

फाईल पोजिशनिंग ऑपरेशन्स हाताळण्यासाठी C भाषा खालील फंक्शन्स पुरवते:

- फाईलच्या सुरुवातीस फाईल पॉइंटर सेट करण्यासाठी फंक्शन *rewind()* .
- फाईलमधील फाईल पॉइंटरची सद्य स्थिती जाणून घेण्यासाठी फंक्शन *ftell()*.
- फाईलमधील फाईल पॉइंटरची स्थिती बदलण्यासाठी फंक्शन *seek()*

10.4.1 रिवाइंड फाइल: *rewind()* फंक्शन

फाईलच्या सुरुवातीस फाईल पॉइंटरला स्थितीत ठेवण्याचा एक मार्ग म्हणजे फाईल बंद करणे आणि पुन्हा ती पुन्हा उघडणे. तथापि, आपण हे काम रिवाइंड फंक्शन वापरून फाइल बंद न करता पूर्ण करू शकतो. हे फंक्शन फाईलच्या सुरुवातीस फाईल पॉइंटरला स्थान देते. हे फंक्शन अगदी सुरुवातीपासूनच कॅसेट ऐकण्यासाठी किंवा पाहण्यासाठी ऑडिओ किंवा व्हिडिओ कॅसेट रिवाइंड बटण सारखे काम करते.

रिवाइंड () फंक्शनचा सिंटॅक्स आहे.

```
rewind(fp);
```

जेथे *fp* सध्या उघडलेल्या फाईलसाठी फाइल पॉइंटर आहे.

10.4.2 सद्य स्थान (Current Location): `ftell ()` फंक्शन

काही घटनांमध्ये फाईलमध्ये असलेल्या फाईल पॉइंटरचे सद्य स्थान शोधण्याची आवश्यकता असू शकते. `ftell ()` फंक्शन मुळे आपण फाईल पॉइंटरची सद्य स्थिती जाणून घेऊ शकतो.

`ftell()` फंक्शनचा सिंटॅक्स आहे.

```
long k = ftell(fp);
```

जिथे `fp` सध्या उघडलेल्या फाईलसाठी एक फाईल पॉइंटर आहे तेथे `ftell ()` फंक्शन एक लॉन्ग इन्टिजर दाखवेल. हे आवश्यक आहे कारण बऱ्याच फायलींमध्ये 32767 बाइटपेक्षा जास्त डेटा असू शकतो.

लक्षात ठेवा की सी I/O सिस्टम फायलींना डेटा बाइटचे प्रवाह मानते. हे फाईलच्या प्रारंभापासून शून्य, म्हणजेच, बाइटच्या संख्येने फाईलमधील स्थितीचे मोजमाप करते. जेव्हा फाईल पॉइंटर फाईलच्या सुरुवातीला असतो, `ftell ()` फंक्शन 0 रिटर्न करते. फाईल पॉइंटर दुसऱ्या बाइटवर असल्यास, `ftell ()` फंक्शन व्हॅल्यू 1 देते.

10.4.3 स्थान सेट करा (Set Position): `fseek ()` फंक्शन

फाईलमध्ये कुठूनही डेटा आयटम वाचण्यासाठी, आपल्याला फाईल पॉइंटर त्या डेटा आयटमच्या सुरुवातीस हलवावे लागेल. हे कार्य पूर्ण करण्यासाठी आपण `fseek ()` फंक्शन वापरू शकतो.

`fseek()` फंक्शनचा सिंटॅक्स आहे.

```
fseek(fp, offset, wherefrom);
```

फंक्शन तीन आर्ग्युमेंट्स घेते, जिथे प्रथम आर्ग्युमेंट `fp` फाईल पॉइंटर असते, तर दुसरे आर्ग्युमेंट `offset` लॉन्ग इन्टिजरचे प्रकार असते जे फाईल पॉइंटर ने जायचे त्या बाइट्सची संख्या निर्दिष्ट करते. तिसरा आर्ग्युमेंट `wherefrom` ज्यामधून ऑफसेट कोणत्या स्थानावरून मोजले जाते ते निर्दिष्ट करते.

आर्ग्युमेंटची विविध मूल्ये Table 10.3. मध्ये दिलेली आहेत.

Table 10.3: `fseek ()` फंक्शनसाठी `wherefrom` विविध मूल्ये

Mode	Offset is measured from
SEEK_SET	Beginning of the file
SEEK_CUR	Current position of the file
SEEK_END	End of the file

Table 10.4: `fseek ()` फंक्शनचा वापर स्पष्ट करणारी काही उदाहरणे

Seek Call	Action performed
<code>fseek(fp, 0, SEEK_SET);</code>	Moves the file pointer <code>fp</code> to the beginning of the file. If the file pointer is currently at the beginning of the file, it results in no action.
<code>fseek(fp, n, SEEK_SET);</code>	Moves the file pointer <code>fp</code> forward by <code>n</code> bytes, i.e., moves the file pointer to <code>(n+1)</code> the bytes in the file.
<code>fseek(fp, -n, SEEK_CUR);</code>	Moves the file pointer <code>fp</code> backward by <code>n</code> bytes from the current position.

<code>fseek(fp, 0, SEEK_END);</code>	Moves the file pointer <code>fp</code> to the end of the file. If the file pointer is currently at the end of the file, it results in no action.
<code>fseek(fp, 0, SEEK_END);</code>	Moves the file pointer <code>fp</code> to the end of the file. If the file pointer is currently at the end of the file, it results in no action.
<code>fseek(fp, n, SEEK_CUR);</code>	Moves the file pointer <code>fp</code> forward by <code>n</code> bytes from the current position.
<code>fseek(fp, 1, SEEK_CUR);</code>	Moves the file pointer <code>fp</code> to the next byte.
<code>fseek(fp, -1, SEEK_CUR);</code>	Moves the file pointer <code>fp</code> to the previous byte.
<code>fseek(fp, m, SEEK_END);</code>	Moves the file pointer <code>fp</code> backward by <code>m</code> bytes from the end of the file.

10.5 फाइल स्थिती फंक्शन (FILE STATUS FUNCTIONS)

C भाषा फाइल स्थिती बाबत केरी हाताळण्यासाठी खालील फंक्शन्स पुरवते:

- फाइलचा शेवट तपासण्यासाठी फंक्शन `feof()`.
- त्रुटी तपासण्यासाठी फंक्शन `ferror()`.
- त्रुटी साफ करण्यासाठी फंक्शन `clearerr()`.

10.5.1 फाइलचा शेवट तपासण्यासाठी (Test End of File): *feof()* फंक्शन

feof() फंक्शन फाइलचा शेवट गाठला गेला आहे का हे तपासण्यासाठी वापरला जातो. फाइल पॉइंटर शेवटी असल्यास, म्हणजेच, सर्व डेटा वाचला गेला आहे, फंक्शन व्हॅल्यू 1 परत करेल. जर फाइलचा शेवट गाठला नाही तर तो व्हॅल्यू 0 परत करेल.

feof() फंक्शनचा सिंटॅक्स आहे.

```
feof(fp);
```

जेथे *fp* सध्या उघडलेल्या फाइलसाठी फाइल पॉइंटर आहे.

10.5.2 चाचणी त्रुटी (Test Error): *ferror()* फंक्शन

The *ferror()* फंक्शन फाइलच्या त्रुटी स्टेटसची चाचणी घेण्यासाठी वापरले जाते. आधी सांगितल्याप्रमाणे, खराब मीडिया (डिस्क, सीडी, इत्यादी) पासून लिहिलेल्या अवस्थेत फाइल वाचण्यासारख्या बेकायदेशीर ऑपरेशन्सपर्यंत अनेक कारणांमुळे त्रुटी निर्माण केल्या जाऊ शकतात.

समजा *ferror()* फंक्शन 1 परत करते जर फाइल ऑपरेशन नंतर त्रुटी आली असेल. कोणतीही त्रुटी आली नसल्यास, फाइल *ferror()* मूल्य 0 परत करते.

ferror() फंक्शनचा सिंटॅक्स आहे.

```
ferror(fp);
```

जेथे *fp* सध्या उघडलेल्या फाइलसाठी फाइल पॉइंटर आहे.

येथे हे लक्षात घेणे महत्वाचे आहे की एखाद्या त्रुटीची चाचणी केल्याने त्रुटीची स्थिती रीसेट होत नाही. एकदा एखादी त्रुटी आली की ती नंतर वर्णन केलेल्या *clearerr()* फंक्शनचा वापर करून त्रुटीची स्टेट क्लीयर केल्यानंतर केवळ स्टॅंडर्ड वाचन किंवा लेखन स्थितीवर परत येऊ शकते.

13.5.3 क्लियर त्रुटी (Clear Error): `clearerr()` Function

जेव्हा एखादी त्रुटी उद्भवली जाते, तेव्हापर्यंतच्या `ferror()` फंक्शन रिटर्न 1 जोपर्यंत नंतरची कॉल फाईलची त्रुटी स्थिती रीसेट होईपर्यंत परत येते नाही. `clearerr()` फंक्शन या उद्देशाने वापरले जाते.

`clearerr()` फंक्शनचा सिंटॅक्स आहे.

```
clearerr(fp);
```

जेथे `fp` सध्या उघडलेल्या फाईलसाठी फाईल पॉइंटर आहे.

येथे हे लक्षात ठेवणे महत्वाचे आहे की आपण त्रुटी साफ केल्या असूनही आपण समस्या दूर झाली असणं आवश्यक नाही. आपल्याला असे आढळले आहे की त्यानंतरचे वाचन किंवा लेखन ऑपरेशन त्रुटी स्थितीत परत येऊ शकतात.

स्पष्टीकरणात्मक उदाहरणे

Example 10.1: दिलेल्या फाईलचा आकार शोधण्याचा प्रोग्राम लिहा, जेथे युझर फाईलचे नाव प्रदान करतो.

निर्दिष्ट फाईलचा आकार जाणून घेण्यासाठी, फाईलच्या शेवटी फाईल पॉइंटरला `fseek()` फंक्शन वापरा आणि फाईलचा आकार देणाऱ्या `ftell()` फंक्शनचा वापर करून फाईल पॉइंटरच्या मूल्यामध्ये प्रवेश करा.

Listing 10.10

```
/*
   Program to find the size of a given file
*/
#include <stdio.h>
int main()
{
    FILE *fptr;
    char fname[30];
    printf("\nEnter name of file : ");
    gets(fname);
    fptr = fopen(fname, "r");
    if (!fptr) {
        printf("\nFile %s does not exist\n", fname);
        return 1;
    }
    fseek(fptr, 0L, 2);
    printf("\nSize of file = %ld bytes.\n", ftell(fptr));
    fclose(fptr);
    return 0;
}
```

Test Run

```
Enter name of file : fsize.c
Size of file = 443 bytes.
```

वरील टेस्ट रन दर्शविते की Listing 10.7 मध्ये तयार केलेल्या प्रोग्राम फाईलचा आकार (सोर्स कोड) 443 बाइट आहे..

Example 10.2: प्रोग्राम लिहा जो कीबोर्डवरून काही मजकूर वाचतो आणि त्यास टेक्स्ट फाईलमध्ये लिहितो. प्रोग्राम नंतर ही फाईल वाचतो आणि त्यातील मजकूर स्क्रीनवर प्रदर्शित करतो.

Listing 10.11

```
/*
   Program that creates a file and then reads its contents
   and display them on the computer screen
*/
#include <stdio.h>
int main()
{
    char ch;
    FILE *fp1;
    fp1 = fopen("TEXT", "w");    /* open file for writing */
    if ( fp1 == NULL ) {
        printf("\nUnable to open file TEXT for writing\n");
        return 1;
    }
    printf("\nType some text and terminate");
    printf(" the input by Enter key\n\n");
    while ( ( ch = getche() ) != '\r' )
        fputc(ch, fp1);
    fclose(fp1);    /* close file */
    fp1 = fopen("TEXT", "r");    /* open file for reading */
    if ( fp1 == NULL ) {
        printf("\nUnable to open file TEXT for reading\n");
        return;
    }
    printf("\n\nContents of file TEXT are\n\n");
    while ( !feof(fp1) ) {
        ch = fgetc(fp1);
        putchar(ch);
    }
    fclose(fp1);
    return 0;
}
```


Test Run

```
Type some text and terminate the input by Entering key
Don't do anything with others that you wish others should not
do with you. ↵
Contents of file TEXT are
Don't do anything with others that you wish others should not
do with you.
```

Example 10.3: बाईट-बाय-बाईट एका फाईलमधील मजकूर कॉपी करण्यासाठी एक प्रोग्राम लिहा. युझर फायलीची नावे पुरवतो.

Listing 10.12

```
/*
    Program to copy the contents of a given file to another file.
*/
#include <stdio.h>
int main()
{
    char ch;
    FILE *fp1, *fp2;
    char file1[30], file2[30];
    printf("\nEnter name of source file : ");
    gets(file1);
    printf("\nEnter name of destination file : ");
    gets(file2);
    fp1 = fopen(file1, "r");
    if ( fp1 == NULL ) {
        printf("\nUnable to open file: %s for reading\n", file1);
        return 1;
    }
    fp2 = fopen(file2, "w");
    if ( fp2 == NULL ) {
        printf("\nUnable to open file: %s for writing\n", file2);
        return 1;
    }
    while ( !feof(fp1) ) {
        ch = fgetc(fp1);
        fputc(ch, fp2);
    }
}
```

```
printf("\nFile copied successfully...\n");
fclose(fp1);
fclose(fp2);
return 0;
}
```

Test Run

```
Enter the name of the source file: file_copy.c
Enter the name of destination file: temp
File copied successfully...
```

आपण फाइल *kk* चा मजकूर *file_copy.c*. प्रमाणेच असल्याचे पडताळू शकता.

युनिट सारांश

या अध्यायात आपण हे शिकलो आहोत

- ❑ जर इनपुट डेटाचे परिमाण अफाट असेल तर ते डेटा फाइल्स वापरून उत्तम प्रकारे हाताळले जाऊ शकते.
- ❑ जर एखाद्या इन्स्ट्रुमेंटद्वारे निर्मित केलेला डेटा मशीन-वाचन करण्यायोग्य फॉर्ममध्ये (बायनरी फॉर्म) असेल तर आपल्याला हा डेटा आपल्या प्रोग्रामला पाठवण्यासाठी डेटा फाइल वापरावी लागेल.
- ❑ जर आउटपुट व्हॉल्यूम अफाट असेल आणि स्क्रीनवर योग्यप्रकारे पाहिले जाऊ शकत नसेल तर डेटा प्रोग्राममध्ये आउटपुट लिहिणे चांगले आहे आपण आपला प्रोग्राम पुन्हा कार्यान्वित केल्याशिवाय कधीही संदर्भ घेऊ शकता.
- ❑ जर एका प्रोग्रामद्वारे निर्मित केलेले आउटपुट दुसऱ्या प्रोग्रामसाठी इनपुट म्हणून वापरायचे असेल तर पुन्हा डेटा फाइल्सचा उपयोग फायदेशीर ठरेल
- ❑ डेटा फाइल टेक्स्ट फाईल किंवा बायनरी फाईल असू शकते.

अभ्यास करा

व्यक्तिनिष्ठ प्रश्न

1. C मध्ये I/O फाइल करण्यासाठी विविध प्रणाल्यांची नावे द्या.
2. लेखनासाठी उघडलेली फाईल बंद करण्याचा सल्ला दिला आहे काय?
3. फाईल बंद करण्यासाठी फंक्शन्सची नाव द्या.
4. फाइल उघडल्यास कोणती माहिती प्रणाली मिळते?
5. फाइल पॉईंटर म्हणजे काय?
6. *fopen()* फंक्शनसह उघडलेली फाईल प्रोग्रॅममध्ये कशी संदर्भित केली जाते?
7. समजा एखाद्या प्रोग्रामला फाईलच्या प्रत्येक बाईटची तपासणी करायची आहे; कोणता मोड अधिक योग्य असेल?
8. फाईलच्या संदर्भात टर्म ऑफसेट म्हणजे काय?
9. *fseek()* फंक्शन द्वारे कोणते कार्य केले जाते?

10. *ftell()* फंक्शनद्वारे कोणते कार्य केले जाते?
11. फाईल उघडण्यात काही त्रुटी आढळल्यास *fopen()* ने कोणते मूल्य परत करेल?
12. *w+* आणि *r+* मोडमध्ये काय फरक आहे?
13. *rewind()* काय करते?

एकाधिक निवड प्रश्न

1. खालीलपैकी कोणते डीफॉल्ट फाईल पॉइंटर आहे?
 (a) *stdin* (b) *stdout* (c) *stderr* (d) रील सर्व
2. *fopen()* फंक्शन फाईल उघडण्यास अपयशी ठरते तेव्हा ते व्हॅल्यू मिळवते
 (a) *NULL* (b) *-1* (c) *Null* (d) *void*
3. *fclose()* सहसा वापरला जातो
 (a) फाईल डिलिट करण्यासाठी (b) प्रोग्राममधून फाईल डिस्कनेक्ट करा
 (c) फाईलमधील डेटा वाचा (d) एखाद्या फाईलवर डेटा लिहा
4. खालीलपैकी कोणती मूल्ये *seek(fp, offset, from)* फंक्शनमधील मूल्य नाहीत?
 (a) 2 (b) 0 (c) 1 (d) EOF
5. फंक्शन *fseek(fp, 0, 0)* द्वारे केलेले कार्य आहे
 (a) *fclose(fp)* (b) *ftell(fp)* (c) *search(fp)* (d) *rewind(fp)*
6. खालीलपैकी कोणते वैध फाईल उघडण्याचे मोड नाही?
 (a) *r+* (b) *+r* (c) *r* (d) *rb*
7. जर फाईल ओपनिंग मोडमध्ये अक्षर बीचा प्रत्यय आला असेल तर तो काय दर्शवितो?
 (a) फाईल केवळ वाचनासाठी उघडली पाहिजे (b) फाईल फक्त लिहिण्यासाठी उघडली पाहिजे
 (c) फाईल वाचन-लेखन दोन्हीसाठी उघडायची आहे (d) फाईल बायनरी फाईल आहे
8. खालील कोड विभागाचा विचार करा

```
FILE *fp;
fp = fopen("inventory.dat", "rb");
```

“rb” मध्ये लेटर ‘b’ ची भूमिका काय आहे?

- (a) वाचनासाठी बायनरी मोडमध्ये *inventory.dat* फाईल उघडा.
 - (b) *inventory.dat* फाईल वाचन आणि लेखन उघडा.
 - (c) लेखनासाठी नवीन फाईल *inventory.dat* तयार करा.
 - (d) लिहिणे आणि वाचण्यासाठी *inventory.dat* फाईल बायनरी मोडमध्ये उघडा.
9. खालील कोड विभागाचा विचार करा

```
FILE *fp;
fp = fopen("notes.txt", "r+");
```

खालीलपैकी कोणते ऑपरेशन्स *notes.txt* फाईलवर करता येतील?

- (a) Reading (b) Writing (c) Appending (d) रील सर्व

10. फाइल _____ प्रकारची आहे.
 (a) *int* type (b) *struct* type (c) *string* type (d) *structure* type
11. फाइल उघडताना काही त्रुटी असल्यास, fopen _____ परत करेल.
 FILE *fp;
 (a) null (b) NULL (c) Null (d) EOF
12. ऍड-ऑफ-फाइल शोधण्यासाठी खालीलपैकी कोणत्या लायब्ररी फंक्शनचा वापर केला जाऊ शकतो?
 (a) eof() (b) isend() (c) feof() (d) end()
13. वाचन आणि लेखन या दोन्हीसाठी विद्यमान फाइल उघडण्यासाठी वापरलेला एक ___ मोड आहे.
 (a) +r (b) r+ (c) w+ (d) w
14. खालीलपैकी कोणत्या फंक्शनचा उपयोग फाइल न बंद केल्यावर आणि पुन्हा उघडल्याशिवाय पुन्हा वाचण्यासाठी केला जाऊ शकतो?
 (a) fseek() (b) rewind() (c) ftell() (d) Both (a) and (b)
15. पुढीलपैकी कोणते फाइल स्थिती(status) फंक्शन नाही?
 (a) ferror() (b) ftell() (c) clearerr() (d) feof()

ANSWERS															
1.	(d)	2.	(a)	3.	(b)	4.	(d)	5.	(d)	6.	(b)	7.	(d)	8.	(d)
9.	(d)	10.	(b)	11.	(b)	12.	(c)	13.	(b)	14.	(d)	15.	(b)		

प्रोग्रामिंग समस्या

- काही टेक्स्ट असलेली टेक्स्ट फाइल दिली. एक प्रोग्राम लिहा जो युझरला टेक्स्ट फाइलचे नाव इनपुट करण्यास प्रॉम्प्ट करतो, ही फाइल वाचतो आणि टेक्स्टमधील स्वरांची(vowles) संख्या आणि शब्दांची संख्या दर्शवितो.
- काही टेक्स्ट असलेली टेक्स्ट फाइल दिली. एक प्रोग्राम लिहा जो युझरला टेक्स्ट फाइलचे नाव इनपुट करण्यास प्रवृत्त करतो, ही फाइल वाचतो आणि प्रत्येक अक्षर 'x' ला अपरकेस 'X' ने बदलतो.
- C भाषेच्या नवशिक्याने संपूर्ण प्रोग्राम अप्परकेसमध्ये टाईप केला आहे; तुम्हाला माहिती आहेच की C भाषेतील नेहमीचे म्हणजे मॅक्रो आणि डिफाईंड कॉन्स्टन्ट वगळता लोअरकेस अक्षरे वापरून सौर्स कोड टाईप करणे. एखादा प्रोग्राम लिहा जो सौर्स कोडला लोअरकेस लेटर मध्ये रूपांतरित करतो. कमेंट्स मध्ये बदल न करता.
- फाइलमधून अंक वाचण्यासाठी प्रोग्राम लिहा आणि फाइलमध्ये समान, विषम आणि प्राइम नंबर लिहा.
- दोन फाइल्स एका टोकापासून तिसऱ्या फाइलमध्ये विलीन(merge) करण्यासाठी प्रोग्राम लिहा.
- स्क्रीनवर आउटपुट म्हणून स्वतःचा सौर्स कोड प्रिंट करण्यासाठी प्रोग्राम लिहा.
- टेक्स्ट फाइलमध्ये कॅरेक्टर, शब्द(words) आणि ओळी(line) मोजण्यासाठी प्रोग्राम लिहा.

प्रॉक्टिकल

- एक प्रोग्राम लिहा जो कीबोर्डमधून काही टेक्स्ट वाचतो आणि त्यास sample.txt. नावाच्या फाइलमध्ये लिहितो. प्रोग्राम नंतर ही फाइल वाचतो आणि फाइल बंद न करता आणि न उघडता त्यातील टेक्स्ट स्क्रीनवर प्रदर्शित करतो.

फाईल “W +” मोडमध्ये उघडली जाते आणि एकदा फाईल तयार झाल्यावर फाईलच्या सुरुवातीला फाईल पॉइंटर बदलण्यासाठी आपण ती `rewind()` फंक्शन वापरून रिवाइंड करतो.

Listing 10.13

```
/*
Program that creates a file and then reads its contents without
closing and re-opening, and display them on the computer screen
*/
#include <stdio.h>
int main()
{
    char ch;
    FILE *fp;
    fp = fopen("sample.txt", "w+");
    if ( fp == NULL ) {
        printf("\nUnable to open file sample.txt\n");
        return 1;
    }
    printf("\nType some text...\n\n");
    while ( ( ch = getche() ) != '\r' )
        fputc(ch, fp);

    rewind(fp);
    printf("\n\nContents of file...\n\n");
    while ( !feof(fp) ) {
        ch = fgetc(fp);
        putchar(ch);
    }
    fclose(fp);
    return 0;
}
```

Test Run

```
Type some text...
This is a sample file.
Contents of file...
This is a sample file.
```

2. फाईलमधील प्रथम n अक्षरे उलट करण्यासाठी C प्रोग्राम लिहा. युझर n चे फाईल नाव आणि मूल्य प्रदान करतो.

Listing 10.14

```

/*
    Program to reverse the first n characters in a file
*/
#include <stdio.h>
int main()
{
    char str[80], filename[30], ch;
    int i, n;
    FILE *fp;
    printf("\nEnter file name : ");
    gets(filename);
    printf("\nEnter value for n : ");
    scanf("%d", &n);

    fp = fopen(filename, "r+");
    if ( fp == NULL ) {
        printf("\nUnable to open file: %s\n", filename);
        return 1;
    }
    i = 0;
    while ( ( !feof(fp) ) && ( i < n ) )
    {
        str[i] = fgetc(fp);
        i++;
    }
    rewind(fp); /* reposition the file pointer to the beginning */
    i = n-1;
    while ( i >= 0 )
    {
        fputc( str[i], fp);
        i--;
    }
    fclose(fp);
    return 0;
}

```

Test Run

```
Enter file name: testfile.txt
```

```
Enter the value for n: 10
```

फाईलची मूळ माहिती ही होती

```
1234567890abcdefghijklmnopqrstuvwxyz
```

प्रोग्राम एक्झिक्युट झाल्यानंतर, फाईलमधील माहिती ही आहे

```
0987654321abcdefghijklmnopqrstuvwxyz
```

कमांड प्रॉम्प्टवर टाईप कमांड वापरून आपण प्रोग्राम एक्झिक्युशन फाईलची पडताळणी करू शकता.

वरील प्रोग्राममध्ये प्रथम दिलेली फाईल वाचन / लेखन मोडमध्ये उघडली जाईल. पुढे, फाईलचे प्रथम n बाइट्स डायनॅमिकली वाटप केलेल्या कॅरेक्टर अ‍ॅरमध्ये कॉपी केले जातात. पुढे दिलेली फाईल रिवाइंड करू. म्हणजेच फाईलच्या सुरुवातीला फाईल पॉइंटर पुन्हा ठेवू.

अखेरीस, आपण अ‍ॅर कॅरेक्टरची सामग्री फाईलवर उलट क्रमाने लिहितो. अशा प्रकारे आपण इच्छित कार्य साध्य करू शकतो.

- कीबोर्ड वरून विद्यार्थ्यांची नावे व त्यांची संख्या वाचणारी प्रोग्राम लिहा व फाईलमध्ये ही माहिती साठवा. जर फाईल आधीपासून अस्तित्वात असेल तर फाईलमध्ये माहिती जोडा.

Listing 10.15

```
/*
   Program to store names and marks of students in file
*/
#include <stdio.h>
int main()
{
    FILE *fp;
    char name[50], filename[30];
    int i, n, marks;
    printf("Enter name of file : ");
    gets(filename);
    printf("Enter number of students to add : ");
    scanf("%d", &n);
    fp = fopen(filename, "a");
    if (fp == NULL) {
        printf("\nUnable to open file: %s\n", filename);
        return 1;
    }
    for(i = 0; i < n; i++)
    {
        printf("\nEnter data for student %d\n", i+1);
        printf("\nEnter name : ");
```

```
scanf("%s", name);  
printf("Enter marks: ");  
scanf("%d", &marks);  
fprintf(fp, "\nName: %s \nMarks=%d \n", name, marks);  
}  
fclose(fp);  
return 0;  
}
```

आणखी माहिती

शिक्षकांनी डेटा फायलींच्या संकल्पना आणि C भाषेत फाईल हाताळण्याशी संबंधित विविध पैलू समजून घेणे अपेक्षित आहे.

विद्यार्थ्यांच्या सहभागासह फायलींवर प्रक्रिया करण्यासाठी शिक्षकांनी योग्य उदाहरणे घेऊन C प्रोग्राम तयार करून फाइल्स दाखवाव्यात.

संदर्भ आणि सूचविलेले वाचन

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. <https://ocw.mit.edu/courses/intro-programming/>
7. <https://www.programiz.com/c-programming>
8. <https://www.javatpoint.com/c-programming-language-tutorial>

पुढील शिक्षणासाठी संदर्भ

1. Brain W. Kernighan and Deniss M. Ritchie, The C Programming Language, 2nd Edition, Prentice Hell.
2. Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill.
3. R. S. Salaria, Test Your Skills in C, Khanna Book Publishing Co(P) Ltd.
4. R. S. Salaria, Cracking IT Interviews, Khanna Book Publishing Co(P) Ltd.
5. Stephen Prata, C Primer Plus, Addison-Wesley Professional.
6. David Griffiths and Dawn Griffiths, Head First C, Shroff Publishers & Distributors Pvt. Ltd.
7. Antti Laaksonen, Guide to Competitive Programming, 2nd Edition, Springer.
8. <https://www.greatlearning.in/academy/learn-for-free/courses/c-programming>
9. <https://www.udemy.com/topic/c-programming/>
10. <https://www.edx.org/learn/c-programming>
11. <https://www.coursera.org/courses?query=c%20programming>
12. <https://www.learnonline.com/>
13. <https://www.codecademy.com/>

CO आणि PO अटेन्मेन्ट तक्ता

या कोर्सच्या समाप्तीनंतर कोर्ससाठीचे कोर्स आऊटकम्स (COs) यांचे प्रोग्रॅम आऊटकम्स सोबत मॅपिंग केले जाऊ शकते आणि त्या अनुषंगाने POs च्या अटेन्मेन्टबाबतीत विश्लेषण केले जाऊ शकते. या संपूर्ण विश्लेषणामार्फत POs च्या अटेन्मेन्टमधील तफावतीवर सुधारण्यासाठीच्या आवश्यक उपाययोजना केल्या जाऊ शकतील.

CO आणि PO अटेन्मेन्ट तक्ता

कोर्स आऊटकम्स	प्रोग्रॅम आऊटकम्सचे अटेन्मेन्ट (1- किमान परस्परसंबंध; 2- मध्यम परस्परसंबंध; 3- घनिष्ट परस्परसंबंध)											
	PO-1	PO-2	PO-3	PO-4	PO-5	PO-6	PO-7	PO-8	PO-9	PO-10	PO-11	PO-12
CO-1												
CO-2												
CO-3												
CO-4												
CO-5												
CO-6												

या तक्त्यातील तपशीलानुसार तफावती सुधारता येतील

सूची

अ

अल्गोरिदम
अरीथमेटिक एक्सप्रेसनस
अरीथमेटिक ऑपरेटर
अरीथमेटिक आणि लॉजिक युनिट
अरे
असाइनमेंट ऑपरेटर
अससोसिएटिव्हिटी
अकर्मन फंक्शन
ऑप्लिकेशन सॉफ्टवेअर
अर्ग्युमेण्ट

आ

आउटपुट
आर्मस्ट्रॉंग नंबर
आयडेंटिफायर्स

इ

इन्क्रीमेंट आणि डिक््रीमेंट ऑपरेटर
इंडिरेक्शन ऑपरेटर
इनपुट युनिट
इंटरप्रीटर
इफ स्टेटमेंट
इफ - एल्स स्टेटमेंट
इफ-एल्स इफ लाडर
इनिशियलिझेशन
इंसरशन सॉर्ट
इन्टिजर
इंटीग्रेशन

ए

एंड्रेस ऑफ ऑपरेटर
एसेम्बलर
एडिटर
एक्सेसइंग
एक्झिक्यूशन
एक्सप्रेसनस

ऑ

ऑपरेटिंग सिस्टम
ऑब्जेक्ट कोड

क

कंडिशनल ऑपरेटर
कमांड
कंट्रोल युनिट
कंडिशनल ब्रैचिंग
कंपायलर
कोल्ड बूटिंग
कंटेन्यू स्टेटमेंट
क्विक सॉर्ट
कॉन्स्टन्ट
कॉलइंग फंक्शन
कॅरेक्टर सेट
कीवर्ड

ग

ग्लोबल डेटा

ज

जम्पिंग

ट

ट्रान्सलेटर

टेस्टिंग

टोकन्स

टर्नरी ऑपरेटर

टाईप कन्वर्जन

टू डायमेशनल

ट्रॅपेझॉइडल नियम

टॅग स्ट्रक्चर

टाइप- डिफाइंड स्ट्रक्चर

टेक्स्ट फाईल

ड

डेटा टाइप

डू - व्हाइल स्टेटमेंट

डिक्लरेशन

डायनॅमिक मेमरी

डी-वाटप मेमरी

न

नेस्टेड इफ

नेस्टेड इफ- एल्स

नैसर्गिक संख्या

प

पॅलिंड्रोम

पंकचूएटर्स

पॉईंटर्स

प्र

प्राइम नंबर

प्रोग्राम

प्रीप्रोसेसर डिरॅक्टिव्ह्स

प्रिसिडन्स

फ

फ्लोचार्ट

फॉर्मेट स्पेसिफायर्स

फॉर्मेट

फॉर स्टेटमेंट

फिबोनेकी अनुक्रम

फिबोनाची नंबरस

फंक्शन्स

फंक्शन हेडर

फंक्शन बॉडी

फॅक्टोरियल फंक्शन

फाईल

ब

बिटवाईस ऑपरेटर

बिल्ट-इन

बायनरी ऑपरेटर

ब्रेक स्टेटमेंट

बायनरी सर्च

बबल सॉर्ट

बेस केस

बायनरी फाईल

म

मायक्रोप्रोसेसर

मिक्स्ड-मोड

मल्टि डायमेशनल

मूलभूत

मूल्यानुसार कॉल करा

मर्ज सॉर्ट

मेमरी वाटप

य

युझर- डिफाइंड
युनेरी ऑपरेटर
युटिलिटी सॉफ्टवेअर

र

रिअल
रिटर्न स्टेटमेंट
रिलेशनल ऑपरेटर
रिकर्सन
रिकर्सिव्ह स्टेप

ल

लैंग्वेज ट्रांसलेटर
लिंकर
लायब्ररी फंक्शन
लॉजिकल ऑपरेटर
लोडर
लिटरल्स
लिनिअर सर्च
लोकल डेटा

व

वॉर्म बूटिंग
व्हेरिएबल
विरामचिन्हे
व्हाइल स्टेटमेंट
वन डायमैन्शनल

स

सिस्टम सॉफ्टवेअर
संदर्भानुसार कॉल करा
सिन्टाक्स एरॉर
सॉफ्ट बूटिंग
स्यूडोकोड
सिन्टाक्स
सौर्स कोड
स्ट्रिंग
साईझऑफ ऑपरेटर
स्विच स्टेटमेंट
सर्चिंग
सिलेक्शन सॉर्ट
स्ट्रक्चर

ह

हार्डवेअर
हार्ड बूटिंग

