Bachelor of Computer Applications (BCA)

OOPs Using C++ Lab (DBCACO307P24)

Self-Learning Material (SEM III)

Jaipur National University
Centre for Distance and Online Education

Established by Government of Rajasthan
Approved by UGC under Sec 2(f) of UGC ACT 1956
&
NAAC A+ Accredited

TABLE OF CONTENTS

EXPERT COMMITTEE

Prof. Sunil Gupta
(Computer and Systems Sciences, JNU Jaipur)

Dr. Satish Pandey
(Computer and Systems Sciences, JNU Jaipur)

Dr. Shalini Rajawat
(Computer and Systems Sciences, JNU Jaipur)

COURSE COORDINATOR

Ms. Heena Shrimal
(Computer and Systems Sciences, JNU Jaipur)

UNIT PREPARATION

Unit Writer(s)	**Assisting & Proofreading**	**Unit Editor**
Ms. Heena Shrimal (Computer and Systems Sciences, JNU Jaipur)	Ms. Rachana yadav (Computer and Systems Sciences, JNU Jaipur)	Dr. Deepak Shekhawat (Computer and Systems Sciences, JNU Jaipur)

Secretarial Assistance

Mr. Mukesh Sharma

COURSE INTRODUCTION

"Object-oriented programming is an exceptionally bad idea which could only have originated in California."

- Edsger W. Dijkstra

Object-oriented programming (OOP) has revolutionized software development by focusing on objects and classes to create scalable and maintainable applications. This course explores the key OOP principles and their implementation in C++, providing students with a comprehensive understanding of modern programming practices.

The primary objectives of this course are to introduce students to core OOP concepts such as abstraction, encapsulation, inheritance, and polymorphism. Students will build a strong foundation in C++, learning to leverage its features for various programming tasks. Through practical exercises and projects, students will gain experience in flow control, functions, dynamic memory management, file handling, and exception handling.

The course begins with an overview of different paradigms for problem-solving, highlighting the differences between OOP and procedure-oriented programming. Students will explore the principles of abstraction and the foundational concepts of OOP, including encapsulation, inheritance, and polymorphism. The basics of C++ are introduced, covering the structure of a C++ program, data types, variable declaration, expressions, operators, operator precedence, evaluation of expressions, type conversions, pointers, arrays, and strings.

Flow control statements are essential for directing the flow of a program. This course covers the use of if, switch, while, for, do, break, continue, and goto statements. Functions are another crucial aspect, and students will learn about the scope of variables, parameter passing, default arguments, inline functions, recursive functions, and pointers to functions. The course also covers dynamic memory allocation and de-allocation using the new and delete operators, as well as preprocessor directives.

The course then delves into C++ classes and data abstraction. Students will learn about class definitions, class structure, class objects, class scope, the this pointer, friends to a class, static class members, constant member functions, constructors, and destructors. Polymorphism is explored in-depth, covering function overloading, operator overloading, and generic programming. The necessity of templates is discussed, along with function templates and class templates.

Inheritance is another critical concept covered in this course. Students will learn how to define a class hierarchy and explore different forms of inheritance. The course covers defining base and derived classes, accessing base class members, and base and derived class construction. Destructors and virtual base classes are also discussed.

Virtual functions and polymorphism are key topics in this course. Students will learn about static and dynamic bindings, base and derived class virtual functions, dynamic binding through virtual functions, the virtual function call mechanism, and pure virtual functions.

Course Outcomes:

At the completion of the course, a student will be able to:

1. Acquire profound knowledge of object oriented programming.
2. Demonstrate the difference between the solutions offered by traditional imperative problem solving method and object-oriented method
3. Explain the class inheritance, data encapsulation, polymorphism as fundamental building blocks to generate reusable code.
4. Understand and implement error handling and file handling routines.

Acknowledgements:

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

Assignment 1: Implementing a Class and Object

Program Statement: Write a C++ program to create a class **Rectangle** with attributes length and width. Include member functions to:

1. Set the dimensions of the rectangle.
2. Calculate and return the area of the rectangle.
3. Calculate and return the perimeter of the rectangle.
4. Display the dimensions, area, and perimeter.

Solution Description: This program will help students understand how to define a class with private attributes and public member functions. The **Rectangle** class will encapsulate the properties of a rectangle and provide methods to manipulate and access these properties. The program will include a constructor to initialize the rectangle's dimensions, methods to calculate the area and perimeter, and a display function to output the rectangle's details. This assignment reinforces concepts of encapsulation, data hiding, and basic object manipulation in C++.

Assignment 2: Constructors and Destructors

Program Statement: Create a class **Complex** to represent complex numbers. Implement:

1. A default constructor to initialize the real and imaginary parts to zero.
2. A parameterized constructor to initialize the real and imaginary parts to given values.
3. A destructor to display a message when an object is destroyed.
4. A member function to display the complex number in the form **a + bi**.

Solution Description: This assignment emphasizes the use of constructors and destructors in a class. The **Complex** class will have attributes for the real and imaginary parts. Constructors will initialize these attributes, either to default values or to user- provided values. The destructor will be used to demonstrate when an object goes out of scope and is destroyed. The display function will format and print the complex number. This task helps in understanding object lifecycle management and resource cleanup in C++.

Assignment 3: Implementing Inheritance

Program Statement: Create a base class **Shape** with a pure virtual function **area()**. Derive two classes **Circle** and **Square** from **Shape**. Implement:

1. The constructor for each derived class.

2. The **area ()** function to calculate and return the area for each shape.
3. A function to display the area.

Solution Description: This assignment introduces the concept of inheritance and polymorphism. The **Shape** class serves as an abstract base class with a pure virtual function **area()**. The derived classes **Circle** and **Square** implement the **area()** function to calculate the area specific to each shape. The constructors initialize the radius and side length, respectively. By using base class pointers to call the **area()** function, students will understand dynamic binding and polymorphism in C++.

Assignment 4: Function Overloading

Program Statement: Write a C++ program to demonstrate function overloading by creating a class **Math** with multiple **add()** functions to handle:

1. Addition of two integers.
2. Addition of two floating-point numbers.
3. Addition of three integers.

Solution Description: Function overloading allows multiple functions with the same name but different parameters. The **Math** class will have overloaded **add()** functions to handle different types and numbers of arguments. This program will show how the same function name can be used to perform different operations based on the input parameters. This assignment helps in understanding the concept of function overloading and its applications in C++.

Assignment 5: Operator Overloading

Program Statement: Create a class **Complex** to represent complex numbers. Overload the **+** operator to add two complex numbers. The program should:

1. Include a constructor to initialize the real and imaginary parts.
2. Overload the **+** operator.
3. Display the result of the addition.

Solution Description: Operator overloading allows operators to be redefined for user-defined types. The **Complex** class will include a constructor for initialization and an overloaded **+** operator to add two **Complex** objects. The program will create **Complex** objects, perform the addition using the overloaded operator, and display the result. This assignment covers operator overloading and custom behavior for operators, allowing

students to extend the functionality of existing operators to work with user-defined types.

Assignment 6: Inheritance with Function Overriding

Program Statement: Create a base class **Animal** with a virtual function **sound()**. Derive two classes **Dog** and **Cat** from **Animal** and override the **sound()** function in each derived class. The program should:

1. Create objects of **Dog** and **Cat**.
2. Call the **sound()** function using a pointer to the base class.

Solution Description: Function overriding allows a derived class to provide a specific implementation of a function already defined in its base class. The **Animal** class will have a virtual **sound()** function, which will be overridden in the **Dog** and **Cat** classes to provide specific sounds. By using base class pointers to call the **sound()** function, the program will demonstrate polymorphism. This assignment helps in understanding inheritance, function overriding, and runtime polymorphism in C++.

Assignment 7: Implementing a Copy Constructor

Program Statement: Create a class **Book** with attributes title, author, and price. Implement:

1. A parameterized constructor to initialize the attributes.
2. A copy constructor to create a copy of a **Book** object.
3. A function to display the book details.

Solution Description: A copy constructor is used to create a new object as a copy of an existing object. The **Book** class will have attributes for the title, author, and price, and a parameterized constructor to initialize them. The copy constructor will create a new **Book** object with the same attribute values as an existing object. The display function will print the details of the book. This assignment helps in understanding deep copying and the role of copy constructors in C++.

Assignment 8: Implementing Dynamic Memory Allocation

Program Statement: Create a class **Student** with attributes name and marks. Implement:

1. A constructor to dynamically allocate memory for the name.
2. A destructor to deal locate the memory.
3. A function to display the student details.

Solution Description: Dynamic memory allocation involves allocating memory at

runtime using pointers. The **Student** class will have a constructor that allocates memory for the name attribute and a destructor that deal locates this memory to prevent memory leaks. The display function will output the student's details. This assignment helps in understanding dynamic memory management, constructors, destructors, and the importance of resource management in C++.

Assignment 9: Friend Function

Program Statement: Create a class **Box** with private attributes length, width, and height. Implement:

1. A constructor to initialize the attributes.
2. A friend function to calculate and return the volume of the box.
3. A function to display the dimensions and volume of the box.

Solution Description: A friend function is a non-member function that has access to the private and protected members of a class. The **Box** class will have a constructor to initialize its dimensions and a friend function to calculate the volume. The display function will print the box's dimensions and volume. This assignment helps in understanding friend functions and their use cases in C++.

Assignment 10: Template Class

Program Statement: Write a template class **Array** that can store elements of any data type. Implement:

1. A constructor to initialize the array with a given size.
2. A function to add elements to the array.
3. A function to display the elements of the array.

Solution Description: Templates allow classes and functions to operate with generic types. The **Array** template class will support any data type, providing flexibility and reusability. The constructor will initialize the array with a specified size, the function to add elements will store values in the array, and the display function will print all elements. This assignment helps in understanding templates and their benefits in creating generic and reusable code in C++.

Assignment 11: Implementing a Linked List

Program Statement: Create a class **Linked List** to represent a singly linked list of integers.

Implement:

1. A constructor to initialize an empty list.
2. A function to add a node at the end.
3. A function to delete a node from the beginning.
4. A function to display the elements of the list.

Solution Description: The **Linked List** class will encapsulate the properties of a singly linked list. The class will have a nested **Node** structure with an integer data field and a pointer to the next node. The constructor will initialize an empty list. The **add Node** function will append nodes to the end of the list, while the **delete Node** function will remove the node at the beginning. The **display** function will traverse and print the list elements. This assignment reinforces concepts of dynamic memory allocation and pointer manipulation in C++.

Assignment 12: Implementing a Stack Using Class

Program Statement: Create a class **Stack** to represent a stack of integers. Implement:

1. A constructor to initialize an empty stack.
2. A function to push an element onto the stack.
3. A function to pop an element from the stack.
4. A function to display the elements of the stack.

Solution Description: The **Stack** class will use an array or a linked list to store stack elements. The constructor will initialize the stack, and the **push** function will add elements to the top. The **pop** function will remove the top element, and the **display** function will print all elements from the top to the bottom. This assignment helps in understanding stack operations and their implementation in C++.

Assignment 13: Implementing a Queue Using Class

Program Statement: Create a class **Queue** to represent a queue of integers. Implement:

1. A constructor to initialize an empty queue.
2. A function to enqueue an element at the end.
3. A function to dequeue an element from the front.
4. A function to display the elements of the queue.

Solution Description: The **Queue** class will use an array or a linked list to manage queue elements. The constructor will initialize the queue, and the **enqueue** function will add

elements to the end. The **dequeue** function will remove elements from the front, and the **display** function will print all elements from the front to the end. This assignment covers the concept of queue operations and their implementation in C++.

Assignment 14: Implementing Polymorphism

Program Statement: Create a base class **Vehicle** with a virtual function **display()**. Derive two classes **Car** and **Bike** from **Vehicle**. Override the **display()** function in each derived class. The program should:

1. Create objects of **Car** and **Bike**.
2. Call the **display()** function using a pointer to the base class.

Solution Description: Polymorphism allows methods to be used interchangeably based on the object type at runtime. The **Vehicle** class will have a virtual **display()** function, which will be overridden in the **Car** and **Bike** classes to provide specific implementations. The program will use base class pointers to demonstrate polymorphism by calling the **display()** function on **Car** and **Bike** objects. This assignment helps understand runtime polymorphism and dynamic binding in C++.

Assignment 15: Implementing Abstract Classes

Program Statement: Create an abstract base class **Employee** with a pure virtual function **calculate Salary()**. Derive two classes **Full Time Employee** and **Part Time Employee** from **Employee**. Implement:

1. The **calculate Salary()** function in each derived class.
2. A function to display the salary details.

Solution Description: Abstract classes cannot be instantiated and are used to define interfaces for derived classes. The **Employee** class will have a pure virtual **calculate Salary()** function, making it abstract. The **Full Time Employee** and **Part Time Employee** classes will provide concrete implementations of the **calculate Salary()** function. The display function will print the salary details. This assignment helps understand abstract classes, pure virtual functions, and their role in defining interfaces in C++.

Assignment 16: Implementing a Template Function

Program Statement: Write a template function **find Max** to find the maximum of two elements. The program should:

1. Use the **find Max** function with different data types (int, float, char).
2. Display the results.

Solution Description: Template functions allow a function to operate with generic types. The **find Max** function will compare two elements of any data type and return the maximum. The program will demonstrate the function with different data types, showcasing its versatility and reusability. This assignment helps understand the concept of templates and their benefits in creating generic functions in C++.

Assignment 17: Exception Handling

Program Statement: Create a C++ program to demonstrate exception handling. The program should:

1. Implement a function that performs division of two numbers.
2. Throw an exception if the divisor is zero.
3. Catch the exception and display an appropriate error message.

Solution Description: Exception handling allows a program to handle runtime errors gracefully. The division function will throw an exception if an attempt is made to divide by zero. The program will catch the exception and display an error message, preventing the program from crashing. This assignment helps in understanding try, catch, and throw mechanisms in C++ for robust error handling.

Assignment 18: Implementing Copy Constructor and Assignment Operator Program Statement: Create a class **String** to represent a dynamic string. Implement:

1. A parameterized constructor to initialize the string.
2. A copy constructor to create a copy of a **String** object.
3. An overloaded assignment operator to assign one **String** object to another.
4. A function to display the string.

Solution Description: The **String** class will manage a dynamically allocated character array. The copy constructor will ensure a deep copy of the string, and the assignment operator will handle assignment between objects, preventing memory leaks. The display function will print the string. This assignment covers dynamic memory management, copy constructors, and assignment operators in C++.

Assignment 19: Implementing Static Members

Program Statement: Create a class **Counter** with a static data member to keep track of the

number of objects created. Implement:

1. A constructor to increment the counter.
2. A static member function to display the count of objects created.

Solution Description: Static members are shared among all objects of a class. The **Counter** class will have a static data member to count the number of objects. The constructor will increment this counter, and the static member function will display the count. This assignment helps in understanding static data members and functions, and their shared nature across all instances of a class in C++.

Assignment 20: File Handling

Program Statement: Create a class **File Handler** to perform basic file operations. Implement:

1. A function to write data to a file.
2. A function to read data from the file and display it.

Solution Description: File handling allows programs to read from and write to files. The **File Handler** class will use file streams to perform these operations. The write function will output data to a file, and the read function will input data from the file and display it. This assignment helps in understanding file I/O operations and their implementation using file streams in C++.