Computer Programming Object Oriented Programming & C++

1st Lecture

엄현상 (Eom, Hyeonsang)
Department of Computer Science and Engineering
Seoul National University

Outline

- Object Oriented Programming (OOP)
  - Basic Terms
  - Class in OOP
  - C++ Examples
  - C++ Constructor and Destructor
  - Other Stuff (Part of Overview)
  - Summary
  - Some Differences between C & C++
- Q&A

Basic Terms

- Object
  - Collection of Data and Operations on This Data
- Type
  - Characteristics Associated with Objects or Data Elements
- OOP
  - Programming with Objects
    - User-defined types

Class

- Means to Define Data Types
  - Collection of Members: Data Elements and Operations
    - E.g., Music CD Class

    - Possibly with Access Control
  - Used for Instantiation of Objects
- Means to Realize OOP Concepts
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Why Class or Why OOP

- Abstraction
  - To Extract the Essential Characteristics
- Encapsulation
  - To Hide Internal, Detailed Information
- Inheritance
  - To Reuse Existing Elements
- Polymorphism
  - To Select Class-Specific Implementations at Runtime

C++ Example: Abstraction

- Online Retailer Such as Amazon.Com
  - Item: Type, Title, Maker, Price, Availability, etc.

```
class Item { // Class definition
    public:
        String title; // String is a class defined earlier
        double price; // double is a predefined data type
        double SalePrice() { return (price*0.9);}
};

Item A;  // Class object definition

// OKAY: A.title, A.price, and A.SalePrice()
```

C++ Example: Encapsulation

- Online Retailer Example Cont'd

```
class Item { // Class definition
    public:
        String title;
        double price;
        double SalePrice() { return (price*0.9);}
        bool isAvailable() { return (inStockQuantity > 0); }
    private:
        int inStockQuantity;
};

Item A;  // Class object definition

// NOT OKAY: A.inStockQuantity
// OKAY: A.isAvailable()
```

C++ Example: Inheritance

- Online Retailer Example Cont'd

```
class MusicCDItem : public Item {
    public:
        String singer_name;
};

MusicCDItem B;   // Class object definition

// OKAY: B.singer_name, B.title, B.price, B.SalePrice(),
// and B.isAvailable()
// NOT OKAY: B.inStockQuantity
```

Derivation
- private: public & protected
  -> private
- protected: public & protected
  -> protected

  - Friendship

```
class Item {
friend class MusicCDItem;
...
```

C++ Example: Polymorphism

- Online Retailer Example Cont'd

```
class Item { // Class definition
    public:
        String title; // String is a class defined earlier
        double price; // double is a predefined data type
        double SalePrice() { return (price*0.9);}
        int isAvailable() { return (inStockQuantity > 0 ? 1 : 0); }
        virtual void specificInfo() {
            cout << "no Info: a base-class object" << endl; }
    private:
        int inStockQuantity;
};
```

```
virtual void specificInfo() = 0;
// pure virtual function:
// making this class be used
// only as a base class
```

C++ Example: Polymorphism Cont'd

- Online Retailer Example Cont'd

```
class MusicCDItem : public Item {
    public:
        String singer_name;
        void specificInfo() { cout << "singer name = " << singer_name
            << " : a derived-class object" << endl; }
};
void printSpecificInfo(Item *P) { P-> specificInfo(); }
Item A;  // Class object definition

MusicCDItem B;   // Class object definition
printSpecificInfo(&A); // Call Item::specificInfo()
printSpecificInfo(&B); // Call MusicCDItem::specificInfo()
// - Another derived class (e.g., MovieDVDItem) with specificInfo()
```

C++ Constructor and Destructor

- Example

```
#include <assert.h>
class String {
public:
    String(const char *s) {
        len = strlen(s);
        str = new char[len + 1];
        assert(str != 0);
        strcpy(str,s);
    }
    ~String() { delete [] str; }
private:
    int len;
    char *str;
};
```

String(int ln) { ... }
// Function overloading
// String buf = 1024;

String() { ... }
// Default constructor
// String st(); -> Error

```
String name0 = String("Andrew");  // Definition
String name1("Karl");
String *name_ptr = new String("Thomas");
delete name_ptr;  // Explicit destruction
```

Other Stuff

- Overloading (w/ Distinguished Argument Lists)
  - Function
    - Operator

Reference type
E.g., int *&ipr;

```
String& String::operator+=(const String &s) {
    len += s.len;
    char *p = new char[len+1];
    assert(p != 0);
    strcpy(p, str);
    strcat(p, s.str);
    delete str;
    str=p;
    return *this;
}
```

Address of the invoking class object

```
String s1("Thank ");
s1 += "you!";
```

call String(const char *s) first

Other Stuff Cont'd

- Reference Type
  - Reference Object to Be Initialized
    - Unable to alias another object once initialized

```
int &refVal = val; // int &const refVal = val;
const int &cir = 1024;
```

OK: uc, d1+d2
// unsigned char uc;
// double d1, d2;

- Class Template
  - Automatic Generation of Class Instances Bound to a Particular Type

```
template <class SDT>
class Stack { ...
```

```
Stack<int> s;                    // typedef int SDT
```

Other Stuff Cont'd

- Multiple Inheritance
  - Child Class as a Composite of Its Multiple Base Classes

```
Class C : public A, public B { ... }
```

    - Qualification to resolve ambiguity

e.g., A::a or B::a in C::func()

- Dominance in the Inheritance Chain
  - Most Derived Instance Dominating

e.g., C::func() dominates over A::func()

Summary

- Class
  - To Define New Types in OOP
  - To Realize OOP Concepts:
    - Abstraction
    - Encapsulation
    - Inheritance
    - Polymorphism

cf, C++ Primer by Stanley B. Lippman, Addison Wesley

Some Differences between C & C++

- Type Checking (Regarding Function Declarations)
  - Meaning of No Argument
    - ANSI C: zero or more arguments of any data type
    - C++: no argument
  - Effect of No Declaration
    - ANSI C: permitted
    - C++: error
- C++ Support for Default Arguments

```
void new_line(int n =1) {
    while (n-- > 0) putchar('\n');
}
```

```
new_line(2);
new_line();
```

- Dynamic Memory Allocation