CS-1201 Object Oriented Programming

Constructors, Objects Manipulation and Static Data Members

Arbish Akram

Department of Computer Science
Government College University

Constructor Overloading

- Constructors can be overloaded like function overloading.
- Overloaded constructors share the same name (the class name) but differ in the number and/or type of arguments.
- The appropriate constructor is called based on the number and type of arguments passed during object creation.

Constructor overloading

```
1  class Person {
2     private:
3          int age;
4     public:
5          // 1. Constructor with no arguments
6          Person() {
7              age = 20;
8          }
9          // 2. Constructor with an argument
10         Person(int a) {
11             age = a;
12         }
13         int getAge() {
14             return age;
15         }
16 };
17 int main() {
18     Person person1, person2(45);
19     cout << "Person1 Age = " << person1.getAge() << endl;
20     cout << "Person2 Age = " << person2.getAge() << endl;
21     return 0;
22 }
```

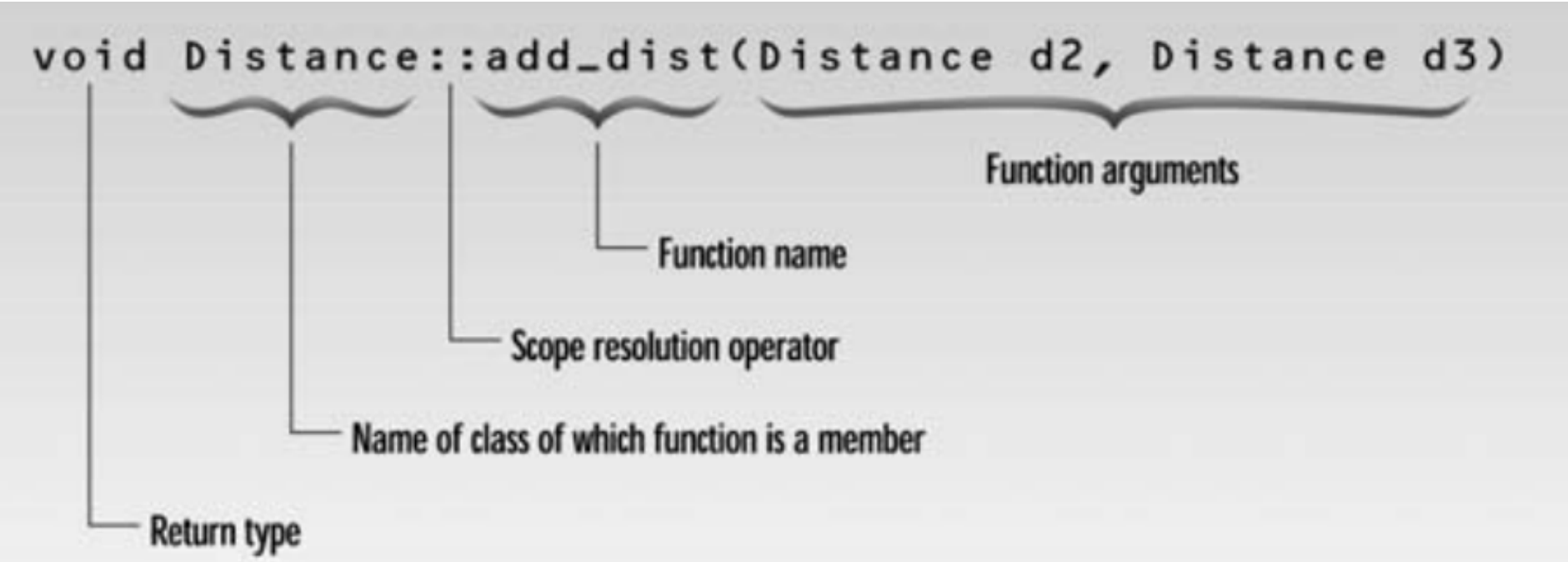
Member functions defined outside the class

- Member function can be defined inside or outside the class.
- Declaring functions in the class keeps the interface clear, while definitions outside can keep implementation details separate.
- Definitions outside can make the class definition itself more concise and easier to read.

Member Functions Defined Outside the Class

```
class Distance {
    private:
        int feet;
        float inches;
    public:
          // Constructor declaration
        Distance();
          // Member function declaration
        void show_dist();
    };
    // Member function definitions outside the class
Distance::Distance() : feet(0), inches(0.0) { }
void Distance::show_dist() {
        cout << feet << "'-" << inches << "\"" << endl;
    }
```

Member Functions Defined Outside the Class



Pass Objects to Function

- Pass an object as an argument within the member function of the class.
- Used to initialize all data members of an object with another object.
- Pass objects and assign the values of supplied object to the current object.

Example 1: Pass Objects to Function

```
1  // C++ program to calculate the average marks of two students
2  #include <iostream>
3  using namespace std;
4
5  class Student {
6      public:
7          double marks;
8          // constructor to initialize marks
9          Student(double m) : marks{m} { }
10 };
11
12 // function that has objects as parameters
13 double average_marks(Student s1, Student s2) {
14     // return the average of marks of s1 and s2
15     return (s1.marks + s2.marks)/2 ;
16 }
```

Example 1: Pass Objects to Function

```
#include<iostream>

class Student {...};

void calculateAverage(Student s1, Student s2) {
    // code
}

int main() {
    ... ...
    calculateAverage(student1, student2);
    ... ...
}
```

```
1  int main() {
2      Student student1(88.0), student2(56.0);
3      // pass the objects as arguments
4      cout << "Average Marks = " << average_marks(student1, student2) << "\n";
5      return 0;
6  }
```

Example 2: Pass Objects to Function

```
1class Distance {
2  private:
3      int feet;
4      float inches;
5  public:
6      Distance() : feet(0), inches(0.0)
7      { }
8      Distance(int ft, float in) : feet(ft), inches(in)
9      { }
10     void getdist() {
11         court << "\nEnter feet: "; cin >> feet;
12         court << "Enter inches: "; cin >> inches;
13     }
14     void showdist() {
15       cout << feet << "\'-" << inches << '\"'; }
16     void add_dist(Distance, Distance );
17 };
```

Example 2: Pass Objects to Function

```
1  void Distance::add_dist(Distance d2, Distance d3)
2  {
3   inches = d2.inches + d3.inches; //add the inches
4   feet = 0;
5   if(inches >= 12.0) {
6       inches -= 12.0;
7       feet++;
8   }
9   feet += d2.feet + d3.feet;
10  }
11  int main() {
12      Distance dist1, dist3;
13      Distance dist2(11, 6.25);
14      dist1.getdist();
15      dist3.add_dist(dist1, dist2);
16      court << "\ndist1 = ";
17      court << "\ndist2 = ";
18      court << "\ndist3 = "<<endl;
19      return 0;
20  }
```

Example 1: Return Objects from Function

```
1  class Student {
2     public:
3      double marks1, marks2;
4  };
5  // function that returns object of Student
6  Student createStudent() {
7      Student student;
8      // Initialize member variables of Student
9      student.marks1 = 96.5;
10     student.marks2 = 75.0;
11     // print member variables of Student
12     cout << "Marks 1 = " << student.marks1 << endl;
13     cout << "Marks 2 = " << student.marks2 << endl;
14     return student;
15 }
16 int main() {
17     Student student1;
18     // Call function
19     student1 = createStudent();
20     return 0;
21 }
```

Example 2: Return Objects from Function

```
1  Distance Distance::add_dist(Distance d2)
2      {
3      Distance temp;   //temporary variable
4      temp.inches = inches + d2.inches; //add the inches
5      if(temp.inches >= 12.0)
6      {
7          temp.inches -= 12.0;
8          temp.feet = 1;
9      }
10     temp.feet += feet + d2.feet;      //add the feet
11     return temp;
12     }
13     int main() {
14         Distance dist1, dist3;
15         Distance dist2(11, 6.25);
16         dist1.getdist();
17         dist3 = dist1.add_dist(dist2);
18         court << "\ndist1 = ";
19         court << "\ndist2 = ";
20         court << "\ndist3 = "<<endl;
21         return 0;
22     }
```

Array of Objects

- An array of objects is a collection of multiple instances of a class stored in contiguous memory locations.
- Syntax: `ClassName arrayName[arraySize];`

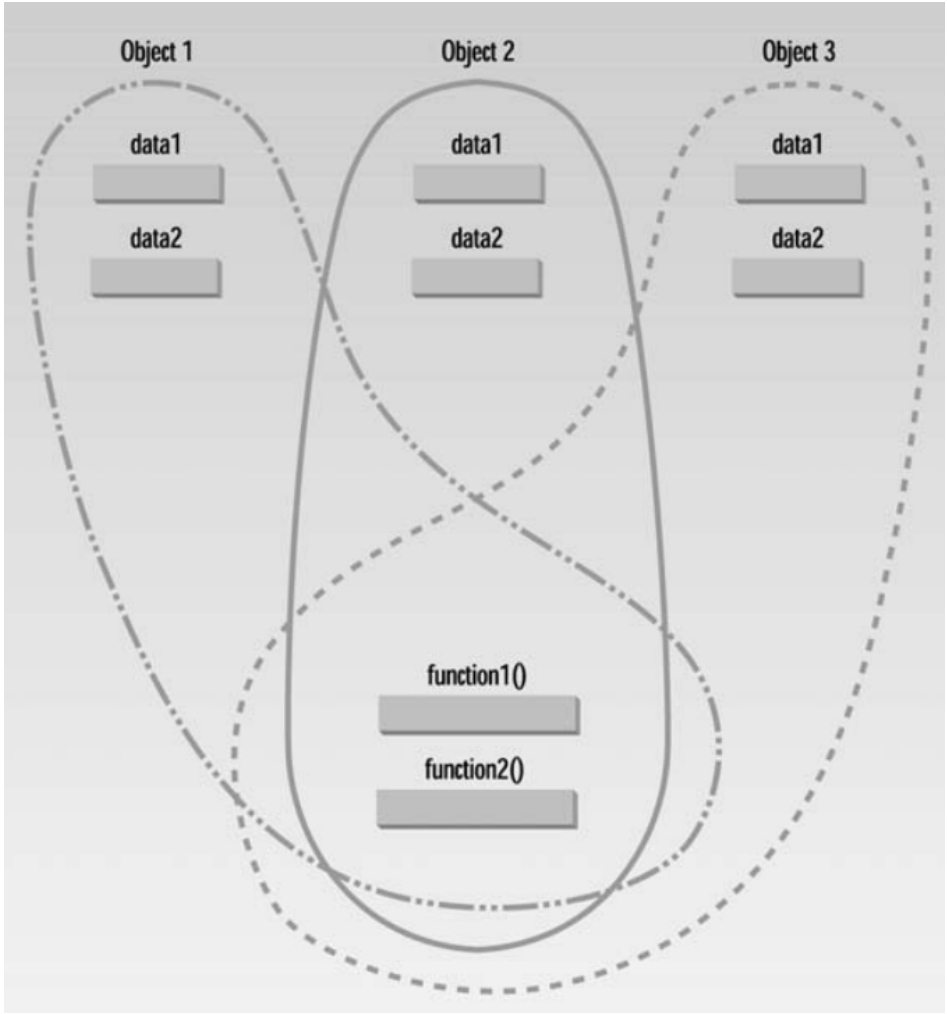
Array of Objects

```
1  class Employee
2  {
3      int id;
4      int salary;
5      public:
6          void setId() {
7              salary = 122;
8              cout << "Enter the id of employee" << endl;
9              cin >> id;
10         }
11         void getId() {
12             cout << "The id of this employee is " << id << endl;
13         }
14 };
15 int main() {
16     Employee arr[4];
17     for (int i = 0; i < 4; i++) {
18         arr[i].setId();
19         arr[i].getId();
20     }
21     return 0;
22 }
```

Objects, Classes and Memory

- Objects are self-contained entities, created according to a class definition.
- All objects of a given class share the same member functions.
- Member functions are created once when defined in the class and are not duplicated for each object.
- This is efficient, as functions are identical across objects.
- Each object has its own separate data items (also known as member variables or attributes).
- While the member functions are shared, the data values differ for each object.
- Separate instances of data are created in memory for each object when it is defined.

Objects, Classes and Memory



Static Class Data

- A static data member is shared among all objects of a class.
- Only one instance of the static data exists, regardless of how many objects are created.
- Static data is visible only within the class, but its lifetime spans the entire program.
- Continues to exist even if there are no objects of the class.
- Used to share information among all objects of a class.
- Similar to normal static variables, but specifically for class members.

Static Class Data

- Static data is useful when all objects need to share a common piece of information.
- Example: A static variable ‘count‘ can be used to track how many objects of a class exist.
- In a game, a race car object might use a static variable to know how many other cars are still in the race.

Non-static Data Members

```
1  class foo
2  {
3      private:
4          int count = 0;
5      public:
6          foo() { //increments count when object created
7              count++;
8          }
9          int getcount() {  //returns count
10             return count;
11         }
12     };
13 int main()
14 {
15     foo f1, f2, f3; //create three objects
16     cout << "count is " << f1.getcount() << endl;
17     cout << "count is " << f2.getcount() << endl;
18     cout << "count is " << f3.getcount() << endl;
19     return 0;
20 }
```

Static Data Members

```
1  class foo
2  {
3      private:
4          static int count; //only one data item for all objects
5                            //note: "declaration" only!
6      public:
7          foo() {//increments count when object created
8              count++;
9          }
10         int getcount() { //returns count
11             return count;
12         }
13 };
14 int foo::count = 0; //*definition* of count
15 int main()
16 {
17     foo f1, f2, f3; //create three objects
18     cout << "count is " << f1.getcount() << endl;
19     cout << "count is " << f2.getcount() << endl;
20     cout << "count is " << f3.getcount() << endl;
21     return 0;
22 }
```

Static Members Declaration

- Ordinary Variables:
  - Declared and defined in the same statement.
  - The compiler knows their name, type, and allocates memory.
- Static Member Data:
  - Requires two separate statements.
  - Declaration: Appears in the class definition.
  - Definition: Done outside the class, similar to a global variable.

Static vs Automatic Data Members