When do we use Initializer List in C++?

Initializer List is used to initialize 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 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
*/
```

Run on IDE

The above code is just an example for syntax of 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.

```
#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
*/
```

Run on IDE

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
 */
```

Run on IDE

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 << end
}

// 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
*/
```

Run on IDE

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, parameterized constructor of 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 << end
}

// 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;
}
```

Run on IDE

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 point
/* 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
*/
```

Run on IDE

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;
    }
};
```

Run on IDE

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
                    // declared class and it has appropriate
                    // constructors and operators
    }
};
```

Run on IDE

With the Initializer List, following steps are followed by compiler:

1. Copy constructor of “Type” class is called to initialize : variable(a). The arguments in initializer list are used to copy construct “variable” directly.
2. 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.