Object Oriented Programming

Using C++ Programming Language

Lecture # 6

Copy Constructor

The Default Copy Constructor

- Two ways to initialize objects.

- A no–argument constructor can initialize data members to constant values,
- A multi–argument constructor can initialize data members to values passed as arguments.

- Another way to initialize an object: you can initialize it with *another object of the same type*
- You don't need to create a special constructor for this; one is already built into all classes. It's called the *default copy constructor*

The Default Copy Constructor
Example(1)

```
#include <iostream>
using namespace std;
class Distance                  //English Distance class
{
private:
int feet;
float inches;
Public:                                 //constructor (no args)
Distance() : feet(0), inches(0.0)
{  }//Note: no one–arg constructor
//constructor (two args)
Distance(int ft, float in)  : feet(ft), inches(in)
{  }
void getdist()              //get length from user
{
cout << “\nEnter feet: ”;  cin >> feet;
cout << “Enter inches: ”;  cin >> inches;
}
void showdist()             //display distance
{ cout << feet << “\’–” << inches << ‘\”’; }
};
```

The Default Copy Constructor Example(2)

```
int main()
{
Distance dist1(11, 6.25);      //two–arg constructor
Distance dist2(dist1);         //one–arg constructor
Distance dist3 = dist1;        //also one–arg constructor
//display all lengths


cout << “\ndist1 = ”;  dist1.showdist();
cout << “\ndist2 = ”;  dist2.showdist();
cout << “\ndist3 = ”;  dist3.showdist();
cout << endl;
return 0;
}
```

Returning Objects from Functions Example(1)

```
#include <iostream>
using namespace std;
class Distance                    //English Distance class
{
private:
int feet;
float inches;
public:                           //constructor (no args)
Distance() : feet(0), inches(0.0)
{  }                              //constructor (two args)
Distance(int ft, float in) : feet(ft), inches(in)
{  }
void getdist()                    //get length from user
{
cout << "\nEnter feet: ";  cin >> feet;
cout << "Enter inches: ";  cin >> inches;
}
void showdist()                   //display distance
{ cout << feet << "\'-" << inches << '\"'; }
Distance add_dist(Distance);    //add
};
```

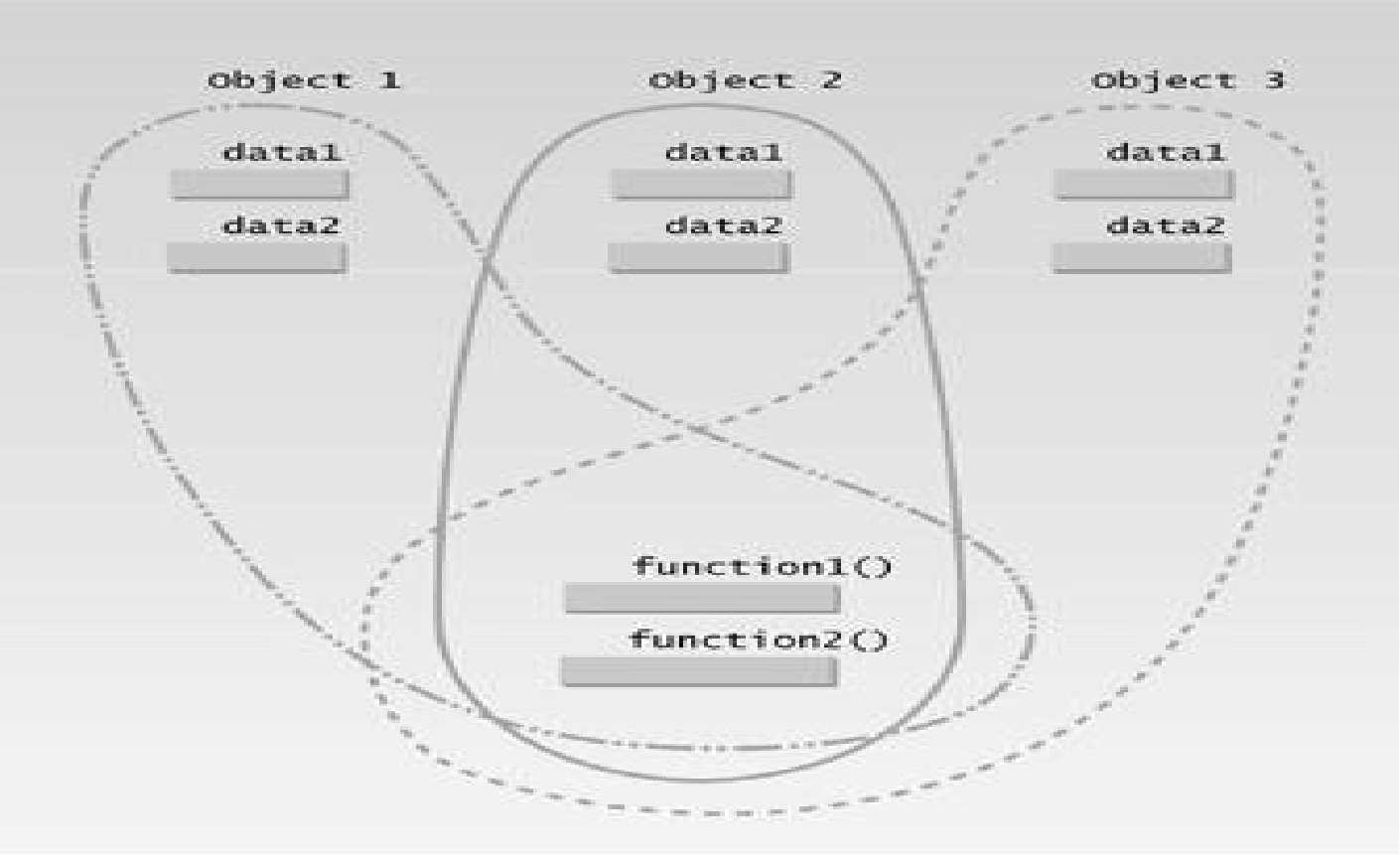
Returning Objects from Functions Example(2)

```
//add this distance to d2, return the sum
Distance Distance::add_dist(Distance d2)
{
Distance temp;                    //temporary variable
temp.inches = inches + d2.inches;  //add the inches
if(temp.inches >= 12.0)           //if total exceeds 12.0,
{                            //then decrease inches
temp.inches –= 12.0;           //by 12.0 and
temp.feet = 1;                 //increase feet
}                            //by 1
temp.feet += feet + d2.feet;       //add the feet
return temp;
}
int main()
{
Distance dist1, dist3;          //define two lengths
Distance dist2(11, 6.25);       //define, initialize dist2
dist1.getdist();               //get dist1 from user
dist3 = dist1.add_dist(dist2);  //dist3 = dist1 + dist2
//display all lengths
cout << “\ndist1 = ”;  dist1.showdist();
cout << “\ndist2 = ”;  dist2.showdist();
cout << “\ndist3 = ”;  dist3.showdist();
cout << endl;
return 0;
}
```

Classes, Objects, and Memory

- Each object created from a class contains separate copies of that class’s data.
- Objects are complete, self–contained entities, designed using the class declaration
- Each object has its own separate data items. On the other hand, all the objects of a class use the same member functions.

Classes, Objects, and Memory



Static Class Data

- If a data item in a class is declared as static, then only one such item is created for the entire class, no matter how many objects there are.

- static class member data is used to share information among the objects of a class.

Uses of Static Class Data

- In a road–racing game, for example,
- A race car might want to know how many other cars were still in the race.
- In this case a static variable count could be included as a member of the class. All the objects would have access to this variable.
- It would be the same variable for all of them; they would all see the same count

An Example of Static Class Data

```
#include <iostream>
using namespace std;
class foo
{
private:
static int count;   //only one data item for all objects
//note: *declaration* only!
public:
foo()               //increments count when object created
 { count++; }
int getcount()      //returns count
{ return count; }
};

int foo::count = 0;       //*definition* of count
int main()
{
foo f1, f2, f3;        //create three objects
cout << “count is ” << f1.getcount() << endl;  //each object
cout << “count is ” << f2.getcount() << endl;  //sees the
cout << “count is ” << f3.getcount() << endl;  //same value
return 0;
}
```

Separate Declaration and Definition

- Ordinary variables are declared (the compiler is told about their name and type) and defined (the compiler sets aside memory to hold the variable) in the same statement.

- Static member data, requires two separate statements. The variable's declaration appears in the class declaration, but the variable is actually defined outside the class, in much the same way as an external variable.

Returning an object from Class

```
Distance CalcDistance(Point A, Point B)
{
int X = abs(A.x-B.x);
int Y = abs(A.y-B.y);
 Distance  d(X,Y);


return d;
}
```

Q & A