Class

Prof. Jinkyu Jeong ([link](mailto:jinkyu@skku.edu))
TA -- Minwoo Ahn ([link](mailto:minwoo.ahn@csl.skku.edu))
TA -- Donghyun Kim ([link](mailto:donghyun.kim@csl.skku.edu))
Computer Systems Laboratory
Sungkyunkwan University
[link](http://csl.skku.edu)
Agenda

• Class definition
• Class examples
• Objects
• Constructors
• Destructors
Object oriented programming

Vehicle class

Inheritance

Car class

Car Objects

Green
Ford
Mustang
Gasoline

Red
Toyota
Prius
Electricity

Blue
Volkswagen
Golf
Deisel

C code vs. C++ code

• C code

int funcA(...) {...}
int funcB(...) {...}
int funcC(...) {...}

int main() {
    funcA();
    funcB();
    funcC();
}

• C++ code

class classA {
    private: …

    public:
    int methodA() {...}
    int methodB() {...}
    int methodC() {...}
};

int main() {
    classA A;
    A.methodA();
    A.methodB();
    A.methodC();
}

Classes & Objects

- Objects: Instance of a class

```cpp
class Rectangle 
{
  private:
    int width;
    int length;
  public:
    void set(int w, int l);
    int area();
};
```

Rectangle r1;
Rectangle r2;
Rectangle r3;

```
  r1
  length = 3;
  width = 5;

  r2
  length = 4;
  width = 2;

  r3
  length = 7;
  width = 6;
```
Define a Class Type

**Header**

```cpp
class class_name
{
    
};
```

**Body**

```cpp
permission_label:
    member;

permission_label:
    member;

...
```

```cpp
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```
Class Definition – Data Members

- Can be of any type, built-in or user-defined

- *non-static* data member
  - Each class object has its own copy

- *static* data member
  - Acts as a global variable
  - One copy per class type, e.g. counter
class Rectangle {
    private:
        int width;
        int length;
        static int count;
    public:
        void set(int w, int l);
        int area();
};

Rectangle r1;
Rectangle r2;
Rectangle r3;
Class Definition – Member Functions

• Used to
  – Access the values of the data members (accessor)
  – Perform operations on the data members (implementor)

• Are declared inside the class body

• Their definition can be placed inside the class body, or outside the class body

• Can access both public and private members of the class

• Can be referred to using dot or arrow member access operator
Define a Member Function

class Rectangle
{
    private:
        int width, length;
    public:
        void set (int w, int l);
        int area() {return width*length; }
};

void Rectangle :: set (int w, int l)
{
    width = w;
    length = l;
}

r1.set(5,8);
rp->set(8,10);
Define a Member Function (cont’d)

• const member function
  – Declaration
    • `return_type func_name (param_list) const;`
  – Definition
    • `return_type func_name (param_list) const {…}`  Inline definition
    • `return_type class_name::func_name (param_list) const {…}
  – It is illegal for a `const` member function to modify a class data member
**Const Member Function**

```cpp
class Time {
private:
    int hrs, mins, secs;
public:
    void Write() const;
};

void Time::Write() const {
    cout << hrs << ":" << mins << ":" << secs << endl;
}
```
Class Definition – Access Control

• Information hiding
  – To prevent the internal representation from direct access from outside the class

• Access specifiers
  – public
    • May be accessible from anywhere within a program
  – private
    • May be accessed only be the member functions, and friends of this class
  – protected
    • Acts as public for derived classes
    • Behaves as private for the rest of the program
Class Time Specification

class Time
{

public :
    void Set ( int hours , int minutes , int seconds ) ;
    void Increment ( ) ;
    void Write ( ) const ;
    Time ( int initHrs , int initMins , int initSecs ) ; // constructor
    Time ( ) ; // default constructor

private :
    int hrs ;
    int mins ;
    int secs ;

} ;
Class Interface Diagram

Time class

- Set
- Increment
- Write
- Time

Private data:
- hrs
- mins
- secs

Cannot access
What is an object?

OBJECT

- Operations (set of methods, member functions)
- Data (internal state, values of private data members)
Declaration of an Object

class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

main()
{
    Rectangle r1;
    Rectangle r2;
    r1.set(5, 8);
    cout<<r1.area()<<endl;
    r2.set(8,10);
    cout<<r2.area()<<endl;
}
Another Example

```cpp
#include <iostream.h>

class circle
{
    private:
        double radius;

    public:
        void store(double);
        double area(void);
        void display(void);
};

// member function definitions
void circle::store(double r)
{
    radius = r;
}

double circle::area(void)
{
    return 3.14*radius*radius;
}

void circle::display(void)
{
    cout << "r = " << radius << endl;
}

int main(void) {
    circle c;   // an object of circle class
    c.store(5.0);
    cout << "The area of circle c is " << c.area() << endl;
    c.display();
}
```
Declaration of an Object

class Rectangle
{
  private:
    int width;
    int length;
  public:
    void set(int w, int l);
    int area();
};

main()
{
  Rectangle r1;
  r1.set(5, 8);
}

r1 is statically allocated

r1
width = 5
length = 8
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

r2 is a pointer to a Rectangle object

main()
{
    Rectangle r1;
    r1.set(5, 8);  //dot notation
    Rectangle *r2;
    r2 = &r1;
    r2->set(8,10);  //arrow notation
}

width = 8
length = 10
Declaration of an Object (cont’d)

class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

r3 is dynamically allocated

main()
{
    Rectangle *r3;
    r3 = new Rectangle();
    r3->set(80,100);  //arrow notation
    delete r3;
    r3 = NULL;
}

r3  6000
    NULL
Object Initialization

```cpp
#include <iostream.h>

class circle
{
    public:
        double radius;
};

int main()
{
    circle c1;       // Declare an instance of the class circle
    c1.radius = 5;   // Initialize by assignment
}
```

1. By Assignment

- Only work for public data members
- No control over the operations on data members

• Only work for public data members
• No control over the operations on data members
Object Initialization (cont’d)

```cpp
#include <iostream.h>

class circle {
    private:
        double radius;
    
    public:
        void set (double r) {
            radius = r;
        }
        double get_r () {
            return radius;
        }
};

int main(void) {
    circle c;  // an object of circle class
    c.set(5.0);  // initialize an object with a public member function
    cout << "The radius of circle c is " << c.get_r() << endl;
    // access a private data member with an accessor
}
```

2. By Public Member Functions
Object Initialization (cont’d)

class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle();
        Rectangle(const Rectangle &r);
        Rectangle(int w, int l);
        void set(int w, int l);
        int area();
};

3. By Constructor

- Default constructor
- Copy constructor
- Constructor with parameters

They are publicly accessible
Have the same name as the class
There is no return type
Are used to initialize class data members
They have different signatures
When a class is declared with no constructors, the compiler automatically assumes default constructor and copy constructor for it.

- **Default constructor**

  ```cpp
  Rectangle :: Rectangle() { }
  ```

- **Copy constructor**

  ```cpp
  Rectangle :: Rectangle (const Rectangle & r)
  {
      width = r.width;  length = r.length;
  }
  ```
Constructor (cont’d)

class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

• Initialize with default constructor

Rectangle r1;
Rectangle *r3 = new Rectangle();

• Initialize with copy constructor

Rectangle r4;
r4.set(60,80);
Rectangle r5 = r4;
Rectangle r6(r4);
Rectangle *r7 = new Rectangle(r4);
Constructor (cont’d)

If any constructor with any number of parameters is declared, no default constructor will exist, unless you define it.

```cpp
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle(int w, int l)
        {
            width = w; length = l;
        }
        void set(int w, int l);
        int area();
};
```

- Initialize with constructor

```cpp
Rectangle r4;  // error
Rectangle r5(60, 80);
Rectangle *r6 = new Rectangle(60, 80);
```
Define Constructors

class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle();
        Rectangle(int w, int l);
        void set(int w, int l);
        int area();
};

Write your own constructors

Rectangle :: Rectangle()
{
    width = 20;
    length = 50;
};

Rectangle *r7 = new Rectangle();

r7

5000
width = 20
length = 50

5000
6000
Multiple Constructors

With constructors, we have more control over the data members

```cpp
class Account
{
    private:
        char *name;
        double balance;
        unsigned int id;
    public:
        Account();
        Account(const Account &a);
        Account(const char *person);

Account :: Account()
{
    name = NULL; balance = 0.0;
    id = 0;
};

Account :: Account(const Account &a)
{
    name = new char[strlen(a.name)+1];
    strcpy (name, a.name);
    balance = a.balance;
    id = a.id;
};

Account :: Account(const char *person)
{
    name = new char[strlen(person)+1];
    strcpy (name, person);
    balance = 0.0;
    id = 0;
};
```
Destructor

• An object can be initialized by a class constructor
  – Default constructor
  – Copy constructor
  – Constructor with parameters

• (Constructor) Resources are allocated when an object is initialized

• (Destructor) Resources should be revoked when an object is about to end its lifetime
Cleanup of an Object

class Account
{
    private:
        char *name;
        double balance;
        unsigned int id;  //unique
    public:
        Account();
        Account(const Account &a);
        Account(const char *person);
        ~Account();
    }

Destructor

Account :: ~Account()
{
    delete[] name;
}

• Its name is the class name preceded by a ~ (tilde)
• It has no argument
• It is used to release dynamically allocated memory and to perform other "cleanup" activities
• It is executed automatically when the object goes out of scope
Putting them Together

class Str
{
    char *pData;
    int nLength;

public:
    //constructors
    Str();
    Str(char *s);
    Str(const Str &str);

    //accessors
    char* get_Data();
    int get_Len();

    //destructor
    ~Str();
};

Str :: Str() {
    pData = new char[1];
    *pData = '\0';
    nLength = 0;
};

Str :: Str(char *s) {
    pData = new char[strlen(s)+1];
    strcpy(pData, s);
    nLength = strlen(s);
};

Str :: Str(const Str &str) {
    int n = str.nLength;
    pData = new char[n+1];
    strcpy(pData, str.pData);
    nLength = n;
};
Putting them Together (cont’d)

class Str
{
    char *pData;
    int nLength;
public:
    //constructors
    Str();
    Str(char *s);
    Str(const Str &str);
    //accessors
    char* get_Data();
    int get_Len();
    //destructor
    ~Str();
};

char* Str :: get_Data()
{
    return pData;
};

int Str :: get_Len()
{
    return nLength;
};

Str :: ~Str()
{
    delete[] pData;
};
class Str
{
    char *pData;
    int nLength;
public:
    //constructors
    Str();
    Str(char *s);
    Str(const Str &str);
    //accessors
    char* get_Data();
    int get_Len();
    //destructor
    ~Str();
};

int main()
{
    int x=3;
    Str *pStr1 = new Str("Joe");
    Str *pStr2 = new Str();
}

Interacting Objects

Class A

Private: data members

Member methods

Public:
Constructor
Destructor
Other public methods

Private: methods

Class B

Private: data members

Member methods

Public:
Constructor
Destructor
Other public methods

Private: methods

Message passing

Cannot access!
Working with Multiple Files

• To improve the readability, maintainability and reusability, codes are organized into modules

• When working with complicated codes,
  – A set of .cpp and .h files for each class groups
    • .h file contains the prototype of the class
    • .cpp contains the definition/implementation of the class
  – A .cpp file containing main() function, should include all the corresponding .h files where the functions used in .cpp files are defined
Example: time.h

// SPECIFICATION FILE (time.h)
// Specifies the data members and
// member functions prototypes.

#ifndef _TIME_H
#define _TIME_H

class Time
{
    public:
        ...

    private:
        ...

};

#endif
Example: time.cpp

// IMPLEMENTATION FILE ( time.cpp )
// Implements the member functions of class Time

#include <iostream.h>
#include "time.h" // also must appear in client code
... ...

bool Time :: Equal ( Time otherTime ) const

// Function value == true, if this time equals otherTime
// == false , otherwise
{
    return ( (hrs == otherTime.hrs) && (mins == otherTime.mins)
    && (secs == otherTime.secs) ) ;
}
...
Example: main.cpp

// Client Code (main.cpp)
#include "time.h"

// other functions, if any

int main()
{
    ...
    ...
}

Compile and Run

g++ -std=c++11 -o mainExec main.cpp time.cpp
Separate Compilation and Linking of Files

- **main program**
  - `main.cpp`
  - Compiler
  - `main.o`

- **specification file**
  - `time.h`

- **implementation file**
  - `time.cpp`
  - Compiler
  - `time.o`

- Linker
- `mainExec`

The `time.h` file acts as an interface to the implementation details in `time.cpp`. The `#include "time.h"` directive in `main.cpp` allows `main.cpp` to use the functionalities defined in `time.h`. After compilation, the object files `main.o` and `time.o` are created and linked together by the Linker to produce the executable `mainExec`. This process demonstrates the separation of concerns between the specification and implementation of the code.
[Lab – Practice #1]

• Calculator
  – Input
    • Formula (string)
    • Maximum 5 numbers

$ ./calculator
Insert Formula: 16+20*5-8/2
Result: 112

Execution example