Review: C++ Basic Concepts

Dr. Yingwu Zhu

Outline

- C++ class declaration
- Constructor
- Overloading functions
- Overloading operators
- Destructor
- Redundant declaration

A Real-World Example

- Question #1: How to use a C++ class to represent a simple polynomial $ax + b$?

C++ Class Declaration

```
class ClassName
{
   public:
      Declarations of public members
   private:
      Declarations of private members
};
```

- Data members normally placed in private: section of a class (information hiding)
- Function members usually in public: section (exported for external use)

Polynomial *ax+b* Class

```
class   SimplePoly {
    private:
        int   m_a;   //coefficient for degree 1
        int   m_b;   //coefficient for degree 0
    public:
        /* other members later */
};
```

Constructor

- A function member
  - Initialize data members
  - Optional: allocate (dynamic) memory

```
ClassName::ClassName (parameter_list)
 : member_initializer_list
 {
     // body of constructor definition
    // member initialization can be
    // done here!
 }
```

- Default constructor: constructor with no arguments (automatically generated in the absence of explicit constructors; otherwise you need to provide if you want to use it)

Polynomial *ax+b* Class: Constructor

```
class   SimplePoly {
    private:
        int   m_a;   //coefficient for degree 1
        int   m_b;   //coefficient for degree 0
    public:
        SimplePoly(int  a, int b); //constructor
        /* other members later */
};

SimplePoly::SimplePoly(int a, int b) : m_a(a), m_b(b) {
}
```

Polynomial *ax+b* Class: Constructor

```
class   SimplePoly {
    private:
        int   m_a;   //coefficient for degree 1
        int   m_b;   //coefficient for degree 0
    public:
        SimplePoly(int  a, int b); //constructor
        /* other members later */
};

SimplePoly::SimplePoly(int a, int b) {
    m_a = a;
    m_b = b;
}
```

Overloading Functions

- Same function name, different signatures
- Compilers compare number and types of arguments (signature) of overloaded functions
- Note two constructors:

```
SimplePoly();
SimplePoly(int a, int b);
```

A Real-World Example

- Question #1: How to use a C++ class to represent a simple polynomial *ax + b* ?
- Question #2: How to make operators like +, -, *, /, <<, and >> work for SimplePoly *ax + b* class?

E.g.

```
SimplePoly p1 = p2 + p3;
cout << p1;
cin >> p1;
```

Overloading Operators

- Same symbol can be used more than one way
  - "+" used for int, float, double, …
- Operator → function

  △ → operator△()
- Two cases
  - If △ is a function member, compilers transform a △b → a.operator△(b)
  - Otherwise, a △b → operator△(a,b)
- Operators: +,-,*,/
- Operators: <<, >>

Overloading Operators

```
class   SimplePoly {
    private:
        int   m_a;
        int   m_b;
    public:
        SimplePoly(int  a, int b); //constructor
        SimplePoly() { m_a = m_b = 0; }   //default constructor
        SimplePoly   operator+(const  SimplePoly&  p);
};
ifstream&    operator>>(ifstream&  in, const  SimplePoly&  p);
ofstream&   operator<<(ofstream& out, const  SimplePoly&  p);
```

Overloading Operator+

```
SimplePoly  SimplePoly::operator+(const  SimplePoly&  p) {
    SimplePoly   p1;   //default constructor will be called!
    p1.m_a = m_a + p.m_a;
    p1.m_b = m_b + p.m_b;
    return p1;
}
```

Q: why pass p as a const reference?

Overloading Operator<<

```
ostream&  operator<<(ostream& out, const  SimplePoly&  p) {
    out << p.m_a << “x + ” << p.m_b << endl;
    return out;
}
```

Q: Anything wrong?

Overloading Operator<<

```
//add a function member
void  SimplePoly::display(ostream& out) {
     out << m_a << “x + ” << m_b << endl;
}

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

Q: Why pass p as a const reference?

Exercises

- Overload operator>>
- Overload operator-
- If you want to overload operator*, how would you do that?

Default Destructor

```
class   SimplePoly {
    private:
        int   m_a;
        int   m_b;
    public:
        SimplePoly() { m_a = m_b = 0; }  //default constructor
        SimplePoly(int  a, int b); //constructor
        ~SimplePoly() {};  //No need here!
          /* other members*/
};
```

Redundant Declarations

- Causes "redeclaration" errors at compile time
- Solution is to use conditional compilation
  - Use #ifndef and #define and #endif compiler directives

Redundant Declarations

```
//SimplePoly.h
#ifndef   _SIMPLE_PLOY_H
#define  _SIMPLE_PLOY_H

class   SimplePoly {
    private:
            int   m_a;
            int   m_b;
    public:
            SimplePoly(int  a, int b); //constructor
            /* other members*/
};

#endif
```

After-Class Practice

- Run this ax+b program and complete the three problems in the exercises