GIRIJANANDACHOWDHURYUNIVERSITY
Hathkhowapara, Azara, Guwahati781017, Assam

GIRIJANANDA CHOWDHURY UNIVERSITY,GUWAHATI(ASSAM)

OBJECT ORIENTED PROGRAMING

LAB MANUAL

COURSE CODE: BCS23202P

Prepared by:-

RUBI KALITA
Laboratory Instructor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEING
GUWAHATI-781017

GIRIJANANDACHOWDHURYUNIVERSITY
Hathkhowapara, Azara, Guwahati781017, Assam

	OBJECT ORIENTED PROGRAMMING USING C++ Lab	L	T	P	C
		0	0	4	2

Pre-requisite:

1. Basic of computer knowledge.
2. Knowledge of C programming.

Course Objectives:

1. To introduce the concept of object orientation to C++.
2. To introduce solving real world problems.
3. It will help in acquainting the techniques and applications of C++ for programming based onchallenging tasks.

Course Outcome:

After successful completion of the course, the students will learn

CO1: Create and explain Basic C++ Program using i/o variables and structures.
CO2: Apply object oriented programming concepts using class and objects.
CO3: Design and assess the classes for code reuse.
CO4: Analyze and Apply the generic classes and exception handling concepts in programming problem.

List of Experiments:

Lab 1	Basics of C++ Programming.	**2hours**
Lab 2	WAP using Switch Case to add, subtract, multiply and divide of two numbers.	**1 hour**
Lab 3	WAP to illustrate Class Declaration, Definition, Member function, objects. (Area of Trapezium, Rhombus, Circle, Triangle).	**4 hours**
Lab 4	WAP to create a simple class named Account and write methods to deposit and withdraw amount from the account.	**2 hours**
Lab 5	WAP to demonstrate the usage of a Constructor and Destructor in a class.	**1 hour**
Lab 6	WAP to illustrate parameterized constructor.(default, copy constructor)	**2 hours**
Lab 7	WAP to demonstrate: a) Operator Overloading b) Function Overloading.	**2 hours**

Lab 8	WAP to demonstrate Hybrid Inheritance. (Single, Multiple, Multi-level, Hierarchical)	**5 hours**
Lab 9	WAP to demonstrate Friend Function and Friend class.	**2 hours**
Lab 10	WAP to demonstrate polymorphism by calculating area of a rectangle and triangle using Shape class.	**2 hours**
Lab 11	WAP to demonstrate Virtual function.	**2 hours**
Lab 12	WAP to overload +operator to add two numbers.	**1hour**
Lab 13	WAP to create a Class Template.	**2 hours**
Lab 14	WAP to demonstrate exception handling.	**2 hours**
	Total	**30 hours**

Text Book:

1.E Balaguruswamy (2013), Object-oriented programming with C++, 6th Edition, Mc GrawHill Education.

2. Bjarne Stroustrup (2013), The C++ Programming Language, 4th Edition, Addison-Wesly.

3.Herbert Schildt (2017), C++: The Complete Reference, 4th Edition, McGraw Hill Education.

Reference Books:

1.Reema Thareja,(2016),Object Oriented Programming with C++,1st Edition, Oxford University.

GIRIJANANDACHOWDHURYUNIVERSITY
Hathkhowapara, Azara, Guwahati781017, Assam

LIST OF EXPERIMENTS

I)CLASS-OBJECT:

a) Create a class area with suitable data member and member functions. Find out area of a triangle, circle and rectangle using switch case (menu base).
b) Write a C++ program to implement a class called BankAccount that has private member variables for account number and balance. Include member functions to deposit and withdraw money from the account.

II) CONSTRUCTOR-DESTRUCTOR:

Define a class in C++ to represent Employee. Include the following data members:

1. Name of the Employee
2. Employee ID
3. Basic Salary

Constructor &Member functions:
1. To assign initial values.
2. Find out Gross and Net Salary If
   DA 100% of Basic Salary
   Medical 10 % of Basic Salaray
   HRA 20% of Basic Salary
   PF 10% of Basic Salary
   Tax 5% of Basic Salary

Use destructor to deallocate memory.
Write a main program to test the class. Modify the program for handling 10 Employee.

III) INHERITANCE:

a) Design C++ program to demonstrate example of Private and Protected simple Inheritance.
b) Create a base class Student. Derived class Marks from student to calculate total marks and percentage. Derived a class Display from class marks to display details student information.
c) Create a base class Student to input student's details. Create another base class Marks to calculate total marks and percentage of three subjects. Derived a class

Display from base class Student and Marks to display details student's information

d) Create base class Shape with suitable data member and member function, Create two derived class Triangle and Rectangle from base class Shape to calculate area and perimeter of triangle and rectangle.

e) Create a base class Student to input student's details. Derived a class Marks from base class Student to calculate total marks and percentage. Create another base class Sports to input grade securing in sports. Derived a class Result from class Marks and Sports to calculate result.

IV) POLYMORPHISM:

a) Design a C++ program to swap integer, float and character variable using function overloading.

b) Create a C++ class that contains one data member to overload unary "++" operator.

c) Design a class to use binary "+" operator to add two timestamp.

d) Create a base class called 'SHAPE' having
-two data members of type double
- member function get-data( ) to initialize base class data members - **pure virtual member function** display-area( ) to compute and display the area of the geometrical object. Derive two specific classes 'TRIANGLE' and 'RECTANGLE' from the base class. Using these three classes design a program that will accept dimension of a triangle / rectangle interactively and display the area.

V) Design a **template class** to implement stack operation using array.

VI) Write a C++ program to demonstrate the use of try, catch and throw in **exception handling.**

VII) Write a C++ Program to Maintain Book Records using **File Handling**.

Class Object:

Theoretical Background:

Classes and objects are the two main aspects of object-oriented programming.

Class: A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

Object: An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Excerise:-

1. Write a c++ program to declare a class "student" having data members student name, rollno and branch. Accept and display data for student.

Program Code:

```
#include<iostream>
using namespace std;
class student
{
        char st_name[20],branch[20];
        int rollno;
void input()
{
        cout<<"Enter student name::";
        cin>>st_name;
        cout<<"Enter branch";
        cin>>branch;
        cout<<"Enter Rollno::";
        cin>>rollno;
}
void display()
{
        cout<<"Student name:"<<st_name;
        cout<<"Branch::"<<branch;
        cout<<"Roll no::"<<rollno;
}
};
int main()
{
        student st;
        st.input();
        st.display();
}
```

GIRIJANANDACHOWDHURYUNIVERSITY
Hathkhowapara, Azara, Guwahati781017, Assam

AIM:

I) Define a class in C++ to represent a bank account. Include the following data members:

1. Name of the depositor
2. Account number
3. Type of account
4. Balance amount in the account

Member functions:

1. To assign initial values
2. To deposit an amount
3. To withdraw an amount after checking the balance
4. To display name and balance

Write a main program to test the class. Modify the program for handling 10 customers.

Step 1:

Create a class bank_account with following data member
Depositor name, Account number, account type, Balance amount

Step 2:

Create suitable member functions to
1.assign initial values
2.deposit an amount
3. withdraw an amount after checking the balance
4.display name and balance

Step 3:

Create a object for class bank_account and call all member functions.

Step 4:

Modify the program for handling 10 customers.

Constructor-Destructor:
Theoretical Background:

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.
The main purpose of the
class constructor in C++ programming is to construct an object of the class. In other word, it is used to initialize all class data members. ...

Note that if we don't write a constructor in the class, compiler will provide default constructor in C++ programming.

Types of Contractor

Default Constructor

Example:

```
#include <iostream>
using namespace std;
class DemoDC {
 private:
 int num1, num2 ;
public:
DemoDC() {
 num1 = 10;
 num2 = 20;
}
void display() {
cout<<"num1 = "<< num1 <<endl;
cout<<"num2 = "<< num2 <<endl;
 }
};
int main()
{
DemoDC obj;
obj.display();
return 0;
}
```

Parameterized Constructor

Example:

```
#include <iostream>
using namespace std;
class ParamA {
private:
int b, c;
public:
ParamA (int b1, int c1)
{
b = b1;
c = c1;
}
int getX ()
{
return b;
}
int getY ()
{
return c;
}
};
int main ()
{
ParamA p1(10, 15);
cout << "p1.b = " << p1. getX() << ", p1.c = " << p1.getY();
return 0;
}
```

Copy Constructor

Example:

```
#include <iostream>
using namespace std;
class Check
{
public:
int val;
Check(int a)
{
val=a;
}
Check(Check &i)
```

```
{
val = i.val;
}
};
int main()
{
int add_val;
Check a1(50);
Check a2(a1);
add_val = a2.val + 10;
cout<<add_val;
}
```

What is destructor?
Destructor is a member function which destructs or deletes an object.
When is destructor called?
A destructor function is called automatically when the object goes out of scope:
(1) the function ends
(2) the program ends
(3) a block containing local variables ends
(4)a delete operator is called

AIM:

III) A book shop maintains the inventory of books that are being sold at the shop. The list includes details such as author, title, price, publisher and stock position. Whenever a customer wants a book, the sales person inputs the title and author and the system searches the list and displays whether it is available or not. If it is not, an appropriate message is displayed. If it is, then the system displays the book details and requests for the number of copies required. If the required copies are available, the total cost of the requested copies is displayed; otherwise the message "Required copies not in stock" is displayed. Design a system in C++ using a class called "books" with suitable member functions **and constructors. Use new operator in constructors to allocate memory space required.**

STEP 1:Create a class book with data members author, title, price, publisher and stock position

STEP 2:Include suitable constructor and member functions to

i. **input book details**
ii. **find out customer required book is available or not.**
iii. **If not message is displayed**
iv. **If yes display book details and request no of copies.**
v. **If no of copies available find out total cost**
vi. **else display suitable message.**

STEP 3: Modify it with more than one books record.

Operator Overloading:
Theoretical Background:

Using **operator overloading** in C++, you can specify more than one meaning for an operator in one scope. The purpose of operator overloading is to provide a special meaning of an operator for a user-defined data type.

With the help of operator overloading, you can redefine the majority of the C++ operators. You can also use operator overloading to perform different operations using one operator.

Syntax

To overload a C++ operator, you should define a special function inside the Class as follows:

```
class class_name
{
   ... .. ...
   public
      return_type operator symbol (argument(s))
      {
         ... .. ...
      }
   ... .. ...
};
```

Here is an explanation for the above syntax:

- The return_type is the return type for the function.
- Next, you mention the operator keyword.
- The symbol denotes the operator symbol to be overloaded. For example, +, -, <, ++.
- The argument(s) can be passed to the operator function in the same way as functions.

Example 1:

```
#include <iostream>

using namespace std;

class TestClass {

private:

    int count;

public:

    TestClass() : count(5) {}

    void operator --() {

        count = count - 3;

    }

    void Display() {

        cout << "Count: " << count; }

};

int main() {

    TestClass tc;

    --tc;

    tc.Display();

    return 0;

}
```

GIRIJANANDACHOWDHURYUNIVERSITY
Hathkhowapara, Azara, Guwahati781017, Assam

AIM:

IV) Create a C++ class FLOAT that contains one float data member. Overload all the four arithmetic
operators so that they operate on the objects of FLOAT.

STEP 1:Create a class FLOAT with one float type data member
STEP 2: Overload arithmetic operator + to add two float object.
STEP 3: Overload arithmetic operator - to Subtract two float object.
STEP 4: Overload arithmetic operator * to add multiply two float object.
STEP 5: Overload arithmetic operator / to divide two float object.

Inheritance:

Theoretical Background:

Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

The idea of inheritance implements the **is a** relationship.

When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance.

Type of Inheritance

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Single inheritance Syntax:

```
class A   // base class
{
```

.....

};

class B : acess_specifier A // derived class

{

.....

} ;

Multiple Inheritance Syntax:

class A

{

.....

};

class B

{

.....

} ;

class C : acess_specifier A,access_specifier A // derived class from A and B

{

.....

} ;

Multilevel Inheritance Syntax

class A // base class

{

.....

};

class B : acess_specifier A // derived class

{

.....

} ;

class C : access_specifier B // derived from derived class B

{

.....

} ;

Hierarchical Inheritance Syntax:

class A // base class

{

.....

};

class B : access_specifier A // derived class from A

{

.....

} ;

class C : access_specifier A // derived class from A

{

.....

} ;

class D : access_specifier A // derived class from A

{

.....

} ;

Hybrid Inheritance Syntax:

class A

{

```
    .....
};
class B : public A
{
    .....
} ;
class C
{
    .....
};
```

AIM:

v)Assume that a bank maintains two kinds of accounts for customers, one called as Savings account
and the other as current account. The savings account gives compound interest and withdrawal facilities but no cheque book facility. The current account provides cheque book facility but no interest. Current account holders should also maintain a minimum balance and if the balance falls
below this level, a service charge is imposed. Create a C++ class account that stores customer name,account number and type of account. From this, derive the classes curr_acct and sav_acct to make
them more specific to their requirements. Include necessary member functions in order to achieve the
following tasks:

a) Accept deposit from a customer and update the balance
b) Display the balance
c) Compute and deposit interest
d) Permit withdrawal and update the balance
e) Check for the minimum balance, impose penalty and update the balance

STEP 1: Create a base class **Account** with suitable data members and member functions.
STEP 2: Derived a class **sav_acct** from base class **Account**
Add data member for calculating compound interest and withdraw for withdrawal money.
Display a message no cheque book facility.
STEP 3: Derived a class **curr_acct** from base class Account
add suitable data members and member functions

for check the minimum balance
impose penalty if balance falls and
update the balance

Virtual Function:
Theoretical Background:

A virtual function is a member function that is declared within a base class and redefined by a derived class. To create virtual function, precede the function's declaration in the base class with the keyword virtual. When a class containing virtual function is inherited, the derived class redefines the virtual function to suit its own needs.
- Base class pointer can point to derived class object. In this case, using base class pointer if we call some function which is in both classes, then base class function is invoked. But if we want to invoke derived class function using base class pointer, it can be achieved by defining the function as virtual in base class, this is how virtual functions support runtime polymorphism.

Virtual Function:
Theoretical Background:

What is a Virtual Function in C++?

A virtual function in C++ is a base class member function that we can redefine in a derived class to achieve polymorphism.We can declare the function in the base class using the virtual keyword. Once we declare the function in the base class, We can use a pointer or reference to call the virtual class and execute its virtual version in the derived class. Thus, it asks the compiler to determine the object's type during run-time and create a function bind (late binding or dynamic linkage).

A virtual function in C++ helps ensure us call the correct function via a reference or pointer. The C++ programming language allows we only to use a single pointer to refer to all the derived class objects. Since the pointer refers to all the derived objects, calling it will consistently execute the function in the base class. We can overcome this challenge with a virtual function in C++ as it helps execute the virtual version of the derived class, which is done at the run-time.

Rules of Virtual Function in C++

There are a few rules we need to follow to create a virtual function in C++. These rules are:

- The functions cannot be static

- You derive them using the “virtual” keyword
- Virtual functions in C++ needs to be a member of some other class (base class)
- They can be a friend function of another class
- The prototype of these functions should be the same for both the base and derived class
- Virtual functions are accessible using object pointers
- Redefining the virtual function in the derived class is optional, but it needs to be defined in the base class
- The function call resolving is done at run-time
- You can create a virtual destructor but not a constructor

Using a Virtual Function in C++

```
#include <iostream>
using namespace std;
class base {
public:
virtual void show(){
cout << "show base class" << endl;
}
void print(){

cout << "print base class" << endl;

}

};

class derived : public base {

public:
```

```
void show(){

cout << "show derived class" << endl;

}

void print(){

cout << "print derived class" << endl;

}

};

int main(){

base* bpointr;

derived dev;

bpointr = &dev;

// runtime binding

bpointr->show();

// compile time binding

bpointr->print();

}
```

What is a Pure Virtual Function in C++?

A pure virtual function in C++, also known as the do-nothing function, is a virtual function that does not perform any task. It is only used as a placeholder and does not contain any function definition (do-nothing function). It is declared in an abstract base class. These types of classes cannot declare any objects of their own. You derive a pure virtual function in C++ using the following syntax:

```
Virtual void class_name() = 0;
```

Example of Pure Virtual Functions in C++

```
#include <iostream>

using namespace std;

class Base{

   public:

   virtual void Output() = 0;

};

class Derived : public Base{

   public:

   void Output()

   {

      std::cout << "Class derived from the Base class." << std::endl;
```

```
    }
};
int main(){
    Base *bpointr;
    Derived dpointr;
    bpointr = &dpointr;
    bpointr->Output();
    return 0;
}
```

AIM:

VI) Create a base class in C++ called shape. Use this class to store two double type values that could
be used to compute the area of figures. Derive two specific classes called triangle and rectangle from
the base shape. Add to the base class a member function get_data () to initialize base class data members and another member function display_area () to compute and display the area of figures.
Make display_area () a virtual function and redefine this function in the derived classes to suit their
requirements. Using these classes, design a program that will accept the dimensions of a triangle or a
rectangle interactively and display the area.

STEP 1: Create a base class Shape with two double type data members.
Add member function get_data to initialize double type data members.
Add another virtual member function display_area to compute and display area of figures.
STEP 2: Create two derived classes triangle and rectangle from base class shape.
STEP 3: Redefine virtual function display_area in the both derived classes .
STEP 4: Compute area of a triangle and area of a rectangle using display_area function.

Template Class:
Theoretical Background:

A template is a simple yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. For example, a software company may need to sort() for different data types.

A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept. There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector or vector .
Function Template: The general form of a template function definition is shown here:

```
template ret-type func-name(parameter list)
{ // body of function }
```

Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition. The following is the example of a function template that returns the maximum of two values:

```
#include <iostream>
#include <string>
using namespace std;
template inline T const& Max (T const& a, T const& b)
{ return a < b ? b:a; }
int main ()
{ int i = 39; int j = 20; cout << "Max(i, j): " << Max(i, j) << endl;
double f1 = 13.5;
double f2 = 20.7;
cout << "Max(f1, f2): " << Max(f1, f2) << endl;
string s1 = "Hello";
string s2 = "World";
cout << "Max(s1, s2): " << Max(s1, s2) << endl;
return 0; }
```

If we compile and run above code, this would produce the following result:

Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World

Class Template: Just as we can define function templates, we can also define class templates. The general form of a generic class declaration is shown here:

template class class-name
{
.
.
.
}

AIM:
VII) Implement a template class in C++ for stack data structure and show how it can be used inside main function.

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

                    // Class Stack with default parameter
                    template<typename T, int N=10>
                    class Stack
                    {
                    public:
                    Stack();
                    ~Stack();
                    void Push(T data);
                    T Pop();
                    private:
                    T *Data;
                    int count;
                    };

                    // Constructor
                    template<typename T, int N>
```

```
Stack<T,N>::Stack()
{
Data = new T[N];
count = 0;
}

// Destructor
template<typename T, int N>
Stack<T,N>::~Stack()
{
delete [] Data;
}

// Method to push data
template<typename T, int N>
void Stack<T,N>::Push(T data)
{
if (count < N)
{
Data[count++] = data;
}
Else
{
cout<<"\n stack Overflow!!!"<<endl;
}
}

// Methos to pop data
template<typename T, int N>
T Stack<T,N>::Pop()
{
if (count > 0)
{
return Data[--count];
}
Else
```

```
{
cout<<"\n stack is Empty"<<endl;
}
}

int main()
{
Stack<int,5> stack;
Stack<int> stack1;
stack.Push(10);
stack.Push(20);
stack.Push(70);
stack.Push(50);
stack.Push(40);

cout<<stack.Pop()<<endl;
cout<<stack.Pop()<<endl;
cout<<stack.Pop()<<endl;
cout<<stack.Pop()<<endl;
cout<<stack.Pop()<<endl;
return 0;
}
```

```
{
```

Exception Handling:
Theoretical Background:

Exception handling in C++ is a mechanism to handle runtime errors, maintaining program flow and preventing crashes. It uses try, catch, and throw keywords. The try block encloses code that might throw an exception. When an error occurs within the try block, a throw statement raises an exception. The catch block then handles this exception.

```
#include <iostream>
#include <stdexcept>

int main() {
   try {
      int age = 15;
      if (age < 18) {
         throw std::runtime_error("Access denied - You must be at least 18 years old.");
      }
      std::cout << "Access granted." << std::endl;
   } catch (const std::runtime_error& error) {
      std::cerr << "Caught exception: " << error.what() << std::endl;
      return 1;
   }
   return 0;
}
```

In this example, if age is less than 18, a std::runtime_error is thrown. The catch block catches this exception, prints an error message, and the program exits gracefully. If no exception is thrown, "Access granted" is printed.

AIM:

IX) Write a program Handling the Divide by Zero Exception in C++

```
// Program to show division without using

// Exception Handling



#include <iostream>

using namespace std;



// Defining function Division

float Division(float num, float den)

{

   // return the result of division

   return (num / den);
```

```
} // end Division


int main()
{
   // storing 12.5 in numerator
   // and 0 in denominator
   float numerator = 12.5;
   float denominator = 0;
   float result;

   // calls Division function
   result = Division(numerator, denominator);

   // display the value stored in result
   cout << "The quotient of 12.5/0 is "
        << result << endl;

} // end main
```

Output

```
The quotient of 12.5/0 is inf
```