Classes

Lecture 22
Sections 7.1 - 7.4

Robb T. Koether

Hampden-Sydney College

Mon, Oct 24, 2018

Outline

Abstract Data Types

- An abstract data type (ADT) is a type of object that is described only by its behavior.
- We could describe sets abstractly by their behavior:
  - $\{a, b, c\} + \{a, c, d\} = \{a, b, c, d\}$. (set union)
  - $\{a, b, c, d\} - \{a, c\} = \{b, d\}$. (set difference)
  - $\{a, b, c\} * \{a, c, d\} = \{a, c\}$. (set intersection)
- We do not need to know the details of how sets are stored or how these operations are carried out

Outline

Classes

- A powerful feature of C++ is its mechanism to allow the programmer to create new data types.
- With a “little effort,” these new data types can be made as functional as the built-in types.

The `Point` Type

- For example, the programmer may need to write a program that involves points (x, y).
- One choice (not recommended) would be to create a pair of **`double`**s `x` and `y` and have the programmer "simply" remember that `x` and `y` are the coordinates of the same point.
- The program may involve hundreds of points (hundreds of `x`'s and `y`'s).

The `Point` Type

- The other choice (recommended) is to create a new `Point` data type.
- A `Point` object will be a *single object* with two components: **`doubles`** `x` and `y`.
- They behave as a "package;" where the point goes, `x` and `y` both go. We never have one without the other.
- This is accomplished through the class mechanism.

The `Point` Type

Using a `Point` Object

```
cout << "Enter two points:  ";
Point p;
Point q;
cin >> p >> q;
Point mid = (p + q)/2;
cout << "The midpoint is " << mid << endl;
```

- Once the `Point` class has been created, the programmer may work with `Point` objects as he would other objects.

The `Point` Type

Using a `Point` Object

```
cout << "Enter two points:  ";
Point p;
Point q;
cin >> p >> q;
Point mid = (p + q)/2;
cout << "The midpoint is " << mid << endl;
```

- Once the `Point` class has been created, the programmer may work with `Point` objects as he would other objects.
- What operators must be defined on points for this example to work?

Classes

- A class is another name for a data type.
- An object is an instance of a class.
- A class consists of
  - Data members.
  - Member functions.

Outline

Data Members

- Every object in a class has a specific set of data members, which are themselves objects (possibly instances of other classes).
- Each instance of the class has its own set of values of the data members, distinct from other instances of that class.
- For example, a `Point` object would have two **`doubles`**, with specific values.
- These data members record the state of the object.

Outline

Member Functions

- The member functions define the actions that are permissible on the object.
- For example, the `Point` class might have
  - A `getX()` function that will return the `x` coordinate of a `Point`.
  - A `setX()` function that will set the `x` coordinate of a `Point`.
  - An `output()` function that will output a `Point`.
  - An `isEqual()` function that will determine whether two `Point`s are equal.

Member Functions

Using a `Point` Object

```
double getX() const;
void setX(double x);
void output(ostream& out) const;
bool isEqual(const Point& p) const;
```

- These four functions would have the above prototypes.

Member Functions

Using a `Point` Object

```
double getX() const;
void setX(double x);
void output(ostream& out) const;
bool isEqual(const Point& p) const;
```

- These four functions would have the above prototypes.
- The keyword **`const`** at the end of the prototype means that the member function will not change the invoking object.

Member Functions

Using a `Point` Object

```
cout << "Enter two points:  ";
Point p, q;
p.input(cin);
q.input(cin);
double x = p.getX();
p.setX(x + 1.0);
if (p.isEqual(q))
   p.output(cout);
```

- Member functions are accessed through the dot operator.

Member Functions

Using a `Point` Object

```
bool operator==(const Point& p, const Point& q);
istream& operator>>(istream& in, Point& p);
ostream& operator<<(ostream& out, const Point& p);
```

- We can also define operators on class objects.

Member Functions

Using a `Point` Object

```
cout << "Enter two points:  ";
Point p, q;
cin >> p >> q;
double x = p.getX();
p.setX(x + 1.0);
if (p == q)
    cout << p << endl;
```

- Then the previous example becomes much more readable.

Outline

Member Access

- Access to each data member and each member function is controlled by the programmer.
- There are three levels of access.
  - Public access – The member may be accessed by any function.
  - Private access – The member may be accessed only by its own member functions (class scope).
  - Protected access – The member may be accessed only by its own member functions and member functions of derived classes (discussed in CS II).

Member Access

- Typically, data members are private.
  - This guarantees the integrity of the object.
  - Non-member functions can't change them.
- Typically, member functions are public.
  - This allows the rest of the program to perform the necessary actions on the objects.

Outline

Example of Using a Class

- Example
  - `point.h`
  - `point.cpp`
  - `Arclength.cpp`

Assignment

Assignment

- Read Sections 7.1 - 7.4.