1. Simple hello world program

here we write iostream which we have to include to access the cout method and syntax is much similar to c language

#include <iostream>

using *namespace* std;

*int* main() {

  cout << "Hello World!";

  return 0;

}

2. Comment

Comment in c++ is similar to c language // for single line comment /\*\*/ for the multi line comment.

#include <iostream>

using *namespace* std;

*int* main() {

  cout << "Student bread";//This is a single line comment

  /\*cout << "Student bread"; This is a double line comment

  cout << "Student bread";\*/

  return 0;

}

3. Data types

Here we have same data type as a c language which are int float double string and many more available

#include <iostream>

using *namespace* std;

*int* main() {

*int* a=15;

*float* b=25.25;

*double* c=25;

  string d="Student bread";

  cout<< a<<"\n";

  cout<< b<<"\n";

  cout<< c<<"\n";

  cout<< d<<"\n";

  return 0;

}

4. User input

In c++ user input can taken by writing cout function and variable name

#include <iostream>

using *namespace* std;

*int* main() {

*int* a;

  cout<<"Enter a: ";

  cin>>a;

  cout<<"A is "<<a;

  return 0;

}

5. Operator

5.1 Arithmetic operator

+ operator will add, - operator will subtract , \* operator will multiply, / will divide, % operator will modulo means gives the remainder

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a=20,b=10;

  cout<<"Your + operator is "<<a+b<<"\n";

  cout<<"Your - operator is "<<a-b<<"\n";

  cout<<"Your \* operator is "<<a\*b<<"\n";

  cout<<"Your / operator is "<<a/b<<"\n";

  cout<<"Your % operator is "<<a%b<<"\n";

  return 0;

}

5.2 Comparison operator

This operator can compare a two number and give the output generally this operator used with if else statement. It gives output in bool value 1 or 0.

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a=20,b=10;

  cout<<"Your == operator is "<<(a==b)<<"\n";

  cout<<"Your != operator is "<<(a!=b)<<"\n";

  cout<<"Your < operator is "<<(a<b)<<"\n";

  cout<<"Your > operator is "<<(a>b)<<"\n";

  cout<<"Your <= operator is "<<(a<=b)<<"\n";

  cout<<"Your >= operator is "<<(a>=b)<<"\n";

  return 0;

}

5.3 Logical operator

First is and operator if both condition is true then it will true second is or operator only one condition is need to true then it becomes true.

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a=2,b=10;

  cout<<"Your && operator is "<<(a<5 && b<10)<<"\n";

  cout<<"Your || operator is "<<(a<5 || b<4)<<"\n";

  return 0;

}

6 String

String means it is one type of data type which can store a word or a sentences.

In this program we simple initialize a variable string and print it

#include <iostream>

using *namespace* std;

*int* main()

{

  string a="Student bread";

  cout<<a;

  return 0;

}

6.1 String concat

String concat means to join a two different variable string we see it with a program. In this we concat a and b.

#include <iostream>

using *namespace* std;

*int* main()

{

  string a="Student";

  string b=" Bread";

  string c= a+b;

  cout<<c;

  return 0;

}

6.2 Size function

To find the size of the string we have the size method

#include <iostream>

using *namespace* std;

*int* main()

{

  string a="Student";

  cout<<"Size is "<<a.size();

  return 0;

}

6.3 Access string using it’s index

You can access the string one character as written in [0] means first character you can access

#include <iostream>

using *namespace* std;

*int* main()

{

  string a="Student bread";

  cout<<a[0];

  return 0;

}

7 math function

This type of function is include in the cmath header file

#include <iostream>

#include <cmath>

using *namespace* std;

*int* main()

{

  cout<<min(50,51)<<"\n";

  cout<<max(50,51)<<"\n";

  cout<<sqrt(49)<<"\n";

  return 0;

}

8. If else statement

If else statement is used when program takes a decision from multiple choice if choice is selected based on the condition

8.1 if else

In here we simple check the number a is greater or b is greater a is greater so if condition will executed otherwise else is executed

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a=150,b=100;

  if(a>b)

  {

    cout<<"A is greatest";

  }

  else

  {

    cout<<"B is greatest";

  }

  return 0;

}

8.2if else if else statement

This statement is used when we have more then 2 condition. Here we check the number is negative, positive or zero.

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a;

  cout<<"Enter your number: ";

  cin>>a;

  if(a<0)

  {

    cout<<"Number is negative";

  }

  else if(a>0)

  {

    cout<<"Number is positive";

  }

  else

  {

    cout<<"Number is zero";

  }

  return 0;

}

8.3 short if else statement

The if else statement are written on one line. Here we check if you are greater then 18 then you can vote otherwise not

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a;

  cout << "Enter your age: ";

  cin >> a;

  string result = (a > 18) ? "You can vote." : "You can not vote.";

  cout << result;

  return 0;

}

9. switch case

This type of statement is used when we have condition of particular integer or character or a string.

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a;

  cout << "Enter your age: ";

  cin >> a;

  switch (a)

  {

  case 1:

    cout<<"One";

    break;

  case 2:

    cout<<"Two";

    break;

  case 3:

    cout<<"Three";

    break;

  case 4:

    cout<<"Four";

    break;

  default:

    cout<<"please enter from 1 to 4";

    break;

  }

  return 0;

}

10.Loops

Loop is used when we need to print or do a task repeatedly then loop is used there are three types of loops is available do- while , while loop, for loop .  
  And for loop we need to initialize it check the condition and increment/decrement it.

10.1 while loop

Here we initialize i =1 and condition i<=5 when condition hit loop is terminated then we increment it

#include <iostream>

using *namespace* std;

*int* main()

{

*int* i=1;

  while (i<=5)

  {

    cout<<i<<"\n";

    i++;

  }

  return 0;

}

10.2 do while loop

Here we initialize i =1 and condition i<=5 when condition hit loop is terminated then we increment it

#include <iostream>

using *namespace* std;

*int* main()

{

*int* i=1;

  do

  {

    cout<<i<<"\n";

    i++;

  } while (i<=5);

  return 0;

}

10.3 For loop

Here we initialize i =1 and condition i<=5 when condition hit loop is terminated then we increment it

#include <iostream>

using *namespace* std;

*int* main()

{

*int* i;

  for (i = 1; i <=5; i++)

  {

    cout<<i<<"\n";

  }

  return 0;

}

11 Break/ continue

11.1 Break will terminated the whole loop when condition is hit

#include <iostream>

using *namespace* std;

*int* main()

{

*int* i;

  for (i = 1; i <=5; i++)

  {

    if(i==4)

    {

      break;

    }

cout<<i<<"\n";

  }

  return 0;

}

11.2 Continue

It will just don’t print the condition we define it will not terminate the whole loop

#include <iostream>

using *namespace* std;

*int* main()

{

*int* i;

  for (i = 1; i <=5; i++)

  {

    if(i==4)

    {

      continue;

    }

  cout<<i<<"\n";

  }

  return 0;

}

12 Array

Array means one type of variable which can store multiple value of single data types.

12.1 Array initialize and accessing

Here we declare array of size 3 12,13,14 we initialize a variable then simply we access it by writing it’s index in cout

#include <iostream>

using *namespace* std;

*int* main()

{

*int* i[3]={12,13,14};

  cout<<i[0]<<"\n";

  cout<<i[1]<<"\n";

  cout<<i[2]<<"\n";

  return 0;

}

12.2 Array and loops

#include <iostream>

using *namespace* std;

*int* main()

{

*int* a[3]={12,13,14};

  for(*int* i=0;i<3;i++)

  {

    cout<<a[i]<<"\n";

  }

  return 0;

}

12.3 Two dimension Array

2 d array it has 01,02,03 .. that much of index in the 2 d array here we initialize x and then simple print it.

#include<iostream>

using *namespace* std;

*int* main()

{

*int* x[3][2] = {{0,1}, {2,3}, {4,5}};

  for (*int* i = 0; i < 3; i++)

  {

    for (*int* j = 0; j < 2; j++)

    {

      cout << x[i][j]<<endl;

    }

  }

  return 0;

}

13 Structure

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.

Unlike an array, a structure can contain many different data types (int, string, bool, etc.).

#include<iostream>

using *namespace* std;

*struct* Employee

{

*int* a;

  string name;

*int* salary;

};

*int* main()

{

  Employee e;

  e.a=15;

  e.name="Loy";

  e.salary=50000;

  cout<<e.a<<endl;

  cout<<e.name<<endl;

  cout<<e.salary<<endl;

  return 0;

}

14 References and memory address

A reference variable is a "reference" to an existing variable, and it is created with the & operator:

14.1 References

Here we gave the references of the a to b variable when we print b it prints the value of a

#include<iostream>

using *namespace* std;

*int* main()

{

*int* a=15;

*int* &b=a;

  cout<<b;

  return 0;

}

14.2 Memory address

Here we have print the memory address of a by writing &b.

#include<iostream>

using *namespace* std;

*int* main()

{

*int* a=15;

*int* &b=a;

  cout<<&b;

  return 0;

}

15 pointer

A pointer is a variable that stores the memory address of an object. Pointers are used extensively in both C and C++ for three main purposes: to allocate new objects on the heap, to pass functions to other functions.

Here we make a pointer variable b and print a reference of a and variable b

#include<iostream>

using *namespace* std;

*int* main()

{

*int* a=15;

*int* \*b=&a;

  cout<<a<<endl;

  cout<<&a<<endl;

  cout<<b<<endl;

  return 0;

}

16 Function

Function means a block of code which executes when its call if we need function so we need to follow 3 steps function declaration, function call, function definition.

16.1 Function without argument without return type

myfunc is a function with no argument and no return type. Here we declare function call it and then define it.

#include<iostream>

using *namespace* std;

*void* myFunc();

*int* main()

{

  myFunc();

  return 0;

}

*void* myFunc()

{

    cout<<"I am function";

}

16.2 Function with argument without return type

here we make a function which takes 2 argument and add it

#include<iostream>

using *namespace* std;

*void* myFunc(*int* *a*,*int* *b*);

*int* main()

{

  myFunc(12,12);

  return 0;

}

*void* myFunc(*int* *a*,*int* *b*)

{

    cout<<"Answer is "<<(*a*+*b*);

}

16.3 Function with return type with argument

This is a function where we pass two argument and add it because we return it with a+b

#include<iostream>

using *namespace* std;

*int* myFunc(*int* *a*,*int* *b*);

*int* main()

{

  cout<<"Answer is "<<myFunc(24,24);

  return 0;

}

*int* myFunc(*int* *a*,*int* *b*)

{

    return *a*+*b*;

}

16.4 Function with return type without argument

Here we write a function which print the value of a.

#include<iostream>

using *namespace* std;

*int* myFunc();

*int* main()

{

  cout<<"A is "<<myFunc();

  return 0;

}

*int* myFunc()

{

*int* a=5;

    return a;

}

17 Oops concept

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

OOP is faster and easier to execute

OOP provides a clear structure for the programs

OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug

OOP makes it possible to create full reusable applications with less code and shorter development time

17.1 Classes and object

Here we define the myclass in that we have 2 variable which we can access by making it’s object in main method and print it by mc.a and mc.name. Here mc is a object or instances of a class.

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

    string name="Student bread";

};

*int* main()

{

    MyClass mc;

    cout<<mc.a<<endl;

    cout<<mc.name<<endl;

    return 0;

}

17.2 Access specifier

There are 3 access specifier are available in c++ public, private and protected

Public: you can access data to outside of program also

Private: It can not accessed from outside of the class

Protected: it cannot accessed by outside of class but it can used by inherited the above class

When we access private member in main class it gives a error

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

    string name="Student bread";

*private:*

*int* b=25;

};

*int* main()

{

    MyClass mc;

    cout<<mc.a<<endl;

    cout<<mc.b<<endl;

    cout<<mc.name<<endl;

    return 0;

}

Now you can access all member by having a getdata method in public

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

    string name="Student bread";

*void* getdata()

    {

        cout<<"Your a is "<<a<<endl;

        cout<<"Your b is "<<b;

    }

*private:*

*int* b=25;

*protected:*

*int* c=50;

};

*int* main()

{

    MyClass mc;

    cout<<mc.a<<endl;

    cout<<mc.name<<endl;

    mc.getdata();

    return 0;

}

17.3 Constructor

Constructor is executed when instances is created the class is define but it memory is allocated when the instances is make. Two types of constructor is parameterized and simple constructor. You can define a constructor by write a same name as class name in class in public

17.3.1 Simple constructor

It is simple constructor it doesn’t have any argument.

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

    MyClass()

    {

        cout<<"I am a constructor";

    }

};

*int* main()

{

    MyClass mc;

    return 0;

}

17.3.2 Parameterized constructor

This constructor have a argument in it. Simple gave a argument and pass at when object is created.

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

    MyClass(*int* *a*,*int* *b*)

    {

        cout<<"I am a parameterised constructor";

    }

};

*int* main()

{

    MyClass mc(2,2);

    return 0;

}

17.4 Encapsulation

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class). If you want others to read or modify the value of a private member, you can provide public get and set methods.

#include<iostream>

using *namespace* std;

*class* MyClass{

*private:*

*int* a;

*public:*

*void* seta(*int* *x*)

    {

        a=*x*;

    }

*void* display()

    {

        cout<<"A is "<<a;

    }

};

*int* main()

{

    MyClass mc;

    mc.seta(15);

    mc.display();

    return 0;

}

18 Inheritances

The one class member and method can used by another class. There are 5 types of inheritance is supported by the c++ Single inheritances, Multiple inheritances ,Multilevel inheritances, Hierarchical inheritances and Hybrid inheritances.

18.1 Single inheritances

It is one level inheritances one class derived from another that simple. You can write : public previous class name and access it member by making object of class 2.

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

    string name="student bread";

};

*class* MyClass2 : *public* MyClass{

};

*int* main()

{

    MyClass2 mc;

    cout<<"Your a is "<<mc.a<<endl;

    cout<<"Your name is "<<mc.name<<endl;

    return 0;

}

18.2 Multiple inheritances

In this type of inheritances 2 class are inherited by it’s child class.

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

};

*class* MyClass2 {

*public:*

    string name="student bread";

};

*class* MyClass3 : *public* MyClass , *public* MyClass2{

};

*int* main()

{

    MyClass3 mc;

    cout<<"Your a is "<<mc.a<<endl;

    cout<<"Your name is "<<mc.name<<endl;

    return 0;

}

18.3 Multilevel inheritances

In this type of inheritances every class are inherited by it’s child class.

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

};

*class* MyClass2: *public* MyClass{

*public:*

    string name="student bread";

};

*class* MyClass3 :  *public* MyClass2{

};

*int* main()

{

    MyClass3 mc;

    cout<<"Your a is "<<mc.a<<endl;

    cout<<"Your name is "<<mc.name<<endl;

    return 0;

}

18.4 Hierarchical inheritances

In this type of inheritances the one class is derived all their child class

#include<iostream>

using *namespace* std;

*class* MyClass{

*public:*

*int* a=15;

    string email="student bread@gmail.com";

};

*class* MyClass2: *public* MyClass{

*public:*

    string name="student bread";

};

*class* MyClass3 :  *public* MyClass{

};

*int* main()

{

    MyClass3 mc;

    cout<<"Your a is "<<mc.a<<endl;

    cout<<"Your a is "<<mc.email<<endl;

    return 0;

}

18.5 Hybrid inheritances

It is a combination of more the 2 inheritances it is called hybrid inheritances.

#include <iostream>

using *namespace* std;

*class* vehicle

{

*public:*

  vehicle()

  {

    cout << "This is a vehicle\n";

  }

};

*class* Car : *public* vehicle

{

*public:*

  Car()

  {

    cout << "This is a car\n";

  }

};

*class* Racing

{

*public:*

  Racing()

  {

    cout << "This is for Racing\n";

  }

};

*class* Ferrari : *public* Car, *public* Racing

{

*public:*

  Ferrari()

  {

    cout << "Ferrari is a Racing Car\n";

  }

};

*int* main()

{

  Ferrari f;

  return 0;

}

19 Polymorphism

It means define a same function in different class and object of class first get their own function

#include <iostream>

using *namespace* std;

*class* A{

*public:*

*void* getdata()

    {

        cout<<"I am getdata from A"<<endl;

    }

};

*class* B:*public* A{

*public:*

*void* getdata()

    {

        cout<<"I am getdata from B"<<endl;

    }

};

*class* C:*public* B{

*public:*

*void* getdata()

    {

        cout<<"I am getdata from C"<<endl;

    }

};

*int* main()

{

  A obj;

  B obj1;

  C obj2;

  obj.getdata();

  obj1.getdata();

  obj2.getdata();

  return 0;

}

20 File handling

File handling means to read and write a txrt file or other file using the C++ language

The fstream library allows us to work with files.

To use the fstream library, include both the standard <iostream> AND the <fstream> header file:

20.1 Writing on the file

#include <iostream>

#include <fstream>

using *namespace* std;

*int* main() {

  ofstream MyFile("Student\_Bread.txt");

  MyFile << "Our websites provide a college search engine as well as Programming language tutorials";

  if(MyFile)

  {

      cout<<"Your text is written";

  }

  else

  {

      cout<<"Your text is written";

  }

  MyFile.close();

}

20.2 Reading from the file

#include <iostream>

#include <fstream>

using *namespace* std;

*int* main()

{

  string myText;

  ifstream MyReadFile("Student\_Bread.txt");

  while (getline(MyReadFile, myText))

  {

    cout << myText<<endl;

  }

  MyReadFile.close();

}