K. K. Wagh Institute of Engineering Education and Research, Nashik

(An Autonomous Institute from A. Y. 2022-23)

Exam Seat No.:

End-Sem Examination-II | Winter 2025

Academic Year: 2025-26 | Sem: II

Class: F.Y | Program: B.Tech

Branch Code: INT | Pattern: 2023

Name of Course: Object Oriented Programming | Course Code: 2300118F

Max. Marks: 60 | Duration: 2:30 Hrs.

Solution

Q. No.	Details	Max. Marks	CO	BL

Q. 1
 	
Q. 2
	a) Explain class, object with the help of sample code in C++.
	1. Class
	• A **class** is a **user-defined data type** that acts as a **blueprint** for objects.
	• It can contain **data members** (variables) and **member functions** (methods).
	• Declared using the class keyword
	class ClassName {
	public:
	// data members
	int data;
	
	// member function
	void display() {
	cout << "Data: " << data << endl;
	}
	};
	2. Object
	• An **object** is an **instance of a class**.
	• Objects store actual data and can use the class's functions.
	ClassName obj; // Creating an object
		[06]	CO 1	1- Understand

	b) Explain friend function with help of sample code in C++. A **friend function** is a function that is **not a member of a class** but is **allowed to access private and protected members** of that class.	[06]	CO 2	1- Understand

	Key Points
	• Declared inside the class using the **friend** keyword.
	• Defined **outside the class**.
	• Useful for **functions that need access to multiple classes** or private data.
	• Does **not** have a this pointer.
	class ClassName {
	int data;
	public:
	friend void friendFunction(ClassName &obj); // declaration
	};
	#include <iostream>
	using namespace std;
	
	class Box {
	private:
	int length;
	public:
	Box(int l) { length = l; }
	
	// Friend function declaration
	friend void displayLength(Box &b);
	};
	
	// Friend function definition
	void displayLength(Box &b) {
	cout << "Length of Box: " << b.length << endl;
	}
	
	int main() {
	Box b1(10);
	displayLength(b1); // Accesses private member
	return 0;
	}			

Q. 3	a) Explain runtime and compile time polymorphism in C++. **(08 marks)** **Polymorphism** means "**many forms**". In C++, polymorphism allows functions or objects to behave differently based on context. Polymorphism is classified into **compile-time (static)** and **runtime (dynamic)**. **. Compile-Time Polymorphism** **Definition** • Polymorphism resolved **at compile time**. • Also called **static polymorphism**. • Achieved using: ○ **Function overloading**	[16]	CO 3	1-Understand 1-Understand

- **Operator overloading**

Example: Function Overloading

2. Runtime Polymorphism

Definition

- Polymorphism resolved **at runtime**.
- Also called **dynamic polymorphism**.
- Achieved using:
  - **Function overriding** with **virtual functions** in inheritance.

Example: Function Overriding

Feature	Compile-Time	Runtime
Also called	Static Polymorphism	Dynamic Polymorphism
Resolution	At Compile time	At Runtime
Achieved by	Function & Operator Overloading	Function Overriding with `virtual`
Flexibility	Less flexible	More flexible
Performance	Faster	Slightly slower due to dynamic dispatch

OR

b) Explain call by reference and call by value with the help of sample code in C++.

Call by Value:

In call by value, a **copy of the actual argument** is passed to the function. The function works on this copy, so any changes made inside the function **do not affect the original variable**. It is safer to use because the original data remains unchanged, but it may require more memory.

```
void change(int x) {
   x = 20;
}
```

Call by Reference:

In call by reference, the **reference (address)** of the variable is passed to the function. The function directly accesses and modifies the **original variable**. It is more efficient and is used when the function needs to return multiple values or modify data.

```
void change(int &x) {
   x = 20;
}
```

c) Write a C++ program to implement a class Complex which represents the Complex Number data type. Implement the following: **(08marks)**

a. Constructor

3-Apply

b. Overload operator - (minus) to subtract two complex numbers
c. Overload operator + (plus) to add two complex numbers

```
#include <iostream>
using namespace std;

class Complex {
private:
   int real, imag;

public:
   // Constructor
   Complex(int r = 0, int i = 0) {
      real = r;
      imag = i;
   }

   // Overload + operator
   Complex operator + (Complex c) {
      Complex temp;
      temp.real = real + c.real;
      temp.imag = imag + c.imag;
      return temp;
   }

   // Overload - operator
   Complex operator - (Complex c) {
      Complex temp;
      temp.real = real - c.real;
      temp.imag = imag - c.imag;
      return temp;
   }

   // Function to display complex number
   void display() {
      cout << real << " + " << imag << "i" << endl;
   }
};

int main() {
   Complex c1(5, 3), c2(2, 1), c3, c4;

   c3 = c1 + c2;   // addition
   c4 = c1 - c2;   // subtraction

   cout << "First Complex Number: ";
   c1.display();
```

```
    cout << "Second Complex Number: ";
    c2.display();

    cout << "Addition of Complex Numbers: ";
    c3.display();

    cout << "Subtraction of Complex Numbers: ";
    c4.display();

    return 0;
}
```

OR

d) Write a C++ program to implement a class student. Implement function overloading to find total of marks of students for academic subjects and sports marks. Write constructor and 3 functions for the same.

```
#include <iostream>
using namespace std;

class Student {
private:
    int roll;
    string name;

public:
    // Constructor
    Student(int r, string n) {
        roll = r;
        name = n;
    }

    // Function 1: Total of academic subjects (3 subjects)
    int total(int m1, int m2, int m3) {
        return m1 + m2 + m3;
    }

    // Function 2: Total of sports marks
    int total(int sports) {
        return sports;
    }

    // Function 3: Total of academic + sports marks
    int total(int m1, int m2, int m3, int sports) {
        return m1 + m2 + m3 + sports;
    }
```

```
    // Display student details
    void display() {
        cout << "Roll No: " << roll << endl;
        cout << "Name   : " << name << endl;
    }
};

int main() {
    Student s1(1, "Rahul");

    int academicTotal = s1.total(70, 80, 75);
    int sportsMarks = s1.total(20);
    int grandTotal = s1.total(70, 80, 75, 20);

    s1.display();
    cout << "Academic Total Marks: " << academicTotal
<< endl;
    cout << "Sports Marks        : " << sportsMarks << endl;
    cout << "Overall Total Marks : " << grandTotal <<
endl;

    return 0;
}
```

Q. 4	a) Explain exception handling in C++. Write all required blocks to implement it in C++.**(08 marks)** **Exception handling** in C++ is a mechanism used to handle **runtime errors** and abnormal conditions in a program so that normal program flow can be maintained. It improves program **reliability** and **readability** by separating error-handling code from normal logic. C++ uses three main keywords for exception handling: **try**, **throw**, and **catch**. **Required Blocks in Exception Handling** 1. **try block** o Contains the code that may generate an exception. o It is followed by one or more catch blocks. 2. **throw block** o Used to **explicitly generate an exception**. o Can throw built-in data types or user-defined objects. 3. **catch block** o Used to **handle the exception** thrown in the try block. o Different catch blocks can be used for different exception types.	[16]	CO 4	1-Understand 1-Understand

```
#include <iostream>
using namespace std;

int main() {
   int a = 10, b = 0;

   try {
      if (b == 0)
         throw b;
      cout << a / b;
   }
   catch (int e) {
      cout << "Division by zero error!" << endl;
   }

   return 0;
}
```

OR

b) What are types of templates? Explain its types with suitable sample code in C++.

A **template** in C++ is a feature that allows writing **generic programs**, meaning the same code can work with different data types. Templates help in **code reusability** and **type safety**.

There are **two main types of templates** in C++:

1. Function Templates

Explanation

A **function template** is used to create a generic function that works with **different data types**. The data type is decided at the time of function call.

```
#include <iostream>
using namespace std;

template <class T>
T add(T a, T b) {
   return a + b;
}

int main() {
   cout << add(10, 20) << endl;       // int
   cout << add(5.5, 2.5) << endl;     // double
   return 0;
}
```

A class template is used to define a class that can work with **different data types**. The data type is specified while creating the object.

```
#include <iostream>
```

```
using namespace std;

template <class T>
class Sample {
private:
   T value;

public:
   Sample(T v) {
      value = v;
   }

   void display() {
      cout << "Value: " << value << endl;
   }
};

int main() {
   Sample<int> s1(10);
   Sample<float> s2(3.14f);

   s1.display();
   s2.display();

   return 0;
}
```

e) Write a C++ program to Create a class for integers operations. Write functions to accept the numbers and do the all arithmetic operations. Use exception handling for division operation. **(08 marks)**

3-Apply

```
#include <iostream>
using namespace std;

class IntegerOperations {
private:
   int a, b;

public:
   // Function to accept numbers
   void accept() {
      cout << "Enter two integers: ";
      cin >> a >> b;
   }

   // Addition
   void add() {
      cout << "Addition: " << a + b << endl;
```

```
    }

    // Subtraction
    void subtract() {
        cout << "Subtraction: " << a - b << endl;
    }

    // Multiplication
    void multiply() {
        cout << "Multiplication: " << a * b << endl;
    }

    // Division with exception handling
    void divide() {
        try {
            if (b == 0)
                throw b;
            cout << "Division: " << a / b << endl;
        }
        catch (int) {
            cout << "Error: Division by zero is not allowed." <<
endl;
        }
    }
};

int main() {
    IntegerOperations obj;
    obj.accept();
    obj.add();
    obj.subtract();
    obj.multiply();
    obj.divide();
    return 0;
}
```

OR

f) Write a C++ program with a function which can find maximum and minimum of numbers with data types like integer, double, float. Use templates in C++.

```
#include <iostream>
using namespace std;

// Function template to find maximum
template <class T>
T findMax(T a, T b) {
    return (a > b) ? a : b;
}
```

```
// Function template to find minimum
template <class T>
T findMin(T a, T b) {
   return (a < b) ? a : b;
}

int main() {
   int i1 = 10, i2 = 20;
   float f1 = 5.5f, f2 = 2.3f;
   double d1 = 9.8, d2 = 4.6;

   cout << "Integer Max: " << findMax(i1, i2) << endl;
   cout << "Integer Min: " << findMin(i1, i2) << endl;

   cout << "Float Max: " << findMax(f1, f2) << endl;
   cout << "Float Min: " << findMin(f1, f2) << endl;

   cout << "Double Max: " << findMax(d1, d2) << endl;
   cout << "Double Min: " << findMin(d1, d2) << endl;

   return 0;
}
```

Q.5	a) Explain role of istream and ostream classes for file handling in C++. Explain seekp() and seekg() functions with suitable code in C++. **(08 marks)**	[16]	CO 5	1- Understand
	istream and ostream in File Handling In C++, file handling is performed using stream classes. The base classes istream and ostream play an important role 1. istream • Used for input operations • Reads data from files or input streams • ifstream (input file stream) is derived from istream • Supports functions like read(), getline(), tellg(), seekg() 2. ostream • Used for output operations • Writes data to files or output streams • ofstream (output file stream) is derived from ostream • Supports functions like write(), tellp(), seekp() seekp() and seekg() in C++ • seekp(): Moves the put (write) pointer to a specific position in a file. Syntax: file.seekp(offset, ios::beg/ios::cur/ios::end);			1- Understand

	• seekg(): Moves the get (read) pointer to a specific position in a file.
	Syntax: file.seekg(offset, ios::beg/ios::cur/ios::end);
	OR
	b) Explain all error handling functions used during file operations with suitable code syntax in C++.
	In C++, file handling is done using stream classes like ifstream, ofstream, and fstream. During file operations, **error handling functions** are used to check the **state of the file stream** and detect errors such as file not found, read/write failure, or end of file.
	Common Error Handling Functions
	1. fail()
	• Returns **true** if an input/output operation fails.
	• Used to check errors like file not opening or invalid data read.
	Syntax:
	file.fail();
	2. bad()
	• Returns **true** if a **serious error** occurs (e.g., hardware failure).
	• Indicates loss of stream integrity.
	Syntax:
	file.bad();
	3. good()
	• Returns **true** if **no error** has occurred.
	• Indicates the stream is in a normal state.
	Syntax:
	file.good();
	4. eof()
	• Returns **true** when the **end of file** is reached.
	• Commonly used while reading files in a loop.
	Syntax:
	file.eof();			

	e) Write C++ Program for writing students information (Name, RollNo, CGPA) to a file and reading the same information. **(08 marks)** #include <iostream> #include <fstream> using namespace std; class Student { public: string name; int rollNo;			3-Apply

```
    float cgpa;

    void getData() {
        cout << "Enter Name: ";
        cin >> name;
        cout << "Enter Roll No: ";
        cin >> rollNo;
        cout << "Enter CGPA: ";
        cin >> cgpa;
    }

    void displayData() {
        cout << "Name    : " << name << endl;
        cout << "Roll No : " << rollNo << endl;
        cout << "CGPA    : " << cgpa << endl;
    }
};

int main() {
    Student s;
    ofstream fout;
    ifstream fin;

    // Writing to file
    fout.open("student.txt");
    s.getData();
    fout << s.name << " " << s.rollNo << " " << s.cgpa <<
endl;
    fout.close();

    // Reading from file
    fin.open("student.txt");
    fin >> s.name >> s.rollNo >> s.cgpa;
    fin.close();

    cout << "\nStudent Information Read from File:\n";
    s.displayData();

    return 0;
}
```

OR

f) There is a file containing books- Title, Price and Author names. Write a C++ program to search Book of a given. Print its all details.

```
#include <iostream>
#include <fstream>
#include <string>
```

```
using namespace std;

class Book {
public:
   string title;
   string author;
   float price;

   void getData() {
      cin.ignore();
      cout << "Enter Book Title: ";
      getline(cin, title);
      cout << "Enter Author Name: ";
      getline(cin, author);
      cout << "Enter Price: ";
      cin >> price;
   }

   void display() {
      cout << "\nBook Details:\n";
      cout << "Title : " << title << endl;
      cout << "Author: " << author << endl;
      cout << "Price : " << price << endl;
   }
};

int main() {
   fstream file;
   Book b;
   int numBooks;
   string searchTitle;

   // Writing sample book data to file
   file.open("books.txt", ios::out);
   cout << "Enter number of books to store: ";
   cin >> numBooks;

   for (int i = 0; i < numBooks; i++) {
      cout << "\nBook " << i + 1 << " details:\n";
      b.getData();
      file << b.title << "," << b.author << "," << b.price <<
endl;
   }
   file.close();

   // Searching for a book by title
   cin.ignore();
```

```
    cout << "\nEnter Book Title to search: ";
    getline(cin, searchTitle);

    file.open("books.txt", ios::in);
    string line;
    bool found = false;

    while (getline(file, line)) {
        size_t pos1 = line.find(',');
        size_t pos2 = line.find(',', pos1 + 1);

        string t = line.substr(0, pos1);
        string a = line.substr(pos1 + 1, pos2 - pos1 - 1);
        float p = stof(line.substr(pos2 + 1));

        if (t == searchTitle) {
            b.title = t;
            b.author = a;
            b.price = p;
            found = true;
            break;
        }
    }
    file.close();

    if (found)
        b.display();
    else
        cout << "Book not found!\n";

    return 0;
}
```