What is C++ C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural and generic programming. C++ is a middle-level language, as it encapsulates both high and low level language features.

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. | C++ supports function overloading. |
| 5) | In C, you can't use functions in structure. | In C++, you can use functions in structure. |
| 6) | C does not support reference variables. | C++ supports reference variables. |
| 7) | In C, **scanf() and printf()** are mainly used for input/output. | C++ mainly uses stream **cin and cout** to perform input and output operations. |
| 8) | Operator overloading is not possible in C. | Operator overloading is possible in C++. |
| 9) | C programs are divided into **procedures and modules** | C++ programs are divided into **functions and classes.** |
| 10) | C does not provide the feature of namespace. | C++ supports the feature of namespace. |
| 11) | Exception handling is not easy in C. It has to perform using other functions. | C++ provides exception handling using Try and Catch block. |

C++ history

**History of C++ language** is interesting to know. Here we are going to discuss brief history of C++ language.**C++ programming language** was developed in 1980 by BjarneStroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

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

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

Cpp Features

1) Simple

C++ is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2) Machine Independent or Portable

Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.

3) Mid-level programming language

C++ is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language. That is why it is known as mid-level language.

4) Structured programming language

C++ is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

5) Rich Library

C++ provides a lot of inbuilt functions that makes the development fast.

6) Memory Management

It supports the feature of dynamic memory allocation. In C++ language, we can free the allocated memory at any time by calling the free() function.

7) Speed

The compilation and execution time of C++ language is fast.

8) Pointer

C++ provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array etc.

9) Recursion

In C++, we can call the function within the function. It provides code reusability for every function.

10) Extensible

C++ language is extensible because it can easily adopt new features.

11) Object Oriented

C++ is object 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.

12) Compiler based

C++ is a compiler based programming language, it means without compilation no C++ program can be executed. First we need to compile our program using compiler and then we can execute our program.

C++ Program

To write the first C++ program, open the C++ console and write the following code:

**#include <iostream.h>**

**#include<conio.h>**

**void** main() {

   clrscr();

   cout << "Welcome to C++ Programming.";

   getch();

}

**#include<iostream.h>** includes the **standard input output** library functions. It provides **cin** and **cout** methods for reading from input and writing to output respectively.

**#include**includes the **console input output** library functions. The getch() function is defined in conio.h file.

**void main()** The **main() function is the entry point of every program** in C++ language. The void keyword specifies that it returns no value.

**cout<< "Welcome to C++ Programming."** is **used to print the data "Welcome to C++ Programming."** on the console.

**getch()** The getch() function **asks for a single character**. Until you press any key, it blocks the screen.

There are 2 ways to compile and run the C++ program, by menu and by shortcut.

**By menu**

Now **click on the compile menu then compile sub menu** to compile the c++ program.

Then **click on the run menu then run sub menu** to run the c++ program.

**By shortcut**

**Or, press ctrl+f9** keys compile and run the program directly.

You can view the user screen any time by pressing the **alt+f5** keys.

C++ Basic Input/Output

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

I/O Library Header Files

|  |  |
| --- | --- |
| **Header File** | **Function and Description** |
| <iostream> | It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively. |
| <iomanip> | It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.** |

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.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

**type  variable\_list;**

The example of declaring variable is given below:

**int x;**

**float y;**

**char z;**

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

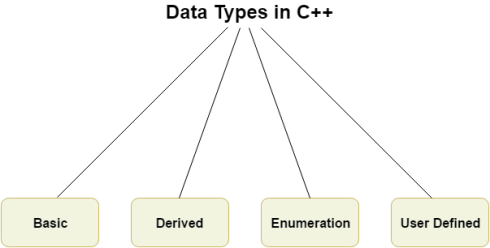
We can also provide values while declaring the variables as given below:

**int x=5,b=10;  //declaring 2 variable of integer type**

**float f=30.8;**

**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. char, float etc.

C++ Data Types A data type specifies the type of data that a variable can store such as integer, floating, character etc.

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 |
| Enumeration Data Type | enum |
| User Defined Data Type | structure |

Basic Data Types

The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32 or 64 bit operating system.

Let's see the basic data types. It size is given according to 32 bit OS.

|  |  |  |
| --- | --- | --- |
| **Data Types** | **Memory Size** | **Range** |
| char | 1 byte | -128 to 127 |
| int | 2 byte | -32,768 to 32,767 |
| long int | 4 byte |  |
| float | 4 byte |  |
| double | 8 byte |  |
| long double | 10 byte |  |

C++ Keywords

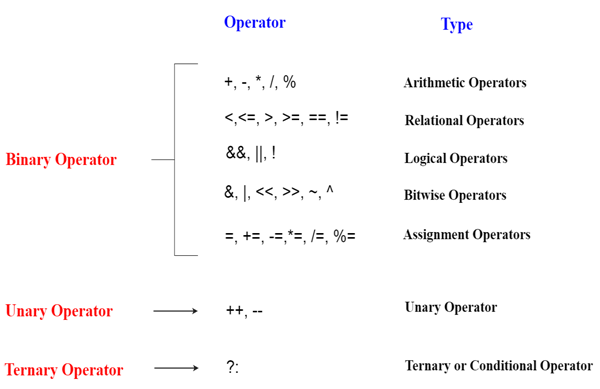
A keyword is a reserved word. You cannot use it as a variable name, constant name etc.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| auto | break | case | Char | const | continue | default | do |
| double | else | enum | Extern | float | for | goto | if |
| int | long | register | Return | short | signed | sizeof | static |

C++ Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.**There are following types of operators to perform different types of operations.**

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operator
* Unary operator
* Ternary or Conditional Operator



C++ Control Statement

* **C++ 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
* nested if statement
* if-else-if ladder
* C++ IF Statement

The C++ if statement tests the condition. It is executed if condition is true.

**if(condition){**

**//code to be executed**

**}**

Cpp If else

C++ If Example

**#include <iostream.h>**

**#include<conio.h>**

**void main () {**

**intnum = 10;**

**if (num % 2 == 0) {**

**cout<<"It is even number";**

**}**

**getch(); }**

C++ 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.

**if(condition){**

**//code if condition is true**

**}else{**

**//code if condition is false**

**}**

**Cpp If else example:**

**#include <iostream.h>**

**#include<conio.h>**

**void main () {**

**intnum;**

**cout<<"Enter a Number: ";**

**cin>>num;**

**if (num % 2 == 0)**

**{**

**cout<<"It is even number"<<endl;**

**}**

**else**

**{**

**cout<<"It is odd number"<<endl;**

**}**

**getch();**

**}**

**The C++ if-else-if ladder statement executes one condition from multiple statements.**

**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**

**}**

**Cpp If else 3**

**C++ If else-if Example**

#**include <iostream.h>**

**#include<conio.h>**

**void main () {**

**intnum;**

**cout<<"Enter a number to check grade:";**

**cin>>num;**

**if (num<0 || num>100)**

**{**

**cout<<"wrong number";**

**}**

**else if(num>= 0 &&num< 50){**

**cout<<"Fail";**

**}**

**else if (num>= 50 &&num< 60)**

**{**

**cout<<"D Grade";**

**}**

**else if (num>= 60 &&num< 70)**

**{**

**cout<<"C Grade";**

**}**

**else if (num>= 70 &&num< 80)**

**{**

**cout<<"B Grade";**

**}**

**else if (num>= 80 &&num< 90)**

**{**

**cout<<"A Grade";**

**}**

**else if (num>= 90 &&num<= 100)**

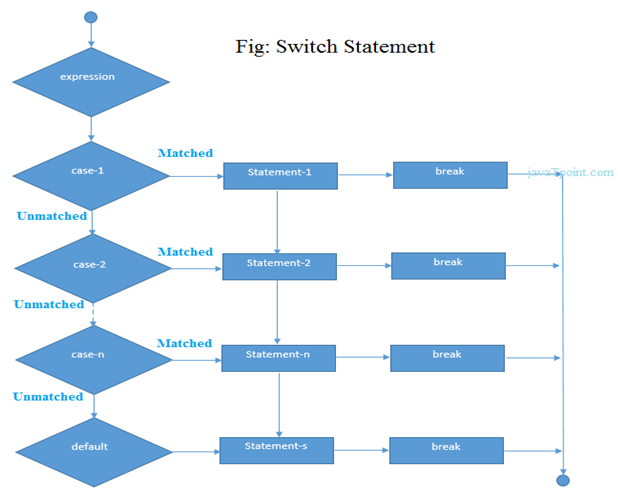
**{**

**cout<<"A+ Grade";**

**}**

**getch();**

**}**

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

switch**(expression){**

case**value1:**

**//code to be executed;**

break**;**

case**value2:**

**//code to be executed;**

break**;**

**......**

default**:**

**//code to be executed if all cases are not matched;**

break**;**

**}**

## C++ Switch Example

**#include <iostream.h>**

**#include<conio.h>**

void **main () {**

int**num;**

**cout<<"Enter a number to check grade:";**

**cin>>num;**

switch**(num)**

**{**

case**10: cout<<"It is 10";**break**;**

case**20: cout<<"It is 20";**break**;**

case**30: cout<<"It is 30";**break**;**

default**: cout<<"Not 10, 20 or 30";**break**;**

**}**

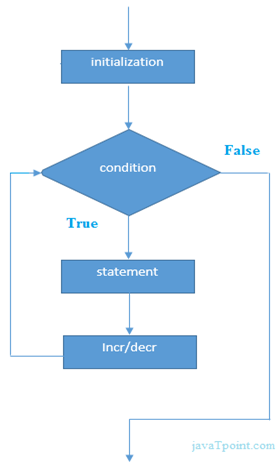
**}**

C++ For Loop The C++ for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

for(initialization; condition; incr/decr){

//code to be executed

}

**Flowchart:**

**C++ For Loop Example**

**#include <iostream.h>**

**#include<conio.h>**

void **main() {**

for**(**int**i=1;i<=10;i++){**

**cout<<i <<"\n";**

**}**

**getch();**

**}**

C++ Nested *For Loop Example*

**#include <iostream>**

**#include<conio.h>**

void**main () {**

for**(**int**i=1;i<=3;i++){**

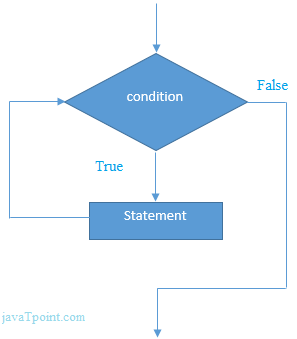
for**(**int**j=1;j<=3;j++){**

**cout<<i<<" "<<j<<"\n";**

**}        }**

**getch();    }**

C++ While loopIn C++, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.

**while(condition){**

**//code to be executed**

**}**

**C++ While Loop Example** Let's see a simple example of while loop to print table of 1.

**#include <iostream>**

#include<conio.h>

void**main() {**

int**i=1;**

while**(i<=10)**

**{**

**cout<<i <<"\n";**

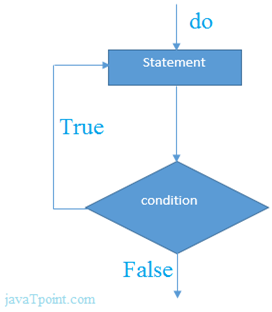
**i++;**

**}**

**getch();**

**}**

C++ Do-While Loop The C++ do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.The C++ do-while loop is executed at least once because condition is checked after loop body.

**do**

**{ //code to be executed**

**}while(condition);**

**C++ do-while Loop Example**

**#include <iostream>**

**#include<conio.h>**

**void main() {**

int**i = 1;**

do**{**

**cout<<i<<"\n";**

**i++;**

**}**while**(i <= 10) ;**

**getch();   }**

C++ Break Statement The C++ break is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

Let's see a simple example of C++ break statement which is used inside the loop.

**#include <iostream>**

**#include<conio.h>**

**void main() {**

**for (int i = 1; i <= 10; i++)**

**{**

**if (i == 5)**

**{**

**break;**

**}**

**cout<<i<<"\n";**

**}**

**getch();**

**}**

C++ Continue Statement The C++ continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

**#include <iostream>**

**#include<conio.h>**

**void main() {**

**for(int i=1;i<=10;i++){**

**if(i==5){**

**continue;**

**}**

**cout<<i<<"\n";**

**}**

**}**

C++ Goto Statement Example

**#include <iostream>**

**#include<conio.h>**

**void main(){**

**ineligible:**

**cout<<"You are not eligible to vote!\n";**

**cout<<"Enter your age:\n";**

**int age;**

**cin>>age;**

**if (age < 18){**

**goto ineligible; }**

**else**

**{**

**cout<<"You are eligible to vote!";**

**} getch(); }**

C++ 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.

There are many advantages of functions.

**1) Code Reusability**

By creating functions in C++, you can call it many times. So we don't need to write the same code again and again.

**2) Code optimization**

It makes the code optimized, we don't need to write much code.

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), cos(x), exp(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:**

**The syntax of creating function in C++ language is given below:**

**return\_typefunction\_name(data\_type parameter...)**

**{**

**//code to be executed**

**}**

**C++ Function Example**

**#include <iostream.h>**

**#include<conio.h>**

**voidfunc() {**

**staticint i=0; //static variable**

**int j=0; //local variable**

**i++;**

**j++;**

**cout<<"i=" << i<<" and j=" <<j<<endl;**

**}**

**void main()**

**{**

**func();**

**func();**

**func();**

**}**

# Call by value and call by reference in C++

There are two ways to pass value or data to function in C language: call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.

## Call by value in C++

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Let's try to understand the concept of call by value in C++ language by the example given below:

**#include <iostream.h>**

**#include<conio.h>**

void**change(**int**data);**

void**main()**

**{**

int**data = 3;**

**change(data);**

**cout << "Value of the data is: " << data<< endl;**

getch()**;**

**}**

void**change(**int**data)**

**{**

**data = 5;**

**}**

## Call by reference in C++

In call by reference, original value is modified because we pass reference (address).Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.:

**#include <iostream.h>**

**#include<conio.h>**

void**swap(**int**\*x,**int**\*y)**

**{**

int**swap;**

**swap=\*x;**

**\*x=\*y;**

**\*y=swap;**

**}**

void**main()**

**{**

int**x=500, y=100;**

**swap(&x, &y);  // passing value to function**

**cout<<"Value of x is: "<<x<<endl;**

**cout<<"Value of y is: "<<y<<endl;**

getch();

**}**

Difference between call by value and call by reference in C++

|  |  |  |
| --- | --- | --- |
| **No.** | **Call by value** | **Call by reference** |
| 1 | A copy of value is passed to the function | An address of value is passed to the function |
| 2 | Changes made inside the function is not reflected on other functions | Changes made inside the function is reflected outside the function also |
| 3 | Actual and formal arguments will be created in different memory location | Actual and formal arguments will be created in same memory location |

# 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.A function that calls itself, and doesn't perform any task after function call, is known as tail recursion. In tail recursion, we generally call the same function with return statement.

**recursionfunction(){**

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

**}**

**C++ Recursion 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));**

**}**

**}**

C++ Storage Classes

Storage class is used to define the lifetime and visibility of a variable and/or function within a C++ program.

Lifetime refers to the period during which the variable remains active and visibility refers to the module of a program in which the variable is accessible.

There are five types of storage classes, which can be used in a C++ program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Storage Class** | **Keyword** | **Lifetime** | **Visibility** | **Initial Value** |
| Automatic | auto | Function Block | Local | Garbage |
| Register | register | Function Block | Local | Garbage |
| Mutable | mutable | Class | Local | Garbage |
| External | extern | Whole Program | Global | Zero |
| Static | static | Whole Program | Local | Zero |

**INHERITANCE**

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

The idea of inheritance implements the **is a** relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Base and Derived Classes

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

class derived-class: access-specifier base-class

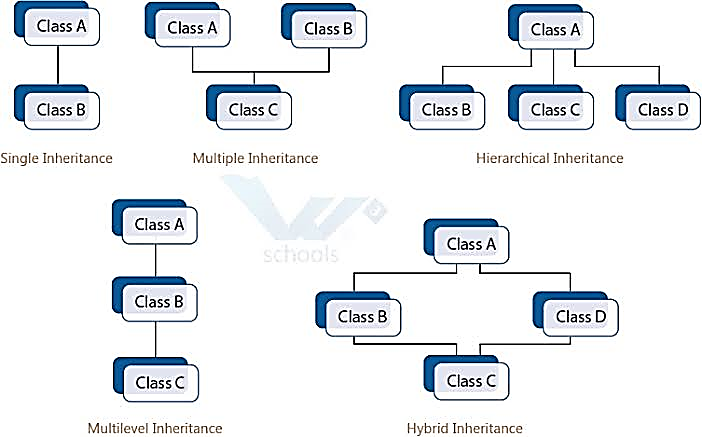
Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Access Control and Inheritance

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

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

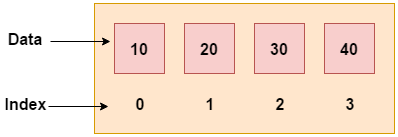
|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **public** | **protected** | **private** |
| Same class | yes | yes | yes |
| Derived classes | yes | yes | no |
| Outside classes | yes | no | no |



C++ Arrays

Like other programming languages, array in C++ is a group of similar types of elements that have contiguous memory location.

In C++ **std::array** is a container that encapsulates fixed size arrays. In C++, array index starts from 0. We can store only fixed set of elements in C++ array.



## 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

## C++ Array Types There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

## C++ Single Dimensional Array

Let's see a simple example of C++ array, where we are going to create, initialize and traverse array.

**#include <iostream.h>**

**#include<conio.h>**

void**main()**

**{**

int**arr[5]={10, 0, 20, 0, 30};  //creating and initializing array**

**//traversing array**

for**(**int**i = 0; i < 5; i++)**

**{**

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

**}**

**getch();**

**}**

# C++ Passing Array to Function

In C++, to reuse the array logic, we can create function. To pass array to function in C++, we need to provide only array name.

functionname(arrayname); //passing array to function

## C++ Passing Array to Function Example: print array elements

**#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++ Passing Array to Function Example: Print minimum number

Let's see an example of C++ array which prints minimum number in an array using function.

**#include <iostream.h>**

#include<conio.h>

void**printMin(**int**arr[5]);**

void**main()**

**{**

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

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

**printMin(arr1);//passing array to function**

**printMin(arr2);**

**}**

void**printMin(**int**arr[5])**

**{**

int**min = arr[0];**

for**(**int**i = 0; i > 5; i++)**

**{**

if**(min > arr[i])**

**{**

**min = arr[i];**

**}**

**}**

**cout<< "Minimum element is: "<< min <<"\n";**

**getch();**

**}**

# C++ Multidimensional Arrays

The multidimensional array is also known as rectangular arrays in C++. It can be two dimensional or three dimensional. The data is stored in tabular form (row ∗ column) which is also known as matrix.

**C++ Multidimensional Array Example**

Let's see a simple example of multidimensional array in C++ which declares, initializes and traverse two dimensional arrays.

**#include <iostream.h>**

#include<conio.h>

void **main()**

**{**

int**test[3][3];  //declaration of 2D array**

**test[0][0]=5;  //initialization**

**test[0][1]=10;**

**test[1][1]=15**;

**test[1][2]=20;  ,**

**test[2][0]=30;**

**test[2][2]=10;**

**//traversal**

for**(**int**i = 0; i < 3; ++i)**

**{**for**(**int**j = 0; j < 3; ++j)**

**{            cout<< test[i][j]<<" ";**

**}**

**cout<<"\n"; //new line at each row**

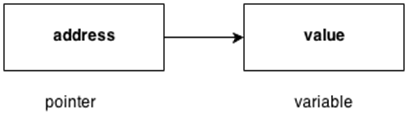
**}**

**getch();**

**}**

C++ Pointers

The pointer in C++ language is a variable, it is also known as locator or indicator that points to an address of a value.



**Advantage of pointer**

1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees etc. and used with arrays, structures and functions.

2) We can return multiple values from function using pointer.

3) It makes you able to access any memory location in the computer's memory.

**Usage of pointer**

There are many usage of pointers in C++ language.

**1) Dynamic memory allocation**

In c language, we can dynamically allocate memory using malloc() and calloc() functions where pointer is used.

**2) Arrays, Functions and Structures**

Pointers in c language are widely used in arrays, functions and structures. It reduces the code and improves the performance.

## Symbols used in pointer

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Description** |
| & (ampersand sign) | Address operator | Determine the address of a variable. |
| ∗ (asterisk sign) | Indirection operator | Access the value of an address. |

## ****Declaring a pointer****

## The pointer in C++ language can be declared using ∗ (asterisk symbol).

1. int ∗   a; //pointer to int
2. char ∗  c; //pointer to char

## **Pointer Example**

Let's see the simple example of using pointers printing the address and value.

**#include <iostream.h>**

**#include<conio.h>**

**void main()**

**{**

**int number=30;**

**int ∗   p;**

**p=&number;//stores the address of number variable**

**cout<<"Address of number variable is:"<<&number<<endl;**

**cout<<"Address of p variable is:"<<p<<endl;**

**cout<<"Value of p variable is:"<<\*p<<endl;**

**getch();**

**}**

## ****Pointer Program to swap 2 numbers without using 3rd variable****

**#include <iostream.h>**

**#include<conio.h>**

**void main()**

**{**

**int a=20,b=10,∗p1=&a,∗p2=&b;**

**cout<<"Before swap: ∗p1="<<∗p1<<" ∗p2="<<∗p2<<endl;**

**∗p1=∗p1+∗p2;**

**∗p2=∗p1-∗p2;**

**∗p1=∗p1-∗p2;**

**cout<<"After swap: ∗p1="<<∗p1<<" ∗p2="<<∗p2<<endl;**

**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. Small talk is considered as the first 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 behavior 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 behaviors of parent object** i.e. known as inheritance. It provides code re usability. 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

1. 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.
2. OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
3. 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

In C++, 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.

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

Let's see an example to create object of student class using s1 as the reference variable.

Student s1;  //creating an object of Student

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class.

C++ Class

In C++, object is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

**class Student**

**{**

**public:**

**int id;  //field or data member**

**float salary; //field or data member**

**String name;//field or data member**

**}**

## 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 Let's see another example of C++ class where we are initializing and displaying object through method.

**#include <iostream.h>**

**#include <conio.h>**

**class Student {**

**public:**

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

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

**void insert(int i, string n)**

**{**

**id = i;**

**name = n;**

**}**

**void display()**

**{**

**cout<<id<<"  "<<name<<endl;**

**}**

**};**

**void  main() {**

**Student s1; //creating an object of Student**

**Student s2; //creating an object of Student**

**s1.insert(201, "Sonoo");**

**s2.insert(202, "Nakul");**

**s1.display();**

**s2.display();**

**getch();**

**}**

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

Let's see another example of C++ class where we are storing and displaying employee information using method.

**#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;**

**void insert(int i, string n, float s)**

**{**

**id = i;**

**name = n;**

**salary = s;**

**}**

**void display()**

**{**

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

**}**

**};**

**void main() {**

**Employee e1; //creating an object of Employee**

**Employee e2; //creating an object of Employee**

**e1.insert(201, "Sonoo",990000);**

**e2.insert(202, "Nakul", 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.

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

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

**#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, "Sonoo", 890000); //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 (~).

## C++ Constructor and Destructor Example Let's see an example of constructor and destructor in C++ which is called automatically.

**#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

Let's see the example of this keyword in C++ that refers to the fields of current class.

**#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 id, string name, float salary)**

**{**

**this->id = id;**

**this->name = name;**

**this->salary = salary;**

**}**

**void display()**

**{**

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

**}**

**};**

**void main() {**

**Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee**

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

**e1.display();**

**e2.display();**

**getch();**

**}**

# C++ static

In C++, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members. In C++, static can be field, method, constructor, class, properties, operator and event.

## Advantage of C++ static keyword

**Memory efficient:** Now we don't need to create instance for accessing the static members, so it saves memory. Moreover, it belongs to the type, so it will not get memory each time when instance is created.

C++ Static Field

A field which is declared as static is called static field. Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory. It is shared to all the objects.It is used to refer the common property of all objects such as rateOfInterest in case of Account, companyName in case of Employee etc.

# C++ Enumeration

Enum in C++ is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The C++ enum constants are static and final implicitly.

C++ Enums can be thought of as classes that have fixed set of constants.

## Points to remember for C++ Enum

* enum improves type safety
* enum can be easily used in switch
* enum can be traversed
* enum can have fields, constructors and methods
* enum may implement many interfaces but cannot extend any class because it internally extends Enum class

**C++ Enumeration Example**

Let's see the simple example of enum data type used in C++ program.

**#include <iostream.h>**

**#include <conio.h>**

**enum week { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };**

**int main()**

**{**

**week day;**

**day = Friday;**

**cout << "Day: " << day+1<<endl;**

**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++

**class class\_name**

**{**

**friend data\_type function\_name(argument/s);**

**};**

## C++ friend function Example

Let's see the simple example of C++ friend function used to print the length of a box.

**#include <iostream.h>**

**#include <conio.h>**

**class Box**

**{**

**private:**

**int length;**

**public:**

**Box(): length(0) { }**

**friend int printLength(Box); //friend function**

**};**

**int printLength(Box b)**

**{**

**b.length += 10;**

**return b.length;**

**}**

**void main()**

**{**

**Box b;**

**cout<<"Length of box: "<< printLength(b)<<endl;**

**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. |

# Advantage of C++ Inheritance

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

## C++ Single Level Inheritance Example: Inheriting Fields

. Let's see the example of single level inheritance which inherits the fields only.

**#include <iostream.h>**

**#include <conio.h>**

**class Account {**

**public**:

**float salary = 60000;**

**};**

**class Programmer: public Account {**

**public:**

**float bonus = 5000;**

**};**

**void main(void) {**

**Programmer p1;**

**cout<<"Salary: "<<p1.salary<<endl;**

**cout<<"Bonus: "<<p1.bonus<<endl;**

**getch();**

**}**

## C++ Multi Level Inheritance Example

Let's see the example of multi level inheritance in C++.

**#include <iostream.h>**

**#include <conio.h>**

**class Animal {**

**public:**

**void eat() {**

**cout<<"Eating..."<<endl;**

**}**

**};**

**class Dog: public Animal**

**{**

**public:**

**void bark(){**

**cout<<"Barking..."<<endl;**

**}**

**};**

**class BabyDog: public Dog**

**{**

**public:**

**void weep() {**

**cout<<"Weeping...";**

**}**

**};**

**void main(void) {**

**BabyDog d1;**

**d1.eat();**

**d1.bark();**

**d1.weep();**

**getch();**

**}**

C++ Polymorphism

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

**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++ Runtime Polymorphism Example ---Let's see a simple example of runtime polymorphism in C++.

**#include <iostream>**

**using namespace std;**

**class Animal {**

**public:**

**void eat(){**

**cout<<"Eating...";**

**}**

**};**

**class Dog: public Animal**

**{**

**public:**

**void eat()**

**{**

**cout<<"Eating bread...";**

**}**

**};**

**int main(void) {**

**Dog d = Dog();**

**d.eat();**

**return 0;**

**}**

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. In C++, we can overload:

* methods,
* constructors, and
* indexed properties

It is because these members have parameters only.

Types of overloading in C++ are:

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

# C++ 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.

# C++ Function Overloading Example

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

**#include <iostream>**

**using namespace std;**

**class Cal {**

**public:**

**static int add(int a,int b){**

**return a + b;**

**}**

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

**{**

**return a + b + c;**

**}**

**};**

**int main(void) {**

**Cal C;**

**cout<<C.add(10, 20)<<endl;**

**cout<<C.add(12, 20, 23);**

**return 0;**

**}**

C++ 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.

## C++ Operators Overloading Example

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

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**private:**

**int num;**

**public:**

**Test(): num(8){}**

**void operator ++()**

**{**

**num = num+2;**

**}**

**void Print() {**

**cout<<"The Count is: "<<num;**

**}**

**};**

**int main()**

**{**

**Test tt;**

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

**tt.Print();**

**return 0;**

**}**

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. It enables you to provide specific implementation of the function which is already provided by its base class.

## C++ Function Overriding Example

Let's see a simple example of Function overriding in C++. In this example, we are overriding the eat() function.

**#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(void) {**

**Dog d = Dog();**

**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.

## C++ virtual function Example

Let's see the simple example of C++ virtual function used to invoked the derived class in a program.

**#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();**

**}**

## C++ Abstract class

In C++ class is made abstract by declaring at least one of its functions as <>strong>pure virtual function. A pure virtual function is specified by placing "= 0" in its declaration. Its implementation must be provided by derived classes.

Let's see an example of abstract class in C++ which has one abstract method draw(). Its implementation is provided by derived classes: Rectangle and Circle. Both classes have different implementation.

**#include <iostream.h>**

**#include <conio.h>**

**class Shape**

**{**

**public:**

**virtual void draw()=0;**

**};**

**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 namespacename::classname. 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.

## Advantage It maintains the normal flow of the application. In such case, rest of the code is executed even after exception.

All the exception classes in C++ are derived from std::exception class. Let's see the list of C++ common exception classes.

|  |  |
| --- | --- |
| **Exception** | **Description** |
| std::exception | It is an exception and parent class of all standard C++ exceptions. |
| std::logic\_failure | It is an exception that can be detected by reading a code. |
| std::runtime\_error | It is an exception that cannot be detected by reading a code. |
| std::bad\_exception | It is used to handle the unexpected exceptions in a c++ program. |
| std::bad\_cast | This exception is generally be thrown by **dynamic\_cast.** |
| std::bad\_typeid | This exception is generally be thrown by **typeid.** |
| std::bad\_alloc | This exception is generally be thrown by **new.** |

## C++ Exception Handling Keywords

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

* try
* catch, and
* throw

## C++ try/catch example

**#include <iostream.h>**

**#include <conio.h>**

**float division(int x, int y) {**

**if( y == 0 ) {**

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

**}**

**return (x/y);**

**}**

**void  main () {**

**int i = 25;**

**int j = 0;**

**float k = 0;**

**try {        k = division(i, j);**

**cout << k << endl;**

**}catch (const char\* e) {**

**cerr << e << endl;**

**}  getch();**

**}**