Lecture : Constructor 2

Quick Recap

- Constructor
- Types of Constructor
- Default Constructor
- Parameterized Constructor
- Copy Constructor

Today’s Agenda

Today we are going to cover -

- Initializer List
- Constructor with default argument
- Destructor

Let’s Get Started-

Initializer List

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:
```

Initializer List

```
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;}
};
```

Initializer List

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

OUTPUT:

x = 10, y = 15

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

2) For initialization of reference members

3)When constructor’s parameter name is same as data member

For initialization of non-static const data members

```
#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;
}
```

For initialization of non-static const data

```
/* OUTPUT:
10
*/
```

For initialization of reference members.

```
// 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; }
};
```

For initialization of reference members.

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

OUTPUT:

20

30

When constructor’s parameter name is same as data.

```
#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;
}
```

When constructor’s parameter name is same as data.

```
// 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

Default Argument

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn't provide a value for the argument with a default value.

```
// A function with default arguments, it can be called with
// 2 arguments or 3 arguments or 4 arguments.
int sum(int x, int y, int z=0, int w=0)
{
    return (x + y + z + w);
}
```

Default Argument

```
/* Driver program to test above function*/
int main()
{
    cout << sum(10, 15) << endl;
    cout << sum(10, 15, 25) << endl;
    cout << sum(10, 15, 25, 30) << endl;
    return 0;
}
```

Output

```
25
50
80
```

Destructors

Destructor is a member function which destructs or deletes an object.

Syntax:-

~constructor-name();

Properties of Destructors

- Destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or const.
- The destructor does not have arguments.
- It has no return type not even void.
- An object of a class with a Destructor cannot become a member of the union.
- A destructor should be declared in the public section of the class.
- The programmer cannot access the address of destructor.

When a Destructors is called

A destructor function is called automatically when the object goes out of scope:

- the function ends
- the program ends
- a block containing local variables ends
- a delete operator is called

Destructors

```
class String {
private:
   char* s;
   int size;
public:
   String(char*); // constructor
   ~String(); // destructor
};

String::String(char* c)
{
   size = strlen(c);
   s = new char[size + 1];
   strcpy(s, c);
}
String::~String() { delete[] s; }
```

Can there be more than one destructor in a class?

No, there can only one destructor in a class with classname preceded by ~, no parameters and no return type.

MCQ 1

Can destuctors be private in C++?

(A) Yes

(B) No

MCQ 1

Can destuctors be private in C++?

(A) Yes

(B) No

MCQ 2

Like constructors, can there be more than one destructors in a class?

(A) Yes

(B) No

MCQ 2

Like constructors, can there be more than one destructors in a class?

(A) Yes

(B) No

Guess the output 1

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

int i;

class A
{
public:
   ~A()
   {
      i=10;
   }
};
```

Guess the output 1

```
int foo()
{
    i=3;
    A ob;
    return i;
}

int main()
{
    cout << foo() << endl;
    return 0;
}
```

Guess the output 1

3

Guess the output 2

```
class A
{
    int id;
    static int count;
public:
    A() {
        count++;
        id = count;
        cout << "constructor for id " << id << endl;
    }
    ~A() {
        cout << "destructor for id " << id << endl;
    }
};
```

Guess the output 2

```
int A::count = 0;

int main() {
    A a[3];
    return 0;
}
```

Guess the output 2

constructor for id 1

constructor for id 2

constructor for id 3

destructor for id 3

destructor for id 2

destructor for id 1

QNA Time

Any Questions ??

Any Questions??

Thank You!

See you guys in next class.