JAIN COLLEGE OF ENGINEERING RESEARCH, BELAGAVI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2023-24

LABORATORY MANUAL

SUBJECT: Object Oriented Programming with C++

SUB CODE: BCS306B

SEMESTER: III

Prepared By,

Prof. Srinidhi Kulkarni

Approved By,

Dr. Pritam D.

HOD, CSE

INSTITUTIONAL MISSION AND VISION

Vision of the Institute

- To become a leading institute that offers quality technical education and research, nurturing professionally competent graduates with strong ethical values.

Mission of the Institute

- Achieve academic excellence through innovative and effective teaching-learning practices.
- Establish industry and academia collaborations to cultivate a culture of research.
- Ensure professional and ethical values to address societal needs.

Department of ComputerScience & Engineering

VisionoftheDepartment

- To produce competent computer science and engineering graduates by imparting technical knowledge and research to meet industrial needs with moral values.

MissionoftheDepartment

- To facilitate learning opportunities through teaching and mentoring.
- Impart skills in the areas of Computer Science and Engineering through research and industry collaborations.
- To inculcate strong ethical values, understand the societal needs in the field of Computer Science and Engineering.

ProgramOutcomes(POs)

EngineeringGraduateswillbeableto:

PO1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineeringfundamentals,andanengineeringspecializationtothesolutionofcomplexengineering problems.

PO2. **Problem analysis**: Identify, formulate, review research literature, and analyze complexengineering problems reaching substantiated conclusions using first principles of mathematics,naturalsciences, andengineeringsciences.

PO3. **Design/development of solutions**: Design solutions for complex engineering problemsand design system components or processes that meet the specified needs with

appropriateconsideration for the public health and safety, and the cultural, societal, and environmentalconsiderations.

PO4. **Conduct investigations of complex problems**: Use research-based knowledge andresearch methods including design of experiments, analysis and interpretation of data, andsynthesisof the information to providevalid conclusions.

PO5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, andmodern engineering and IT tools including prediction and modeling to complex engineeringactivitieswith an understandingofthe limitations.

PO6. **The engineer and society**: Apply reasoning informed by the contextual knowledge toassess societal, health, safety, legal and cultural issues and the consequent responsibilitiesrelevantto theprofessional engineeringpractice.

PO7.**Environmentandsustainability**:Understandtheimpactoftheprofessionalengineering solutions in societal and environmental contexts, and demonstrate the knowledgeof,and needforsustainabledevelopment.

PO8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilitiesandnorms of the engineeringpractice.

PO9. **Individual and team work**: Function effectively as an individual, and as a member orleaderin diverseteams,and in multidisciplinarysettings.

PO10. **Communication**: Communicate effectively on complex engineering activities with theengineering community and with society at large, such as, being able to comprehend and writeeffectivereportsanddesigndocumentation,makeeffectivepresentations,andgiveandreceiveclearinstructions.

PO11. **Project management and finance**: Demonstrate knowledge and understanding of theengineering and management principles and apply these to one‘s own work, as a member andleaderin ateam, to manageprojects and inmultidisciplinaryenvironments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability toengageinindependentandlife-longlearninginthe broadest contextoftechnologicalchange.

ProgramSpecificOutcomes(PSOs)

PSO1: Apply mathematical and scientific skills in the area of computer science and engineering to design, develop, analyze software and hardware based systems.

PSO2: Provide solutions using networking and database administration skills, and address the needs of environmental and societal issues through entrepreneurial practices.

ProgramEducationalObjectives(PEOs):

1. To produce graduates who have a strong foundation of knowledge and Engineering skills that will enable them to have successful career in the field of computer science and engineering.
2. To expose graduates to professional and team building skills along with ideas of innovation and invention.
3. To prepare graduates with an ethics, social responsibilities, and professional concerns.

Introduction to C++

C++ is a general-purpose, object-oriented programming language. It was developed in 1979 by Bjarne Stroustrup at AT & T Bell Laboratory. It is a high-level programming language & advanced version of C programming language. As Compared to other programming languages like Java, and Python, it is the fastest programming language.

Basic Syntax of a C++ Program

```
#include<iostream>
using namespace std;
main()
{
        //body;
}
```

In **line #1**, we used the **#include <iostream>** statement to tell the compiler to include an **iostream** header file library which stores the definition of input and output statements.

In **line #2**, we have used the **using namespace std** statement for specifying that we will be the standard namespace where all the standard library functions are defined.

In **line #3**, we defined the main function as **int main().** The main function is the most important part of any C++ program. The program execution always starts from the main function. All the other functions are called from the main function.

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**.

- **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.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.

The two instances **cout in C++** and **cin in C++** of iostream class are used very often for printing outputs and taking inputs respectively.

- **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.
- **Identifiers**

We use identifiers for the naming of variables, functions, and other user-defined data types. An identifier may consist of uppercase and lowercase alphabetical characters, underscore, and digits. The first letter must be an underscore or an alphabet.

Example:
int **num1** = 24;
int **num2** = 34;

C++ Control Statements:

The *if* statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the C/C++ **else statement**. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)

{

   // Executes this block if

   // condition is true

}

else

{

   // Executes this block if

   // condition is false

}
```

Using Loops

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below. In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

There are mainly two types of loops:

1. **Entry Controlled loops**: In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loop** is entry-controlled loops.
2. **Exit Controlled Loops**: In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do-while **loop** is exit controlled loop.

S.No.	Loop Type and Description
1.	**while loop** – First checks the condition, then executes the body.
2.	**for loop** – firstly initializes, then, condition check, execute body, update.
3.	**do-while** **loop** – firstly, execute the body then condition check

For Loop-

A *For loop* is a repetition control structure that allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

Syntax:

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

While Loop-

In **for loop** the number of iterations is ***known beforehand***, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where **we do not know** the exact number of iterations of the loop **beforehand**. The loop execution is terminated on the basis of the test conditions.

Syntax:

```
initialization expression;
while (test_expression)
{
  // statements

 update_expression;
}
```

Do-while loop

In Do-while loops also the loop execution is terminated on the basis of test conditions. The main difference between a do-while loop and the while loop is in the do-while loop the condition is tested at the end of the loop body, i.e do-while loop is exit controlled whereas the other two loops are entry-controlled loops.

Note: In a do-while loop, the loop body will ***execute at least once*** irrespective of the test condition.

Syntax:

```
initialization expression;
do
{
  // statements
  update_expression;
} while (test_expression);
```

1) Develop a C++ program to find the largest of three numbers

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

int main()
{

   double n1, n2, n3;

   cout << "Enter three numbers: ";
   cin >> n1 >> n2 >> n3;

   // check if n1 is the largest number
   if(n1 >= n2 && n1 >= n3)
      cout << "Largest number: " << n1;

   // check if n2 is the largest number
   else if(n2 >= n1 && n2 >= n3)
      cout << "Largest number: " << n2;

   // if neither n1 nor n2 are the largest, n3 is the largest
   else
      cout << "Largest number: " << n3;

   return 0;
}
```

Output:

```
Enter three numbers: 36   96   57
Largest number: 96
```

2) Develop a C++ program to sort the elements in ascending and descending order.

```
#include <iostream>

using namespace std;

int main(){

        int num[100],n;

        int i,j,man;

        cout<<"enter n for the numbers you want to sort"<<endl<<endl;

        cin>>n;

        for(i=0;i<n;i++){

                cout<<"enter number"<<endl;

                cin>>num[i];

        }

        for(i=0;i<n;i++){

                for(j=0;j<n;j++){

                        if(num[i]<num[j]){

                                man=num[i];

                                num[i]=num[j];

                                num[j]=man;

                        }

                }

        }

        cout<<"ascending "<<endl;

        for(i=0;i<n;i++){

                cout<<" "<<num[i]<<endl;;
```

```
        }


        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        if(num[i]>num[j]){
                                man=num[i];
                                num[i]=num[j];
                                num[j]=man;
                        }
                }
        }
        cout<<" descending"<<endl;
        for(i=0;i<n;i++){
                cout<<" "<<num[i]<<endl;
        }
        return 0;
}
```

Output:

```
enter n for the numbers you want to sort
5
enter number
5
enter number
3
enter number
9
enter number
4
enter number
```

1

ascending
1
3
4
5
9
descending
9
5
4
3
1

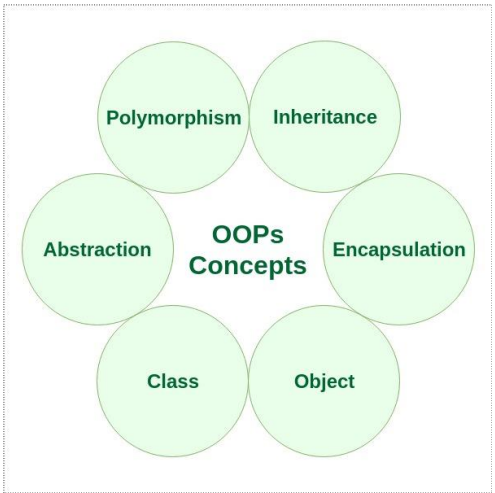
Object Oriented Programming in C++

Object-oriented programming – As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

There are some basic concepts that act as the building blocks of OOPs i.e.

1. Class
2. Objects
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance

Characteristics of an Object-Oriented Programming Language



Class

The building block of C++ that leads to Object-Oriented programming is a Class. 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.

Object

An Object is an identifiable entity with some characteristics and behavior. 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.

Encapsulation

In normal terms, Encapsulation is defined as wrapping up data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

Abstraction

Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Inheritance

The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

- **Sub Class**: The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class**: The class whose properties are inherited by a sub-class is called Base Class or Superclass.

3) Develop a C++ program using classes to display student name, roll number, marks obtained in two subjects and total score of student

```
#include<iostream>
using namespace std;
class student
{
        public:
                int rno;
                string name;
                float marks1,marks2;
           void get_no(string b,int a)
        {
                name=b;
                rno=a;
        }
        void get_marks(float x,float y)
        {
                        marks1=x;
                        marks2=y;
        }
         void put_no(void)
        {
                cout<<"Student name: "<<name<<endl;
                cout<<"Roll No: "<<rno<<endl;
                cout<<"Marks 1: "<<marks1<<endl;
```

```
            cout<<"Marks 2: "<<marks2<<endl;
            cout<<"Total score: "<<marks1+marks2<<endl;
        }
};
int main()
{
   student std1;
   std1.get_no("abc",1234);
   std1.get_marks(27.5,33.0);
   std1.put_no();
   return 0;
}
```

Output:

```
Student name: abc
Roll No: 1234
 Marks1: 27.5
 Marks2: 33.0
total score: 60.5
```

4) Develop a C++ program for a bank empolyee to print name of the employee, account_no. & balance. Print invalid balance if amount <500, Display the same, also display the balance after withdraw and deposit.

```
#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;


class bank
{
   public:
      int acno;
      string nm;
      float bal;
  public:
      void get_data(int,string,float);
      void deposit();
      void withdraw();
      void display();
      void Invalid();
};
void bank::get_data(int acc_no, string name, float balance)
{
      acno=acc_no;
            nm=name;
```

```
        bal=balance;
}
void bank::deposit()   //depositing an amount
{
        int damt1;
        cout<<"\n Enter Deposit Amount = ";
        cin>>damt1;
        bal+=damt1;
}
void bank::withdraw()  //withdrawing an amount
{
        int wamt1;
        cout<<"\n Enter Withdraw Amount = ";
        cin>>wamt1;
        if(wamt1>bal)
                cout<<"\n Cannot Withdraw Amount";
        bal-=wamt1;
}
void bank::display()  //displaying the details
{
        cout<<"\n ----------------------";
        cout<<"\n Accout No. : "<<acno;
        cout<<"\n Name : "<<nm;
        cout<<"\n Balance : "<<bal;
}
```

```
void bank::Invalid()
{
    if(bal<500)
      {
         cout<<"Invalid balance";
      }
}
int main()
{
      bank b1;  //object is created
      b1.get_data(7858585,"ststst",400);
      b1.Invalid();
      b1.deposit(); //
      b1.withdraw(); // calling member functions
      b1.display(); //

      return 0;
}
```

Output:

```
Invalid balance
 Enter Deposit Amount = 25000
 Enter Withdraw Amount = 3000
 ----------------------
 Accout No. : 7858585
 Name : ststst
```

Balance : 22400

Function Overloading in C++

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. Function overloading can be considered as an example of a polymorphism feature in C++.

The parameters should follow any one or more than one of the following conditions for Function overloading:

- Parameters should have a different type

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

- Parameters should have a different number

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

5) Develop a C++ program to demonstrate function overloading for the following prototypes.
add(int a, int b)
add(double a, double b)

```
#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
```

Operator Overloading in C++

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.
In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big integers, etc.

What is the difference between operator functions and normal functions?
Operator functions are the same as normal functions. The only differences are, that the name of an operator function is always the operator keyword followed by the symbol of the operator, and operator functions are called when the corresponding operator is used.

Operators that can be overloaded	Examples
Binary Arithmetic	+, -, *, /, %
Unary Arithmetic	+, -, ++, —
Assignment	=, +=,*=, /=,-=, %=
Bitwise	& , | , << , >> , ~ , ^
De-referencing	(->)
Dynamic memory allocation, De-allocation	New, delete
Subscript	[ ]
Function call	()
Logical	&, II, !
Relational	< ,<=, > ,>=, ==

6) Develop a C++ program using Operator overloading for overloading Unary minus operator.

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

class Numbers
{
      int x, y, z;
   public:
      void accept()
      {
            cout<<"\n Enter Three Numbers";
            cout<<"\n -------------------------";
            cout<<"\n First Number   :  ";
            cin>>x;
            cout<<"\n Second Number  :  ";
            cin>>y;
            cout<<"\n Three Number   :  ";
              cin>>z;
            cout<<"\n -------------------------";
      }
      void display()
      {
            cout<<" ";
            cout<<x<<"\t"<<y<<"\t"<<z;
      }
      void operator-()
      {
            x=-x;
            y=-y;
            z=-z;
      }
};
int main()
{
      Numbers num;
      num.accept();
      cout<<"\n Numbers are :\n\n";
```

```
        num.display();
        -num;    //Overloaded Unary (-) Operator
        cout<<"\n\n Negated Numbers are :\n\n";
        num.display();
        return 0;
}
```

Output:

```
Enter Three Numbers
 -------------------------
 First Number   :  5

 Second Number  :  6

 Three Number   :  7

 -------------------------
 Numbers are :

 5      6       7

 Negated Numbers are :

 -5     -6      -7
--------------------------------
```

Implementing inheritance in C++:

For creating a sub-class that is inherited from the base class we have to follow the below syntax.

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

Syntax:

class <derived_class_name> : <access-specifier> <base_class_name>

{

//body

}

Where

class — keyword to create a new class

derived_class_name — name of the new class, which will inherit the base class

access-specifier — either of private, public or protected. If neither is specified, PRIVATE is taken as default

base-class-name — name of the base class

Modes of Inheritance: There are 3 modes of inheritance.

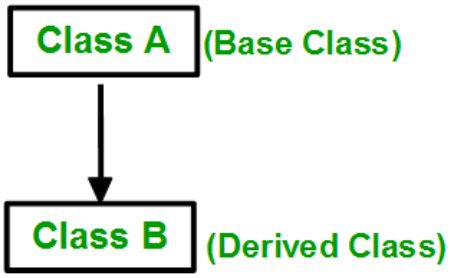
1. **Public Mode**: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.
2. **Protected Mode**: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.
3. **Private Mode**: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

Types Of Inheritance:-

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

Types of Inheritance in C++

1. Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.



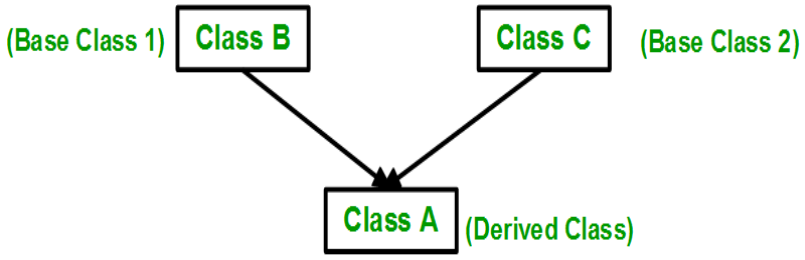
Syntax:

```
class subclass_name : access_mode base_class
{
  // body of subclass
};
```

or

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

2. Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one **subclass** is inherited from more than one **base class**.

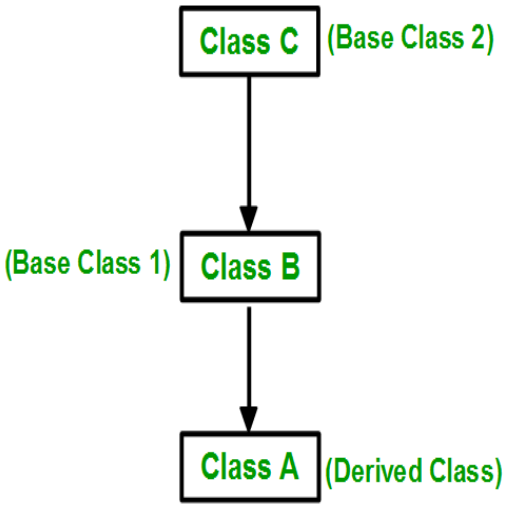


Syntax:

```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
```

```
  // body of subclass
};
class B
{
... .. ...
};
class C
{
... .. ...
};
class A: public B, public C
{
... ... ...
};
```

3. Multilevel Inheritance: In this type of inheritance, a derived class is created from another derived class.



Syntax:-

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

{

... .

};

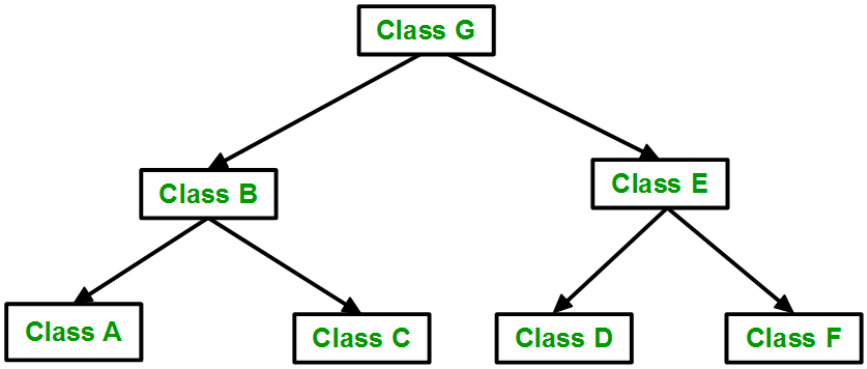
class A: public B

{

... ...

};

4. Hierarchical Inheritance: In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.

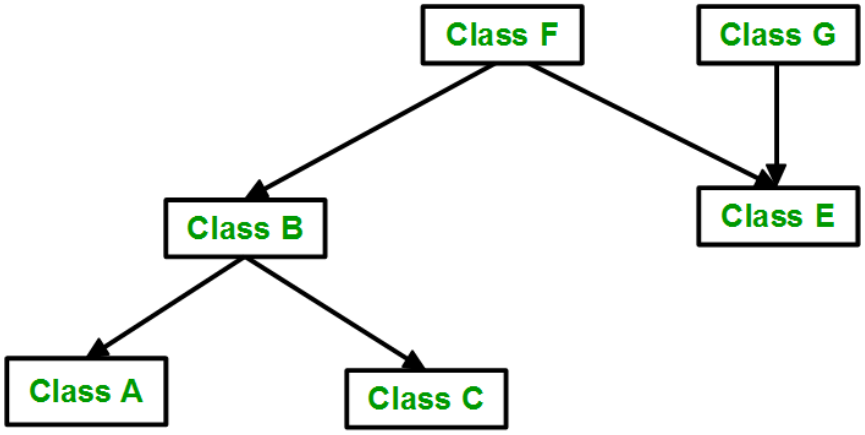


Syntax:-

```
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.
}
```

5. Hybrid (Virtual) Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.
Below image shows the combination of hierarchical and multiple inheritances:



7) Develop a C++ program to implement Multiple inheritance for performing arithmetic operation of two numbers

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

class M
{
   protected:
      int m;
      public :
         void get_M(int );
};

class N
{
   protected:
      int n;
      public:
         void get_N(int);
};

class P: public M, public N
{
   public:
      void display(void);
};

void M::get_M(int x)
{
   m=x;
}

void N::get_N(int y)
{
   n=y;
}

void P::display(void)
{
```

```
    cout<<"\n\tm = "<<m<<endl;
    cout<<"\n\tn = "<<n<<endl;
    cout<<"\n\tm*n = "<<m*n<<endl;
}

int main()
{
    P p;
    p.get_M(10);
    p.get_N(20);
    p.display();
    return 0;
}
```

Output:

```
m = 10

n = 20

 m*n = 200
```

Constructors in C++

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. It constructs the values i.e. provides data for the object which is why it is known as constructor.

• 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.

The prototype of the constructor looks like

<class-name> (list-of-parameters);

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
}
```

Characteristics of constructor

• The name of the constructor is same as its class name.

• Constructors are mostly declared in the public section of the class though it can be declared in the private section of the class.

• Constructors do not return values; hence they do not have a return type.

• A constructor gets called automatically when we create the object of the class.

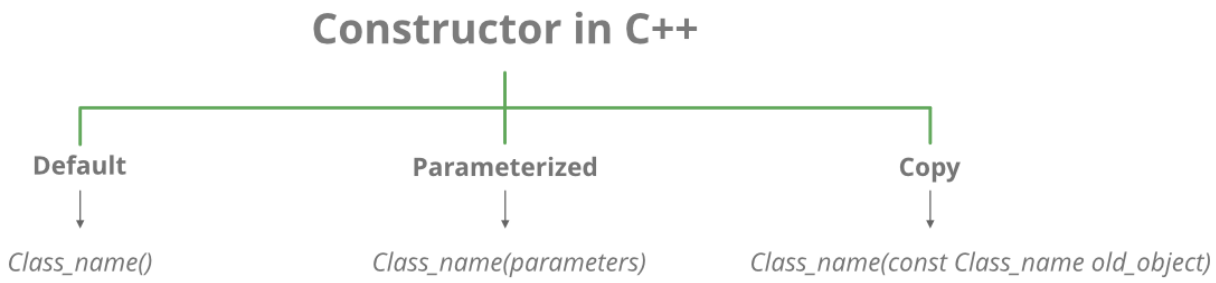
• Constructors can be overloaded.

- Constructor can not be declared virtual.

Types of constructor

- Default constructor
- Parameterized constructor
- Copy constructor

Constructor does not have a return value, hence they do not have a return type.



8) Develop a C++ program using Constructor in Derived classes to initialize alpha, beta and gamma and display corresponding values.

```
#include <iostream>
using namespace std;
class alpha
{
   int x;
   public:
   alpha (int i)
   {
      x=i;
      cout<<"Alpha Initialized";
   }
         void show_x()
   {
      cout<<"\n x="<<x;
   }
};

class beta
{
   float y;
   public:
   beta(float j)
   {
      y=j;
      cout<<"\n Beta Initialized \n";
   }
   void show_y()
   {
      cout<<"\n y="<<y;
   }
};

class gamma:public beta,public alpha
{
   int m,n,c,d;
   public:
```

```
    gamma(int a,float b,int c,int d):alpha(a),beta(b)
    {
        m=c;
        n=d;
        cout<<"\n gamma initialized \n";
    }
    void show_mn()
    {
            cout<<"\n m="<<m;
        cout<<"\n n="<<n;
           }
};

int main()
{
    gamma g(5,7.65,30,100);
    g.show_x();
    g.show_y();
    g.show_mn();
    return 0;

}
```

Output:

```
Beta Initialized
Alpha Initialized
 gamma initialized

 x=5
 y=7.65
 m=30
 n=100
```

File Handling through C++ Classes

File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory (Hard disk).
How to achieve the File Handling
For achieving file handling we need to follow the following steps:-
STEP 1-Naming a file
STEP 2-Opening a file
STEP 3-Writing data into the file
STEP 4-Reading data from the file
STEP 5-Closing a file.

ifstream:-

- This class provides input operations.
- It contains open() function with default input mode.
- Inherits the functions get(), getline(), read(), seekg() and tellg() functions from the istream.

ofstream:-

- This class provides output operations.
- It contains open() function with default output mode.
- Inherits the functions put(), write(), seekp() and tellp() functions from the ostream.

fstream:-

- This class provides support for simultaneous input and output operations.
- Inherits all the functions from istream and ostream classes through iostream.

Class	Description
`ofstream`	Creates and writes to files
`ifstream`	Reads from files
`fstream`	A combination of ofstream and ifstream: creates, reads, and writes to files

Create and Write To a File

To create a file, use either the ofstream or fstream class, and specify the name of the file.

To write to the file, use the insertion operator (<<).

Read a File

To read from a file, use either the ifstream or fstream class, and the name of the file.

Note that we also use a while loop together with the getline() function (which belongs to the ifstream class) to read the file line by line, and to print the content of the file

9) Develop a C++ program to create a text file, check file created or not, if created it will write some text into the file and then read the text from the file.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
   string filename = "mytextfile.txt";
   ofstream outputFile(filename.c_str()); // Convert std::string to const char*

   if (!outputFile) {
      cerr << "Error creating the file." << endl;
      return 1;
   }

   // Write text into the file
   outputFile << "Hello, this is some text written to the file." << endl;
   outputFile << "You can add more lines if you'd like." << endl;

   outputFile.close();

   // Check if the file was created successfully
   ifstream inputFile(filename.c_str()); // Convert std::string to const char*

   if (!inputFile) {
      cerr << "Error opening the file for reading." << endl;
      return 1;
   }

   cout << "File contents:" << endl;

   // Read and display text from the file
```

```
    string line;
    while (getline(inputFile, line)) {
        cout << line << endl;
    }

    inputFile.close();

    return 0;
}
```

Output:

File contents:
Hello, this is some text written to the file.
You can add more lines if you'd like.

10) Develop a C++ program to write and read time in/from binary file using fstream

```
#include <iostream>
#include <fstream>
#include <ctime>

// Introduce a using directive for the entire std namespace
using namespace std;

int main() {
   // Get the current system time
   time_t currentTime = time(NULL);

   // Define a file stream for writing
   ofstream outputFile("time.bin", ios::binary);

   if (!outputFile) {
      cerr << "Error opening the file for writing." << endl;
      return 1;
   }

   // Write the time to the binary file
   outputFile.write(reinterpret_cast<char*>(&currentTime), sizeof(currentTime));
   outputFile.close();

   // Define a file stream for reading
   ifstream inputFile("time.bin", ios::binary);

   if (!inputFile) {
      cerr << "Error opening the file for reading." << endl;
      return 1;
   }

   // Read the time from the binary file
   time_t readTime;
   inputFile.read(reinterpret_cast<char*>(&readTime), sizeof(readTime));

   if (inputFile) {
      cout << "Time read from the file: " << asctime(localtime(&readTime));
   } else {
```

```
        cerr << "Error reading the time from the file." << endl;
    }

    inputFile.close();

    return 0;
}
```

Output:

Time read from the file: Mon Dec 18 12:57:00 2023

C++ Exceptions

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an **exception** (throw an error).

C++ try and catch

Exception handling in C++ consists of three keywords: try, throw and catch:

- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The throw keyword throws an exception when a problem is detected, which lets us create a custom error.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

```
try {
  // Block of code to try
  throw exception; // Throw an exception when a problem arise
}
catch () {
  // Block of code to handle errors
}
```

Handling the Divide by Zero Exception in C++
Dividing a number by Zero is a mathematical error (not defined) and we can use exception handling to gracefully overcome such operations. If you write a code without using exception handling then the output of division by zero will be shown as infinity which cannot be further processed.

Array Index Out of Bound Exception in C++

The ArrayIndexOutOfBoundsException is a runtime exception that occurs when an array is accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Generally, an array is of fixed size and each element is accessed using the indices. For example, we have created an array with size 9. Then the valid expressions to access the elements of this array will be a[0] to a[8] (length-1).

Whenever you used an –ve value or, the value greater than or equal to the size of the array, then the **ArrayIndexOutOfBoundsException** is thrown.

For Example, if you execute the following code, it displays the elements in the array asks you to give the index to select an element. Since the size of the array is 7, the valid index will be 0 to 6.

11) Develop a function which throws a division by zero exception and catch it in catch block. Write a C++ program to demonstrate usage of try, catch and throw to handle exception.

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

using namespace std;

double divide(int numerator, int denominator) {
   if (denominator == 0) {
      throw runtime_error("Division by zero is not allowed.");
   }
   return static_cast<double>(numerator) / denominator;
}

int main() {
   int numerator, denominator;

   cout << "Enter the numerator: ";
   cin >> numerator;

   cout << "Enter the denominator: ";
   cin >> denominator;

   try {
      double result = divide(numerator, denominator);
      cout << "Result of division: " << result << endl;
   } catch (const exception& e) {
      cerr << "Exception caught: " << e.what() << endl;
   }
```

```
    return 0;
}
```

Output:

Enter the numerator: 9

Enter the denominator: 0

Exception caught: Division by zero is not allowed.

12) Develop a C++ program that handles array out of bounds exception using C++.

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

using namespace std;

int main() {
    int myArray[] = {1, 2, 3, 4, 5};
    int index;

    // Display the array contents and size
    cout << "Array Contents: ";
    for (int i = 0; i < sizeof(myArray) / sizeof(myArray[0]); ++i) {
        cout << myArray[i] << " ";
    }
    cout << endl;
    cout << "Array Size: " << sizeof(myArray) / sizeof(myArray[0]) << endl;

    cout << "Enter an index to access: ";
    cin >> index;

    try {
        if (index >= 0 && index < sizeof(myArray) / sizeof(myArray[0])) {
            int value = myArray[index];
            cout << "Value at index " << index << " is: " << value << endl;
        } else {
            throw out_of_range("Index is out of bounds.");
        }
    } catch (const out_of_range& e) {
```

```
        cerr << "Exception caught: " << e.what() << endl;
    }

    return 0;
}
```

Output:

Array Contents: 1 2 3 4 5

Array Size: 5

Enter an index to access: 6

Exception caught: Index is out of bounds.