Vision of the Department

To be recognized for keeping innovation, research and excellence abreast of learning in the field of computer science & engineering to cater the global society.

Mission of the Department

M1: To provide an exceptional learning environment with academic excellence in the field of computer science and engineering.

M2: To facilitate the students for research and innovation in the field of software, hardware and computer applications and nurturing to cater the global society.

M3: To establish professional relationships with industrial and research organisations to enable the students to be updated of the recent technological advancements.

M4: To groom the learners for being the software professionals catering the needs of modern society with ethics, moral values and full of patriotism.

Program Educational Objectives (PEO's)

PEO1: The graduate will have the knowledge and skills of major domains of computer science and engineering in providing solution to real world problems most efficiently.

PEO2: The graduate will be able to create and use the modern tools and procedures followed in the software industry in the relevant domain.

PEO3: The graduate will be following the ethical practices of the software industry and contributing to the society as a responsible citizen.

PEO4: The graduate will have the innovative mindset of learning and implementing the latest developments and research outcomes in the computer hardware and software to keep pace with the fast changing socio economic world.

COURSE OUTCOMES

CO1: Implement encapsulation, abstraction of object oriented technology

CO2: Develop classes, their relationships and associativity of objects with classes.

CO3: Discuss various object oriented features like polymorphism, inheritance and abstract methods with example.

CO4: Perform operations with container classes and templates.

CO5: Perform the file operations with the help of various class library.

List of Experiments

1. Write a program to find the largest number using function.

2. Write a program to find the program the area of circle, rectangle and triangle using function overloading.

3. Write a program to implement complex numbers using operator overloading and type conversion.

4. Write a program illustrating class declarations, definition, and accessing class members.

5. Write a Program to illustrate default constructor, parameterized constructor and copy constructors.

6. Write a Program to demonstrate Friend function and Friend class.

7. Write a Program to access members of a STUDENT class using pointer to object Members

8. Write a C++ program that illustrates the order of execution of constructors and destructors when new class is derived from more than one base class

9. Write a Program containing a possible exception. Use a try block to throw it and a catch block to handle it properly.

10. Write C++ programs that illustrate how the following forms of inheritance are supported:

Single inheritance B) Multiple inheritance Multi level inheritance D) Hierarchical inheritance

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

EXPERIMENT -1

Functions:

The function in C++ language is also known as procedure or subroutine in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Function Declaration:

A function declaration tells the compiler about the number of parameters function takes data-types of parameters, and returns the type of function. Putting parameter names in the function declaration is optional in the function declaration, but it is necessary to put them in the definition. Below is an example of function declarations. (Parameter names are not there in the below declarations)

Syntax:

```
Return-type function-name(data-type parameter...)
        {
        //code to be executed
        }
```

Function body: is the part where the code statements are written.

```
#include <iostream>
Usingnamespacestd;

// Following function that takes two parameters 'x' and 'y'
// as input and returns max of two input numbers
```

Parameters:

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

```
intmax(intx, inty)
{
   if(x > y)
      returnx;
   else
      returny;
}

// main function that doesn't receive any parameter and
// returns integer
intmain()
{
int a = 10, b = 20;

// Calling above function to find max of 'a' and 'b'
  intm = max(a, b);

 cout<< "m is "<< m;
 return0;
}
```

Output :m is 20

Parameter Passing to Functions

- **Formal Parameter :** A variable and its type as they appear in the prototype of the function or method.
- **Actual Parameter :** The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.
- **Modes:**
- **IN:** Passes info from caller to callee.
- **OUT:** Call writes values in caller.
- **IN/OUT:** Caller tells call value of variable, which may be updated by callee.

There are two most popular ways to pass parameters:

1. **Pass by Value*:*** In this parameter passing method, values of actual parameters are copied to the function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in the actual parameters of the caller.

PROGRAM:

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

```
#include<stdio.h>

#include<conio.h>

void main(){
        int num1, num2 ;
      void swap(int,int) ; // function declaration
      clrscr() ;
        num1 = 10 ;
        num2 = 20 ;

      printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
      swap(num1, num2) ; // calling function
      printf("\nAfter swap: num1 = %d\nnum2 = %d", num1, num2);
      getch() ;
      }
      void swap(int a, int b)  // called function
      {
        int temp ;
        temp = a ;
        a = b ;
        b = temp ;
      }
```

Output: Before swap num1=10, num2=20
After swap num1=20, num2=10

2. **Pass by Reference*:*** Both actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in the actual parameters of the caller.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main(){
  int num1, num2 ;
void swap(int *,int *) ; // function declaration
clrscr() ;
  num1 = 10 ;
  num2 = 20 ;
printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
swap(&num1, &num2) ; // calling function
printf("\nAfter swap: num1 = %d, num2 = %d", num1, num2);
getch() ;
}
void swap(int *a, int *b)  // called function
{
  int temp ;
  temp = *a ;
  *a = *b ;
  *b = temp ;
}
```

Output: Before swap num1=10, num2=20
After swap num1=20,num2=10

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

EXPERIMENT -2

Function Overloading:

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different.

If multiple functions having same name but parameters of the functions should be different is known as Function Overloading.If we have to perform only one operation and having same name of the functions increases the readability of the program.Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the function such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you to understand the behavior of the function because its name differs.

Syntax:

```
add(int a, int b)
add(double a, double b)
```

PROGRAM

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

void add(int a, int b)
{
  cout<< "sum = " << (a + b);
}

void add(double a, double b)
{
    cout<<endl<< "sum = " << (a + b);
}
 // Driver code
```

```
int main()
{
   add(10, 2);
   add(5.3, 6.2);

   return 0;
}
```

OUTPUT:sum = 12
sum = 11.5

EXPERIMENT -3

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.

Syntax:

```
Return_type class_name  : : operator op(argument_list)
{
    // body of the function.
}
```

PROGRAM:

```
#include <iostream>
using namespace std;
class Test
{
private:
int num;
```

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

```
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

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

EXPERIMENT -4

Objects:

Object is a real world entity, for example, chair, car, pen, mobile, laptop etc. In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality. Object is a runtime entity, it is created at runtime.

Class:

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

Properties:

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

- Data members are the data variables and member functions are the functions used to manipulate these variables together.
- these data members and member functions define the properties and behavior of the objects in a Class.

Class Definition:

A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semi colon or a list of declarations.

For example, we defined the Box data type using the keyword class as follows:

```
class Box{

    public:

double length; // Length of a box double breadth; // Breadth of a box double height;

    };
```

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

The keyword public determines the access attributes of the members of the class that follow it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as private or protected.

Define C++ Objects:

In order to use the class functionality, we need to instantiate the class to create an object. An object is an instance of a class. I simple words, we can say that an object is a variable of type class.

The general syntax to create an object is:
classname object_name;
Once the object is created, it can be used to access the data members and functions of that class.Accessing the members of the class (data and functions) is done using the dot (.) operator, which is also called as the member access operator.If obj is the name of the object and there is a function "display ()" in the class, then the function can be accessed as "obj.display ()".However, there is a catch in the above statement. We can access the function display () using an object and the dot operator if the function is "public".

PROGRAM:

```
#include <iostream>
using namespace std;
class Room {
public:
   double length;
   double breadth;
   double height;
   double calculateArea() {
      return length * breadth;
   }
 double calculateVolume() {
      return length * breadth * height;
   }
};
int main() {
   // create object of Room class
```

```
    Room room1;
    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;
    // calculate and display the area and volume of the room
cout<< "Area of Room =  "<< room1.calculateArea() <<endl;
cout<< "Volume of Room =  "<< room1.calculateVolume() <<endl;
 return 0;
}
```

OUTPUT: Area of Room = 1309

Volume of Room = 25132.8

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

EXPERIMENT -5

Constructor:

Constructor in C++ is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as the class or structure. Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object which is why it is known as constructors.

- Constructor is a member function of a class, whose name is same as the class name.
- Constructor is a special type of member function that is used to initialize the data members for an object of a class automatically, when an object of the same class is created.
- Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.
- Constructor do not return value, hence they do not have a return type.

Features of Constructor:

- The same name as the class itself
- No return type

SYNTAX:

Constructor can be defined inside the class declaration or outside the class declaration

a. Syntax for defining the constructor within the class

```
<class-name>(list-of-parameters)
    {
        //constructor definition
    }
```

b. Syntax for defining the constructor outside the class

```
<class-name>: :<class-name>(list-of-parameters)
    {
        //constructor definition
    }
```

TYPES OF CONSTRUCTORS:

There are 3 types of constructors in C++, They are:

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Constructor Overloading:

We can have more than one constructor in a class with same name, as long as each has a different list of arguments.This concept is known as Constructor Overloading and is quite similar to function overloading.

- Overloaded constructors essentially have the same name (exact name of the class) and different by number and type of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

Destructor:

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

- Destructor is also a special member function like constructor. Destructor destroys the class objects created by constructor.
- Destructor has the same name as their class name preceded by a tilde (~) symbol.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy the object create by constructor. Hence destructor can-not be overloaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when object goes out of scope.
- Destructor release memory space occupied by the objects created by constructor.
- In destructor, objects are destroyed in the reverse of an object creation.

SYNTAX:

Syntax for defining the destructor within the class

~ <class-name>()

{

}

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

Syntax for defining the destructor outside the class

<class-name>: : ~ <class-name>()

{

}

PROGRAM:

```
#include<iostream>
using namespace std;
class Test
{
public:
Test()
{
cout<<"\n Constructor executed";
}
~Test()
{
cout<<"\n Destructor executed";
}
};

int main()
{
Test t,t1,t2,t3;
return 0;
}
```

OUTPUT: Constructor executed

Constructor executed

Constructor executed

Constructor executed

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

Destructor executed

Destructor executed

Destructor executed

Destructor executed

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

EXPERIMENT -6

Friend Function:

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.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

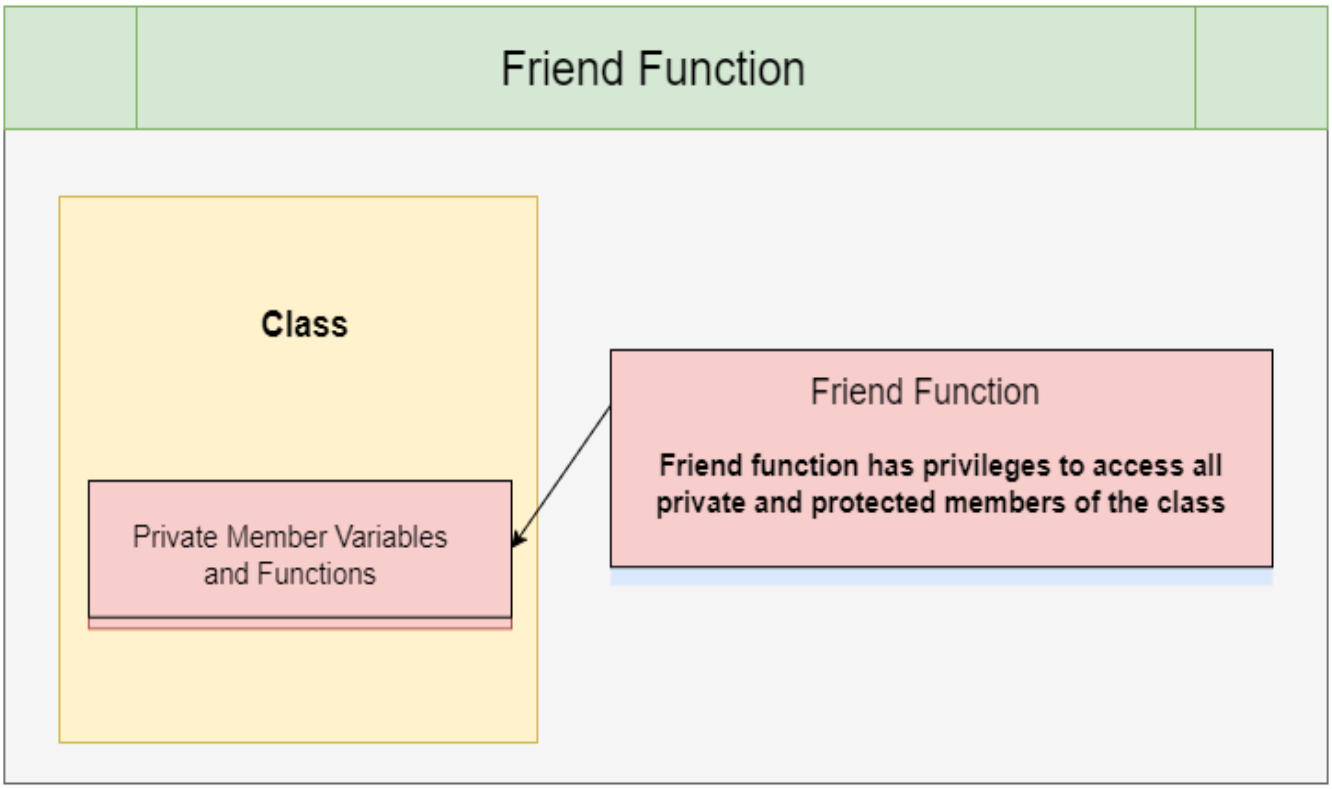
Characteristics of Friend Function:

- A friend function is not in the scope of the class, in which it has been declared as friend.
- It cannot be called using the object of that class.
- It can be invoked like a normal function without any object.
- Unlike member functions, it cannot use the member names directly.
- It can be declared in public or private part without affecting its meaning.
- Usually, it has objects as arguments.

SYNTAX:

```
classclassName {
   ... .. ...
   friendreturnType functionName(arguments);
   ... .. ...
}
```

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY



Friend Class:

A **friend class** can access private and protected members of other classes in which it is declared as a friend. It is sometimes useful to allow a particular class to access private and protected members of other classes. For example, a LinkedList class may be allowed to access private members of Node.
We can declare a friend class in C++ by using the **friend** keyword.

```
Class A{
friend class B;
// class B is a friend class
…… …… ……
}
class B{
……. ……. ……
}
```

As you can see in the syntax, all you need to do is use the keyword friend in front of a class to make it a friend class. Using this syntax will make ClassB a friend class of ClassA. Since ClassB becomes the friend class, it will have access to all the public, private, and protected members of ClassA. However,

the opposite will not be true. That's because C++ only allows granting the friend relation and not taking it. Hence, ClassA will not have access to private members of ClassB.

PROGRAM:

```
#include<iostream>
usingnamespacestd;
classB; //forward declaration.
classA
{
   intx;
   public:
       voidsetdata (inti)
        {
          x=i;
        }
   friendvoidmax (A, B); //friend function.
} ;
classB
{
    inty;
    public:
        voidsetdata (inti)
         {
           y=i;
         }
    friendvoidmax (A, B);
};
voidmax (A a, B b)
```

```
{
  if(a.x>= b.y)
        cout<<a.x<<endl;
  else
       cout<<b.y<<endl;
}
 intmain ()
{
  A a;
  B b;
   a. setdata (10);
   b. setdata (20);
   max (a, b);
   return0;
}
```

OUTPUT:20

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

Expirement-7

Virtual function:

A virtual function in C++ is a base class member function that you can redefine in a derived class to achieve polymorphism. You can declare the function in the base class using the virtual keyword.

Syntax:

```
virtual return_typefunction_name()
{
.....
.....
}
```

Virtual Base Class:

Suppose you have two derived classes B and C that have a common base class A, and you also have another class D that inherits from B and C. You can declare the base class A as virtual to ensure that B and C share the same subobject of A.

In the following example, an object of class D has two distinct subobjects of class L, one through class Bl and another through class B2. You can use the keyword virtual in front of the base class specifiers in the base lists of classes B1 and B2 to indicate that only one subobject of type L, shared by class B1 and class B2, exists.

Object Member function:

Member functions are operators and functions that are declared as members of a class. Member functions do not include operators and functions declared with the friend specifier. These are called friends of a class. You can declare a member function as static ; this is called a static member function.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

Define C++ Objects:

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box .

Box Box1;	// Declare Box1 of type
Box Box2;	// Declare Box2 of type

Accessing Member:

To access a member function by pointer, we have to declare a pointer to the object and initialize it (by creating the memory at runtime, yes! We can use new keyboard for this). The second step, use arrow operator -> to access the member function using the pointer to the object.

Access a member function by pointer:

To **access a member function by pointer**, we have to declare a pointer to the object and initialize it (by creating the memory at runtime, yes! We can use new keyboard for this).

The second step, use arrow operator -> to access the member function using the pointer to the object.

Program:

```
#include <iostream>
using namespace std;
class Number {
private:
   int num;

public:
Number() {
        num = 0;
        };
```

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

```
    void inputNumber(void)
  {
cout<< "Enter an integer number: ";
cin>> num;
  }
    void displayNumber()
  {
cout<< "Num: " << num <<endl;
  }
};


  int main()
  {

    Number N;
N.inputNumber();
N.displayNumber();
    Number* ptrN;
ptrN = new Number;
cout<< "Default value... " <<endl;
ptrN->displayNumber();
ptrN->inputNumber();
ptrN->displayNumber();

  return 0;
}
```

OUTPUT:

Enter the integer number:20
Num:20
Defualt value….

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

EXPERIMENT -8

Constructor:

A constructor is a special member function of a class and shares the same name as of class, which means the constructor and class have the same name. Constructor is called by the compiler whenever the object of the class is created, it allocates the memory to the object and initializes class data members by default values or values passed by the user while creating an object. Constructors don't have any return type because their work is to just create and initialize an object.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

Important Points about Constructors

Access specifiers: Constructors can be defined as public, protected, or private as per the requirements. By default or default constructors, which are created by the compiler are declared as the public. If the constructor is created as private, then we are not able to create its object.

When there is no requirement of the object of a class (in a situation when all class members are static) we can define its constructor as private.

Inheritance: As a derived class can access all the public properties of the base class, it can call its constructor also if it is not declared as private. Also, the constructor's address cannot be referenced.

Virtual: Constructor in C++ cannot be declared as virtual because when we declare any function of a class as a virtual function, at compile time compiler creates a virtual table to store the address of each function that is declared as virtual. Also, it creates a data member of the class virtual pointer to points towards the virtual table. But as we have discussed we cannot refer to the address of the constructor, which means we are not able to declare the constructor as virtual.

Dynamic Constructor

When memory is allocated dynamically to the data members at the runtime using a new operator, the constructor is known as the dynamic constructor. This constructor is similar to the default or parameterized constructor; the only difference is it uses a new operator to allocate the memory.

SYNTAX:

```
classclass_name{

 private:
// private members

public:

// declaring dynamic constructor
class_name({parameters}){// constructor body where data members are initialized using new operator
}
};
```

How Constructor and Destructor are called when the object is Created and Destroyed:

As constructor is the first function called by the compiler when an object is created and the destructor is the last class member called by the compiler for an object. If the constructor and destructor are not declared by the user, the compiler defines the default constructor and destructor of a class object.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

Let's see a code to get the proper idea of how constructor and destructor are called:
First, we will create a class with single parametrized constructors and a destructor. Both of them contain print statements to give an idea of when they are called.

PROGRAM:

```
#include <iostream>
using namespace std;
class class_name{
  // declaring private class data members
private:
  int a,b;
  public:
    // declaring Constructor
class_name(int aa, int bb)
  {
cout<<"Constructor is called"<<endl;
      a = aa;
      b = bb;
cout<<"Value of a: "<<a<<endl;
cout<<"Value of b: "<<b<<endl;
cout<<endl;
  }
    // declaring destructor
  ~class_name()
  {
cout<<"Destructor is called"<<endl;
cout<<"Value of a: "<<a<<endl;
cout<<"Value of b: "<<b<<endl;
  }
```

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

```
};
int main()
{
   // creating objects of class using parameterized constructor
class_nameobj(5,6);

   return 0;
}
```

OUTPUT: Constructor is called
Value of a: 5
Value of b: 6
Destructor is called
Value of a: 5
Value of b: 6

CONCLUSION:

- Constructor and Destructor are the special member functions of the class which are created by the C++ compiler or can be defined by the user.
- Constructor is called by the compiler whenever the object of the class is created, it allocates the memory to the object and initializes class data members.
- A destructor is called by the compiler when the object is destroyed and its main function is to deallocate the memory of the object.
- Constructors have the same as of class while destructors have the same name of the class with the prefix a tilde (~) operator.

- Both Constructor and destructor can be defined as public, private, or protected. But it is better to declare the constructor as public.
- The constructor can have parameters but the destructor doesn't receive any parameters.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

EXPERIMENT -9

Exception:

Exceptions occur for numerous reasons, including invalid user input, code errors, device failure, the loss of a network connection, insufficient memory to run an application, a memory conflict with another program, a program attempting to divide by zero or a user attempting to open files that are unavailable.

When an exception occurs, specialized programming language constructs, interrupt hardware mechanisms or operating system interprocess communication facilities handle the exception..

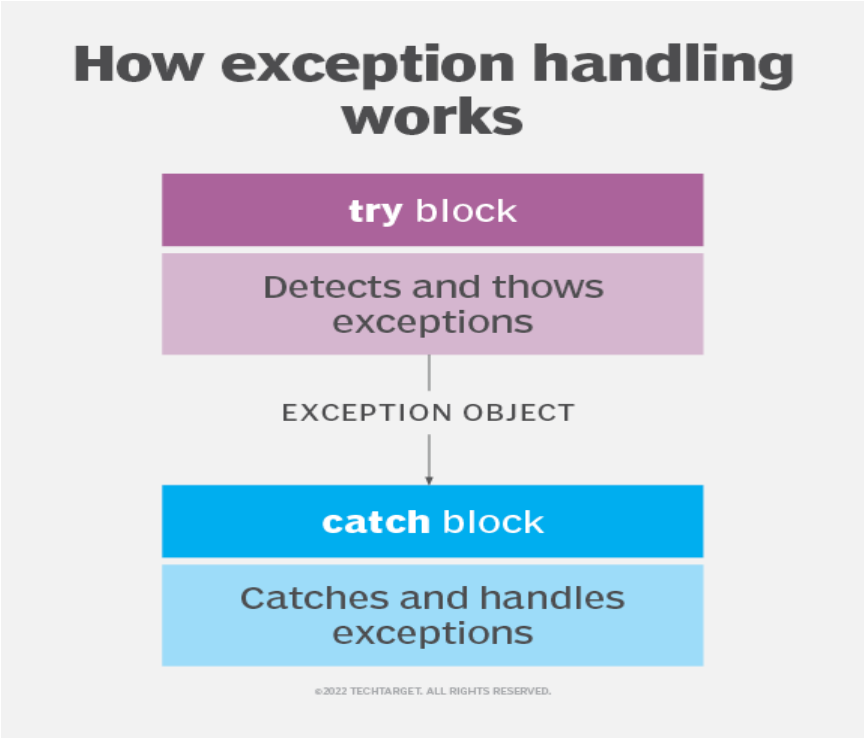
Exception Handling:

Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

How is exception handling used?

If a program has a lot of statements and an exception happens halfway through its execution, the statements after the exception do not execute, and the program crashes. Exception handling helps ensure this does not happen when an exception occurs.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY



Exception handling can catch and throw exceptions. If a detecting function in a block of code cannot deal with an anomaly, the exception is thrown to a function that can handle the exception. A catch statement is a group of statements that handle the specific thrown exception. Catch parameters determine the specific type of exception that is thrown.

Exception handling is useful for dealing with exceptions that cannot be handled locally. Instead of showing an error status in the program, the exception handler transfers control to where the error can be handled. A function can throw exceptions or can choose to handle exceptions.

Error handling code can also be separated from normal code with the use of try blocks, which is code that is enclosed in curly braces or brackets that could cause an exception. Try blocks can help programmers to categorize exception objects.

Examples of exception handling:

The following are examples of exceptions:

- **SQLException** is a checked exception that occurs while executing queries on a database for Structured Query Language syntax.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

- **ClassNotFoundException** is a checked exception that occurs when the required class is not found -- either due to a command-line error, a missing CLASS file or an issue with the classpath.
- **IllegalStateException** is an unchecked exception that occurs when an environment's state does not match the operation being executed.
- **IllegalArgumentException** is an unchecked exception that occurs when an incorrect argument is passed to a method.
- **NullPointerException** is an unchecked exception that occurs when a user tries to access an object using a reference variable that is null or empty.

PROGRAM:

```
#include <iostream>

using namespace std;

int main()

{

int x = -1;

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 Caught (Will be executed) \n";

return 0;
}
```

OUTPUT:

```
Before try
Inside try
Exception Caught
After catch (Will be executed)
```

Expirement-10

Inheritance:

Inheritance is a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them. Inheritance is almost like embedding an object into a class.

Syntax :

```
class base-class-name
{
  // members...
```

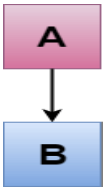
```
  // member function
}
class derived-class-name : visibility-mode base-class-name
{
  // members....
  // member function
}
```

Types Of Inheritance:

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

SINGLE INHERITANCE

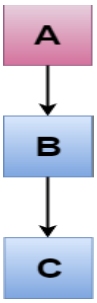
In single inheritance there exists single base class and single derived class.



MULTIPLE INHERITANCE

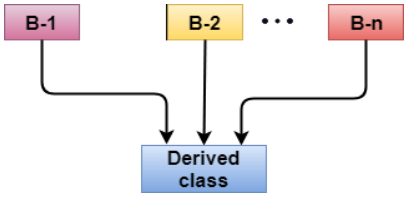
In multipal inheritance there exist multipal base classes and single derived class.

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY



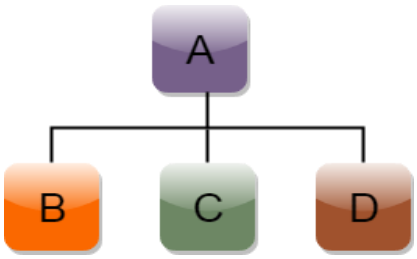
MULTI LEVEL INHERITANCE:

In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The super class for one is sub class for the other.



HIERARCHICAL INHERITANCE:

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



PROGRAM:

Single inheritance

```
#include <iostream>
using namespace std;
class base
{
  public:
  int x;
```

```
  void getdata()
  {
cout<< "Enter the value of x = ";
cin>> x;
  }
};
class derive : public base
{
  private:
   int y;
  public:
  void readdata()
  {
cout<< "Enter the value of y = "; cin>> y;
  }
  void product()
  {
cout<< "Product = " << x * y;
  }
};

int main()
{
   derive a;
a.getdata();
a.readdata();
a.product();
   return 0;
}
```

Output

```
Enter the value of x = 3
Enter the value of y = 4
Product = 12
```

Multiple inheritance

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

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY

```
                                                                    public:
                                                                    int x;
                                                                    void getx()
    {
                                                                    cout<<    "enter
value of x: "; cin>> x;
    }
};
class B
{
                                                                    public:
                                                                    int y;
                                                                    void gety()
                                                                    {
                                                                    cout<<    "enter
value of y: "; cin>> y;
                                                                    }
};
class C : public A, public b
{
                                                                    public:
                                                                    void sum()
                                                                    {
                                                                    cout<< "Sum =
" << x + y;
                                                                    }
};

int main()
{
                                                                     C obj1;
                                                                     obj1.getx();
                                                                     obj1.gety();
                                                                    obj1.sum();
                                                                     return 0;
}
```

Output:

```
enter value of x: 5
enter value of y: 4
Sum = 9
```

CS-305
OBJECT ORIENTED PROGRAMMING AND METHODOLOGY