Classes

Lecture 1
Sections 13.1 - 13.14

Robb T. Koether

Hampden-Sydney College

Mon, Jan 15, 2018

Outline

Classes

Class Construct

```
class Name
{
    // Member function prototypes
    // Declarations of data members
};
```

- The **`class`** construct in C++ is an enhancement of the **`struct`** construct in C.

Classes

The Class `Point`

```
class Point
{
    public:

    // Various member functions

    private:
        int m_x;
        int m_y;
};
```

Class Members

- Members may be designated **`public`**, **`private`**, or **`protected`**.
- Members are *private* by default.
- Members may be objects or functions.

Outline

Class Design

- Many of the member functions of a class fall into one of the following categories.
  - Constructor
  - Destructor
  - Inspector
  - Mutator
  - Operator
  - Facilitator

Outline

Constructors and Destructors

- A constructor creates a new object in a class.
- A destructor destroys an object.
- A class may have many constructors, but only one destructor.

Constructors and Destructors

Example (Point Class)

```
Point() : m_x(0), m_y(0) {}
Point(int x, int y) : m_x(x), m_y(y) {}
~Point() {}
```

Outline

Inspectors and Mutators

- An inspector returns an attribute of an object.
- A mutator modifies an attribute of an object.
- Typically, the attributes involved are the data members.
- Typically, inspectors are **`const`** and take no parameters.
- Typically, mutators take one or more parameters and they return **`void`**.
- Whenever appropriate, the mutators should test the parameters for validity.

Inspectors and Mutators

Example (Point Class)

```
int x() const {return m_x;}
void x(int x) {m_x = x;}
```

Outline

Operators and Facilitators

- An operator is a function that is invoked by a symbol such as + or *.
- A facilitator is invoked by a non-member operator to perform the function of the operator.
- Nearly every class should have the following operators.
  - The assignment operator =.
  - The output operator <<.

Equality and Relational Operators

- It is always sensible to define the operators == and !=.
- It is not always sensible to define the operators <, >, <=, and >=.
- If there is a sensible meaning of <, then the other three can be defined as well.
- If < is undefined, then operations such as sorting are impossible.

Input and Output Operators

- The output operator `<<` can be very useful for debugging.
- The input and output operators `>>` and `<<` should always be "compatible."
- That is, the input operator should be designed to read the same format that the output operator uses.
- The `Point` class outputs a point as `(2, 3)`.

Input and Output Operators

Example (Point Class)

```
void output(ostream& out) const
{
    out << '(' << m_x << ", " << m_y << ')';
}

ostream& operator<<(ostream& out, const Point& p)
{
    p.output(out);
    return out;
}
```

Outline

Member-Access Operators

Member-Access Operators

```
Point p;
cout << p.x() << ' ' << p.y();
p.x(5);
p.y(6);
```

- Use the dot operator . to access members through an object.

Outline

Structs

Struct Construct

```
struct Name
{
    // Declarations of data members
};
```

- C and C++ support the **`struct`** construct.

Structs

The Struct `Point`

```
struct Point
{
    int m_x;
    int m_y;
};
```

Struct Members

- In C:
  - Struct members are public.
  - Members may be objects, but not functions.
- In C++:
  - Struct members may be designated **`public`**, **`private`**, or **`protected`**.
  - Members are public by default.
  - Members may be objects or functions.

Outline

Assignment

Assignment

- Read Sections 13.1 - 13.14.