Data Structures and Object Oriented Programming using C++

Ahsan Ijaz

Topics Covered

- Classes
- Objects
- Data Hiding

Topics Covered

- Classes
- Objects
- Data Hiding

Topics Covered

- Classes
- Objects
- Data Hiding

Lecture Details

- Constructors
- Destructors

Constructors

- Member Function
- **Primary use**-Object Initialization

Example Class

```
1   class Distance
2   {private:
3   int feet;
4   float inches;
5    public:
6    void setdistance(int ft, float in)  //Function Used to Initialize Object
7   {feet = ft;
8    inches = in;}
9    void getdistance()
10  {cout << "Enter feet : ";
11   cin >> feet;
12   cout << "Enter inches : ";
13   cin >> inches;}
14   void showdistance()
15  {cout << "feet"<< feet << "inches" << inches << endl;}
16  };
17  int main()
18  { Distance d1, d2;
19    d1.setdistance(1,2);
20  d1.showdistance();
21  d2.getdistance();
22  d2.showdistance();
23  }
```

Constructors Definition

- Name Should be same as Class
- No Return Type
- Function called at time of Object creation
- Increases program readability

Constructors Definition

- Name Should be same as Class
- No Return Type
- Function called at time of Object creation
- Increases program readability

Constructors Definition

- Name Should be same as Class
- No Return Type
- Function called at time of Object creation
- Increases program readability

Constructors Definition

- Name Should be same as Class
- No Return Type
- Function called at time of Object creation
- Increases program readability

Writing Constructors

- Counter() count=0;
- Counter(): count(0) {};

Writing Constructors

- Counter() count=0;
- Counter(): count(0) {};

Constructors Example 1

```
1   class Counter
2   {
3    private:
4   unsigned int count;
5    public:
6   Counter() {count=0;}     //CONSTRUCTOR DEFINITION
7   void inc_count()
8   {count++;}  // increment counter
9   int get_count()
10  {return count;}  // return counter
11  };
12
13  int main()
14  {
15  Counter c1, c2;
16
17   cout << "\n c1 = " << c1.get_count();     // display counter 1
18   cout << "\n c2 = " << c2.get_count();     // display counter 2
19
20   c1.inc_count();     // increment counter 1
21   c2.inc_count();     // increment counter 2
22   c2.inc_count();     // increment counter 2
23
24   cout << "\n c1 = " << c1.get_count();     // display again
25   cout << "\n c2 = " << c2.get_count() << endl;
26  }
```

Constructors Example 2

```
1   class Distance
2   {private:
3   int feet;
4   float inches;
5    public:
6    Distance(int ft, float in):feet(ft),inches(in)
7   {};  //CONSTRUCTOR DEFINITION
8    void getdistance()
9   {cout << "Enter feet : ";
10   cin >> feet;
11   cout << "Enter inches : ";
12   cin >> inches;}
13   void showdistance()
14  {cout << feet << "\' - " << inches << "\" "<< endl;}
15  };
16
17  int main()
18  { Distance d1(11,6.25);
19    d1.showdistance();
20  d1.getdistance();
21  d1.showdistance();
22  }
```

Constructor Overloading

- Initialize one of the objects, instantiate another one.
- Use multiple constructors
- Similar as Function Overloading

Constructor Overloading Example

```
1  class Distance
2  {private:
3    int feet;
4    float inches;
5  public:
6    Distance()    //CONSTRUCTOR 1
7        {}
8       Distance(int ft, float in): feet(ft),inches (in){};
9       . //CONSTRUCTOR 2
10 };
11
12 int main()
13 {
14   Distance dist1, dist3;
15 Distance dist2(11, 6.25);
16      ..
17 }
```

Destructor

- Opposite of Constructors
- Destructors should be called before object goes out of scope
- No arguments, no return types
- De-allocation of memory

Destructor

- Opposite of Constructors
- Destructors should be called before object goes out of scope
- No arguments, no return types
- De-allocation of memory

Destructor

- Opposite of Constructors
- Destructors should be called before object goes out of scope
- No arguments, no return types
- De-allocation of memory

Destructor

- Opposite of Constructors
- Destructors should be called before object goes out of scope
- No arguments, no return types
- De-allocation of memory

Destructor Example

```
1  class Counter
2  {private:
3  int *data;
4   public:
5  Counter():count (0) {}
6  ~Counter()
7  {delete data;}
8  };
```

Copy Constructor

- Initialize one object from another of the same type.
- Created by default

Complete Example for a Class

```
1   #include <iostream>
2   using namespace std;
3   class Line
4   {
5      public:
6         Line( int len );             // simple constructor
7         Line( const Line &obj);  // copy constructor
8         ~Line();                     // destructor
9     private:
10        int *ptr;
11  };
12  // Member functions definitions including constructor
13  Line::Line(int len)
14  {
15      cout << "Normal constructor allocating ptr" << endl;
16      // allocate memory for the pointer;
17      ptr = new int;
18      *ptr = len;
19  }
20  Line::Line(const Line &obj)
21  {
22      cout << "Copy constructor allocating ptr." << endl;
23      ptr = new int;
24     *ptr = *obj.ptr; // copy the value
25  }
26  Line::~Line(void)
27  {
28      cout << "Freeing memory!" << endl;
29      delete ptr;
30  }
```

Namespaces

- Resolves Conflicts while using functions of variables defined at different places but having the same name.

```
1  #include <iostream>
2  using namespace std;
3
4  // first name space
5  namespace first_space{
6     void func(){
7        cout << "Inside first space" << endl;
8     }
9  }
10 // second name space
11 namespace second_space{
12    void func(){
13       cout << "Inside second space" << endl;
14    }
15 }
16 int main ()
17 {
18
19    // Calls function from first name space.
20    first_space::func();
21
22    // Calls function from second name space.
23    second_space::func();
24
25    return 0;
26 }
```