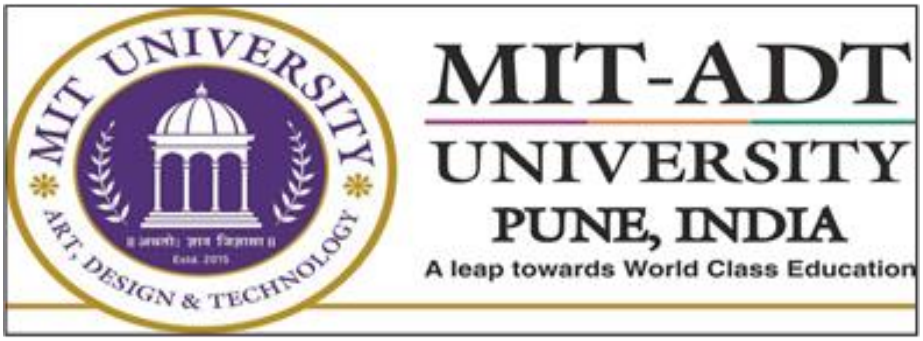
21BTCS202: Object Oriented Programming

AY 2021-22, SEM II



MIT School of Engineering

B. Tech First Year

LABORATORY ANUAL

(Student Copy)

21BTCS202

Object Oriented Programming

SEM II, AY 2021-22

Name: Sourav Shailesh Toshniwal

Class: FY-CSE A

Batch: A2

Roll no: 2213047

Assignment No. 1

Title: Basics of object-oriented programming

Concept: Class, data member, object
Problem statement: Develop a hotel bill application with the following details: table no., customer name, customer contact, details of order. Compute the bill amount by offering possible discounts on the given order.

Objective: To explore the principles of Object-Oriented Programming (OOP): Class and object.

Theory:
Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Input/ Output in C++
C++ comes with libraries that provide us with many ways for performing input and output. In C++ input and output are performed in the form of a sequence of bytes or more commonly known as streams.

1. Input Stream: If the direction of flow of bytes is from the device (for example , Keyboard) to the main memory then this process is called input.
2. Output Stream: If the direction of flow of bytes is opposite, 1.e., from mam memory to device (display screen) then this process is called output.
3. iostream: iostream stands for standard input-output stream. This header file contains definitions of objects like cin, cout, cerr, etc.

The two instances cout in C++ and cin in C++ of iostream class are used very often for printing outputs and taking inputs respectively. These two are the most basic methods of taking input and printing output in C++. To use cin and cout in C++ one must include the header file *iostream* in the program.

Standard output stream (cout): Usually the standard output device is the display screen. The C++ **cout** statement is the instance of the ostream class. It is used to produce output on the standard output device which is usually the display screen. The data needed to be

displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

standard input stream (cin): Usually the input device in a computer is the keyboard. C++ cin statement is the instance of the class **istream** and is used to read input from the standard input device which is usually a keyboard. The extraction operator (>>) is used along with the object **cin** for reading inputs. The extraction operator extracts the data from the object **cin** which is entered using the keyboard.

Classes and Objects:

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.

For Example: Consider the class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, 'Cars' is class and wheels, speed limits, mileage are their properties.

4. A Class is a user defined data-type which has data members and member functions.

5. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.
6. In the above example of class Car, the data member will be speed limit, mileage etc and member functions can apply brakes, increase speed etc.

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.

Classes are defined using keyword 'class' as shown below with sample example

Syntax and example

class class name

I

access specifi,er 1:

data member(s); memberJunction(s);

class ***Rectangl el***

I

public:
int width, height; void set values 0;

In C++, there are three access specifiers:

Public (+) - Members are accessible from outside the class

Private (-) - Members cannot be accessed (or viewed) from outside the class

Protected (#)- Members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

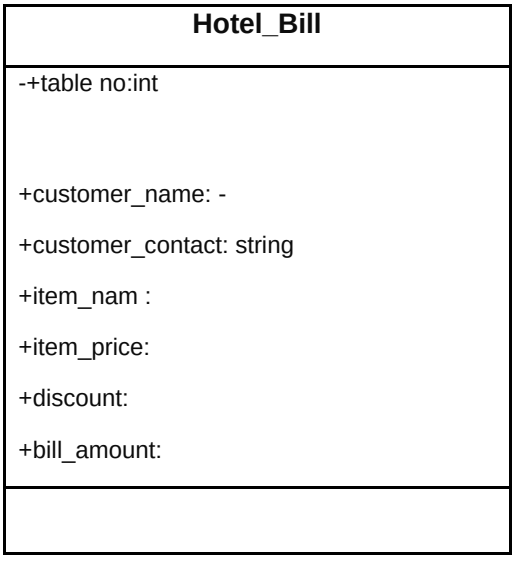
Algorithm:
I.START

1. Read information table no., customer name, customer contact, order (item name and item price).
2. Compute the bill amount based on offered discount (assume 5%)
3. Display the table no., customer name, customer contact, order (item name and item price).

   along with bill amount

4. STOP

Class diagram and Flowchart:
"+" Public



Conclusion: Basic input output operations are performed. Learned use of class and object in simple hotel bill application.

Code with output:

CODE:

```
#include <iostream>
using namespace std;
class hotel
{
public:
    int table_num;
    string cust_name;
    long int cust_contact;
    string item_name;
    int item_price;
    int final_price;
};
int main()
{
    hotel h;
    cout<<"Enter the table no:";
    cin>> h.table_num;
    cout<<"Enter the customer name:";
    cin>> h.cust_name;
    cout<<"Enter the customer contact:";
    cin>> h.cust_contact;
    cout<<"Enter the Item name:";
    cin>> h.item_name;
    cout<<"Enter the item price:";
    cin>> h.item_price;
    // for discount 10%
    h.final_price=0.9*h.item_price;
    cout<<"-----DETAILS-----"<<endl;
    cout<<"Table no:"<<h.table_num<<endl;
    cout<<"Customer name:"<<h.cust_name<<endl;
    cout<<"Customer contact:"<<h.cust_contact<<endl;
    cout<<"Total amount:"<<h.final_price<<endl;
    return 0;
}
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass1_2213047.cpp -o Ass1_2213047 } ; if ($?) { .\Ass1_2213047 }
Enter the table no:23
Enter the customer name:Sourav
Enter the customer contact:2343241
Enter the Item name:Pizza
Enter the item price:300
-----DETAILS-----
Table no:23
Customer name:Sourav
Customer contact:2343241
Total amount:270
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

Problem statement: Write a CPP program to generate a student mark sheet. Create a class student with the following members: name, branch, division, roll no, and marks in three subjects. Compute total and percentage . Display complete student information on screen.

(Student answer section)

Code with output:

```cpp
#include<iostream>

using namespace std;
class StudentInfo
{
public:
    string Name;
    string Branch;
    char Div;
    int Rno;
    float OOP_marks, ODEAC_marks, DELD_marks, Total=0;
    double Percent;
};
int main()
{
 StudentInfo s;
 cout<<"Enter the name of student:"<<endl;
 cin>>s.Name;
 cout<<"Enter the branch of student:"<<endl;
 cin>>s.Branch;
 cout<<"Enter the division of student:"<<endl;
 cin>>s.Div;
 cout<<"Enter the roll no of student:"<<endl;
 cin>>s.Rno;
 cout<<"Enter the OOP marks of student:"<<endl;
```

```
cin>>s.OOP_marks;
cout<<"Enter the ODEAC marks of student:"<<endl;
cin>>s.ODEAC_marks;
cout<<"Enter the DELD marks of student:"<<endl;
cin>>s.DELD_marks;
s.Total=s.OOP_marks+s.ODEAC_marks+s.DELD_marks;
s.Percent=s.Total/300*100;

cout<<"-STUDENT DETAILS-"<<endl;
cout<<"Name: "<<s.Name<<endl;
cout<<"Brach: "<<s.Branch<<endl;
cout<<"Division: "<<s.Div<<endl;
cout<<"OOP Marks: "<<s.OOP_marks<<endl;
cout<<"ODEAC Marks: "<<s.ODEAC_marks<<endl;
cout<<"DELD Marks: "<<s.DELD_marks<<endl;
cout<<"Total: "<<s.Total<<endl;
cout<<"Percentage: "<<s.Percent<<endl;

return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass1_2213047Prob.cpp -o Ass1_2213047Prob } ; if ($?) { .\Ass1_2213047Prob }
Enter the name of student:
Sourav
Enter the branch of student:
CSE
Enter the division of student:
A
Enter the roll no of student:
2213047
Enter the OOP marks of student:
99
Enter the ODEAC marks of student:
100
Enter the DELD marks of student:
89
-STUDENT DETAILS-
Name: Sourav
Brach: CSE
Division: A
OOP Marks: 99
ODEAC Marks: 100
DELD Marks: 89
Total: 288
Percentage: 96
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

Assignment No. 2

Title: Object and Class (data members and member functions)

Problem statement: Concept: Class, data members, member functions, object

Create a bank application that contains details of bank account holder:

i. Name of the depositor

ii. Account number

and provide the following services

i. Deposit an amount

ii. Withdraw an amount

iii. Display name and balance

Objective:

To understand the implementation of Object and Class along with data members and member functions.

Theory:

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.

For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

1. A Class is a user defined data-type which has data members and member functions.
2. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviour of the objects in a Class.
3. In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc and member functions can be *apply_brakes*, *increase_speed* etc.

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.

Defining Class and Declaring Objects:

A class is defined in C++ using the keyword class followed by the name of the class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

Declaring Objects:

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects

Syntax: Class_Name ObjectName;

.

Class AND Class diagram

A class is represented by a rectangle having three sections −

1. The top section containing the name of the class
2. The middle section containing class attributes
3. The bottom section representing operations of the class

Access Modifiers in C++:

Access modifiers are used to implement an important aspect of Object-Oriented Programming known as Data Hiding. Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

There are 3 types of access modifiers available in C++:

1. Public

2. Private
3. Protected

The visibility of the attributes and operations can be represented in the following ways −

1. **Public:**

   A public member is visible from anywhere in the system. In class diagram, it is prefixed by the symbol '+'.

   All the class members declared under the public specifier will be available to everyone. The data members and member functions declared as public can be accessed by other classes and functions too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

Example: (Public access specifier):

```
class Student
{
  public:                    // Access specifier
    int rno;                 //public variable
};

int main ()
{
Student stud;
stud.rno=10;                 //Allowed to access

cout<< "Student roll Number ="<<stud.rno;

return 0;
```

```
}
```

2. **Private** :

A private member is visible only from within the class. It cannot be accessed from outside the class. A private member is prefixed by the symbol '−'.

The class members declared as *private* can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

Note: It is possible to access private members of a class using a public method inside the same class.

Example: (Private access specifier):

Example:

```
class Student
{
  int mobilenumber;          //By Default Private Variable

 private:                     // Access specifier
    int rno;                  //private variable
};
```

```
int main()
{
Student stud;
stud.rno=10;                              // Not Allowed to access
stud.mobilenumber=8888882222              // Not Allowed to access
return 0;
}
```

3. **Protected :**

A protected member is visible from within the class and from the subclasses inherited from this class, but not from outside. It is prefixed by the symbol '#'.

Protected access modifier is similar to private access modifier in the sense that it can't be accessed outside of its class unless with the help of friend class, the difference is that the class members declared as Protected can be accessed by any subclass(derived class) of that class as well.

Example:(Protected access specifier):

```
// Base class
class Employee {
  protected: // Protected access specifier
    int salary;
};

// Derived class
class Programmer: public Employee {
```

```
  public:
    int bonus;
    void setSalary(int s) {
      salary = s;
    }
    int getSalary() {
      return salary;
    }
};

int main() {
  Programmer myObj;
  myObj.setSalary(50000);
  myObj.bonus = 15000;
  cout << "Salary: " << myObj.getSalary() << "\n";
  cout << "Bonus: " << myObj.bonus << "\n";
  return 0;
}
```

Example To Draw Class Diagram:

Let us consider the Circle class introduced earlier. The attributes of Circle are x-coord, y-coord, and radius. The operations are findArea(), findCircumference(), and scale(). Let us assume that x-coord and y-coord are private data members, radius is a protected data member, and the member functions are public. The following figure gives the diagrammatic representation of the class.

Object class diagram:
An object is represented as a rectangle with two sections −

1. The top section contains the name of the object with the name of the class or package of which it is an instance of. The name takes the following forms −
    1. object-name − class-name
    2. object-name − class-name:: package-name
    3. class-name − in case of anonymous objects
2. The bottom section represents the values of the attributes. It takes the form attribute-name = value.
3. Sometimes objects are represented using rounded rectangles.

Example − Let us consider an object of the class Circle named c1. We assume that the center of c1 is at (2, 3) and the radius of c1 is 5. The following figure depicts the object.

Accessing data members and member functions:

The data members and member functions of the class can be accessed using the dot('.') operator with the object. For example if the name of object is obj and you want to access the member function with the name printName() then you will have to write obj.printName() .

Accessing Data Members:

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers : public, private and protected.

Member Functions in Classes

There are 2 ways to define a member function:

7. Inside class definition
8. Outside class definition

To define a member function outside the class definition we have to use the scope resolution **::** operator along with class name and function name.

Example for defining member function Inside class definition:

```
class MyClass                    // The class
{

public:                          // Access specifier
   void myMethod()
   {                             // Method/function defined inside the class
     cout << "Hello World!";
   }
};

int main() {
  MyClass myObj;                 // Create an object of MyClass
  myObj.myMethod();              // Call the method
  return 0;
}
```

Note that all the member functions defined inside the class definition are by default **inline**, but you can also make any non-class function inline by using keyword inline with them. Inline functions are actual functions, which are copied everywhere during compilation, like preprocessor macro, so the overhead of function calling is reduced.

Note: Declaring a **friend function** is a way to give private access to a non-member function.

Example for defining member function Outside the class definition:

```
class MyClass
{                                       // The class
  public:                               // Access specifier
    void myMethod();                    // Method/function declaration
};


                                        // Method/function definition outside the class
void MyClass::myMethod()
{
  cout << "Hello World!";
}

int main() {
  MyClass myObj;                        // Create an object of MyClass
  myObj.myMethod();                     // Call the method
  return 0;
}
```

Algorithm:

1.START

2. Read information: Account holder's name, Account number and balance to deposit.

3. Read the user's choice A. Display name and balance B.Deposit an amount C. Withdraw an amount.

4. If user's choice is A Display name and balance

5. If the user's choice is B. Read the deposit amount from the user. Calculate the new balance and display it.

6. If the user's choice is C. Read the withdrawal amount from the user. Calculate the new balance and display it.

7. STOP

Class diagram and Flowchart:

"+" Public

Conclusion:

In this assignment we learnt how to define a class and object also in addition how to access data members and member functions.

Code with output:

```
#include<iostream>
using namespace std;
class BankDetails
{
public:
    string name, name1, name2;
    long long int AccNumber;
    double AccBalance=0;
    float AmountDepo, AmountWith;
```

```
public:
void Details()
{
    cout<<"Enter account holder name:";
    cin>>name;
    cout<<"Enter the account number:";
    cin>>AccNumber;
}
void deposit()
{
    cout<<"Enter the name of depositor:";
    cin>>name1;
    cout<<"Enter the amount to deposit:";
    cin>>AmountDepo;
    AccBalance+=AmountDepo;
    cout<<"The available balance in your account is"<<AccBalance<<endl;
}
void withdraw()
{
    cout<<"Enter the name of withdrawer:";
    cin>>name2;
    cout<<"Enter the amount to withdraw:";
    cin>>AmountWith;
    AccBalance=AccBalance-AmountWith;
    cout<<"The available balance in your account is"<<AccBalance<<endl;
}
};
int main()
{
 int Activity;
 BankDetails B;
 B.Details();
 char OP;
 do
 {
    cout<<"Enter 1 to deposit, 2 to withdraw, 3 to get details, any number to
exit"<<endl;
    cin>>Activity;
    switch (Activity)
     {
     case 1:
         B.deposit();
         break;
     case 2:
         B.withdraw();
```

```
        break;
    case 3:
        B.Details();
        break;
    default:
        break;
    }
   cout<<"Do you want to continue the process for others? Enter Y or N"<<endl;
   cin>>OP;
}while(OP=='Y');
   return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass2_2213047.cpp -o Ass2_2213047 } ; if ($?) { .\Ass2_2213047 }
Enter account holder name:Sourav
Enter the account number:3221341
Enter 1 to deposit, 2 to withdraw, 3 to get details, any number to exit
1
Enter the name of depositor:Naman
Enter the amount to deposit:2132
The available balance in your account is2132
Do you want to continue the process for others? Enter Y or N
Y
Enter 1 to deposit, 2 to withdraw, 3 to get details, any number to exit
2
Enter the name of withdrawer:Paras
Enter the amount to withdraw:231
The available balance in your account is1901
Do you want to continue the process for others? Enter Y or N
Y
Enter 1 to deposit, 2 to withdraw, 3 to get details, any number to exit
3
Enter account holder name:Sourav
Enter the account number:3221341
Do you want to continue the process for others? Enter Y or N
N
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

Problem statement: Write a program to print the area of a rectangle by creating a class named 'Area' having two functions and two data members: Length and breadth. First function named as 'setDim' takes the length and breadth of the rectangle as parameters and the second function named as 'getArea' returns the area of the rectangle. Length and breadth of the rectangle are entered through the keyboard.

(Followed by student answer section)

```
#include<iostream>
using namespace std;
class Area
{
  private:
  float length,breadth,area_rec;
  public:
  void setDim(void);
  void getArea(void);
};
```

```cpp
int main()
{
    Area a1;
    a1.setDim();
    a1.getArea();
    return 0;
}
void Area :: setDim()
{
    cout<<"\n Please enter length of the rectangle :";
    cin>>length;
    cout<<"\n Please enter breadth of the rectangle :";
    cin>>breadth;
}
void Area :: getArea()
{
    area_rec = length*breadth;
    cout<<"\n The area of the rectangle is : "<<area_rec;
}
```

Assignment No. 3

Title: Arithmetic operators and inheritance (Hierarchical) in CPP

Problem statement: Write a CPP program with Employee class with Emp_name, Emp_id, Address, Mail_id, Mobile_no as members. Inherit the classes, Programmer, Assistant Engineer, Business Analyst and Manager from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and net salary

Objective:

To understand how Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application with the use of arithmetic operators.

Theory:

Arithmetic Operators:

An Operator is a symbol which behaves like a function, that tells the computer or the compiler to perform some **mathematical operations** on variables and values. Operators are used in programming in order to manipulate data and variables. C++ has a predefined set of different Operators. One such commonly used Operator is the Arithmetic Operators.

Definition

In C++, Arithmetic Operators are symbols used to perform common arithmetic operations like addition, subtraction, multiplication, division, modulus, etc.

Arithmetic Operators are operators which are used within the equation to perform a number of basic mathematical calculations.

Arithmetic Operators

Arithmetic Operators use a specific symbol in order to be used within the program. Arithmetic Operators use two operands and an operator between them in order to calculate the result. Arithmetic Operators are simply sign's which we use in the mathematical equations in order to compute results.

Basic Arithmetic Operators in C++ are listed below:

Although the Arithmetic Operators are often used to calculate results between 2 values, they can also be used between 2 variables. Let us understand this by taking an example:

Addition Operator

Addition Arithmetic Operators are used to add 2 values and variables in a program statement. The symbol used for this type of operator is '+'. In simple words, Addition combines 2 numbers.

Example

```
#include <iostream>

using namespace std;


int main() {
  int x = 5;
  int y = 3;
  cout << x + y;
  return 0;
}
```

Subtraction Operator

Subtraction Arithmetic Operator is used to subtract 2 values or variables in the program. The symbol used for this operator is '-'.

Example:

```
#include <iostream>

using namespace std;

int main () {
  int x = 5;
  int y = 3;
  cout << x - y;
  return 0;
}
```

Multiplication Operator

This operator is denoted by the '*' or 'x' symbol. It gives the product of two numbers.

Example

```
#include <iostream>
```

```
using namespace std;

int main () {

  int x = 5;

  int y = 3;

  cout << x * y;

  return 0;

}
```

Division Operator

This Arithmetic Operator divides its first operand by the second operand. It is denoted by the '/' symbol. In C++, if we divide An Integer with another Integer then we will get the Quotient as the result. But if either of the operands is a floating-point number, then the result of division will be in Decimals.

Example

```
#include <iostream>

using namespace std;

int main () {

  int x = 12;

  int y = 3;

  cout << x / y;

  return 0;

}
```

Modulus Operator

This type of Arithmetic Operator computes the remainder of the two values when divided. The symbol used to define this operator is '%'. Both the operands must be integer type only. If they are of the floating-point type, then there might arise a compile-time error.

```
#include <iostream>
using namespace std;
int main() {
  int x = 5;
  int y = 2;
  cout << x % y;
  return 0;
}
```

Order of Operations

There can rise an issue when an operand has to be computed with 2 or more other operands at the same time. When more than one operator is applied to the same operand, C++ uses the rules of precedence to decide which operator actions to be applied first.

For example

*int speed = 12 * 5 + 34 ;*

Here the value of speed can be either 94 or 468.

In this above example, we can see that Value 5 is an operand for both the * and + operators. In this situation, C++ follows the Precedence Rules.

The Precedence Rules states that in such situations, the sequence to compute the value is multiplication, division, then modulus before the addition and subtraction mathematical operations.

Inheritance:

One of the most important concepts in object-oriented programming is that of inheritance. 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. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Advantage of C++ Inheritance

Code reusability: Now you can reuse the members of your parent class. So, there is no need to define the member again. So, less code is required in the class.

Types Of Inheritance

C++ supports five types of inheritance:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

Derived Classes

A Derived class is defined as the class derived from the base class.

The Syntax of Derived class:

```
class derived_class_name :: visibility-mode base_class_name
{
   // body of the derived class.
}
```

Where,

derived_class_name: It is the name of the derived class.

visibility mode: The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

base_class_name: It is the name of the base class.

- When the base class is privately inherited by the derived class, public members of the base class become the private members of the derived class. Therefore, the public members of the base class are not accessible by the objects of the derived class only by the member functions of the derived class.
- When the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class. Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the base class.

Note:

- In C++, the default mode of visibility is private.
- The private members of the base class are never inherited.

Base and Derived Classes

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form −

class derived-class: access-specifier base-class

Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

C++ Single Inheritance

Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

Where 'A' is the base class, and 'B' is the derived class.

Example:

```
#include <iostream>
using namespace std;
 class Animal {
   public:
 void eat () {
    cout<<"Eating..."<<endl;
 }
   };
   class Dog: public Animal
   {
      public:
     void bark (){
    cout<<"Barking...";
     }
   };
int main(void) {
    Dog d1;
    d1.eat();
    d1.bark();
    return 0;
}
```

Access Control and Inheritance

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

We can summarize the different access types according to - who can access them in the following way −

Access	public	protected	private
Same class	yes	yes	yes

Derived classes	yes	yes	no
Outside classes	yes	no	no

A derived class inherits all base class methods with the following exceptions −

- Constructors, destructors and copy constructors of the base class.
- Overloaded operators of the base class.
- The friend functions of the base class.

Multilevel inheritance is a process of deriving a class from another derived class.

Example:

```
#include <iostream>
using namespace std;
 class Animal {
   public:
 void eat() {
    cout<<"Eating..."<<endl;
 }
   };
   class Dog: public Animal
   {
       public:
     void bark(){
    cout<<"Barking..."<<endl;
     }
   };
   class BabyDog: public Dog
   {
       public:
     void weep() {
    cout<<"Weeping...";
```

```
    }
  };
int main(void) {
   BabyDog d1;
   d1.eat();
   d1.bark();
    d1.weep();
    return 0;
}
```

Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.

Syntax: class D : visibility B-1, visibility B-2, ?

```
{
   // Body of the class;
}
```

Example:

```
#include <iostream>
using namespace std;
class A
{
   protected:
    int a;
   public:
   void get_a(int n)
   {
       a = n;
```

```
    }
};
  class B
{
    protected:
    int b;
    public:
    void get_b(int n)
    {
        b = n;
    }
};
class C : public A,public B
{
   public:
    void display ()
    {
        std::cout << "The value of a is : " <<a<< std::endl;
        std::cout << "The value of b is : " <<b<< std::endl;
        cout<<"Addition of a and b is : "<<a+b;
    }
};
int main()
{
   C c;
   c.get_a(10);
   c.get_b(20);
   c.display();
```

```
    return 0;
}
```

C++ Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.

Example:

```
#include <iostream>
using namespace std;
class A
{
   protected:
   int a;
   public:
   void get_a()
   {
      std::cout << "Enter the value of 'a' : " << std::endl;
      cin>>a;
   }
};

class B : public A
{
   protected:
   int b;
   public:
```

```
    void get_b()
    {
        std::cout << "Enter the value of 'b' : " << std::endl;
       cin>>b;
    }
};
class C
{
    protected:
    int c;
    public:
    void get_c()
    {
        std::cout << "Enter the value of c is : " << std::endl;
        cin>>c;
    }
};

class D : public B, public C
{
    protected:
    int d;
    public:
    void mul()
    {
         get_a();
         get_b();
         get_c();
```

```
        std::cout << "Multiplication of a,b,c is : " <<a*b*c<< std::endl;
    }
};
int main()
{
    D d;
    d.mul();
    return 0;
}
```

C++ Hierarchical Inheritance

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.

Syntax of Hierarchical inheritance:

```
class A
{
    // body of the class A.
}
class B: public A
{
    // body of class B.
}
class C : public A
{
    // body of class C.
}
class D : public A
{
    // body of class D.
}
```

Example:

```
#include <iostream>
using namespace std;
class Shape                 // Declaration of base class.
{
   public:
   int a;
   int b;
   void get_data(int n,int m)
   {
      a= n;
      b = m;
   }
};
class Rectangle : public Shape  // inheriting Shape class
{
   public:
   int rect_area()
   {
      int result = a*b;
      return result;
   }
};
class Triangle : public Shape    // inheriting Shape class
{
   public:
   int triangle_area()
   {
```

```
        float result = 0.5*a*b;
        return result;
    }
};
int main()
{
    Rectangle r;
    Triangle t;
    int length,breadth,base,height;
    std::cout << "Enter the length and breadth of a rectangle: " << std::endl;
    cin>>length>>breadth;
    r.get_data(length,breadth);
    int m = r.rect_area();
    std::cout << "Area of the rectangle is : " <<m<< std::endl;
    std::cout << "Enter the base and height of the triangle: " << std::endl;
    cin>>base>>height;
    t.get_data(base,height);
    float n = t.triangle_area();
    std::cout <<"Area of the triangle is : "  << n<<std::endl;
    return 0;
}
```

Algorithm:

Step I: Start

Step II: Declare data members as

emp_name, address, emp_id, mail_id, mob_no, net_sal, da, hra, pf, fund, basic pay.

Step III: Read input for data members.

Step IV: Calculate net pay using 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP.

Step V: Display net pay for Programmer, Assistant Engineer, Business Analyst and Manager from employee class.

Step VI. STOP

Flowchart:

Class diagram:

Conclusion: Students were able to learn the concept of base class, derived class using Inheritance.

Code with output:

```
#include <iostream>
using namespace std;
class Employee
{
  private:
  string emp_name,emp_id,address,email,mobile_num;
  float bp,da,hra,gs,pf,cf,ns;
  public:
  void get_details(void);
  void compute_salary(void);
  void put_details(void);
};
void Employee :: get_details()
{
    cout<<"\n Please enter employee name:";
    cin>>emp_name;
    cout<<"\n Please enter employee id:";
    cin>>emp_id;
    cout<<"\n Please enter address :";
    cin>>address;
    cout<<"\n Please enter employee email id :";
    cin>>email;
    cout<<"\n Please enter employee mobile number :";
    cin>>mobile_num;
    cout<<"\n Please enter Basic salary :";
    cin>>bp;
}
void Employee :: compute_salary()
{
    da = 0.97*bp;
    hra = 0.1*bp;
    pf = 0.12*bp;
```

```
    cf = 0.001*bp;
    gs = bp+da+hra;
    ns = gs-pf-cf;
}

void Employee :: put_details()
{
    cout<<"Employee name :"<<emp_name;
    cout<<"Employee id :"<<emp_id;
    cout<<"Employee address :"<<address;
    cout<<"Employee email id :"<<email;
    cout<<"Employee mobile number :"<<mobile_num;
    cout<<"Employee basic salary :"<<bp;
    cout<<"Employee gross salary :"<<gs;
    cout<<"Employee net salary :"<<ns;

}
class Programmer : public Employee
{
    private:
    string language,designation,domain;

    public:
    void get_details(void);
    void put_details(void);
};
void Programmer :: get_details()
{
    cout<<"\n Programming language :";
    cin>>language;
    cout<<"\n Designation :";
    cin>>designation;
    cout<<"\n Domain :";
    cin>>domain;
}
void Programmer :: put_details()
{
    cout<<"\n Language :"<<language;
    cout<<"\n Designation :"<<designation;
    cout<<"\n Domain :"<<domain;
}
int main()
{
    Employee e1;
    Programmer p1;
    e1.get_details();
    e1.compute_salary();
    e1.put_details();
    p1.get_details();
    p1.put_details();
    return 0;
}
```

(Followed by student answer section)

// Provide 01 similar programming-based question (algorithm, flowchart, code & screenshot)

(Followed by student answer section)

Code:

```cpp
#include <iostream>
using namespace std;
class student
{
    private:
        string name,branch;
        int rno;
    public:
        void get_details(void);
        void put_details(void);
};
void student::get_details(void)
{
    cout << "\t---------------Enter student's basic information:----------" << endl;
    cout << "Name: ";
    cin >> name ;
    cout << "Branch: ";
    cin >> branch ;
    cout << "Roll number: ";
    cin >> rno ;
}
void student::put_details(void)
{
    cout <<"\n\t ---------Student details------------";
    cout <<"\nName: " << name;
    cout <<"\nBranch: " << branch;
    cout <<"\nRoll number: " << rno;
}
class result:public student
{
    private:
        int total;
        float  perc;
        char grade;
    public:
        void get_Result(void);
```

```cpp
        void put_Result(void);
};
void result::get_Result(void)
{
    cout << "\t-----------Enter student's result information:----------";
    cout << "\nTotal Marks from one subjects: ";
    cin >> total ;
    perc= total/1;
    cout << "\nGrade: ";
    cin >> grade ;
}
void result::put_Result(void)
{
    cout << "\nTotal Marks: " << total;
    cout << "\nPercentage: " << perc;
    cout << "\nGrade: " << grade;
}
int main()
{
    result r1;
    r1.get_details();
    r1.get_Result();
    r1.put_details();
    r1.put_Result();
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
        --------------Enter student's basic information:----------
Name: Sourav
Branch: CSE
Roll number: 2213047
        -----------Enter student's result information:----------
Total Marks from one subjects: 100

Grade: A

         ---------Student details------------
Name: Sourav
Branch: CSE
Roll number: 2213047
Total Marks: 100
Percentage: 100
Grade: A
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

Assignment No. 4

Title: Constructor

Problem statement: Constructor

Objective:

Student will be able to gain knowledge about various constructor

Theory:

Concept: Constructor

Write a CPP program that represents the complex number (for example: 2 + 3i)

Use

i. Default constructor
ii. Parameterized constructor
iii. Copy constructor

C++ Constructor

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

There can be two types of constructors in C++.

- Default constructor
- Parameterized constructor

C++ Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Let's see the simple example of C++ default Constructor.

```
#include <iostream>

using namespace std;

class Employee

 {

  public:

      Employee ()
```

```
    {
        cout<<"Default Constructor Invoked"<<endl;
    }
};
int main(void)
{
   Employee e1; //creating an object of Employee
   Employee e2;
   return 0;
}
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C++ Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Let's see the simple example of C++ Parameterized Constructor.

```
#include <iostream>
using namespace std;
class Employee {
  public:
     int id;//data member (also instance variable)
     string name;//data member (also instance variable)
     float salary;
     Employee (int i, string n, float s)
      {
         id = i;
```

```
        name = n;
        salary = s;
    }
    void display ()
    {
        cout<<id<<"  "<<name<<"  "<<salary<<endl;
    }
};
int main(void) {
   Employee e1 =Employee (101, "Sonal", 890000); //creating an object of Employee
   Employee e2=Employee (102, "Nakul", 59000);
   e1.display();
   e2. display ();
   return 0;
}
```

Output:

101 Sonal 890000

102 Nakul 59000

C++ Copy Constructor

A Copy constructor is an overloaded constructor used to declare and initialize an object from another object.

Copy Constructor is of two types:

Default Copy constructor: The compiler defines the default copy constructor. If the user defines no copy constructor, compiler supplies its constructor.

User Defined constructor: The programmer defines the user-defined constructor.

C++ Copy Constructor

Syntax Of User-defined Copy Constructor:

Class_name (const class_name &old_object);

Consider the following situation:

```
class A
{
    A (A &x) //  copy constructor.
   {
       // copy constructor.
   }
}
```

In the above case, copy constructor can be called in the following ways:

C++ Copy Constructor

Let's see a simple example of the copy constructor.

```
// program of the copy constructor.

#include <iostream>
using namespace std;
class A
{
   public:
    int x;
    A(int a)               // parameterized constructor.
    {
      x=a;
    }
    A (A &i)               // copy constructor
    {
        x = i.x;
```

```
    }
};
int main()
{
  A a1(20);              // Calling the parameterized constructor.
 A a2(a1);               //  Calling the copy constructor.
 cout<<a2.x;
  return 0;
}
```

Output:

20

When Copy Constructor is called

Copy Constructor is called in the following scenarios:

When we initialize the object with another existing object of the same class type. For example, Student s1 = s2, where Student is the class.

1. When the object of the same class type is passed by value as an argument.
2. When the function returns the object of the same class type by value.

Two types of copies are produced by the constructor:

- Shallow copy
- Deep copy

Shallow Copy

The default copy constructor can only produce the shallow copy.

A Shallow copy is defined as the process of creating the copy of an object by copying data of all the member variables as it is.

Let's understand this through a simple example:

```
#include <iostream>
 using namespace std;
class Demo
{
```

```
    int a;
    int b;
    int *p;
    public:
    Demo()
    {
        p=new int;
    }
    void setdata(int x,int y,int z)
    {
        a=x;
        b=y;
        *p=z;
    }
    void showdata()
    {
        std::cout << "value of a is : " <<a<< std::endl;
        std::cout << "value of b is : " <<b<< std::endl;
        std::cout << "value of *p is : " <<*p<< std::endl;
    }
};
int main()
{
  Demo d1;
  d1.setdata(4,5,7);
  Demo d2 = d1;
  d2.showdata();
    return 0;
```

```
}
```

Output:

```
value of a is : 4

value of b is : 5

value of *p is : 7
```

In the above case, a programmer has not defined any constructor, therefore, the statement Demo d2 = d1; calls the default constructor defined by the compiler. The default constructor creates the exact copy or shallow copy of the existing object. Thus, the pointer p of both the objects point to the same memory location. Therefore, when the memory of a field is freed, the memory of another field is also automatically freed as both the fields point to the same memory location. This problem is solved by the user-defined constructor that creates the Deep copy.

Deep copy

Deep copy dynamically allocates the memory for the copy and then copies the actual value, both the source and copy have distinct memory locations. In this way, both the source and copy are distinct and will not share the same memory location. Deep copy requires us to write the user-defined constructor.

Let's understand this through a simple example.

```
#include <iostream>

using namespace std;

class Demo

{

   public:

   int a;

   int b;

   int *p;


   Demo()

   {
```

```
        p=new int;
    }
    Demo(Demo &d)
    {
        a = d.a;
        b = d.b;
        p = new int;
        *p = *(d.p);
    }
    void setdata(int x,int y,int z)
    {
        a=x;
        b=y;
        *p=z;
    }
    void showdata()
    {
        std::cout << "value of a is : " <<a<< std::endl;
        std::cout << "value of b is : " <<b<< std::endl;
        std::cout << "value of *p is : " <<*p<< std::endl;
    }
};
int main()
{
  Demo d1;
  d1.setdata(4,5,7);
  Demo d2 = d1;
  d2.showdata();
```

```
  return 0;
}
```

Output:

value of a is : 4

value of b is : 5

value of *p is : 7

In the above case, a programmer has defined its own constructor, therefore the statement Demo d2 = d1; calls the copy constructor defined by the user. It creates the exact copy of the value types data and the object pointed by the pointer p. Deep copy does not create the copy of a reference type variable.

Algorithm:

1. Create a class complex and add real ang img variables as float in the public section.
2. Write a default constructor complex to initialize the value of real and img as 0+0i.
3. Write a constructor complex to initialize the value of real and img with some specific values using parameterize constructor..
4. Write a constructor complex to copy the value of object's real and img part using copy constructor..
5. Display the values of all three objects.

Class diagram and Flowchart:

Conclusion:

By above problem statement student can get learning about overloading of '+','*', '<<' and '>>' operators for complex number addition , multiplication, input and output of complex numbers.

Code with output:

```
// Write a CPP program that represents the complex number (for example:  2 + 3i)
#include<iostream>
using namespace std;
```

```cpp
class complex
{
 private:
    int real, imag;
 public:
    complex()
    {
     real=2;
     imag=3;
     cout<<real<<"+"<<imag<<"i"<<endl;
    }
    complex(int a, int b)
    {
     real=a;
     imag=b;
     cout<<real<<"+"<<imag<<"i"<<endl;
    }
    complex(complex &obj)
    {
     real=obj.real;
     imag=obj.imag;
     cout<<real<<"+"<<imag<<"i"<<endl;
    }
};
int main()
{
 complex c;
 complex c1(2,3);
 complex c2(c1);
 return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass4_2213047.cpp -o Ass4_2213047 } ; if ($?) { .\Ass4_2213047 }
2+3i
2+3i
2+3i
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

Assignment No. 5

Title: Inheritance and Exception Handling

Problem statement:

Concept: Inheritance & Exception handling

Imagine a publishing company which does marketing for book and audiocassette versions. It stores information such as the title and price of a publication. Reuse this publication data for book details that in addition includes a page count and tape which includes a playing time in minutes. Write a program that displays error message if data entered is zero.

Objective:

To understand the syntax and concept of Exception Handling in c++

Theory:

Inheritance:

Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes.

New classes inherit some of the properties and behaviour of the existing classes. An existing class that is "parent" of a new class is called a base class.

New class that inherits properties of the base class is called a derived class

. Inheritance is a technique of code reuse.

It also provides the possibility to extend existing classes by creating derived classes.

The basic syntax of inheritance is:

Class DerivedClass : accessSpecifier BaseClass

There are 3 access specifiers: Namely public, private and protected.

Types of Inheritance In C++,

We have 5 different types of Inheritance. Namely,

1. Single Inheritance

2. Multiple Inheritance

3. Hierarchical Inheritance

4. Multilevel Inheritance

5. Hybrid Inheritance

C++ Exception Handling

Exception Handling in C++ is a process to handle runtime errors. We perform exception handling so the normal flow of the application can be maintained even after runtime errors.

In C++, an exception is an event or object which is thrown at runtime. All exceptions are derived from the std::exception class. It is a runtime error which can be handled. If we don't handle the exception, it prints an exception message and terminates the program.

Exception handling is part of C++ and object-oriented programming. they are added in C++ to handle

the unwanted situations during program execution. If we do not type the program correctly then ot

might result in errors. Main purpose of exception handling is to identify and report the runtime error

in the program.

Famous examples are divide by zero, array index out of bound error, file not found, device not found,

etc.

C++ exception handling is possible with three keywords iz. try, catch and throw. Exception handling

performs the following tasks: -

•Find the problem in the given code. It is also called a hit exception.

• It informs me that an error has occurred. It is called throwing the exception.

• We receive the roe info. It is called catching the exception.

•It takes the corrective action.It is called as exception handling.

try:- It is block code in which there are chances of runtime error.This block is followed by one or

more catch block.Most error prone code is added in try block.

catch:- This is used to catch the exception thrown by the try blok. In catch block we take corrective

action on throwing exceptions. If files are opened , we can take corrective action like closing file

handles,closing database connections,saving unsaved work ,etc.

Advantage

It maintains the normal flow of the application. In such cases, the rest of the code is executed even after an exception.

C++ Exception Classes

In C++ standard exceptions are defined in <exception> class that we can use inside our programs. The arrangement of parent-child class hierarchy is shown below:

C++ Exception Handling Keywords

In C++, we use 3 keywords to perform exception handling:

1. try
2. catch, and
3. throw

C++ try/catch

In C++ programming, exception handling is performed using try/catch statement. The C++ try block is used to place the code that may occur exception. The catch block is used to handle the exception.

throw:- Program throws exception when problem occurs.It is possible with throw keyword.

SNYTAX:=

```
//normal program code
try{
throw exception
}
catch(argument)
{
...
...
}
//rest of the code
```

C++ example without try/catch

```
#include <iostream>

using namespace std;

float division(int x, int y) {

  return (x/y);

}

int main () {

  int i = 50;

  int j = 0;

  float k = 0;

    k = division(i, j);

    cout << k << endl;

  return 0;

}
```

Output:

Floating point exception (core dumped)

```
#include <iostream>

using namespace std;

float division(int x, int y) {

  if( y == 0 ) {

    throw "Attempted to divide by zero!";
```

```
  }
  return (x/y);
}
int main () {
  int i = 25;
  int j = 0;
  float k = 0;
  try {
    k = division(i, j);
    cout << k << endl;
  }catch (const char* e) {
    cerr << e << endl;
  }
  return 0;
}
```

Output:

Attempted to divide by zero!

```
// Exception Handling
#include <iostream>
using namespace std;
int main()
{
 int x = -1;
```

```
// Some code
cout<< "Before try \n";
try {
cout<< "Inside try \n";
if (x < 0)
{
throw x;
cout<< "After throw (Never executed) \n";
}
}
catch (int x ) {
cout<< "Exception Caught \n";
}
cout<< "After catch (Will be executed) \n";
return 0;
}
```

Output:

Before try

Inside try

Exception Caught

After catch (Will be executed)

Facilities:

Linux Operating Systems,G++

Input: A class publication that stores the title (a string) and price (type float) of

publications. Derives two classes Book and Tape.

Algorithm:

1.Create a base class publication.

2.title and price data members are declared as protected members.

3.write a default and parameterised constructor in a base class to initialize data members (i.e title and price)

4. write a derived class book which has a page count data member.

5.write a constructor of book class which has 3 parameters from which 2 are used to initialize data members of base class and one for its own.

6.write a display function to display a member of book (title and price inherited from base class publication and pagecount (its own))

7.Follow the same steps for CD class

Class diagram and Flowchart:

Conclusion: For a given application we handled the exception using try, catch and throw keywords.

Code with output:

```
#include<iostream>
using namespace std;
class Publication
{
 public:
  string title;
  float price;
 void getdetails()
 {
 cout<<"Enter the title:";
 cin>>title;
 cout<<"Enter the price:";
 cin>>price;
 }
 void viewdetails()
 {
 cout<<"The title is: "<<title<<endl;
 cout<<"The price is: "<<price<<endl;
 }
};
class book: public Publication
{
 public:
 int pgno;
 void details()
 {
 cout<<"Enter the number of pages:";
 cin>>pgno;
 }
void view()
 {
try{
 if(pgno<=0)
 {
  throw pgno;
 }
 else
 {
  cout<<"Valid\n";
 }
   }
catch(int e)
 {
 cout<<"Invalid page count:"<<e<<endl;
 }
 }
};
class tape: public Publication
{
 public:
 float time;
 void Intime()
 {
```

```
  cout<<"Enter the playing time:";
  cin>>time;
 }
 void outtime()
 {
   try
   {
     if (time<=0)
     {
       throw time;
     }
     else
     {
       cout<<"Valid\n";
     }
   }
   catch(float t)
   {
     cout<<"Invalid time count:"<<t<<endl;
   }

 }
};
int main()
{
 book b;
 tape t;
 b.getdetails();
 b.details();
 b.viewdetails();
 b.view();
 t.getdetails();
 t.Intime();
 t.viewdetails();
 t.outtime();
 return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass5_2213047.cpp -o Ass5_2213047 } ; if ($?) { .\Ass5_2213047 }
Enter the title:ABCD
Enter the price:1000
Enter the number of pages:1234
The title is: ABCD
The price is: 1000
Valid
Enter the title:XYZ
Enter the price:1234
Enter the playing time:12.5
The title is: XYZ
The price is: 1234
Valid
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

(Followed by student answer section)

Create a class to represent a bank account, include the following members:

1. Data members:

   i. Name of the depositor

   ii. Account number

2. Member functions

   i. Deposit an amount Catch an exception If Deposit an amount less than 100)

   ii. Withdraw an amount (Catch an exception If withdraw amount less than balance)

   iii. Display name and balance

(Followed by student answer section)

Code with output:

```
#include<iostream>
using namespace std;
class Bank
{
  private:
    string cus_name,bank_name,branch,acc_type,mobile_num,acc_num,mail_id,ifsc_code;
    float amt,balance=0;

  public:
    void deposit_amt(void);
    void withdraw_amt(void);
    void disp_details(void);
    void get_details(void);
};
void Bank :: get_details()
{
  cout<< "\n\n\t\t----BANK APPLICATION----";
  cout<<" \n\n\t\t------BANK DETAILS-------\n";
  cout << "Enter Bank name:" ;
  cin >> bank_name;
  cout << "Enter Branch:" ;
  cin >> branch;
  cout << "Enter Name depositor:" ;
  cin >> cus_name;
  cout << "Enter Account number:" ;
```

```
  cin >> acc_num;
  cout << "Enter IFSC code:" ;
  cin >> ifsc_code;
  cout << "Enter Account type:" ;
  cin >> acc_type;
  cout << "Enter Mobile number:" ;
  cin >> mobile_num;
  cout << "Enter Email id:" ;
  cin >> mail_id;
}
void Bank :: deposit_amt()
{
  cout << "\nEnter amount to be deposited:" ;
  cin >> amt ;
  try
  {
    if(amt<100)
    throw amt;
  }catch(float e)
  {
    cout<<"\nNot enough deposit";
  }
  balance+=amt;
}
void Bank :: withdraw_amt()
{
 cout << "\nEnter amount to be withdrawed:" ;
 cin >> amt ;
 try
 {
    if(amt>balance)
    throw amt;
 }catch(float e)
 {
    cout<<"\nNot enough balance";
 }
}
void Bank :: disp_details()
{
  cout<<" \n\n\t\t**BANK DETAILS**\n";
  cout << "\n BANK NAME:" << bank_name ;
  cout << "\n CUSTOMER NAME:" << cus_name ;
  cout<< "\n ACCOUNT NUMBER:" << acc_num ;
  cout<< "\n BALANCE:" << balance ;
  cout<< "\n IFSC CODE:" << ifsc_code ;
  cout<< "\n BRANCH:" << branch ;
  cout<< "\n ACCOUNT TYPE:" << acc_type ;
  cout<< "\n MOBILE NUMBER:" << mobile_num ;
  cout<< "\n EMAIL ID:" << mail_id << "\n";
}
int main()
{
  Bank b1;
  b1.get_details();
  b1.deposit_amt();
  b1.withdraw_amt();
  b1.disp_details();
```

```
  return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass5_2213047Prob.cpp -o Ass5_2213047Prob } ; if ($?) { .\Ass5_2213047Prob }


                ----BANK APPLICATION----

                ------BANK DETAILS-------
Enter Bank name:IDBI
Enter Branch:DelhiGate
Enter Name depositor:Sourav
Enter Account number:123244
Enter IFSC code:IkBL2345
Enter Account type:Saving
Enter Mobile number:234567
Enter Email id:sourav@idbi.com

Enter amount to be deposited:123

Enter amount to be withdrawed:12


                **BANK DETAILS**

 BANK NAME:IDBI
 CUSTOMER NAME:Sourav
 ACCOUNT NUMBER:123244
 BALANCE:123
 IFSC CODE:IkBL2345
 BRANCH:DelhiGate
 ACCOUNT TYPE:Saving
 MOBILE NUMBER:234567
 EMAIL ID:sourav@idbi.com
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

Assignment No. 6

Title: Function overloading (polymorphism)

Concept: Polymorphism (Function overloading)

Write a program to compute and display area of different shapes (circle, rectangle etc.)

Objective: To perform a single action in different ways.

Theory:

The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a Greek word. In object-oriented programming, we use 3 main concepts: inheritance, encapsulation, and polymorphism.

Real Life Example of Polymorphism

Let's consider a real-life example of polymorphism. A lady behaves like a teacher in a classroom, mother or daughter in a home and customer in a market. Here, a single person is behaving differently according to the situations.

There are two types of polymorphism in C++:

1. Compile time polymorphism: The overloaded functions are invoked by matching the type and number of arguments. This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time. It is achieved by function overloading and operator overloading which is also known as static binding or early binding. Now, let's consider the case where function name and prototype is same. You invoke the overloaded functions by matching the number and type of arguments. The information is present during compile-time. This means the C++ compiler will select the right function at compile time.

Compile-time polymorphism is achieved through function overloading and operator overloading.

- Run time polymorphism: Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

Function Overloading

Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++. In function overloading, the function is redefined by using either different types of arguments or a different number of arguments. It is only through these differences' compiler can differentiate between the functions.

The advantage of Function overloading is that it increases the readability of the program because you don't need to use different names for the same action.

Let's see the simple example of function overloading where we are changing number of arguments of add () method.

```
// program of function overloading when number of arguments vary.

#include <iostream>

using namespace std;

class Cal {

   public:

static int add (int a, int b)

{

       return a + b;

   }

 static int add (int a, int b, int c)

   {

       return a + b + c;

   }

};

int main(void) {

   cal c;                                  //    class object declaration.

   cout<<c.add (10, 20);

   cout<<c.add (12, 20, 23);

  return 0;

}
```

Let's see the simple example when the type of the arguments vary.

```
// Program of function overloading with different types of arguments.
#include<iostream>
using namespace std;
int mul (int, int);
float mul (float, int);

int mul (int a, int b)
{
   return a*b;
}
float mul (double x, int y)
{
   return x*y;
}
int main ()
{
   int r1 = mul (6,7);
   float r2 = mul (0.2,3);
   cout << "r1 is: " <<r1;
   cout <<"r2 is: “<<r2;
   return 0;
}
```

Function Overloading and Ambiguity

When the compiler is unable to decide which function is to be invoked among the overloaded function, this situation is known as function overloading.

When the compiler shows the ambiguity error, the compiler does not run the program.

Causes of Function Overloading:

1. Type Conversion.

2. Function with default arguments.
3. Function with pass by reference.
4. Type Conversion:

Let's see a simple example.

```
#include<iostream>
using namespace std;
void fun(int);
void fun(float);
void fun(int i)
{
   std::cout << "Value of i is : " <<i<< std::endl;
}
void fun(float j)
{
   std::cout << "Value of j is : " <<j<< std::endl;
}
int main()
{
   fun(12);
   fun(1.2);
   return 0;
}
```

The above example shows an error "call of overloaded 'fun(double)' is ambiguous". The fun(10) will call the first function. The fun(1.2) calls the second function according to our prediction. But, this does not refer to any function as in C++, all the floating point constants are treated as double not as a float. If we replace float to double, the program works. Therefore, this is a type conversion from float to double.

1. Function with Default Arguments

Let's see a simple example.

```
#include<iostream>
using namespace std;
void fun(int);
void fun(int,int);
void fun(int i)
{
   std::cout << "Value of i is : " <<i<< std::endl;
}
void fun(int a,int b=9)
{
   std::cout << "Value of a is : " <<a<< std::endl;
   std::cout << "Value of b is : " <<b<< std::endl;
}
int main()
{
   fun(12);

   return 0;
}
```

The above example shows an error "call of overloaded 'fun(int)' is ambiguous". The fun(int a, int b=9) can be called in two ways: first is by calling the function with one argument, i.e., fun(12) and another way is calling the function with two arguments, i.e., fun(4,5). The fun(int i) function is invoked with one argument. Therefore, the compiler could not be able to select among fun(int i) and fun(int a,int b=9).

1. Function with pass by reference

Let's see a simple example.

```
#include <iostream>
using namespace std;
void fun(int);
```

```
void fun(int &);
int main()
{
int a=10;
fun(a); // error, which f()?
return 0;
}
void fun(int x)
{
std::cout << "Value of x is : " <<x<< std::endl;
}
void fun(int &b)
{
std::cout << "Value of b is : " <<b<< std::endl;
}
```

The above example shows an error "call of overloaded 'fun(int&)' is ambiguous". The first function takes one integer argument and the second function takes a reference parameter as an argument. In this case, the compiler does not know which function is needed by the user as there is no syntactical difference between the fun(int) and fun(int &).

Algorithm:

9. Start
10. Declare two user defined functions with same name area but having different arguments
11. Declare 3 input variables s, l, b;
12. Take input for s (side of square), l (length of rectangle), b (breadth of rectangle)
13. Call the first area function by passing s value.
14. Function calculate the area of square (s*s) returns the output
15. Call the second area function by passing l and b values.
16. Function calculate the area of rectangle(l*b) returns the output
17. Display the output for area of square and area of rectangle.
18. Stop

Flowchart:

Conclusion: Understood the process of using a function or an operator for more than one purpose using polymorphism means the same entity (function or operator) behaves differently in different scenarios.

Code with output:

```
#include <iostream>
using namespace std;
#define PI 3.142
class Shapes
{
    public:
    float r,l,b,a,rad;
    public:
    void area(float i1);
    void area(float i3,float i2);
};
void Shapes :: area(float i1)
{
    r=i1;
    a=3.*r*r;
    cout<<"Area of Circle is: "<<a<<endl;
}
void Shapes::area(float i3,float i2)
{
    l=i3;
    b=i2;
    a=l*b;
    cout<<"Area Of Rectangle is: "<<a<<endl;
}
int main()
{
    cout<<"AREA OF DOFFERENT SHAPES"<<endl;
    cout<<"Polymorphism"<<endl;
    Shapes s1;
    cout<<"AREA OF CIRCLE: "<<endl;
    float rad;
    cout<<"Please Enter Radius Of Circle: "<<endl;
    cin>>rad;
    s1.area(rad);
    cout<<"AREA OF RECTANGLE: "<<endl;
    float len,brd;
    cout<<"Please Enter Length And Breadth Of The Rectangle: "<<endl;
    cin>>len>>brd;
    s1.area(len,brd);
    return 0;
}
```

Write a C++ program to implement function overloading having calculate_percentage () function to display percentage of 10th and 12th.

(Followed by student answer section)

CODE:

```
#include <iostream>
using namespace std;
class Maths
{
    public:
    double perc;
    float per;
    public:
    void calculate_percentage(double p1);
    void calculate_percentage(float p2);
};
void Maths :: calculate_percentage(double p1)
{
    cout<<"\nPercentage(10th): "<<p1;
}
void Maths:: calculate_percentage(float p2)
{
    cout<<"\nPercentage(12th): "<<p2;
}
int main()
{
    Maths m;
    cout<<"\nPercentage scored in 10th:";
    cin>>m.perc;
    cout<<"\nPercentage scored in 12th:";
    cin>>m.per;
    m.calculate_percentage(m.perc);
    m.calculate_percentage(m.per);
    return 0;
}
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Percentage scored in 10th:88.2
Percentage scored in 12th:70.92
Percentage(10th): 88.2
Percentage(12th): 70.92
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>

Assignment No. 7

Title: Operator overloading (polymorphism)

Problem statement: Concept: Polymorphism (Operator overloading)

Develop a complex number application. It must support following operations: 1. Addition of two complex numbers. 2. Multiplication of two complex numbers. 3. Read and write of complex Numbers.

Objective:

Theory:

C++ Constructor

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

There can be two types of constructors in C++.

1. Default constructor
2. Parameterized constructor

•

C++ Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object. Let's see the simple example of C++ default Constructor.

```
#include <iostream>

using namespace std;

class Employee

 {

   public:

       Employee()

       {

          cout<<"Default Constructor Invoked"<<endl;
```

```
        }
};
int main(void)
{
   Employee e1; //creating an object of Employee
   Employee e2;
   return 0;
}
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C++ Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Let's see the simple example of C++ Parameterized Constructor.

```
#include <iostream>
using namespace std;
class Employee {
  public:
      int id;//data member (also instance variable)
      string name;//data member (also instance variable)
      float salary;
      Employee (int i, string n, float s)
       {
           id = i;
           name = n;
           salary = s;
```

```
    }
    void display()
    {
        cout<<id<<"  "<<name<<"  "<<salary<<endl;
    }
};
int main(void) {
    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
    Employee e2=Employee(102, "Nakul", 59000);
    e1.display();
    e2.display();
    return 0;
}
```

Output:

```
101  Sonoo  890000
102 Nakul 59000
```

C++ Operators Overloading

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++. It is used to perform the operation on the user-defined data type. For example, C++ provides the ability to add the variables of the user-defined data type that is applied to the built-in data types.

The advantage of Operators overloading is to perform different operations on the same operand.

Operator that cannot be overloaded are as follows:

1. Scope operator (: :)
2. sizeof
3. member selector(.)
4. member pointer selector (*)
5. ternary operator (? :)

Syntax of Operator Overloading

return_type class_name: operator op(argument_list)

{

    // body of the function.

}

Where the return type is the type of value returned by the function.

class_name is the name of the class.

operator op is an operator function where op is the operator being overloaded, and the operator is the keyword.

Rules for Operator Overloading

1. Existing operators can only be overloaded, but the new operators cannot be overloaded.
2. The overloaded operator contains at least one operand of the user-defined data type.
3. We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
4. When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
5. When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

C++ Operators Overloading Example

Let's see the simple example of operator overloading in C++. In this example, void operator ++ () operator function is defined (inside Test class).

```
// program to overload the unary operator ++.

#include <iostream>

using namespace std;

class Test

{
```

```
  private:
     int num;
  public:
      Test(): num(8){}
      void operator ++()        {
         num = num+2;
      }
      void Print() {
          cout<<"The Count is: "<<num;
      }
};
int main()
{
    Test tt;
    ++tt;  // calling of a function "void operator ++()"
    tt.Print();
    return 0;
}
```

Output:

```
The Count is: 10
```

Let's see a simple example of overloading the binary operators.

```
// program to overload the binary operators.
#include <iostream>
using namespace std;
class A
{
       int x;
```

```
    public:
    A(){}
   A(int i)
   {
     x=i;
   }
   void operator+(A);
   void display();
};

void A :: operator+(A a)
{

   int m = x+a.x;
   cout<<"The result of the addition of two objects is : "<<m;
 }
int main()
{
   A a1(5);
   A a2(4);
   a1+a2;
   return 0;
}
```

Output:

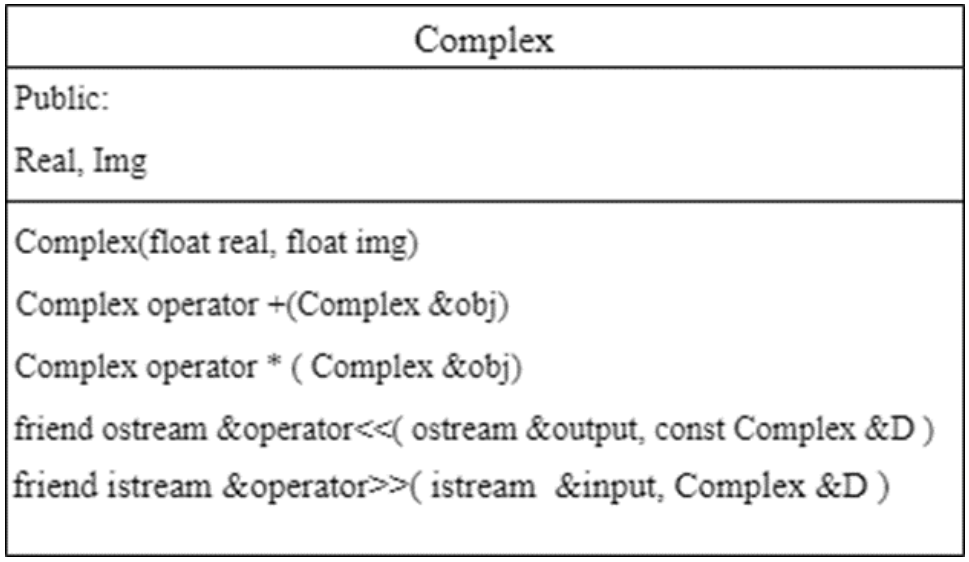
The result of the addition of two objects is: 9

Algorithm:

19. Create a class complex and add real ang img variables as float in the public section.
20. Write a default constructor complex to initialize the value of real and img.

21. write a definition for operator overloading of '+' operator using a single argument.
22. write a definition for operator overloading of '*' operator using single argument.
23. write a definition for operator overloading of '<<' operator using two arguments one is ostream and other complex.
24. write a definition for operator overloading of '>>' operator using two arguments one is istream and other is complex.
25. Declare three objects a, b,c in which a and b are initialized using parameterized constructor and object c is initialized with default constructor.
26. Perform addition of two complex numbers directly using '+' operator.
27. Perform Multiplication of two complex numbers directly using '*' operator.
28. Print the value of addition and multiplication using '<<' operator.

Class diagram



Conclusion:

By above problem statement student can get learning about overloading of '+','*', '<<' and '>>' operators for complex number addition , multiplication, input and output of complex numbers.

Code with output:

```
#include <iostream>
using namespace std;
class Complex
{
private:
float real,img;
public:
void get (void);
void operator+(Complex);
void operator-(Complex);
void operator*(Complex);
void operator/(Complex);
};
void Complex :: get()
{
cout<<"\n Please enter a real number:";
cin>>real;
cout<<"\n Please enter a imaginary number:";
cin>>img;
cout<<"\n"<<real<<"+"<<img<<"i";
}
void Complex :: operator+(Complex c)
{
real = real + c.real;
img = img + c.img;
cout<<"\n Complex number ="<<real<<"+"<<img<<"i";
}
void Complex :: operator-(Complex c)
{
real = real - c.real;
img = img - c.img;
cout<<"\n Complex number ="<<real<<"+"<<img<<"i";
}
void Complex :: operator*(Complex c)
{
real = real * c.real;
img = img * c.img;
cout<<"\n Complex number ="<<real<<"+"<<img<<"i";
}
void Complex :: operator/(Complex c)
{
real = real / c.real;
img = img / c.img;
cout<<"\n Complex number ="<<real<<"+"<<img<<"i";
}
int main()
{
Complex c1,c2,c3;
c1.get();
c2.get();
c1+c2;
c1-c2;
c1*c2;
c1/c2;
return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass7_2213047.cpp -o Ass7_2213047 } ; if ($?) { .\Ass7_2213047 }

 Please enter a real number:1

 Please enter a imaginary number:1

1+1i
 Please enter a real number:1

 Please enter a imaginary number:1

1+1i
 Complex number =2+2i
 Complex number =1+1i
 Complex number =1+1i
 Complex number =1+1i
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

(Followed by student answer section)

CODE:

```cpp
#include <iostream>
using namespace std;
class Maths
{
    private:
    int x=0;
    public:
    void get(void);
    void operator++(void);
};
void Maths:: get()
{
    cout<<"Enter number";
    cin>>x;
}
void Maths :: operator++()
{
    x=x+5;
    cout<<x;
}
int main()
{
    Maths m1;
     m1.get();
    ++m1;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass7_2213047Prob.cpp -o Ass7_2213047Prob } ; if ($?) { .\Ass7_2213047Prob }
Enter number23
28
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

Assignment No. 8

Title: Class template

Problem statement: Create a class template to represent a generic vector. Include following member functions: · To create the vector. · To modify the value of a given element · To multiply by a scalar value

Objective: To explore the principles of Object Oriented Programming (OOP) : templates (Generic classes and functions)

Theory: Templates are powerful features of C++ which allows us to write generic programs.

There are two ways we can implement templates:

- Function Templates
- Class Templates

Function template

•A function template defines a family of functions.

•Function template plays a significant role, but it is neither a type by itself nor a function alone.

Syntax

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

Example:

// If two characters are passed to function template, character with larger ASCII value is displayed.

```
#include <iostream>
 using namespace std;
// template function
```

```
template <class T>
T Large (T n1, T n2)
 {
 return (n1 > n2)? n1: n2;
 }
int main ()
 {
 int i1, i2;
 float f1, f2;
 char c1, c2;
 cout << "Enter two integers:\n";
 cin >> i1 >> i2;
 cout << Large (i1, i2) <<" is larger." << endl;
 cout << "\n Enter two floating-point numbers:\n";
 cin >> f1 >> f2;
 cout << Large (f1, f2) <<" is larger." << endl;
 cout << "\n Enter two characters:\n";
 cin >> c1 >> c2;
 cout << Large (c1, c2) << " has larger ASCII value.";
 return 0;
 }
```

Similar to function templates, we can use class templates to create a single class to work with different data types.

Class templates come in handy as they can make our code shorter and more manageable.

class Template Declaration

A class template starts with the keyword template followed by template parameter(s) inside <> which is followed by the class declaration.

```
template <class T>
class class_name
{
  private:
    T var;
    ...
.. ...
  public:
    T function_name(T arg);
    ...
.. ...
};
```

In the above declaration, T is the template argument which is a placeholder for the data type used, and class is a keyword.

Inside the class body, a member variable var and a member function function_name() are both of type T.

Algorithm:

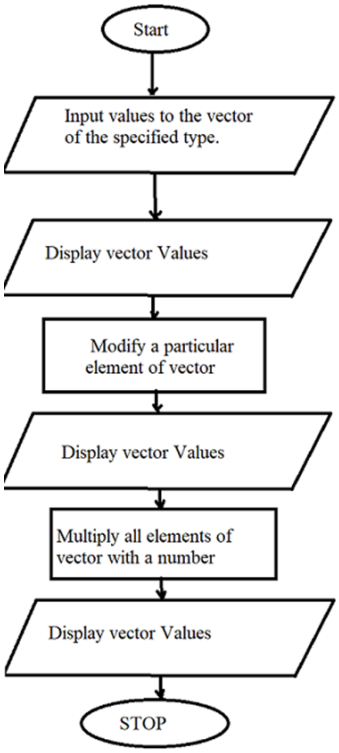
1. create template class for vector
2. create 3 methods
    a. for creation of vector,
    b. for modifying an element in the vector
    c. for scalar multiplication
3. using the above template class call methods with various data type values

Class diagram and Flowchart:

+ Public

Vector
+ value: T (any type) + index: int

Flowchart:

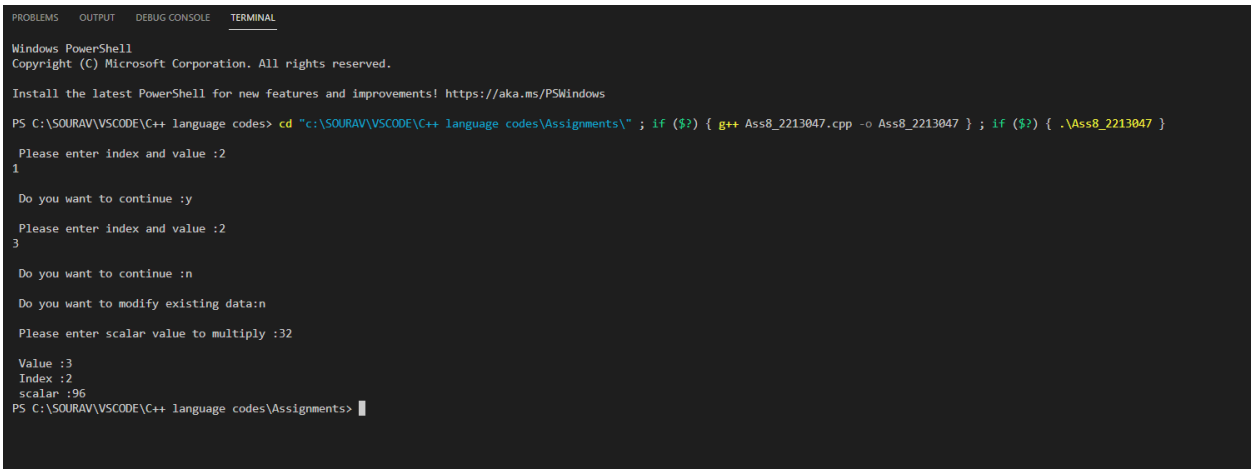


Conclusion: A C++ program with template vector class implemented. Concept of Template in C++ is learnt and implemented too.

Code with output:

```
#include <iostream>
using namespace std;
template <class T>
class Vector
{
```

```
    T v[20],value,scalar,nvalue;
    int i,index;
    char ch;
    public:
    void create (void);
    void modify (void);
    void multiply (void);
    void display (void);
};
template <class T>
void Vector<T> :: create()
{
    do
    {
        cout<<"\n Please enter index and value :";
        cin>>index>>value;
        v[index] = value;
        cout<<"\n Do you want to continue :";
        cin>>ch;
    }while(ch=='y'||ch=='Y');
}
template <class T>
void Vector<T> :: modify()
{
    cout<<"\n Do you want to modify existing data:";
    cin>>ch;
    if(ch=='y'||ch=='Y')
    {
        cout<<"\n Please enter new index and value :";
        cin>>index>>value;
        v[index]=value;
    }
}
template <class T>
void Vector<T> :: multiply()
{
    cout<<"\n Please enter scalar value to multiply :";
    cin>>scalar;
    nvalue = scalar*value;
    v[index] = nvalue;
}
template <class T>
void Vector<T> :: display()
{
    cout<<"\n Value :"<<value;
    cout<<"\n Index :"<<index;
    cout<<"\n scalar :"<<nvalue;
}
int main()
{
    Vector <int> v1;
    v1.create();
    v1.modify();
    v1.multiply();
    v1.display();
    return 0;
}
```

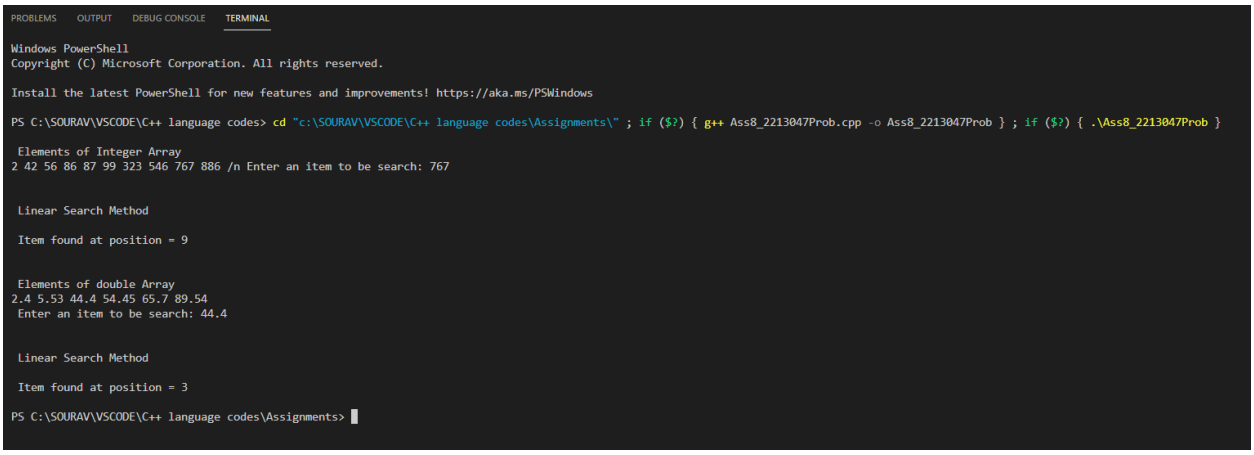


Problem statement: Write C++ program with a function template to perform linear search in an array.

Code with output:

```
#include<iostream>
using namespace std;
template <class T>
void Lsearch(T *a, T item, int n)
{
int z=0;
for(int i=0;i<n;i++)
{
if(a[i]== item)
{
cout<<"\n Item found at position = "<<i+1<<"\n\n";
z=1;
return;
}
}
cout<<"\nNot Found\n\n";
}
int main()
{
int arrayInt[10] = {2,42,56,86,87,99,323,546,767,886};
double arrayDouble[6]= {2.4, 5.53,44.4, 54.45, 65.7,89.54};
int itemI;
double itemD;
cout<<"\n Elements of Integer Array \n";
for(int i=0;i<10;i++)
{
cout<<arrayInt[i]<<" ";
}
cout<<"/n Enter an item to be search: ";
cin>>itemI;
cout<<"\n\n Linear Search Method\n";
Lsearch(arrayInt,itemI,10);
cout<<"\n Elements of double Array \n";
for(int i=0;i<6;i++)
```

```
{
cout<<arrayDouble[i]<<" ";
}
cout<<"\n Enter an item to be search: ";
cin>>itemD;
cout<<"\n\n Linear Search Method\n";
Lsearch(arrayDouble,itemD,6);
return 0;
}
```



Assignment No. 9

Title: Different OOP languages

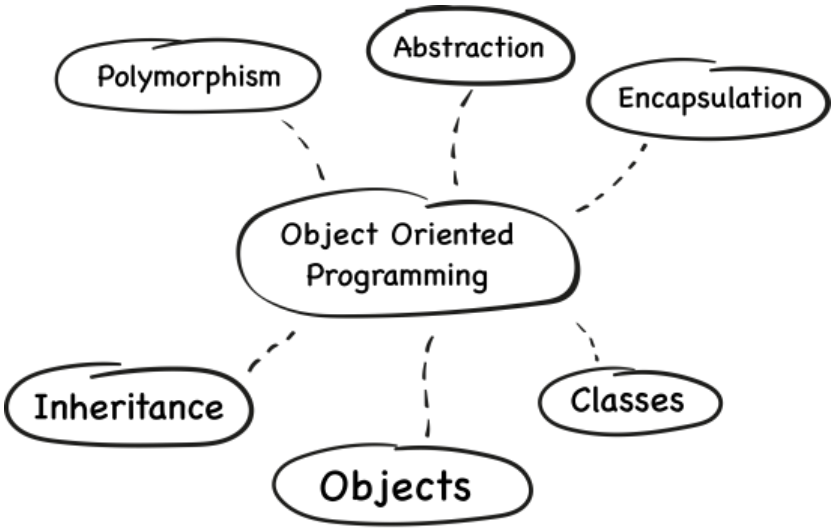
Problem statement: Write a simple OOP program to perform arithmetic operation (simple calculator) in multiple OOP languages to differentiate syntax. (CPP, python and java.)

Objective:

1. To study the syntax of different programming languages as JAVA, C++, Python
2. Compare and understand the way of writing programs in different languages.

Theory:

There are many different types of object-oriented programming languages that all use slightly different syntaxes but share the same general concepts.



C++ being a fast and compiled programming language has gained popularity and is the first programming language that a programmer learns.

Java is popular due to its platform independence and huge number of desktop applications is developed using Java.

Python is interpreted programming language; it is a modern programming language and it fast types language.

Each one of the triad, C++, Java, and Python, has their own advantages and issues that they call forth with their use. C++ is preferred for its speed and memory management, while Java's platform

independency makes it an opportune option for cross-platform development. Python, on the other hand, is more like a human language with high readability, less complex syntax, and an active community support.

C++ Highlights

- A wide range of applications, varying from simple GUI applications to vivid 3D games and real-time mathematical simulations
- Efficient, fast, and powerful
- Highly portable, a top choice for multi-device, multi-platform development
- Object-oriented programming language leveraging classes, data abstraction and encapsulation, inheritance, and polymorphism
- Rich function library
- Supports exception handling and function overloading

Java Highlights

- Designed for distributed computing
- Easy to compile, debug, learn, and write
- High degree of robustness
- Multithreaded, capable of performing different tasks simultaneously within a program
- Object-oriented, allows the creation of modular programs and reusable code
- Platform-independent, easy to migrate from one system to another
- Supports automatic memory allocation and garbage collection

Python Highlights

- Active community support and rapid development
- Batteries are included language
- Easy-to-learn, clear syntax
- Extensible to a greater degree
- Free, open source, cross-platform
- High-level language with high readability and reliability
- Object-oriented nature
- Python programs are easy to modify for supporting different database engines

Algorithm:

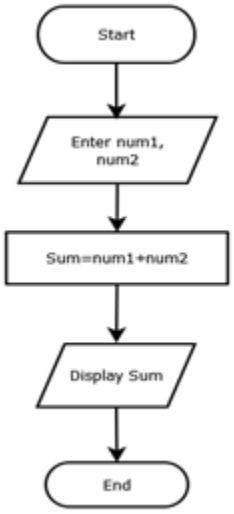
Step1: Take two numeric values from the user

Step2: Verify both values are numeric

Step3: Add both values and store result into a new variable

Step4: Display the result

Flowchart:



[Text Wrapping Break]

Conclusion:

At the end, students can able to know about the OOPs concepts are same, but we can explore it by using different programming language (e.g., syntax) according to the advantages /disadvantages of particular programming language as per programmer's requirement.

Code with output:

```
# include <iostream>
using namespace std;
int main() {
  char op;
  float num1, num2;
  cout << "Enter operator: +, -, *, /: ";
  cin >> op;
  cout << "Enter two operands: ";
```

```
  cin >> num1 >> num2;
  switch(op) {
    case '+':
      cout << num1 << " + " << num2 << " = " << num1 + num2;
      break;
    case '-':
      cout << num1 << " - " << num2 << " = " << num1 - num2;
      break;
    case '*':
      cout << num1 << " * " << num2 << " = " << num1 * num2;
      break;
    case '/':
      cout << num1 << " / " << num2 << " = " << num1 / num2;
      break;
    default:
      cout << "Error! operator is not correct";
      break;
  }
  return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> cd "c:\SOURAV\VSCODE\C++ language codes\Assignments\" ; if ($?) { g++ Ass9_2213047.cpp -o Ass9_2213047 } ; if ($?) { .\Ass9_2213047 }
Enter operator: +, -, *, /: +
Enter two operands: 2
3
2 + 3 = 5
PS C:\SOURAV\VSCODE\C++ language codes\Assignments>
```

CODE 2(python):

```
def add(x, y):
    return x + y
def subtract(x, y):
    return x - y
def multiply(x, y):
    return x * y
def divide(x, y):
    return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
    choice = input("Enter choice(1/2/3/4): ")
    if choice in ('1', '2', '3', '4'):
        num1 = float(input("Enter first number: "))
```

```
        num2 = float(input("Enter second number: "))

        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))

        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))

        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))

        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
          break

    else:
        print("Invalid Input")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\SOURAV\VSCODE\C++ language codes> python -u "c:\SOURAV\VSCODE\C++ language codes\Assignments\Ass9_2213047.py"
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 1
Enter first number: 2
Enter second number: 3
2.0 + 3.0 = 5.0
Let's do next calculation? (yes/no): yes
Enter choice(1/2/3/4): 2
Enter first number: 2
Enter second number: 1
2.0 - 1.0 = 1.0
Let's do next calculation? (yes/no): yes
Enter choice(1/2/3/4): 3
Enter first number: 2
Enter second number: 3
2.0 * 3.0 = 6.0
Let's do next calculation? (yes/no):
Enter choice(1/2/3/4): 4
Enter first number: 2
Enter second number: 2
2.0 / 2.0 = 1.0
Let's do next calculation? (yes/no): no
PS C:\SOURAV\VSCODE\C++ language codes>
```