**What is C++:**

C++ is an object-oriented programming language. It is an extension to C programming. C++ programming language was developed in 1980 by **Bjarne Stroustrup** at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX..

**BjarneStroustrup** is known as the **founder of C++ language.**

**Object-Oriented Programming (OOPs)**

C++ supports the object-oriented programming, the four major pillar of object oriented programming used in C++ are:

**1. Inheritance 2.Polymorphism 3.Encapsulation 4.Abstraction**

**Usage of C++**

By the help of C++ programming language, we can develop different types of secured and robust applications:

* Window application
* Client-Server application
* Device drivers
* Embedded firmware etc

**C vs C++**

|  |  |  |
| --- | --- | --- |
| **No.** | **C** | **C++** |
| 1) | C follows the **procedural style programming.** | C++ is multi-paradigm. It supports both **procedural and object oriented.** |
| 2) | Data is less secured in C. | In C++, you can use modifiers for class members to make it inaccessible for outside users. |
| 3) | C follows the **top-down approach.** | C++ follows the **bottom-up approach.** |
| 4) | C does not support function overloading, Operator Overloading etc. OOP concepts. | C++ supports function overloading, Operator Overloading etc. OOP concepts.. |
| 5) | In C, you can't use functions in structure. | In C++, you can use functions in structure. |
| 6) | In C, **scanf() and printf()** are mainly used for input/output. | C++ mainly uses stream **cin and cout** to perform input and output operations. |

**C++ Features**:

**C++ is object oriented programming language. It provides a lot of features that are given below.**

* The main focus remains on data rather than procedures.
* Object-oriented programs are segmented into parts called objects.
* Data structures are designed to categorize the objects.
* Data member and functions are tied together as a data structure.
* Data can be hidden and cannot be accessed by external functions using access specifier .
* Objects can communicate among themselves using functions.
* New data and functions can be easily added anywhere within a program whenever required.
* Since this is an object-oriented programming language, it follows a bottom up approach, i.e. the execution of codes starts from the main which resides at the lower section and then based on the member function call the working is done from the classes.

**C++ Program:** Basically, a C++ program involves the following section:

Documentation

Pre-processor Statements

Global Declarations

The main () function{

Local Declarations

Program Statements & Expressions}

User Defined Functions

**Example : //program to display a line of text.**

#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

cout<<”welcome to c++ programming language”;

getch();

}

**Standard output stream (cout):**The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream **insertion operator** **(<<)** to display the output on a console

**Standard input stream (cin):**The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream **extraction operator (>>)** to read the input from a console.

**Standard end line (endl):**The **endl** is a predefined object of **ostream** class. It is used to insert a new line characters and flushes the stream.

**C++ Variable:** A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

**Data\_type  variable list;**

The example of declaring variable is given below: **1) int x; 2) float y; 3) char z;**

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

**1) int x=5,b=10;  //declaring 2 variable of integer type    2)float f=30.8;    3)char c='A';**

* A variable can have alphabets, digits and underscore.
* A variable name can start with alphabet and underscore only. It can't start with digit.
* No white space is allowed within variable name.
* A variable name must not be any reserved word or keyword e.g. chars, float etc.

**C++ Data Types : There are 4 types of data types in C++ language.**

|  |  |
| --- | --- |
| **Types** | **Data Types** |
| Basic Data Type | int, char, float, double, etc |
| Derived Data Type | array, pointer, etc |
| User Defined Data Type | structure |

C++ Keywords

A keyword is a reserved word. You cannot use it as a variable name, constant name etc.

Example: if, else,switch,case,do,while,for,class etc

**C++ Operators:** Operators are special type of functions that takes one or more arguments and produces a new value.

* Arithmetic Operators [+ ,- ,\* ,/ ,%]
* Relational Operators ]> ,>= ,< ,<= ,== ,!=]
* Logical Operators [&& , ||, ! ]
* Assignment Operator [=]
* Unary operator [++ ,--]
* Ternary or Conditional Operator [ ? , : ]
* sizeof operator [sizeof(nt)]
* pointer to member operator [ .\* and ->\*]
* **scope resolution** operator [ :: ]
* **address** operator [ & ]
* **new o**perator (memory allocation operator)
* **delete** operator(memory release operator)

**C++ Control Statement**

**if-else**

In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

* **if statement**
* **if-else statement**
* **if-else-if ladder**

**IF Statement** :The C++ if statement tests the condition. It is executed if condition is true.

**Syntax: if(condition) {**

**//code to be executed }**

Example: #include <iostream.h>

#include<conio.h>

void main ()

{

int num = 10;

if (num % 2 == 0) { cout<<"It is even number"; }

getch();

}

**if-else Statement:** The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

Syntax: if(condition) { //code if condition is true }

else { //code if condition is false }

Example:

#include <iostream.h>

#include<conio.h>

void main () {

int num;

cout<<"Enter a Number: ";

cin>>num;

if (num % 2 == 0) { cout<<"It is even number"<<endl; }

else { cout<<"It is odd number"<<endl; }

getch();

}

**if-else-if ladder :** statement executes one condition from multiple statements.

**Syntax:**

if(condition1){ //code to be executed if condition1 is true }

else if(condition2){ //code to be executed if condition2 is true }

else if(condition3){ //code to be executed if condition3 is true }

...

else{//code to be executed if all the conditions are false }

Example:

#include<iostream.h>

#include<conio.h>

void main()

{

int avg;

clrscr();

cout<<"Enter average marks obtained in 5 subjects :";

cin>>avg;

cout<<"Your Grade is ";

if(avg>80)

{ cout<<"A";}

else if(avg>60 && avg<=80)

{ cout<<"B";}

else if(avg>40 && avg<=60)

{ cout<<"C";}

else

{ cout<<"D";}

getch();

}

**switch:** The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

**Syntax:**  switch (expression)

{

case label1: statements ;

break;

case label2: statements;

break;

…..

……

default: staements;

break;

}

**Example**

**// Program to build a simple calculator using switch Statement**

#include <iostream.h>

#include<conio.h>

void main()

{

char op;

float num1, num2;

clrscr();

cout << "Enter an operator (+, -, \*, /): ";

cin >> op;

cout << "Enter two operands: ";

cin >> num1 >> num2;

switch (op)

{

case '+':

cout << num1 << " + " << num2 << " = " << num1+num2;

break;

case '-':

cout << num1 << " - " << num2 << " = " << num1-num2;

break;

case '\*':

cout << num1 << " \* " << num2 << " = " << num1\*num2;

break;

case '/':

cout << num1 << " / " << num2 << " = " << num1/num2;

break;

default:

// operator is doesn't match any case constant (+, -, \*, /)

cout << "Error! operator is not correct";

break;

}

getch();

}

**Looping**:

Loops are used in programming to repeat a specific block until some end condition is met. There are three types of loops in C++ programming:

1. for loop
2. [while loop](https://www.programiz.com/cpp-programming/do-while-loop)
3. [do...while loop](https://www.programiz.com/cpp-programming/do-while-loop)

**While loop:**

**Syntax: while(condition){**

**//code to be executed  }**

**Example**:

// C++ Program to compute factorial of a number

// Factorial of n = 1\*2\*3...\*n

#include <iostream.h>

#include<conio.h>

void main()

{

int number, i = 1, factorial = 1;

clrscr();

cout << "Enter a positive integer: ";

cin >> number;

while ( i <= number)

{

factorial \*= i; //factorial = factorial \* i;

++i;

}

cout<<"Factorial of "<< number <<" = "<< factorial;

getch();

}

**Do-While Loop** :

**Syntax:**  **do**

**{ //code to be executed**

**}while(condition);**

**Example** : //C++ Program to display natural numbers up to 10.

#include <iostream>

#include<conio.h>

void  main()

 {

     int i = 1;

          do{

              cout<<i<<"\n";

              i++;

          } while (i <= 10) ;

getch();

}

**Example:**

**Syntax:** for(initialization; condition; incr/decr) {

//code to be executed    }

**Example: //program to display your name for specified number of time**

#include <iostream.h>

#include<conio.h>

void main()

{

int time,i;

clrscr();

cout << "how many times you want to display your name";

cin >>time;

for(i=0;i<time;i++)

{

0 Cout<<”\n KEONICS”;

}

getch();

}

**C++ Functions**

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main().

**Advantages of using functions in a program**

* You can divide your program in logical blocks. ...
* Use of function avoids typing same pieces of code multiple times. ...
* Individual functions can be easily tested.
* In case of any modification in the code you can modify only the function without changing the structure of the program.

Types of Functions

There are two types of functions in C++ programming:

**1. Library Functions:** are the functions which are declared in the C++ header files such as ceil(x), floor(x), sqrt(x),sin(x) etc.

**2. User-defined functions:** are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.

**Declaration of a function:**

**Syntax:**

return\_type function\_name( parameter list ) {

body of the function}

**Example:**

#include<iostream.h>

#include<conio.h>

// function declaration

int max(int num1, int num2);//function Declaration

void main () {

// local variable declaration:

int a = 100;

int b = 200;

int ret;

// calling a function to get max value.

ret = max(a, b);

cout << "Max value is : " << ret << endl;

getch();

}

// function returning the max between two numbers

int max(int num1, int num2) {

// local variable declaration

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return (result);

}

**C++ Recursion**

When function is called within the same function, it is known as recursion in C++. The function which calls the same function, is known as recursive function.

**recursionfunction(){**

**recursionfunction(); //calling self function**

**}**

**Example**

#include<iostream>

#include<conio.h>

void main()

{

int factorial(int);

intfact,value;

cout<<"Enter any number: ";

cin>>value;

fact=factorial(value);

cout<<"Factorial of a number is: "<<fact<<endl;

getch();

}

int factorial(int n)

{

if(n<0)

return(-1); /\*Wrong value\*/

if(n==0)

return(1); /\*Terminating condition\*/

else

{

return(n\*factorial(n-1));

}

}

**Inline functions**

When a function is declared as inline, the compiler places a copy of the code of that specific function at each point where the function is called at compile time.

**Syntax:**

inline return\_type function\_name(arguments...)

{

//function\_code

//return\_value;

}

**Example:**

//Simple Inline Function Example Program in C++

#include<iostream.h>

#include<conio.h>

inline int square(int x);

void main() {

int a = 100, b = 200;

cout << "Simple Inline Function Example Program in C++\n";

cout << "\nSquare value for " << a << " is :" << square(a);

cout << "\nSquare value for " << b << " is :" << square(b);

getch();

}

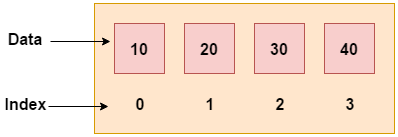
// Inline square function

inline int square(int x) {

return (x \* x);}

**C++ Arrays**

Array is a group of similar types of elements that have contiguous memory location.

In C++, array index starts from **0 to (n-1)** 

***Advantages of C++ Array***

* Code Optimization (less code)
* Random Access
* Easy to traverse data
* Easy to manipulate data
* Easy to sort data etc.

***Disadvantages of C++ Array ->*** Fixed size

***Array Types :***

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

***Single Dimensional Array:***

A one-dimensional array is a group of elements having the same data type and same name.

**Declaration: data\_type array\_name[array\_size];**

**Initialization: data\_type array\_name[array\_size]=comma\_separated\_element\_list};**

**Example :** // C++ One Dimensional Array

#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

int arr[5] = {1, 2, 3, 4, 5};

int i;

for(i=0; i<5; i++)

{ cout<<"arr["<<i<<"] = "<<arr[i]<<"\n";}

getch();

}

**C++ Passing Array to Function**

**functionname(arrayname); //passing array to function**

**Example:**

#include <iostream.h>

#include<conio.h>

void printArray(int arr[5]);

void main()

{

        int arr1[5] = { 10, 20, 30, 40, 50 };

        int arr2[5] = { 5, 15, 25, 35, 45 };

        printArray(arr1); //passing array to function

        printArray(arr2);

}

void printArray(int arr[5])

{

    cout << "Printing array elements:"<< endl;

    for (int i = 0; i < 5; i++)

    {

                   cout<<arr[i]<<"\n";

    }

getch();

}

C++ Multidimensional Arrays:

**Declare Two Dimensional Array->** data\_type array\_name[row\_size][column\_size];

**Initialization:**

data\_type array\_name[row\_size][column\_size] = { comma\_separated\_value\_list} };

**Example :**

// C++ Two Dimensional Array

#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

int arr[5][2] = { {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6} };

int i, j;

for(i=0; i<5; i++)

{

for(j=0; j<2; j++)

{

cout<<"arr["<<i<<"]["<<j<<"] = "<<arr[i][j]<<"\n";

}

}

getch();

}

**C++ OOPs Concepts**

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language. Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism etc.**

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.

**OOPs (Object Oriented Programming System):**

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

**Object**

Any entity that has state and behaviour is known as an object.

For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

**Collection of objects** is called class. It is a logical entity.

**Inheritance**

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**

When **one task is performed by different ways** i.e. known as polymorphism.

For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In C++, we use Function overloading and Function overriding to achieve polymorphism.

**Abstraction**

**Hiding internal details and showing functionality** is known as abstraction.

For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

**Encapsulation**

**Binding (or wrapping) code and data together into a single unit is known as encapsulation.**

For example: capsule, it is wrapped with different medicines.

***Advantage of OOPs over Procedure-oriented programming language***

* OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
* OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
* OOPs provide ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

**C++ Object and Class**

Since C++ is an object-oriented language, program is designed using objects and classes in C++.

**C++ Object:**

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

**C++ Class**

In C++, object is a group of similar objects.

**Syntax to create class:**

**Class class\_name**

**{**

**Data Members;**

**Methods;**

**}**

**Where class-|>keyword**

**Class Name->any valid identifies**

**Example:** class A

{

public:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

}

* **Private**, **Protected**, **Public** is called visibility labels.
* The members that are declared private can be accessed only from within the class.
* Public members can be accessed from outside the class also.
* In C++, data can be hidden by making it private.

**C++ Object and Class Example**

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

#include <iostream.h>

#include <conio.h>

class Student {

   public:

      int id;//data member (also instance variable)

      string name;//data member(also instance variable)

};

void main() {

    Student s1; //creating an object of Student

    s1.id = 201;

    s1.name = "Sonoo Jaiswal";

    cout<<s1.id<<endl;

    cout<<s1.name<<endl;

    getch();

}

**C++ Class Example: Initialize and Display data through method**

#include <iostream.h>

#include <conio.h>

class Student {

   public:

       int id;//data member (also instance variable)

      float per;//data member(also instance variable)

       void insert(int  i, float p)

        {

            id = i;

           per =p;

        }

       void display()

        {

            cout<<”id=”<<id<<"\n percentage= "<<per<<endl;

        }

};

void  main() {

    Student s1; //creating an object of Student

    Student s2; //creating an object of Student

    s1.insert(201, 69.5);

    s2.insert(202, 78.8);

    s1.display();

    s2.display();

    getch();

}

**C++ Class Example: Store and Display Employee Information**

#include <iostream.h>

#include <conio.h>

class Employee {

   public:

       int id;//data member (also instance variable)

       float salary; //data member(also instance variable)

       void insert(int i, float s)

        {

            id = i;

salary = s;

        }

       void display()

        {

            cout<<”id=”<<id<<"\nsalary= "<<salary<<endl;

        }

};

void main() {

    Employee e1; //creating an object of Employee

    Employee e2; //creating an object of Employee

    e1.insert(201,15000);

    e2.insert(202, 29000);

    e1.display();

    e2.display();

    getch(); }

**C++ Constructor**

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

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

1. Default constructor
2. Parameterized constructor

**C++ Default Constructor** :A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Example:

#include <iostream.h>

#include <conio.h>

class Employee

{

public:

Employee()

{

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

}

};

void main()

{

Employee e1; //creating an object of Employee

Employee e2;

getch(); }

**C++ Parameterized Constructor**: A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Example:***.***

#include <iostream.h>

#include <conio.h>

class Employee {

public:

int id;//data member (also instance variable)

string name;//data member(also instance variable)

float salary;

Employee(int i, string n, float s)

{

id = i;

name = n;

salary = s;

}

void display()

{

cout<<id<<" "<<name<<" "<<salary<<endl;

}

};

void main() {

Employee e1 =Employee(101, "Sonu", 89000); //creating an object of Employee

Employee e2=Employee(102, "Nakul", 59000);

e1.display();

e2.display();

getch();

}

**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 (~).**

**Example:** #include <iostream.h>

#include <conio.h>

class Employee

 {

   public:          Employee()

         {               cout<<"Constructor Invoked"<<endl;            }

         ~Employee()

         {                 cout<<"Destructor Invoked"<<endl;            }

};

void main()

{

    Employee e1; //creating an object of Employee

    Employee e2; //creating an object of Employee

    getch();

}

**C++ this Pointer**

In C++ programming, **this** is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

* It can be used **to pass current object as a parameter to another method.**
* It can be used **to refer current class instance variable.**
* It can be used **to declare indexers.**

**C++ this Pointer Example:**

#include <iostream.h>

#include <conio.h>

class Employee {

   public:

       int id; //data member (also instance variable)

       float salary;

       Employee(int id, float salary)

        {

             this->id = id;

            this->salary = salary;

        }

       void display()

        {

            cout<<”\nid”<<id<<”\nsalary=”<<salary<<endl;

        }

};

void main() {

    Employee e1 =Employee(101,  89000); //creating an object of Employee

    Employee e2=Employee(102, 59000); //creating an object of Employee

    e1.display();

    e2.display();

getch();}

**C++ friend function**

If a function is defined as a friend function in C++ then the protected and private data of a class can be accessed using the function. By using the keyword **friend** compiler knows the given function is a friend function .For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword **friend.**

**Declaration of friend function in C++**

**Syntax:**

class class\_name

{

    friend data\_type function\_name(arguments);

};

**Example:** #include<iostream.h>

#include<conio.h>

class base {

int val1, val2;

public:

void get() {

cout << "Enter two values:";

cin >> val1>>val2;

}

friend float mean(base ob);

};

float mean(base ob) {

return float(ob.val1 + ob.val2) / 2;

}

void main() {

clrscr();

base obj;

obj.get();

cout << "\n Mean value is : " << mean(obj);

getch(); }

**C++ Inheritance**

In C++, inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviours which are defined in other class.

In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

**Forms of inheritance in c++:**

|  |  |
| --- | --- |
| Single inheritance | When one class inherits another class, it is known as single level inheritance |
| Multiple inheritance | One derived class with several base classes is called multiple inheritance |
| Hierarchical inheritance | One class inherits more than one derived class is known as hierarchical inheritance |
| Multilevel inheritance | When one class inherits another class which is further inherited by another class, it is known as multi-level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes. |
| Hybrid inheritance | Combination of all forms of inheritance is known as hybrid inheritance. |

**Creation of derived class:**

**Syntax:**

class derived-class: visibility-mode base-class

{

Class body

}

**C++ Single Inheritance Example:**

#include<iostream.h>

#include<conio.h>

class emp {

public:

int eno;

char name[20], des[20];

void get() {

cout << "Enter the employee number:";

cin>>eno;

cout << "Enter the employee name:";

cin>>name;

cout << "Enter the designation:";

cin>>des;

}

};

class salary : public emp

{

float bp, hra, da, pf, np;

public:

void get1() {

cout << "Enter the basic pay:";

cin>>bp;

cout << "Enter the HRA:";

cin>>hra;

cout << "Enter the DA :";

cin>>da;

cout << "Enter the PF";

cin>>pf;

}

void calculate() { np = bp + hra + da - pf; }

void display() {

cout << eno << "\t" << name << "\t" << des << "\t" << bp << "\t" << hra << "\t" << da << "\t" << pf << "\t" << np << "\n";

}

};

void main()

{

int i, n;

char ch;

salary s[10];

clrscr();

cout << "Enter the number of employee:";

cin>>n;

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

{ s[i].get();

s[i].get1();

s[i].calculate();

}

cout << "\ne\_no \t e\_name\t des \t bp \t hra \t da \t pf \t np \n";

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

{ s[i].display(); }

getch();

}

**C++ Multiple Inheritance Example**

#include<iostream.h>

#include<conio.h>

class student {

private:

int rno, m1, m2;

public:

void get() {

cout << "Enter the Roll no :";

cin>>rno;

cout << "Enter the two marks :";

cin >> m1>>m2;

}

};

class sports

{

private:

int sm; // sm = Sports mark

public:

void getsm() {

cout << "\nEnter the sports mark :";

cin>>sm;

}

};

class statement : public student, public sports

{

int tot, avg;

public:

void display() {

tot = (m1 + m2 + sm);

avg = tot / 3;

cout << "\n\n\tRoll No : " << rno << "\n\tTotal : " << tot;

cout << "\n\tAverage : " << avg;

}

};

void main() {

clrscr();

statement obj;

obj.get();

obj.getsm();

obj.display();

getch();

}

**C++ Heirarchical Inheritance**

**Syntax:** Class A{

public void methodA()

{//Do Something }

}

Class B : public A

{

public void methodB()

{//Do Something }

}

Class C : public A

{

public void methodC()

{//Do Something } }

**C++ Multi Level Inheritance**

**Syntax:**  class A

{

... .. ...

};

class B: public A

{

... .. ...

};

class C: public B

{

... ... ...

};

**C++ Polymorphism**

The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a Greek word.

* **There are two types of polymorphism in C++:**
* **Compile time polymorphism:** It is achieved by function overloading and operator overloading which is also known as static binding or early binding.
* **Runtime polymorphism:** It is achieved by method overriding which is also known as dynamic binding or late binding.

**C++ Overloading (Function and Operator)**

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

Types of overloading in C++ are:

* **Function overloading**
* **Operators overloading**
* **Function Overloading**

Having two or more function with same name but different in parameters, is known as function overloading in C++.The **advantage** of Function overloading is that it increases the readability of the program because you don't need to use different names for same action.

**Example:**

* int test() { }
* int test(int a) { }
* float test(double a) { }
* int test(int a, double b) { }

**Function Overloading Example**

#include <iostream>

class Cal {

     public:   int add(int a,int b)

{    return (a + b);     }

 int add(int a, int b, int c)

 {     return (a + b + c);    }

};

void  main(void)

{

    Cal C;

    cout<<C.add(10, 20)<<endl;

    cout<<C.add(12, 20, 23);

   getch();

}

* **Operators Overloading**

Operator overloading is used to overload or redefine most of the operators available in C++. It is used to perform operation on user define data type.

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

**Following are list of operators, which cannot be overloaded:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **::**  Scope resolution operator | **.\***  Pointer to member operator | **.**  Dot membership operatot | **?:**  Conditional Operator | Sizeof() operator |

**Example:**

#include <iostream.h>

#include<conio.h>

class Test

{

   private:   int num;

   public: void getdata(int x ){num=x;}

   void operator ++()

        {

           num = num+2;

        }

        void Print() {   cout<<"The Count is: "<<num;       }

};

void  main()

{

    Test tt;

clrscr( );

tt.getdata(20);

tt.Print();

    ++tt;  // calling of a function "void operator ++()"

Cout<<”\nafter calling operator function”;

    tt.Print();

    getch( );

}

C++ Function Overriding

If derived class defines same function as defined in its base class, it is known as function overriding in C++. It is used to achieve runtime polymorphism.

**Example:**

#include <iostream.h>

#include <conio.h>

class Animal {

     public:

void eat(){  cout<<"Eating...";       }

};

class Dog: public Animal

 {

 public:

 void eat( )  {      cout<<"Eating bread...";      }

};

void  main()

 {

   Dog d = Dog();

clrscr( );

   d.eat();

   getch();

}

**C++ virtual function**

C++ virtual function is a member function in base class that you redefine in a derived class. It is declare using the virtual keyword. It is used to tell the compiler to perform **dynamic linkage or late binding** on the function.

Late binding or Dynamic linkage:In late binding function call is resolved during runtime. Therefore compiler determines the type of object at runtime, and then binds the function call.

**virtual function Example:**

#include <iostream.h>

#include <conio.h>

class A

{

 public:  virtual void display()

 {

  cout << "Base class is invoked"<<endl;

 }

};

class B:public A

{

 public:   void display()

 {

  cout << "Derived Class is invoked"<<endl;

 }

};

void main()

{

 A \* a;    //pointer of base class

 B  b;     //object of derived class

 a = &b;

 a->display();   //Late Binding occurs

getch();

}

**Pure Virtual Function:** A pure virtual function is specified by placing "= 0" in its declaration. Its implementation must be provided by derived classes.

**Example:** #include <iostream.h>

#include <conio.h>

 class Shape

{

    public:

    virtual void draw()=0;    //pure virtual function

};

 class Rectangle : Shape

{

    public:

     void draw()

    {

        cout < <"drawing rectangle..." < <endl;

    }

};

class Circle : Shape

{

    public:

     void draw()

    {

        cout <<"drawing circle..." < <endl;

    }

};

void  main( )

{

    Rectangle rec;

    Circle cir;

    rec.draw();

    cir.draw();

   getch();

}

**C++ Namespaces**

Namespaces in C++ are used to organize too many classes so that it can be easy to handle the application. For accessing the class of a namespace, we need to use namespace name::class name. We can use **using** keyword so that we don't have to use complete name all the time.

In C++, global namespace is the root namespace. The global::std will always refer to the namespace "std" of C++ Framework.

**C++ namespace Example**

Let's see the simple example of namespace which include variable and functions.

#include <iostream.h>

#include <conio.h>

namespace First {

    void sayHello() {

        cout<<"Hello First Namespace"<<endl;

    }

}

namespace Second  {

       void sayHello() {

           cout<<"Hello Second Namespace"<<endl;

       }

}

void main()

{

 First::sayHello();

 Second::sayHello();

getch();}

C++ Exception Handling

Exception Handling in C++ is a process to handle runtime errors. We perform exception handling so the normal flow of the application can be maintained even after runtime errors. In C++, exception is an event or object which is thrown at runtime. All exceptions are derived from std::exception class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

**C++ Exception Handling Keywords:** we use 3 keywords to perform exception handling:

* **throw** − A program throws an exception when a problem shows up. This is done using a **throw** keyword.
* **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
* **try** − A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

try {

// protected code

//throw statements

} catch( ExceptionName e ) {

// code to handle ExceptionName exception

}

**Example:**

#include <iostream.h>

#include<conio.h>

double division(int a, int b)

{

if( b == 0 ) {

throw "Division by zero condition!";

}

return (a/b);

}

void main ()

{

int x = 50;

int y = 0;

double z = 0;

try {

z = division(x, y);

cout << z << endl;

} catch (const char\* msg) {

cerr << msg << endl;

}

getch();

}

we are raising an exception of type **const char\***, so while catching this exception, we have to use const char\* in catch block. If we compile and run above code, this would produce the following result −

Division by zero condition!