
*Object Oriented Programming by C++*

# Software Component (1/2)

**Basic of Class and Object**

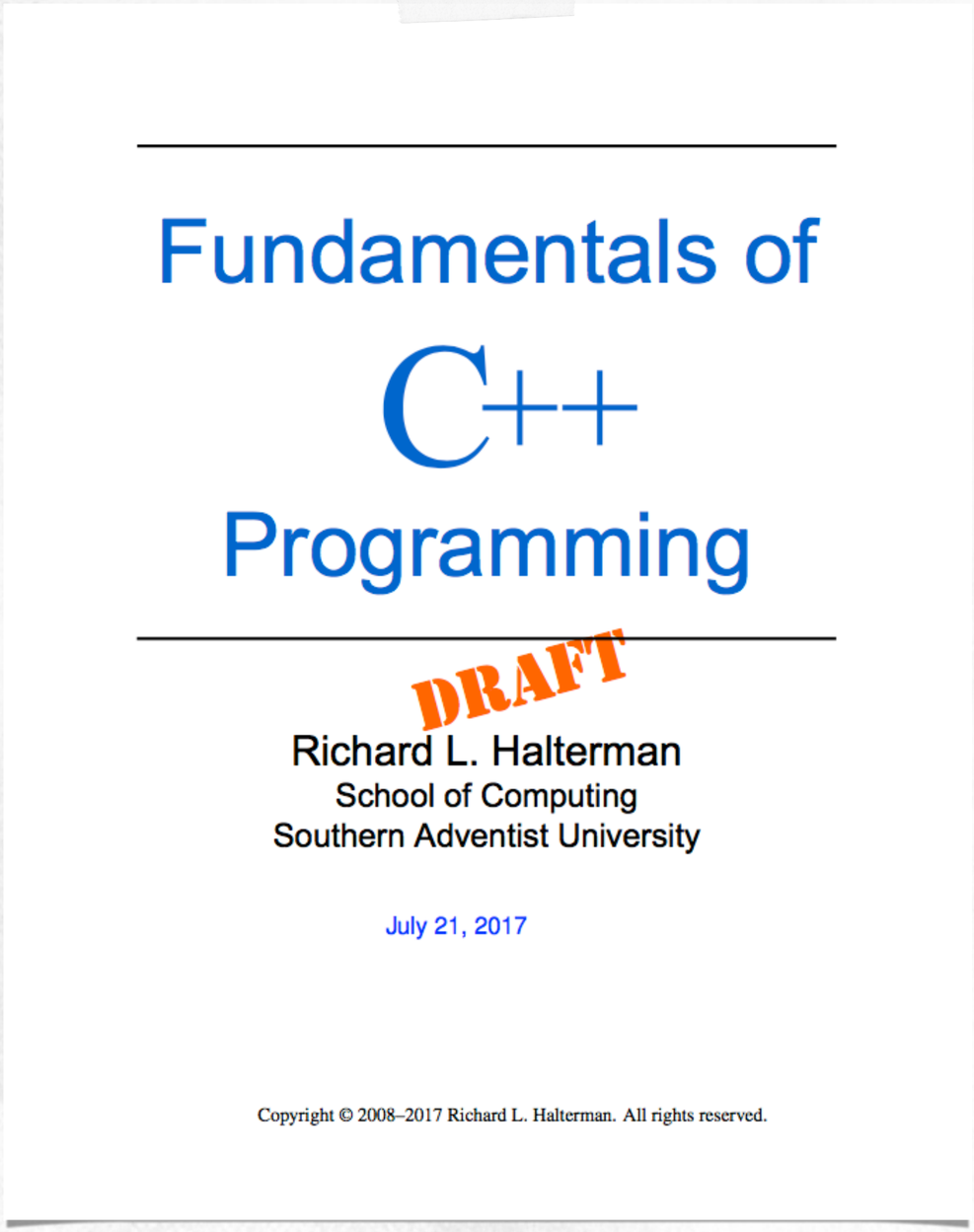
2017. 8.

Sungwon Lee / Professor

Email: drsungwon@khu.ac.kr
Web: http://mobilelab.khu.ac.kr/

# Textbook & Copyright

- Textbook: http://python.cs.southern.edu/cppbook/progcpp.pdf
- Sample Codes: https://github.com/halterman/CppBook-SourceCode



Fundamentals of

C++

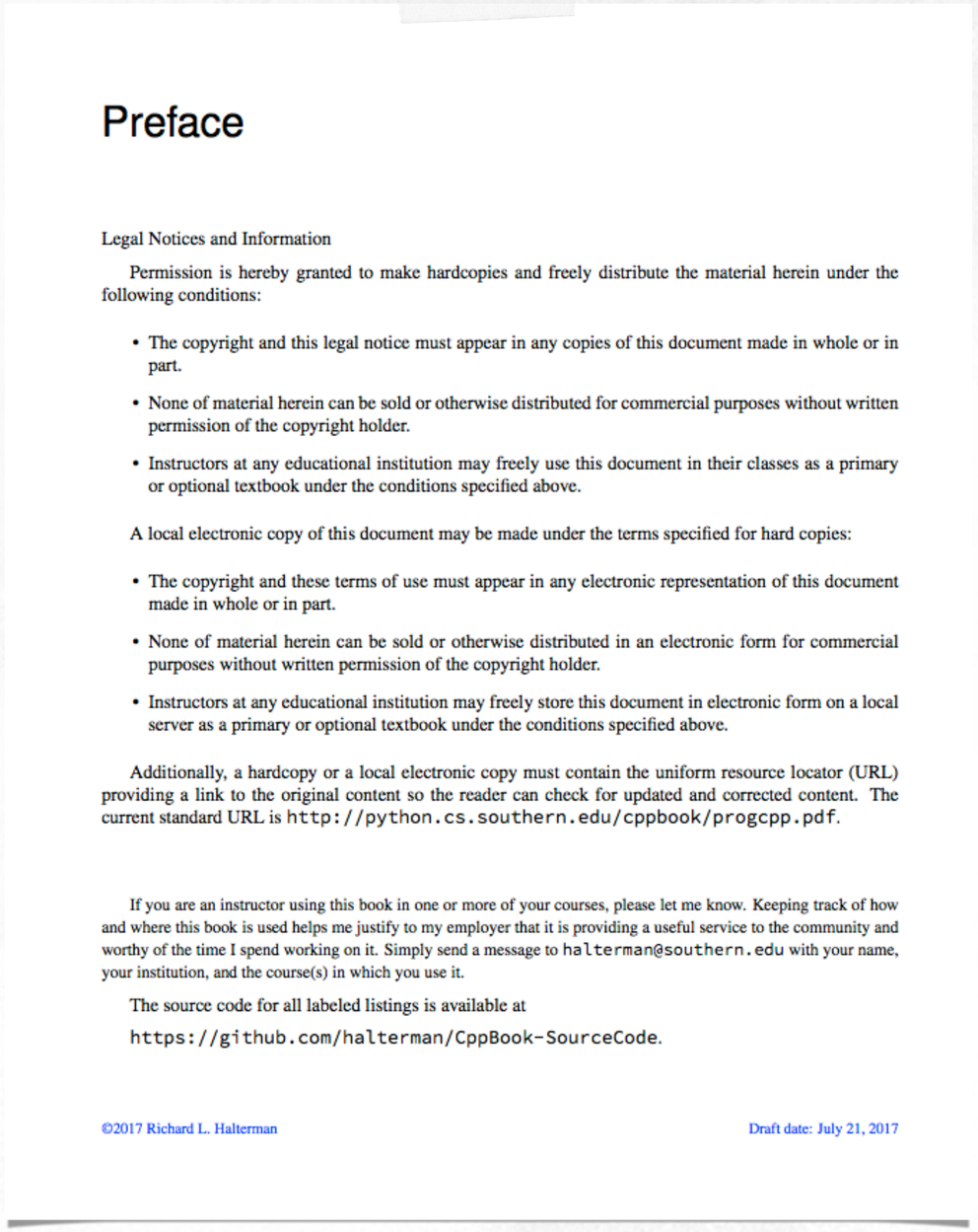
Programming

DRAFT

Richard L. Halterman
School of Computing
Southern Adventist University

July 21, 2017





## Preface



If you are an instructor using this book in one or more of your courses, please let me know. Keeping track of how and where this book is used helps me justify to my employer that it is providing a useful service to the community and worthy of the time I spend working on it. Simply send a message to `halterman@southern.edu` with your name, your institution, and the course(s) in which you use it.

The source code for all labeled listings is available at

`https://github.com/halterman/CppBook-SourceCode`.

 Draft date: July 21, 2017

# Contents

- Remind: Pointer and Array Notation
- Software Component
  - Class
  - Object
- Class Example 1
- Class Definition
- More about Class Definition
- Encapsulation
- Class Example 2
- FYI: auto

Remind

# Pointer and Array Notation

when p is a pointer,

p[i] is *(p+i)

p[j][i] is *(*(p+j)+i)

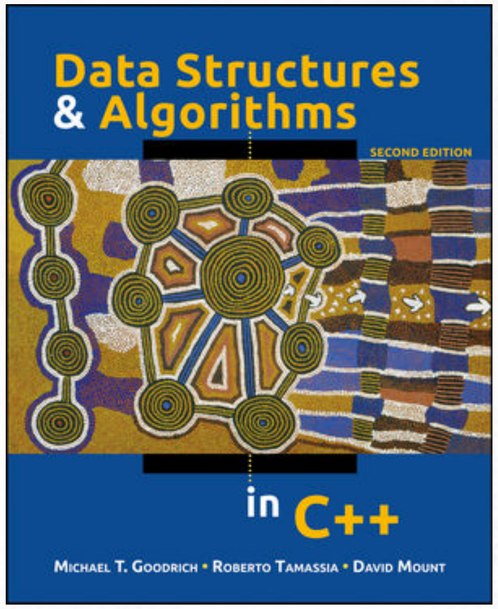
Remind

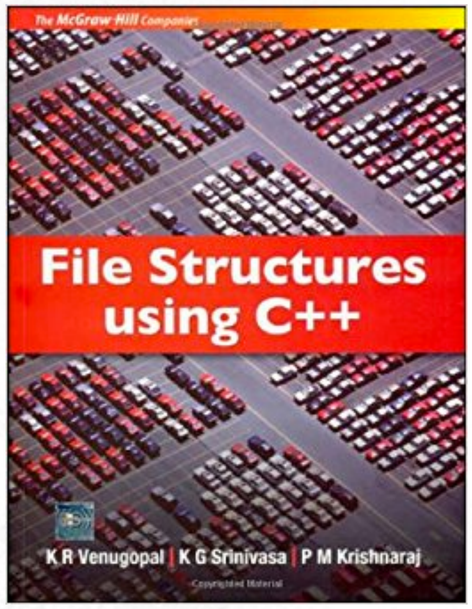
# References

Managing File Intelligently: *FILE STRUCTURE*

Sorting and Searching: *ALGORITHM*

Managing Information Intelligently: *DATA STRUCTURE*



# Software Component = Data + Function

- We already used:
  - cin, cout
  - string
  - fstream, ifstream, ofstream

- Software Component includes:
  - Data: file itself in fstream
  - Function: operations (open, close, etc,.) for file in fstream

# Class and Object

- Class
  - Design document (not a real thing)
  - Example: ***fstream***

- Object
  - Realization of a Class (real thing with specific information)
  - Example: fstream ***myFile***;

- *Object* is an instance of a *Class*
  - ***A class is a programmer-defined type***
  - ***An object is an instance of a class***
    - The terms object and instance may be used interchangeably.

# How can we make Class?

1. ***Specifying the data*** that constitute an object's state,
2. ***Defining the code to be executed on an object***'s behalf that provides services to clients that use the object,
3. ***Defining code that automatically initializes a newly-created object*** ensuring that it begins its life in a well-defined state, and
4. ***Specifying*** which parts of objects are ***visible to clients*** and which parts are ***hidden from clients***.

Class Example 1

# Point Class (with data only = *member data*)

```
class Point {
public:
    double x;
    double y;
};
```

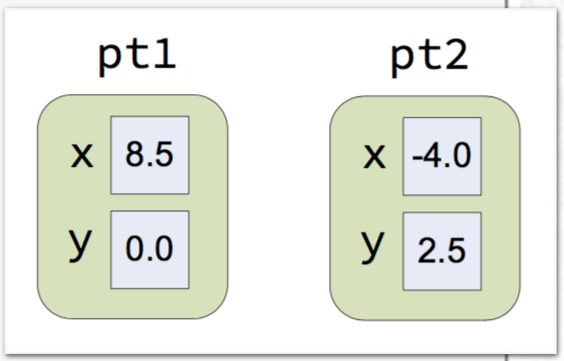
**Listing 14.1: mathpoints.cpp**

```
#include <iostream>

// The Point class defines the structure of software
// objects that model mathematical, geometric points
class Point {
public:
    double x;   // The point's x coordinate
    double y;   // The point's y coordinate
};

int main() {
    // Declare some point objects
    Point pt1, pt2;
    // Assign their x and y fields
    pt1.x = 8.5; // Use the dot notation to get to a part of the object
    pt1.y = 0.0;
    pt2.x = -4;
    pt2.y = 2.5;
    // Print them
    std::cout << "pt1 = (" << pt1.x << "," << pt1.y << ")\n";
    std::cout << "pt2 = (" << pt2.x << "," << pt2.y << ")\n";
    // Reassign one point from the other
    pt1 = pt2;
    std::cout << "pt1 = (" << pt1.x << "," << pt1.y << ")\n";
    std::cout << "pt2 = (" << pt2.x << "," << pt2.y << ")\n";
    // Are pt1 and pt2 aliases?  Change pt1's x coordinate and see.
    pt1.x = 0;
    std::cout << "pt1 = (" << pt1.x << "," << pt1.y << ")\n";
    // Note that pt2 is unchanged
    std::cout << "pt2 = (" << pt2.x << "," << pt2.y << ")\n";

}
```




```
pt1.x = pt2.x;
pt1.y = pt2.y;
```

```
pt1 = pt2;
```

# Account Class (with vector and sorting)

```
class Account {
public:
    std::string name;   // The name of the account's owner
    int id;             // The account number
    double balance;     // The current balance
};
```

```
std::vector<Account> accounts(5000);
```

**Let's see Listing 14.2**

Class Definition

# Account Class (with member function)

- Member Function
  - Function inside Class
  - Manipulate member data
  - '.' notation is not required to access member data *in same Object*
  - *ObjectName.DataName* is required to access member data *in other Object*
  - Member function definition can be *inside or outside of Class declaration*

**Listing 14.3: newaccount.cpp**

```
class Account {
    // String representing the name of the account's owner
    string name;
    // The account number
    int id;
    // The current account balance
    double balance;
public:
    /*********************************************************
     *  deposit(amt)
     *     Adds amount amt to the account's balance.
     *
     *     Author: Sam Coder
     *     Date: April 17, 2017
     *********************************************************/
    void deposit(double amt) {
        balance += amt;
    }

    /*********************************************************
     *  withdraw(amt)
     *     Deducts amount amt from the account's balance,
     *     if possible.
     *     Returns true if successful; otherwise, it returns false.
     *     A call can fail if the withdraw would
     *     cause the balance to fall below zero
     *
     *     amt:  funds to withdraw
     *
     *     Author: Sam Coder
     *     Date: April 17, 2017
     *********************************************************/
    bool withdraw(double amt) {
        bool result = false;  // Unsuccessful by default
        if (balance - amt >= 0) {
            balance -= amt;
            result = true;  // Success
        }
        return result;
    }
};
```

# Account Class (member function calling)

**Listing 14.3: newaccount.cpp**

```
class Account {
    // String representing the name of the account's owner
    string name;
    // The account number
    int id;
    // The current account balance
    double balance;
public:
    /**********************************************************
     *  deposit(amt)
     *     Adds amount amt to the account's balance.
     *
     *     Author: Sam Coder
     *     Date: April 17, 2017
     **********************************************************/
    void deposit(double amt) {
        balance += amt;
    }

    /**********************************************************
     *  withdraw(amt)
     *     Deducts amount amt from the account's balance,
     *     if possible.
     *     Returns true if successful; otherwise, it returns fa
     *     A call can fail if the withdraw would
     *     cause the balance to fall below zero
     *
     *     amt:  funds to withdraw
     *
     *     Author: Sam Coder
     *     Date: April 17, 2017
     **********************************************************/
    bool withdraw(double amt) {
        bool result = false;  // Unsuccessful by default
        if (balance - amt >= 0) {
            balance -= amt;
            result = true;  // Success
        }
        return result;
    }
};
```

**Same usage with standard Classes (fstream, string, etc)**

```
// Affects the balance field of acct1 object
acct1.withdraw(100.00);
// Affects the balance field of acct2 object
acct2.withdraw(25.00);
```

# Account Class (enhance with Constructor - 1/3)

- How can we make sure *the fields of an object have reasonable initial values* before a client begins using the object?
  - A class may define a ***constructor*** that ensures an object will begin in a well-defined sate.
  - A constructor definition *looks similar to a method definition*.
  - The code within a constructor executes on behalf of an object *when a client creates the object*.
  - For some classes, the client can provide information for the constructor to use when initializing the object.
  - ***A constructor has the same name as the class.***
  - ***A constructor has no return type, not even void.***

# Account Class (enhance with Constructor - 2/3)

**Listing 14.4: bankaccountmethods.cpp**

```
#include <iostream>
#include <iomanip>
#include <string>

class Account {
    // String representing the name of the account's owner
    std::string name;
    // The account number
    int id;
    // The current account balance
    double balance;
public:
    // Initializes a bank account object
    Account(const std::string& customer_name, int account_number,
            double amount):
            name(customer_name), id(account_number), balance(amount) {
        if (amount < 0) {
            std::cout << "Warning: negative account balance\n";
            balance = 0.0;
        }
    }

    // Adds amount amt to the account's balance.
    void deposit(double amt) {
        balance += amt;
    }
```

**Initialization list, explained later**

```
Name: Joe, ID: 2312, Balance: 1000
Name: Moe, ID: 2313, Balance: 500.29
--------------------
Name: Joe, ID: 2312, Balance: 200
Name: Moe, ID: 2313, Balance: 522.29
```

# Class Definition

# Account Class (enhance with Constructor - 3/3)

```
    // Deducts amount amt from the account's balance,
    // if possible.
    // Returns true if successful; otherwise, it returns false.
    // A call can fail if the withdraw would
    // cause the balance to fall below zero
    bool withdraw(double amt) {
        bool result = false;  // Unsuccessful by default
        if (balance - amt >= 0) {
            balance -= amt;
            result = true;  // Success
        }
        return result;
    }

    // Displays information about the account object
    void display() {
        std::cout << "Name: " << name << ", ID: " << id
                  << ", Balance: " << balance << '\n';
    }
};


int main() {
    Account acct1("Joe", 2312, 1000.00);
    Account acct2("Moe", 2313, 500.29);
    acct1.display();
    acct2.display();
    std::cout << "--------------------" << '\n';
    acct1.withdraw(800.00);
    acct2.deposit(22.00);
    acct1.display();
    acct2.display();
}
```

```
Name: Joe, ID: 2312, Balance: 1000
Name: Moe, ID: 2313, Balance: 500.29
--------------------
Name: Joe, ID: 2312, Balance: 200
Name: Moe, ID: 2313, Balance: 522.29
```

# More about Constructor (1/3)

- If you do not define a constructor for your class,
  - the compiler automatically will create one for you—a default constructor that accepts no parameters
  - The compiler-generated constructor does not do anything to affect the state of newly created instances.
  - Example: `Account acct1; // Legal, do nothing.`

- If you define any constructor for your class,
  - the compiler will not provide a default constructor
  - Example: ***if you define your own constructor as in previous slide***

```
Account acct1; // Illegal, you should define
               // your own constructor.
```

# More about Constructor (2/3)

- If multiple Constructor is defined, C++ selects approprate one.

```
class S {
    int n;

public:

    S(int) // constructor definition.
    {
        n = 1;
    }

    S() // constructor definition.
    {
        n = 0;
    }
};

int main()
{
    S s; // calls S::S()
    S s2(10); // calls S::S(int)
}
```

# More about Constructor (3/3)

- With constructors, unlike with normal methods, we can use curly braces in place of parentheses, as in

```
// Client creating two Account objects
Account acct1{"Joe", 2312, 1000.00};
Account acct2{"Moe", 2313, 500.29};
```

- Which is same with

```
// Client creating two Account objects
Account acct1("Joe", 2312, 1000.00);
Account acct2("Moe", 2313, 500.29);
```

*Strongly Recommended*

# Member Function definition out of Class

- If multiple Constructor is defined, C++ selects approprate one.

```
class S {
   int n;

public:

   S(int); // constructor declaration.

   S() // constructor definition.
   { n = 0; }
};

S::S(int) // constructor definition.
{
    n = 1;
{

int main()
{
   S s; // calls S::S()
   S s2(10); // calls S::S(int)
}
```

**S::S(int)**

function S(int)
in Class S

# Initialization List

- Easy way to initialize member data in Constructor

```
class S {
   int n;

public:

   S(int); // constructor declaration.

   S() : n(0) // set member 'n' as '0'
   {} // constructor definition.
};

S::S(int) // constructor definition.
{
    n = 1;
{

int main()
{
   S s; // calls S::S()
   S s2(10); // calls S::S(int)
}
```

# What is different?

OK

```
#include <iostream>
using namespace std;

class Adder {
  public:
    // constructor
    Adder(int i = 0) {
      total = i;
    }

    // interface to outside world
    void addNum(int number) {
      total += number;
    }

    // interface to outside world
    int getTotal() {
      return total;
    };

  private:
    // hidden data from outside world
    int total;
};

int main() {
  Adder a;

  a.addNum(10);
  a.addNum(20);
  a.addNum(30);

  cout << "Total " << a.getTotal() <<endl;
  return 0;
}
```

Fail

```
#include <iostream>
using namespace std;

class Adder {
  public:
    // constructor
    Adder(int i = 0) {
      total = i;
    }

    // interface to outside world
    void addNum(int number) {
      total += number;
    }

    // interface to outside world
    int getTotal() {
      return total;
    };

  private:
    // hidden data from outside world
    int total;
};

int main() {
  Adder a;

  a.addNum(10);
  a.addNum(20);
  a.addNum(30);

  cout << "Total " << a.total <<endl;
  return 0;
}
```

# Hided Data and Codes

- **private:**
  - The variables *length*, *breadth*, and *height* are private
  - This means that **they can be accessed only by other members of the Box class**, and not by any other part of your program
  - This is one way ***encapsulation*** is achieved
  - C++'s ***default*** mode

```
class Box {
  public:
    double getVolume(void) {
      return length * breadth * height;
    }

  private:
    double length;     // Length of a box
    double breadth;    // Breadth of a box
    double height;     // Height of a box
};
```

# Opened Data and Codes

- **public:**
  - you must declare them ( *getVolume()* ) after the public keyword
  - All variables or functions defined after the public specifier are **accessible by all other functions in your program**.

```
class Box {
  public:
    double getVolume(void) {
      return length * breadth * height;
    }

  private:
    double length;     // Length of a box
    double breadth;    // Breadth of a box
    double height;     // Height of a box
};
```

# Opened Data and Codes, but not today…

- **protected:**
  - ***explained in Week.12***

# Why Encapsulation?

- Software that is more flexible and resilient to change,
- Software that is more robust and reliable, and
- a software development process that programmers can more easily comprehend and manage.

# Simple but Powerful Rule

A good rule of thumb in class design is this: make data `private`, and make methods that provide a service to clients `public`.

A good rule of thumb in class design is this: make data `private`, and make methods that provide a service to clients `public`.

A good rule of thumb in class design is this: make data `private`, and make methods that provide a service to clients `public`.

Class Example 2

# Code Review

**Let's see Listing 14.5**

# For Your Information

## auto

- The auto keyword allows the compiler to ***automatically deduce the type of a variable*** if it is initialized when it is declared:

```
auto count = 0;
auto ch = 'Z';
auto limit = 100.0;
```

- The auto keyword ***may not be used without an accompanying initialization***; for example, the following declaration is illegal, because the compiler cannot deduce x's type.

```
auto x;
```

Automatic type inference is supported only by compilers that comply with the latest C++11 standard. Programmers using older compilers must specify a variable's exact type during the variable's declaration.

Required

# CODE REVIEW

**Code Review is the best way to understand Class & Object.**

**Please “Step into” your own software component using Visual Studio.**

**{ Recommended example codes are Listing 14.1 to 14.5 in Textbook }**


# *Object Oriented Programming by C++*

Sungwon Lee / Professor

Email: drsungwon@khu.ac.kr

Web: http://mobilelab.khu.ac.kr/