

# Computer Programming

## Lecture 10

이윤진  
서울대학교  
2007.1.21.

# Slide Credits

- 엄현상 교수님
  - 서울대학교 컴퓨터공학부
  - Computer Programming, 2007 봄학기

# Libraries

# 순서

- Libraries
  - Standard Libraries
- Q&A

# Standard Libraries

- Environment That Supports Standard C

- ANSI (American National Standards Institute) C

`/usr/include`

`/usr/lib/gcc-lib/i386-linux/2.95.4/include`

- `<assert.h>` `<float.h>` `<ctype.h>` `<errno.h>` `<limits.h>` `<locale.h>`  
`<math.h>` `<setjmp.h>` `<signal.h>` `<stdarg.h>` `<stddef.h>` `<stdio.h>`  
`<stdlib.h>` `<string.h>` `<time.h>`

- cf, POSIX (Portable Operating System Interface for UNIX) by IEEE

- `<fcntl.h>` `<sys/stat.h>` `<sys/types.h>` etc.

# Standard Libraries (계속)

- `<assert.h>` Diagnostics
  - contains only the `assert` macro
- `<float.h>` Characteristics of Floating Types
  - provides macros that describe the characteristics of floating types, including their range and accuracy
- `<ctype.h>` Character Handling
  - provides functions for classifying characters and for converting letters from lower to upper case or vice versa
- `<errno.h>` Errors
  - provides `errno` (“error number”)
- `<limits.h>` Sizes of Integral Types
  - provides macros that describe the characteristics of integer and character types, including their maximum and minimum values
- `<locale.h>` Localization
  - provides functions to help a program adapt its behavior to a country or other geographic region
- `<math.h>` Mathematics
  - provides a variety of common mathematical functions

# Standard Libraries (계속)

- `<setjmp.h>` Nonlocal Jumps
  - provides the `setjmp` and `longjmp` functions
- `<signal.h>` Signal Handling
  - provides functions that deal with exceptional conditions, including interrupts and run-time errors
- `<stdarg.h>` Variable Arguments
  - provides tools for writing functions that can have a variable number of arguments
- `<stddef.h>` Common Definitions
  - provides definitions of frequently used types and macros
- `<stdio.h>` Input/Output
  - provides a large assortment of I/O functions
- `<stdlib.h>` General Utilities
  - a catch-all header for functions that don't fit into any of the other headers
- `<string.h>` String Handling
  - provides functions that perform string operations
- `<time.h>` Date and Time
  - provides functions for determining the time, manipulating times, and displaying times in a variety of ways

# Errors: <errno.h>

- To Permit Signaling an Error

```
extern int errno;
...
#define ENOENT    2 /* No such file or directory */
...
#define EACCES   13 /* Permission denied */
...
```

- Example

```
#include <stdio.h>
#include <errno.h>
main (int argc, char *argv[]) {
    fprintf(stderr, "EACCES: %s\n", strerror(EACCES));

    perror(argv[0]);
    FILE *fp = fopen("aaa.c", "r");
    perror(argv[0]);
    errno = ENOENT;
    perror(argv[0]);
}
```

```
bacardi:~$ a.out
EACCES: Permission denied
./a.out: Success
./a.out: No such file or directory
./a.out: No such file or directory
```

# Non-Local Jumps: <setjmp.h>

- To Avoid the Normal Function Call and Return Sequence

```
#include <stdio.h>
#include <setjmp.h>
jmp_buf jb;
void prnchar() {
    char c;
    if ((c = getchar()) != '\n') {
        if (c == EOF)
            longjmp(jb,1);
        prnchar();
        putchar(c);
    }
}
main(){
    if (setjmp(jb) != 0) { /* return 0 if called directly */
        printf("ERROR unexpected EOF\n", );
        exit(1);
    }
    prnchar();
    printf("\n");
}
```

Array holding all the information to restore the status of the stack

Recursive call

```
bacardi:~$ cat > temp
abc[Ctrl-D][Ctrl-D]
bacardi:~$ reverse < temp
ERROR unexpected EOF
```

## [참고] Local Jumps

- To Abandon Processing in Some Deeply Nested Structure
  - Not Good: Making Error Handling Non-Trivial

```
for (i=0; i<n; i++)  
  for (j=0; j<m; j++)  
    if (a[i] == b[j])  
      goto found;
```

breaking out of two or more loops at once

...

```
return;
```

```
found:
```

```
/* work to be done if found */
```

Label

```
found = 0;  
for (i=0; i<n && !found; i++)  
  for (j=0; j<m && !found; j++)  
    if (a[i] == b[j])  
      found = 1;
```

Recommended version

...

```
If (found)
```

```
/* work to be done if found */
```

# Variable Arguments: <stdarg.h>

- To Permit Handling a Variable Number of Arguments

```
typedef ... va_list;  
void va_start(valist ap, lastarg);  
type va_arg(va_list ap, type);  
void va_end(va_list ap);
```

Default argument promotion  
char, short int → int  
float → double  
no char, short int or float

- Example

```
#include <stdarg.h>  
int max(int n, ...) {  
    va_list ap;  
    int i, c, l;  
    va_start(ap, n);  
    l = va_arg(ap, int);  
    for(i = 1; i < n; i++){  
        c = va_arg(ap, int);  
        if (c > l) l = c;  
    }  
    va_end(ap);  
    return l;  
}
```

```
max(4, 1, 2, 6, 5);  
max(3, 5, 7, 6);
```

# Input/Output: <stdio.h>

- To Permit Handling Buffer Allocation, and Performing I/O in Optimal-Sized Chunks
  - Stream (File Pointer): e.g., Standard Input
    - Buffering (Standard I/O Buffer; cf, Buffer Cache)
      - Full Buffering (\_IOFBF)
      - Line Buffering (\_IOLBF)
      - No Buffering (\_IONBF)

```
#include <stdio.h>
```

```
...
```

```
fflush(stdout); /* return EOF for a write error, and 0 otherwise */
```

Standard output buffer flush;  
fflush(NULL) for all output streams  
*Note: fflush() only flushes the user space buffers provided by the C library*

```
#include <unistd.h> /* optional */
```

```
...
```

```
fsync(1); /* return 0 if OK, and -1 otherwise */
```

Buffer cache flush;  
sync() for all modified block buffers

# Object-Oriented Programming (1)

# 순서

- Object Oriented Programming (OOP)
  - Basic Terms
  - Class in OOP
  - C++ Examples
  - C++ Constructor and Destructor
  - Multiple Inheritance
  - Other Stuff
  - 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



Title  
Price  
SalePrice()

# 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
```

# 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()

# 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
```

# 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
  - new, delete