



**G. H. Raisoni College of Engineering
and Management, Wagholi, Pune.**

Department of Data Science

Lab Manual (2021-22)

Class: SY DS Term: IV

Sub: Object Oriented Programming

Faculty Name: Bhagyashri Wankar

**G. H. Rasoni College of Engineering and Management, Wagholi,
Pune.**

Department: Data Science

Course Details

Course: Object Oriented Programming Lab (UITP201)

Class: SY BTECH

Internal Marks: 25

Credits: 1

Division: DS

External marks:-25

COURSE OUTCOME	
CO1	Understand the basic principles of object oriented programming
CO2	Apply the concepts of overloading, inheritance, polymorphism
CO3	Appraise memory allocation techniques and usage of exception handling, generic programming
CO4	Develop programs using object oriented concepts

List of Experiments

Sr. No.	Experiment List	CO Mapping	Software Required
1	Write a program to compute the area of triangle and circle by overloading the area() function.	CO1, CO4	DevC++ / IntelliJ Idea / Online Compiler
2	Define a class to represent a bank account. Include the following members : Data members:- Name of depositor, Account number, Type of account, Balance amount in the account Member functions:- To assign initial values, To deposit an amount, To withdraw an amount after checking the balance, To display name & balance Write a main program to test program using class and object.	CO1, CO4	DevC++ / IntelliJ Idea / Online Compiler
3	Create two classes DM and DB which stores values of distances. DM stores distances in meters and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB. Use a friend function to carry out addition operation	CO2, CO4	DevC++ / IntelliJ Idea / Online Compiler
4	Create a class MAT of size m * n. Define all possible matrix operations for MAT type objects	CO2, CO4	DevC++ / IntelliJ Idea / Online Compiler

5	Create Stud class to display student information using constructor and destructor. (Default constructor, Multiple constructor, Copy constructor, Overloaded constructor)	CO2, CO4	DevC++ / IntelliJ Idea / Online Compiler
6	Consider class network of given figure. The class master derives information from both account and admin classes which in turn derive information from the class person. Define all the four classes and write a program to create, update and display the information contained in master objects.	CO2, CO4	DevC++ / IntelliJ Idea / Online Compiler
7	A book shop sells both books and video tapes. Create a class media that stores the title and price of the publication. Create two derived classes, one for storing number of pages in the book and another for storing playing time of tape. A function display() must be defined in all classes to display class contents. Write a program using polymorphism and virtual function.	CO2, CO4	DevC++ / IntelliJ Idea / Online Compiler
8	Write a program to show use of this pointer, new and delete.	CO3, CO4	DevC++ / IntelliJ Idea / Online Compiler
9	Write a function template for finding the minimum value contained in an array	CO3, CO4	DevC++ / IntelliJ Idea / Online Compiler
10	Write a program containing a possible exception. Use a try block to throw it and catch block to handle it properly. Open Ended Experiments / New Experiments	CO3, CO4	DevC++ / IntelliJ Idea / Online Compiler
11	Write a class template to represent a generic vector. Include member functions to perform following tasks -To create a vector -To modify the value of given element -To multiply by scalar value. -To display vector.	CO3, CO4	DevC++ / IntelliJ Idea / Online Compiler
12	Write a C++ program to design a simple calculator	CO3, CO4	DevC++ / IntelliJ Idea / Online Compiler
Content Beyond Syllabus			
1	Demonstrate the steps to install Java on Windows and Write a simple java program to print "HelloWorld"	CO4	JDK,JRE/Eclipse
2	Write a simple java program to perform arithmetic operation on two numbers.	CO4	JDK,JRE/Eclipse

AssignmentNo.1

Title: Write a program to compute the area of triangle and circle by overloading the area() function.

Problem Statement: Implement a C++ program to understand concept of Overloading.

Objective:

1. Understand the basic principles of object oriented programming
2. Implement the concepts of overloading.
3. Develop programs using object oriented concepts.

Theory:

C++Structures

Structureisacollectionofvariables ofdifferentdatatypes underasinglename.
Itissimilartoaclassin that, both holds a collection ofdataof different datatypes.

For example: You want to store some information about a person: his/her name, citizenshipnumber and salary. You can easily create different variables *name*, *citNo*, *salary* to store theseinformationseparately.

However,inthefuture,youwouldwanttostoreinformationaboutmultiplepersons.Now, you'dneed to create different variables for each information per person: *name1*, *citNo1*, *salary1*, *name2*, *citNo2*, *salary2*

You can easilyvisualizehowbigand messythe codewould look. Also,sinceno relationbetweenthe variables (information) wouldexist, it'sgoingtobe adauntingtask.

A better approach will be to have a collection of all related information under a single namePerson,anduseitfor everyperson.Now, thecode looks muchcleaner,readable andefficientaswell.

Thiscollectionofallrelated informationunderasinglenamePersonisa structure.

How to declare a structure in C++ programming?

The `struct` keyword defines a structure type followed by an identifier (name of the structure).

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
    char
    name[50];
    int age;
    float
    salary;
};
```

Here a structure *person* is defined which has three members: *name*, *age* and *salary*.

Sr.No.	Structure	Class
1	A structure is a collection of variables of different data types under a single unit. It is almost similar to a class because both are user-defined data types and both hold a bunch of different data types.	A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods (member function which defines actions) into a single unit.
2	If access specifier is not declared, by default all members are 'public'.	If access specifier is not declared, By default all members are 'private'.
3	Declaration of Structure: struct structure_name { type struct_element1; type struct_element2; type struct_element3; };	Declaration of Class: class class_name { data member; member function; };
4	Instance of 'structure' is called 'structure Variable'.	Instance of a 'class' is called 'object'.
5	Struct has limited features.	Class has limitless features.
6	Struct are used in small programs.	Class is generally used in large programs.

7	Structure does not contain parameter less constructor or destructor, but can contain Parameterized constructor or static constructor.	Classes can contain constructor or destructor.
8	A Struct is not allowed to inherit from another Struct or class.	A Class can inherit from another class.
9	Each variable in struct contains its own copy of data(except in ref and out parameter variable)and any operation on one variable cannot effect another variable.	Two variable of class can containthe reference of the same object and any operation on one variable can affect another variable.

C++ enables two or more functions with the same name but with different types of arguments or different sequence of arguments or different number of arguments. It is also called as “Function Polymorphism”

Program:

Assignment 1 Program

```
#include<iostream>
```

```
Using namespace std;
```

```
Void area(int h, int b)
```

```
{
    Float area;
    Area=(h*b)/2;
    Cout<<"Area of Triangle ="<<area;
}
```

```
Void area(int r)
```

```
{
    Float pi=3.141592653,area;
    Area=pi*(r*r);
    Cout<<"Area of Circle ="<<area;
}
```

```
Int main()
{
```

```
    Float r,h,b,choose;
```

```
    Cout<<"Choose 1 for Area of Triangle\nChoose 2 for Area of circle\n";
```

```
    Cin>>choose;
```

```
    If (choose==1)
```

```
{
        Cout<<"Enter the Height & base\n";
        Cin>>h>>b;
        Area(h,b);
    }
```

```
    Else if (choose==2)
```

```
{
        Cout<<"Enter the Radius\n";
        Cin>>r;
    }
```

```
Area;  
}  
Else  
{  
    Cout<<"Invalid Option\n";  
}  
  
}
```

Output:-

```
Choose 1 for Area of Triangle  
Choose 2 for Area of circle  
1  
Enter the Height & base  
20  
3  
Area of Triangle =30
```

AssignmentNo.2

Title: Define a class to represent a bank account. Include the following members:

Data members: Name of depositor, Account number, Type of account, Balance amount in the account
Member functions: To assign initial values, to deposit an amount, to withdraw an amount after checking the balance, to display name & balance. Write a main program to test program using class and object.

Problem Statement: Implement C++ programs which include a Class having data members and member function and Object to access these members of Class.

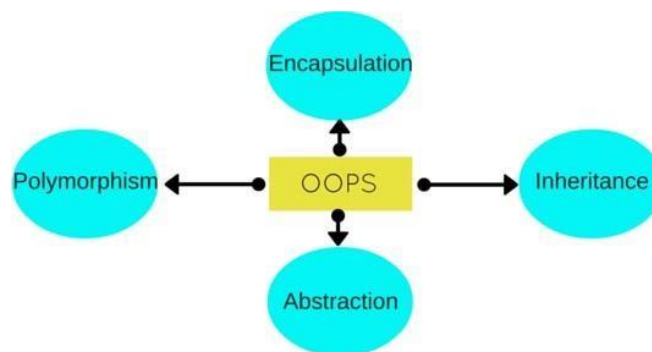
Objective:

1. Understand the basic principles of object oriented programming
2. Develop programs using object oriented concepts

Theory:

ObjectOrientedProgramming

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.



Class

Here we can take **HumanBeing** as a class. A class is a blueprint for any function entity which defines its properties and its functions. Like Human Being, having body parts, and performing various actions.

Inheritance

Considering HumanBeing as a class, which has properties like hands, legs, eyes etc, and functions like walk, talk, eat, see etc. Male and Female are also classes, but most of the properties and functions are included in HumanBeing, Hence, they can inherit everything from class HumanBeing using the concept of **Inheritance**.

Objects

My name is Abhishek, and I am an **instance/object** of class Male. When we say, Human Being, Male or Female, we just mean a kind, you, your friend, me we are the forms of these classes. We have a physical existence while a class is just a logical definition. We are the objects.

Abstraction

Abstraction means, showcasing only the required things to the outside world while hiding the details. Continuing our example, **Human Being's** can talk, walk, hear, and eat, but the details are hidden from the outside world. We can take our skin as the Abstraction factor in our case, hiding the inside mechanism.

Encapsulation

This concept is a little tricky to explain with our example. Our Legs are binded to help us walk. Our hands, help us hold things. This binding of the properties to functions is called Encapsulation.

Polymorphism

Polymorphism is a concept, which allows us to redefine the way something works, by either changing how it is done or by changing the parts using which it is done. Both the ways have different terms for them.

If we walk using our hands, and not legs, here we will change the parts used to perform something. Hence this is called **Overloading**.

And if there is a defined way of walking, but I wish to walk differently, but using my legs, like everyone else. Then I can walk like I want, this will be called as **Overriding**.

OOPs Concept Definitions

Now, let us discuss some of the main features of Object Oriented Programming which you will be using in C++ (technically).

1. Objects
2. Classes
3. Abstraction
4. Encapsulation
5. Inheritance
6. Overloading
7. ExceptionHandling

Objects

Objects are the basic unit of OOP. They are instances of class, which have data members and use various member functions to perform tasks.

Class

It is similar to structures in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declare & defines what data variables the object will have and what operations can be performed on the class's object.

Abstraction

Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes can provide methods to the outside world to access & use the data variables, keeping the variables hidden from direct access, or classes can even declare everything accessible to everyone, or maybe just to the classes inheriting it. This can be done using access specifiers.

Encapsulation

It can also be said databinding. Encapsulation is all about binding the data variables and functions together in class.

Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called the **Base** class & the class which inherits called the **Derived** class. They are also called parent and child class.

So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

Polymorphism

It is a feature, which lets us create functions with same name but different

arguments, which will perform different actions. That means, functions with same name, but functioning in different ways. Or, it also allows us to redefine a function to provide it with a completely new definition. You will learn how to do this in details soon in coming lessons.

Program:

```
#include<iostream.h>
Using namespace std;
Class Bank
{
Public:
Char name[20];
Char account_type[20];
Int account_number;
Int balance;
Void initialize()
{
Cout<<"\nEnter Account Holders Name:";
Gets(name);
Cout<<"\nEnter Account type:";
Gets(account_type);
Cout<<"\nEnter account number:";
Cin>>account_number;
Cout<<"\nEnter balance to deposit:";
Cin>>balance;
}

Void deposit()
{
Int bal;
Cout<<"\nEnter the amout to deposit:";
Cin>>bal;
Balance+=bal;
Cout<<"\nAmount deposited successfully\nYour New Balance:"<<balance;
}

Void check()
{
Int bal;
Cout<<"\nYour balance :"<<balance<<"\nEnter amount to withdraw:";
Cin>>bal;
If(bal<=balance)
{
Balance-=bal;
Cout<<"\nRemaining Balance:"<<balance;
}
Else
{
Exit(0);
}
}
```

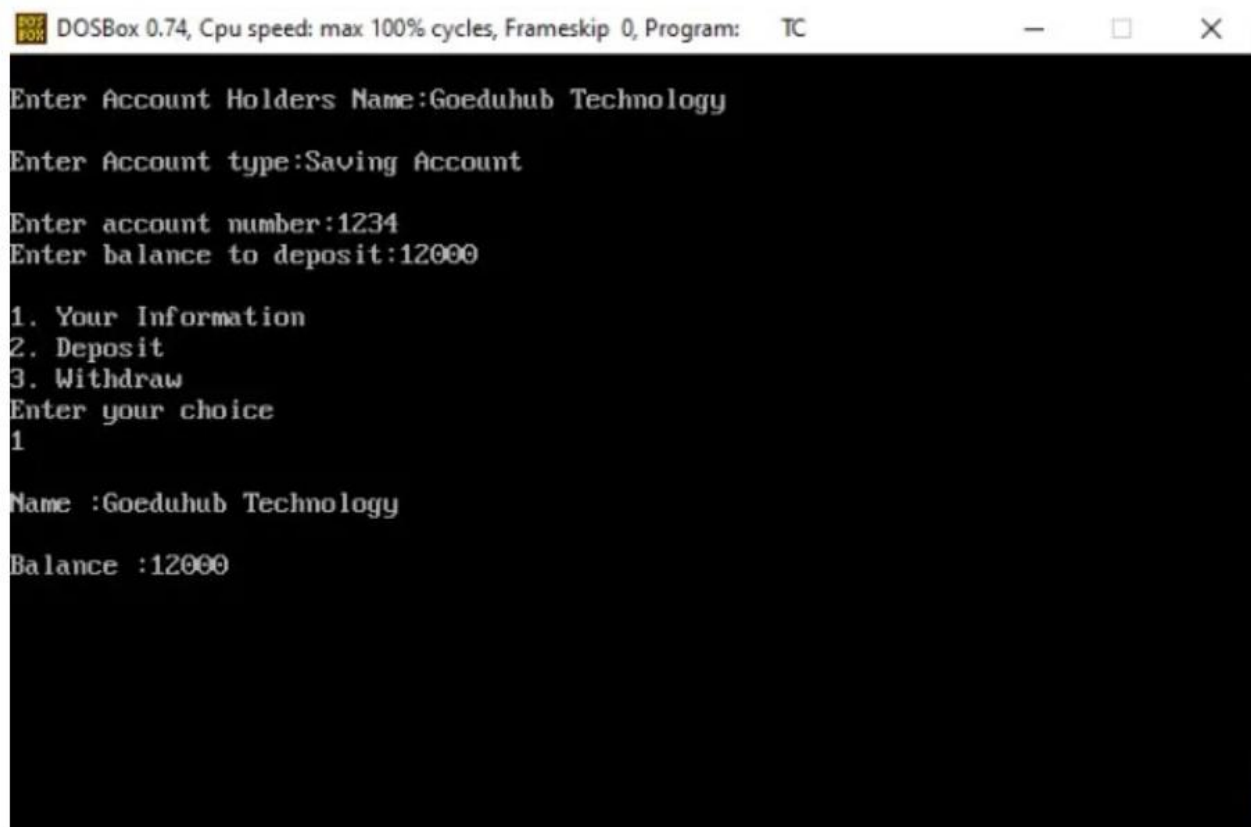
```

Void display()
{
Cout<<"\nName :";
Puts(name);
Cout<<"\nBalance :"<<balance;
}
};

int main()
{
Int I;
Bank bk;
Bk.initialize();
Cout<<"\n1. Your Information\n2. Deposit\n3. Withdraw\nEnter your choice\n";
Cin>>I;
If(i==1)
{
Bk.display();
}
Else if(i==2)
{
Bk.deposit();
}
Else if(i==3)
{
Bk.check();
}
return 0;
}

```

Output :

A screenshot of a DOSBox window. The title bar at the top reads "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window contains a text-based interface for a banking program. The text is as follows:

```
Enter Account Holders Name:Goeduhub Technology
Enter Account type:Saving Account
Enter account number:1234
Enter balance to deposit:12000

1. Your Information
2. Deposit
3. Withdraw
Enter your choice
1

Name :Goeduhub Technology
Balance :12000
```

AssignmentNo.3

Title: Create two classes DM and DB which stores values of distances. DM stores distances in meters and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB. Use a friend function to carry out addition operation

Problem Statement: Implement a C++ program to understand concept of Friend Function.

Objective:

1. Understand the basic principles of object oriented programming
2. Apply the concepts of inheritance and Friend function.
3. Develop programs using object oriented concepts

Theory:

C++Friend Functions

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.

The compiler knows a given function is a friend function by the use of the keyword **friend**.

For accessing the data, the declaration of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section) starting with keyword friend.

Declaration of friend function in C++

```
class class_name
{
    ... ..
    friend return_type function_name(argument/s);
    ... ..
}
```

Now, you can define the friend function as a normal function to access the data of the class. No friend keyword is used in the definition.

Consider in above case,
class className

```

{
    ... ..
    Friend return_type function Name(argument/s);
    ... ..
}
return_type function Name(argument/s)
{
    ... ..
    //Private and protected data of class Name can be accessed from
    //this function because it is a friend function of class Name.
    ... ..
}

```

Program:

```

#include<iostream.h>
Using namespace std;
Class DM
{
Public:
Double meter,centimeter;
};

Class DB
{
Public:
Double feet,inches;
Friend void add(DM,DB);
};

Void add(DM dm,DB db)
{
Double d1,d2;
Cout<<"\nEnter the distance in meter and entimeter:";
Cin>>dm.meter>>dm.centimeter;
Cout<<"\nEnter the distance in feet and inches:";
Cin>>db.feet>>db.inches;
D1=dm.meter+(db.feet)/3.281;
D2=dm.centimeter+(db.inches)*2.54;
Cout<<"\nMeter + Feet = "<<d1<<" meter";
Cout<<"\nCentimeter + inches = "<<d2<<" centimeter";
}
Int main()
{
DM dm;
DB db;
Add(dm,db);
return 0;
}

```

Output:

```
Enter the distance in meter and centimeter:10 12
```

```
Enter the distance in feet and inches:5 4
```

```
Meter + Feet = 11.5239meter
```

```
Centimeter + inches = 22.16 centimeter
```


AssignmentNo.4

Title: Create a class MAT of size $m * n$. Define all possible matrix operations for MAT type objects

Problem Statement: Implement a C++ program to understand concept matrix operations.

Objective:

1. Understand the basic principles of object oriented programming
2. Apply the concepts of overloading, inheritance, polymorphism
3. Develop programs using object oriented concepts

Theory:

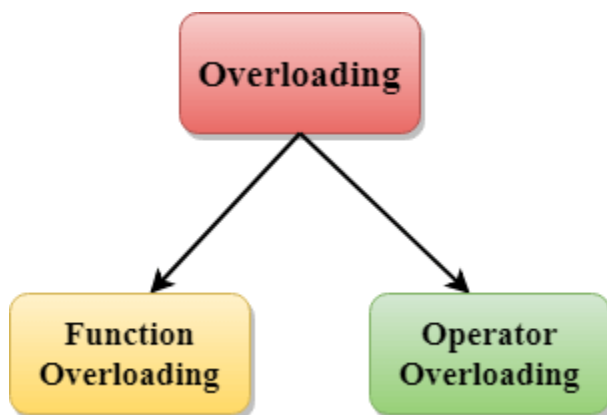
If we create two or more members having the same name but different in number or type of parameter, it is known as C++ overloading. In C++, we can overload:

- methods,
- constructors, and
- indexed properties

It is because these members have parameters only.

Types of overloading in C++ are:

- Function overloading
- Operator overloading



C++ 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.

C++ Function Overloading Example

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)<<endl;
    cout<<C.add(12, 20, 23);
    return 0;
}
```

Output:

```
30
55
```

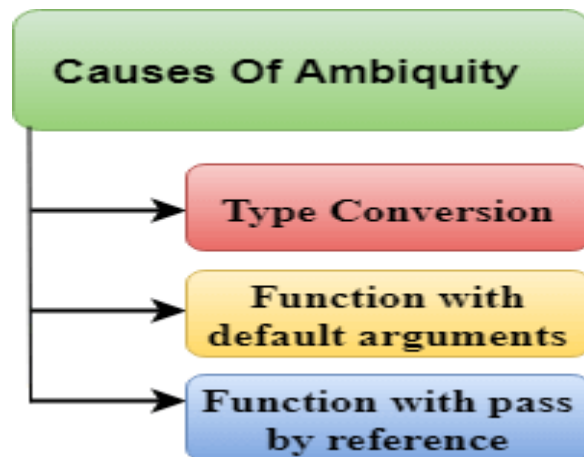
Function Overloading and Ambiguity:

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:

- Type Conversion.
- Function with default arguments.
- Function with pass by reference.



Program:

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

class MAT
{
    int a[2][2];

public:
    void accept()
    {
        cout<<"\n\n Enter 4 element : ";
        for(int i=0;i<2;i++)
        {
            for(int j=0;j<2;j++)
                cin>>a[i][j];
        }
    }

    void display()
    {
        for(int i=0;i<2;i++)
        {
            cout<<endl;
            for(int j=0;j<2;j++)
```

```

        cout<<" "<<a[i][j];
    }
}

MAT operator+(MAT M2)
{
    MAT M3;
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            M3.a[i][j]=a[i][j]+M2.a[i][j];
        }
    }
    return M3;
}

MAT operator*(MAT M2)
{
    MAT M3;

    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            M3.a[i][j]=0;
            for(int k=0;k<2;k++)
            {
                M3.a[i][j]=(a[i][k]*M2.a[k][j])+M3.a[i][j];
            }
        }
    }

    return M3;
}
};

```

```

int main()
{
    MAT M1,M2,M3;

    cout<<"\n\n Enter Matrix M1 value: ";
    M1.accept();

    cout<<"\n\n Enter Matrix M2 value: ";
    M2.accept();

    M3=M1+M2;
    cout<<"\n\n Addition of M1+M2 : ";
    M3.display();

    M3=M1*M2;
    cout<<"\n\n Multiplication of M1*M2 : ";
    M3.display();

    return 0;
}

```

Output:

Enter Matrix M1 value:

Enter 4 element : 1 3 5 7

Enter Matrix M2 value:

Enter 4 element : 2 4 6 8

Addition of M1+M2 :

3 7

11 15

Multiplication of M1*M2 :

20 28

52 76

AssignmentNo.5

Title: Create Stud class to display student information using constructor and destructor. (Default constructor, Multiple constructor, Copy constructor, Overloaded constructor)

Problem Statement: Implement a C++ program to understand concept of constructor and Destructor.

Objective:

1. Understand the basic principles of object oriented programming
2. Apply the concepts of overloading, inheritance, polymorphism
3. Develop programs using object oriented concepts

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++.

- 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.

C++ Parameterized Constructor

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

C++ Destructor

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

Program:

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

class stu
{
    private:
        char name[20],add[20];
        int roll,zip;

    public:
        stu ();//Constructor
        ~stu();//Destructor
        void read();
        void disp();
};

stu :: stu()
{
    cout<<"\nThis is Student Details constructor called....."<<endl;
}

void stu :: read()
{
    cout<<"\nEnter the student Name :: ";
    cin>>name;
    cout<<"\nEnter the student roll no :: ";
    cin>>roll;
    cout<<"\nEnter the student address :: ";
    cin>>add;
    cout<<"\nEnter the Zipcode :: ";
    cin>>zip;
}

void stu :: disp()
{
    cout<<"\nThe Entered Student Details are shown below ::----- \n";
    cout<<"\nStudent Name :: "<<name<<endl;
    cout<<"\nRoll no  is :: "<<roll<<endl;
    cout<<"\nAddress is :: "<<add<<endl;
    cout<<"\nZipcode is :: "<<zip;
}
```

```
stu::~stu()
{
    cout<<"\n\nStudent Detail is Closed.....\n";
}
```

```
int main()
{
    stu s;
    s.read ();
    s.disp ();

    return 0;
}
```


Output:

```
This is Student Details constructor called.....
```

```
Enter the student Name :: abhishek
```

```
Enter the student roll no :: 01
```

```
Enter the student address :: pune
```

```
Enter the Zipcode :: 411002
```

```
The Entered Student Details are shown below ::-----
```

```
Student Name :: abhishek
```

```
Roll no   is :: 1
```

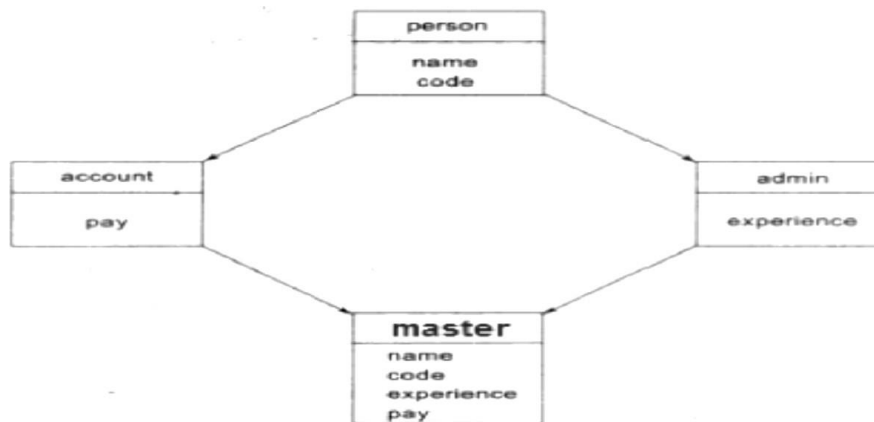
```
Address is :: pune
```

```
Zipcode is :: 411002
```

```
Student Detail is Closed.....
```

Assignment No.6

Title: Consider class network of given figure. The class master derives information from both account and admin classes which in turn derive information from the class person. Define all the four classes and write a program to create, update and display the information contained in master objects.



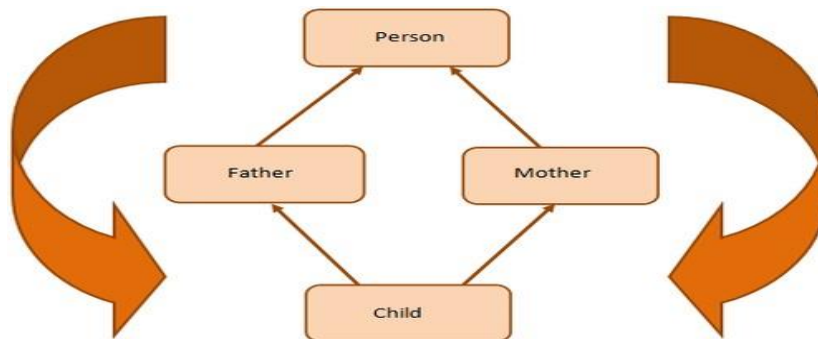
Problem Statement: Implement a C++ program to understand concept of multiple and multilevel inheritance using Virtual function.

Objective:

1. Understand the basic principles of object oriented programming
2. Apply the concepts of inheritance.
3. Develop programs using object oriented concepts

Theory:

The Diamond Problem occurs when a child class inherits from two parent classes who both share a common grandparent class. This is illustrated in the diagram.



Here, we have a class **Child** inheriting from classes **Father** and **Mother**. These two classes, in turn, inherit the class **Person** because both Father and Mother are Person.

As shown in the figure, class Child inherits the traits of class Person twice—once from Father and again from Mother. This gives rise to ambiguity since the compiler fails to understand which way to go.

This scenario gives rise to a diamond-shaped inheritance graph and is famously called “The Diamond Problem.”

The solution to the diamond problem is to use the **virtual** keyword. We make the two parent classes (who inherit from the same grandparent class) into virtual classes in order to avoid two copies of the grandparent class in the child class.

Here we have used the **virtual** keyword when classes Father and Mother inherit the Person class. This is usually called “virtual inheritance,” which guarantees that only a single instance of the inherited class (in this case, the Person class) is passed on.

In other words, the Child class will have a single instance of the Person class, shared by both the Father and Mother classes. By having a single instance of the Person class, the ambiguity is resolved.

Program:

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

class person
{
protected:
    char name[20];
    int code;
public:
    void get_per(int c,char *s)
    {
        code=c;
        strcpy(name,s);
    }
    void put_per()
    {
        cout<<"\nCode : "<<code;
        cout<<"\nName : "<<name;
    }
};

class account : public virtual person
```

```

{
protected:
    float pay;
public:
    void get_pay(float p)
    {
        pay=p;
    }
    void put_pay()
    {
        cout<<"\nPay amount : "<<pay;
    }
};

```

```

class admin : public virtual person
{
protected:
    int exp;
public:
    void get_exp(int e)
    {
        exp=e;
    }
    void put_exp()
    {
        cout<<"\nExperiance : "<<exp;
    }
};

```

```

class master : public account,public admin
{
    private:
        float pay;
        int code,exp;
        char name;

public:
    void display()
    {
        put_per();
        put_pay();
        put_exp();
    }
};

```

```

int main()
{
    master p1;

```

```
        p1.get_per(111,"Piyush");  
    }  
  
    p1.get_pay(501.50);  
    p1.get_exp(2);  
    p1.display();  
  
    return 0;  
  
}
```

Output

```
Code : 111  
Name : Piyush  
Pay amount : 501.5  
Experiance : 2
```

Assignment No. 7

Title: A book shop sells both books and video tapes. Create a class media that stores the title and price of the publication. Create two derived classes, one for storing number of pages in the book and another for storing playing time of tape. A function display() must be defined in all classes to display class contents. Write a program using polymorphism and virtual function.

Problem Statement: Implement a C++ program to understand concept of polymorphism and virtual function.

Objective:

1. Understand the basic principles of object oriented programming
2. Apply the concepts of overloading, inheritance, polymorphism
3. Develop programs using object oriented concepts

Theory:

Static & Dynamic Binding

Polymorphism means „one name“ -, „multiple forms.

The overloaded member functions are „selected“ for invoking by matching arguments, both type and number. This information is known to the compiler at the compile time and compiler is able to select the appropriate function for a particular call at the compile time itself. This is called **Early Binding** or **Static Binding** or **Static Linking**. Also known as **compile time polymorphism**.

Early binding means that an object is bound to its function call at the compile time.

It would be nice if the appropriate member function could be selected while the program is running. This is known as **runtime polymorphism**.

C++ supports a mechanism known as **virtual function** to achieve run time polymorphism.

At the runtime, when it is known what class objects are under consideration, the appropriate version of the function is invoked. Since the function is linked with a particular class much later after the compilation, this process is termed as late binding. It is also known as **dynamic binding** because the selection of the appropriate function is done dynamically at run time.

VIRTUAL FUNCTIONS

Polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but different forms. An essential requirement of polymorphism is therefore the ability to refer to objects without any regard to their classes.

When we use the same function name in both the base and derived classes, the function in the base class is declared as virtual using the keyword virtual preceding its normal declaration. When a function is made virtual, C++ determines which function to use at runtime based on the type of object pointed to by the base pointer, rather than the type of the pointer.

Thus, by making the base pointer to point to different objects, we can execute different versions of the virtual function.

Rules For Virtual Functions:

When virtual functions are created for implementing late binding, observe some basic rules that satisfy the compiler requirements.

1. The virtual functions must be members of some class.
2. They cannot be static members.
3. They are accessed by using object pointers.
4. A virtual function can be a friend of another class.
5. A virtual function in a base class must be defined, even though it may not be used.
6. The prototypes of the base class version of a virtual function and all the derived class versions must be identical. C++ considers them as overloaded functions, and the virtual function mechanism is ignored.
7. We cannot have virtual constructors, but we can have virtual destructors.
8. While a base pointer points to any type of the derived object, the reverse is not true. i.e. we cannot use a pointer to a derived class to access an object of the base class type.
9. When a base pointer points to a derived class, incrementing or decrementing it will not make it to point to the next object of the derived class. It is incremented or decremented only relative to its base type. Therefore we should not use this method to move the pointer to the next object.
10. If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class. In such cases, calls will invoke the base function.

Program:

```
#include <iostream>
#include<string>
using namespace std;
class media
{
protected:
string title;
float price;

public:
media()
{
title=" ";
price=0.0;

}
media(string t,float P)
{
title=t;
price=P;

}
};
class book : public media
{
int P_count;
```

```

public:
book()
{
    P_count=0;

}
book(string t,float P,int pc):media(t,P)
{

    P_count=pc;

}

void display()
{

    cout<<"title : "<<title<<endl;
    cout<<"Price: " <<price<<endl;

    cout<<"Pagecount : "<<P_count<<endl;

}

};

```

```

class CD : public media
{
    float time;
public:
    CD()
    {
        time=0.0;

    }
    CD(string t,float p,float tim):media(t,p)
    {

        time=tim;

    }

void display()
{
    cout<<"title : "<<title<<endl;
    cout<<"Price: " <<price<<endl;

    cout<<"time in minutes : "<<time<<endl;

}

```



```
};  
int main()  
{  
    cout<<endl<<"Book information"<<endl;  
    book Bo("programming in java",1000,500);  
    Bo.display();  
    cout<<endl<<"video information"<<endl;  
    CD cd("programming in c++",100,125);  
    cd.display();  
    return 0;  
}
```

Output:

```
Book information  
title :programming in java  
Price: 1000  
Pagecount :500  
  
video information  
title :programming in c++  
Price: 100  
time in minutes :125
```

Assignment No. 8

Title: Write a program to show use of this pointer, new and delete.

Problem Statement: Implement a C++ program to understand concept of memory allocation using new and delete keyword.

Objective:

1. Understand the basic principles of object oriented programming
2. Implement memory allocation techniques and usage of exception handling, generic programming
3. Develop programs using object oriented concepts

Theory:

C++ supports these functions, it defines two unary operators **new** and **delete** that perform the task of allocating and deallocating the memory in a better and easier way.

An object can be created by using **new**, and destroyed by using **delete**.

A data object created inside a block with **new**, will remain in existence until it is explicitly destroyed by using **delete**.

- **new operator:-**

new operator can be used to create objects of any type .Hence new operator allocates sufficient memory to hold data of objects and it returns address of the allocated memory. Syntax:

pointer-variable = **new** data-type;

Ex: int *p = new int;

To create memory space for arrays:

pointer-variable = **new** data-type[size];

Ex: int *p = new int[10];

- **delete operator:**

If the variable or object is no longer required or needed is destroyed by “**delete**” operator, there by some amount of memory is released for future purpose. Syntax:

delete pointer-variable;

Eg: delete p;

If we want to free a dynamically allocated array:

delete [size] pointer-variable;

Program:

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

```
int main ()
{
```

```
    int* p = NULL;
```

```
    p = new(nothrow) int;
```

```
    if (!p)
```

```

        cout << "allocation of memory failed\n";
    else
    {
        *p = 29;
        cout << "Value of p: " << *p << endl;
    }

    float *r = new float(75.25);

    cout << "Value of r: " << *r << endl;

    int n = 5;
    int *q = new(nothrow) int[n];

    if (!q)
        cout << "allocation of memory failed\n";
    else
    {
        for (int i = 0; i < n; i++)
            q[i] = i+1;

        cout << "Value store in block of memory: ";
        for (int i = 0; i < n; i++)
            cout << q[i] << " ";

    }

    delete p;
    delete r;

    delete[] q;

    return 0;
}

```

Output:

```

Value of p: 29
Value of r: 75.25
Value store in block of memory: 1 2 3 4 5

```

Assignment No. 9

Title: Write a function template for finding the minimum value contained in an array.

Problem Statement: Implement a C++ program to understand concept of template.

Objective:

1. Understand the basic principles of object oriented programming
2. Implement memory allocation techniques and usage of exception handling, generic programming
3. Develop programs using object oriented concepts

Theory:

Instead of writing different functions for the different data types, we can define common function.

For example

```
int max(int a,int b); // Returns maximum of two integers float
```

```
max(float a,float b); // Return maximum of two floats char
```

```
max(char a,char b); // Returns maximum of two characters
```

(this is called as function overloading)

But, instead of writing three different functions as above, C++ provided the facility called "Templates".

With the help of templates you can define only one common function as follows:

```
T max(T a,T b); // T is called generic data type
```

Template functions are the way of making function/class abstracts by creating the behavior of function without knowing what data will be handled by a function. In a sense this is what is known as “generic functions or programming”.

Template function is more focused on the algorithmic thought rather than a specific means of single data type. For example you could make a templated stack push function. This push function can handle the insertion operation to a stack on any data type rather than having to create a stack push function for each different type.

Syntax:

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

Features of templates:-

1. It eliminates redundant code
2. It enhances the reusability of the code.

3. It provides great flexibility to language

Templates are classified into two types. They are

1. Function templates
2. Class Templates.

Function Templates

The templates declared for functions are called as function templates. A function template defines how an individual function can be constructed.

Syntax:

```
template < class type,.....>
ret_type fun_name(arguments)
{
-----// body of the function
-----
}
```

Program:

```
#include <iostream>

template <class T>
class Array {
protected:
    size_t size;
    T* DynamicArray;
public:
    Array() : size( 0 ), DynamicArray( nullptr ) {}

    Array(size_t s) : size(s) {
        DynamicArray = new T[size];
        for ( size_t i = 0; i < size; i++) {
            std::cout << "Element " << i+1 << ": ";
            std::cin >> DynamicArray[i];
        }
    }

    void coutArray() const {
        for ( size_t i = 0; i < size; i++) {
            std::cout << DynamicArray[i] << " ";
        }
    }

    ~Array() {
```

```
delete[]DynamicArray;  
}
```

```
const T * getMin() const  
{  
    T *min = DynamicArray;  
  
    for ( size_t i = 1; i < size; i++ )  
    {  
        if ( DynamicArray[i] < *min ) min = DynamicArray + i;  
    }  
  
    return min;  
}  
};  
  
int main()  
{  
    Array<int> a( 5 );  
  
    const int *min = a.getMin();  
  
    if ( min != nullptr ) std::cout << "The minimum is equal to " << *min << '\n';  
  
    return 0;  
}
```

Output:

```
Element 1: 54  
Element 2: 49  
Element 3: 99  
Element 4: 2  
Element 5: 8  
The minimum is equal to 2  
PS E:\My Codes\C++\opps_assignment> S
```

Assignment No. 10

Title: Write a program containing a possible exception. Use a try block to throw it and catch block to handle it properly.

Problem Statement: Implement a C++ program to understand concept of error handling.

Objective:

1. Understand the basic principles of object oriented programming
2. Implement memory allocation techniques and usage of exception handling, generic programming
3. Develop programs using object oriented concepts

Theory:

Exception handling

Exceptions: Exceptions are runtime anomalies or unusual conditions that a program may encounter while executing. Anomalies might include conditions such as division by zero, accessing an array outside of its bounds or running out of memory or disk space. When a program encounters an exception condition, it must be identified and handled.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

Types of exceptions:

There are two kinds of exceptions

1. Synchronous exceptions
2. Asynchronous exceptions

1. Synchronous exceptions: Errors such as “Out-of-range index” and “over flow” are synchronous exceptions.

2. Asynchronous exceptions: The errors that are generated by any event beyond the control of the program are called asynchronous exceptions.

The purpose of exception handling is to provide a means to detect and report an exceptional circumstance.

Exception Handling Mechanism:

An exception is said to be thrown at the place where some error or abnormal condition is detected. The throwing will cause the normal program flow to be aborted, in a raised exception. An exception is thrown programmatically, the programmer specifies the conditions of a throw.

In handled exceptions, execution of the program will resume at a designated block of code, called a catch block, which encloses the point of throwing in terms of program execution. The catch block can be, and usually is, located in a different function than the point of throwing.

C++ exception handling is built upon three keywords: try, catch, and throw. Try is used to preface a block of statements which may generate exceptions. This block of statements is known as try block. When an exception is detected it is thrown by using throw statement in the try

block. Catch block catches the exception thrown by throw statement in the try block and handles it appropriately.

Program:

```
#include <iostream>
using namespace std;
double zeroDivision(int x, int y) {

    if (y == 0) {
        throw "Division by Zero!";
    }
    return (x / y);
}

int main() {
    int a = 11;
    int b = 0;
    double c = 0;

    try {
        c = zeroDivision(a, b);
        cout << c << endl;
    }
    catch (const char* message) {
        cerr << message << endl;
    }
    return 0;
}
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Division by Zero!

PS E:\My Codes\C++\opps_assignment>

