Some interesting facts about static member functions in C++

Friend class and function in C++

Local Classes in C++

Nested Classes in C++

Simulating final class in C++

Constructors in C++

Copy Constructor in C++

Destructors in C++

Does C++ compiler create default constructor when we write our own?

When should we write our own copy

constructor argument should be const in C++?

time.h header file in C with Examples

scanf("%[^\n]s", str) Vs gets(str) in C with Examples

C program to Insert an element in an Array

Types of Literals in C/C++ with Examples

Conditional or Ternary Operator (?:) in C/C++

Difference between C and C#

time.h localtime() function in C with Examples

asctime() and asctime_s() functions in C with Examples

return statement in

C/C++ with
Examples

When do we use Initializer List in C++?

Initializer List is used in initializing the data members of a class. The list of members to be initialized is indicated with constructor as a comma-separated list followed by a colon. Following is an example that uses the initializer list to initialize x and y of Point class.

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

class Point {
private:
    int x;
    int y;
public:
    Point(int i = 0, int j = 0):x(i), y(j) {}
    /*  The above use of Initializer list is optional as the
        constructor can also be written as:
        Point(int i = 0, int j = 0) {
            x = i;
            y = j;
        }
    */

    int getX() const {return x;}
    int getY() const {return y;}
};

int main() {
  Point t1(10, 15);
  cout<<"x = "<<t1.getX()<<", ";
  cout<<"y = "<<t1.getY();
  return 0;
}

/* OUTPUT:
   x = 10, y = 15
*/
```

The above code is just an example for syntax of the Initializer list. In the above code, x and y can also be easily initialed inside the constructor. But there are situations where initialization of data members inside constructor doesn't work and Initializer List must be used. Following are such cases:

1) For initialization of non-static const data members:
const data members must be initialized using Initializer List. In the following example, "t" is a const data member of Test class and is initialized using Initializer List. Reason for initializing the const data member in initializer list is because no memory is allocated separately for const data member, it is folded in the symbol table due to which we need to initialize it in the initializer list.
Also, it is a copy constructor and we don't need to call the assignment operator which means we are avoiding one extra operation.

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

class Test {
    const int t;
public:
    Test(int t):t(t) {}  //Initializer list must be used
    int getT() { return t; }
};

int main() {
    Test t1(10);
    cout<<t1.getT();
    return 0;
}

/* OUTPUT:
   10
*/
```

2) For initialization of reference members:

Reference members must be initialized using Initializer List. In the following example, “t” is a reference member of Test class and is initialized using Initializer List.

```
// Initialization of reference data members
#include<iostream>
using namespace std;

class Test {
    int &t;
public:
    Test(int &t):t(t) {}  //Initializer list must be used
    int getT() { return t; }
};

int main() {
    int x = 20;
    Test t1(x);
    cout<<t1.getT()<<endl;
    x = 30;
    cout<<t1.getT()<<endl;
    return 0;
}
/* OUTPUT:
    20
    30
 */
```

3) For initialization of member objects which do not have default constructor:

In the following example, an object “a” of class “A” is data member of class “B”, and “A” doesn’t have default constructor. Initializer List must be used to initialize “a”.

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

class A {
    int i;
public:
    A(int );
};

A::A(int arg) {
    i = arg;
    cout << "A's Constructor called: Value of i: " << i << endl;
}

// Class B contains object of A
class B {
    A a;
public:
    B(int );
};

B::B(int x):a(x) {  //Initializer list must be used
    cout << "B's Constructor called";
}

int main() {
    B obj(10);
    return 0;
}
/* OUTPUT:
    A's Constructor called: Value of i: 10
    B's Constructor called
*/
```

If class A had both default and parameterized constructors, then Initializer List is not must if we want to initialize “a” using default constructor, but it is must to initialize “a” using parameterized constructor.

4) For initialization of base class members : Like point 3, the parameterized constructor of the base class can only be called using Initializer List.

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

class A {
    int i;
public:
    A(int );
};

A::A(int arg) {
    i = arg;
    cout << "A's Constructor called: Value of i: " << i << endl;
}

// Class B is derived from A
class B: A {
public:
    B(int );
};

B::B(int x):A(x) { //Initializer list must be used
    cout << "B's Constructor called";
}

int main() {
    B obj(10);
    return 0;
}
```

5) When constructor's parameter name is same as data member

If constructor's parameter name is same as data member name then the data member must be initialized either using this pointer or Initializer List. In the following example, both member name and parameter name for A() is "i".

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

class A {
    int i;
public:
    A(int );
    int getI() const { return i; }
};

A::A(int i):i(i) { }  // Either Initializer list or this pointer must
/* The above constructor can also be written as
A::A(int i) {
    this->i = i;
}
*/

int main() {
    A a(10);
    cout<<a.getI();
    return 0;
}
/* OUTPUT:
    10
*/
```

6) For Performance reasons:

It is better to initialize all class variables in Initializer List instead of assigning values inside body. Consider the following example:

```
// Without Initializer List
class MyClass {
    Type variable;
public:
    MyClass(Type a) {  // Assume that Type is an already
                     // declared class and it has appropriate
                     // constructors and operators
      variable = a;
    }
};
```

Here compiler follows following steps to create an object of type MyClass

1. Type's constructor is called first for "a".

2. The assignment operator of "Type" is called inside body of MyClass() constructor to assign

```
variable = a;
```

3. And then finally destructor of "Type" is called for "a" since it goes out of scope.

Now consider the same code with MyClass() constructor with Initializer List

```
// With Initializer List
class MyClass {
    Type variable;
public:
    MyClass(Type a):variable(a) {   // Assume that Type is an already
                         // declared class and it has appropriate
                         // constructors and operators
    }
};
```

With the Initializer List, the following steps are followed by compiler:
1. Copy constructor of "Type" class is called to initialize: variable(a). The arguments in the initializer list are used to copy construct "variable" directly.
2. The destructor of "Type" is called for "a" since it goes out of scope.

As we can see from this example if we use assignment inside constructor body there are three function calls: constructor + destructor + one addition assignment operator call. And if we use Initializer List there are only two function calls: copy constructor + destructor call. See this post for a running example on this point.

This assignment penalty will be much more in "real" applications where there will be many such variables. Thanks to *ptr* for adding this point.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

list::remove() and list::remove_if() in C++ STL

list::empty() and list::size() in C++ STL

list::push_front() and list::push_back() in C++ STL

list::emplace_front() and list::emplace_back() in C++ STL

list::front() and list::back() in C++ STL

list::pop_front() and list::pop_back() in C++ STL

list::begin() and list::end() in C++ STL

list get_allocator in C++ STL

list end() function in C++ STL

list::clear() in C++ STL

Nested list in C++ STL

list::operator= in C++ STL

list insert() in C++ STL

list::swap() in C++ STL

List in C++ | Set 2 (Some Useful Functions)

Improved By : RajendraStalekar

Article Tags : C C++

Practice Tags : C CPP

35

3.4

☐ To-do ☐ Done

Based on **89** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments