Inner class:

class outer{

public:

class inner{

public:

void m1(){

cout<<"hello world";

}

};

};

int main(){

outer::inner in;

in.m1();

return 0;

}

Local Classes in C++

A class declared inside a function becomes local to that function and is called Local Class in C++. For example, in the following program, Test is a local class in fun().

|  |
| --- |
| #include<iostream>  using namespace std;    void fun()  {        class Test  // local to fun        {          /\* members of Test class \*/        };  }    int main()  {      return 0;  } |

***1)****A local class type name can only be used in the enclosing function.*For example, in the following program, declarations of t and tp are valid in fun(), but invalid in main().

|  |
| --- |
| #include<iostream>  using namespace std;    void fun()  {        // Local class        class Test        {          /\* ... \*/        };          Test t;  // Fine        Test \*tp;  // Fine  }    int main()  {      Test t;  // Error      Test \*tp;  // Error      return 0;  } |

***2)*** All the methods of Local classes must be defined inside the class only

***3)*** A Local class cannot contain static data members. It may contain static functions though.

Local object and global object:

The object declared outside the function bodies is known as global object. All function can access the global object. The object declared inside a function body is known as local object. The scope of local object is limited to its current block.

Ex.

class student{

public:

void display( char \*c){

cout<<"\n"<<c;

}

};

student st; //global object

int main()

{

student st1; // local object

st.display("global");

st1.display("local");

return 0 ;

}

**Static Variables**

* **Static variables in a Function**: When a variable is declared as static, space for **it gets allocated for the lifetime of the program**. Even if the function is called multiple times, space for the static variable is allocated only once and the value of variable in the previous call gets carried through the next function call.

void m1(){

static int x=0;

cout<<x<<endl;

x++;

}

int main()

{

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

m1();

}

return 0 ;

}

We can define class members static using **static** keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.

A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator **::** to identify which class it belongs to.

class student{

public:

static int x;

};

int student::x=5;

int main()

{

cout<<student::x;

return 0 ;

}

## **Static Function**

By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the **static** functions are accessed using only the class name and the scope resolution operator **::**.

A static member function can only access static data member, other static member functions and any other functions from outside the class.

class student{

public:

static void m1(){

cout<<"hello";

}

};

int main()

{

student::m1();

return 0 ;

}