SSE2034: System Software Experiment 3
Spring 2016

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Object-Oriented Programming

- Class Definition
- Class Examples
- Objects
- Constructors
- Destructors

Class

- **The class is the cornerstone of C++**
  - It makes possible encapsulation, data hiding and inheritance
- **Type**
  - Concrete representation of a concept
    - Eg. **float** with operations like -, *, + (math real numbers)
- **Class**
  - A user defined type
  - Consists of both data and methods
  - Defines properties and behavior of that type
- **Advantages**
  - Types matching program concepts
    - Game Program (Explosion type)
  - Concise program
  - Code analysis easy
  - Compiler can detect illegal uses of types
- **Data Abstraction**
  - Separate the implementation details from its essential properties

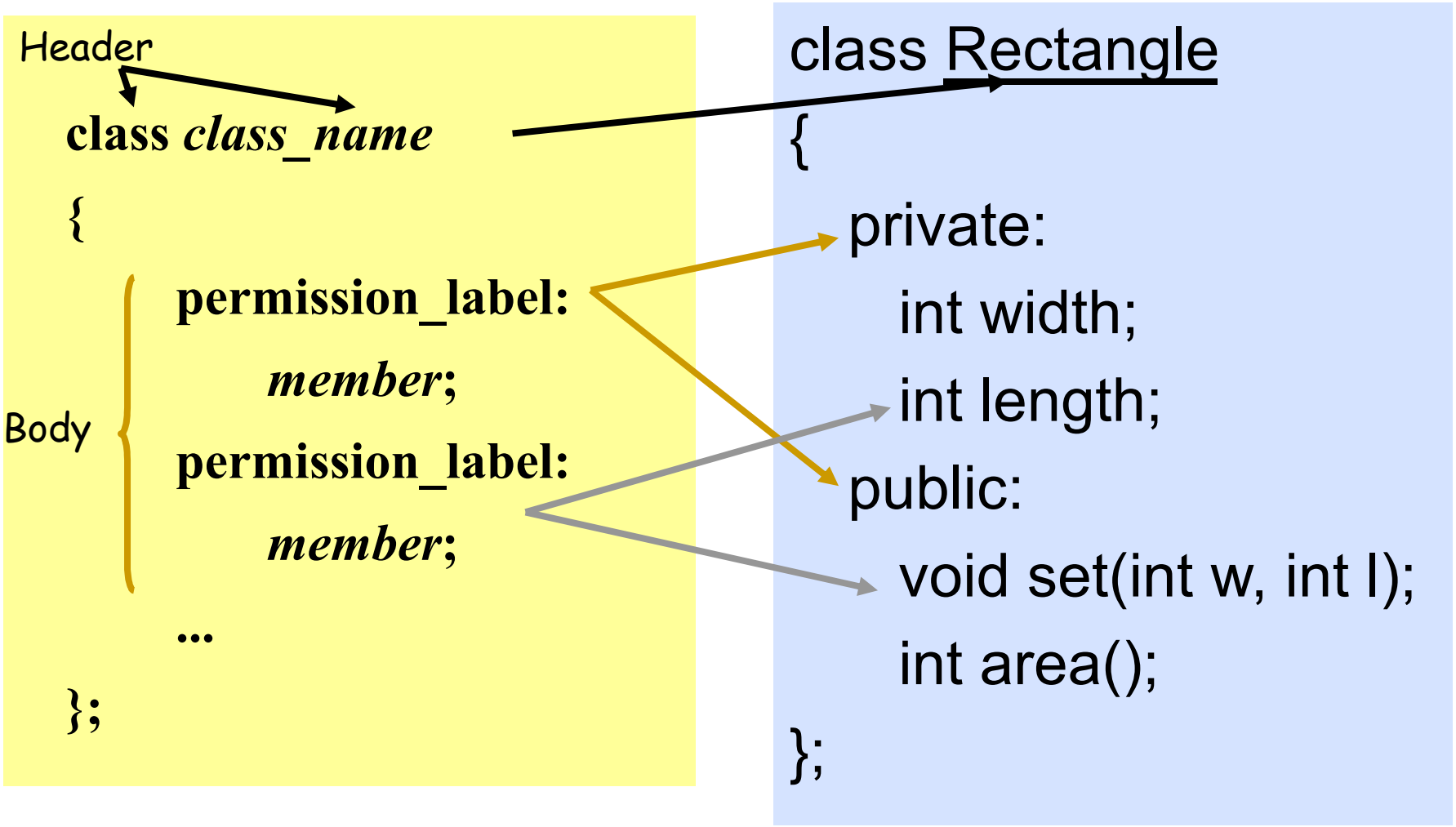
Classes & Objects

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

Objects: Instance of a class

```
Rectangle  r1;
Rectangle  r2;
Rectangle  r3;
```

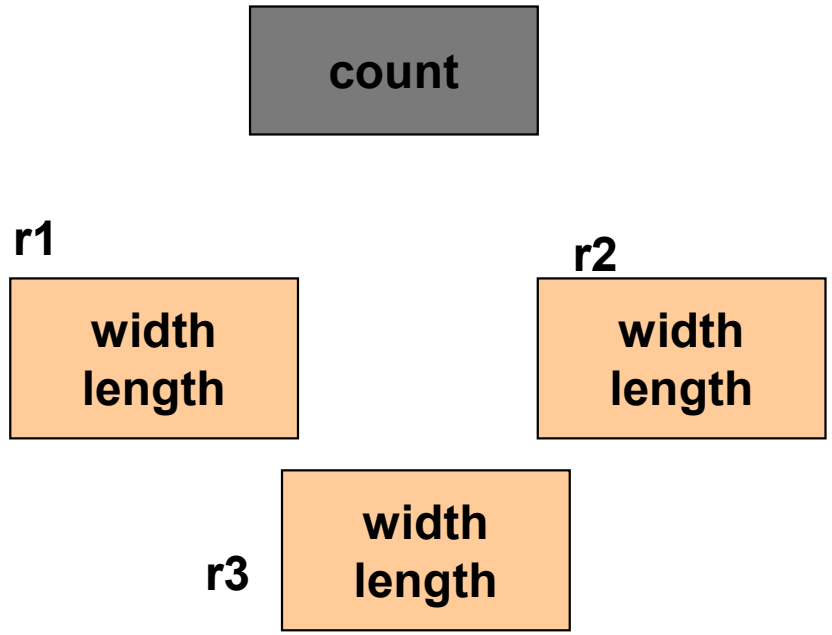
Define a Class Type



Static Data Member

```
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;
```



Define a Member Function

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

class name

member function name

inline

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

scope operator

```
r1.set(5,8);

rp->set(8,10);
```

Const Member Function

```
class  Time
{
  private :
     int     hrs, mins, secs ;

  public :
     void     Write ( )  const ;
} ;
```

function declaration

function definition

```
void Time :: Write( ) const
{
    cout <<hrs << ":" << mins << ":" << secs << endl;
}
```

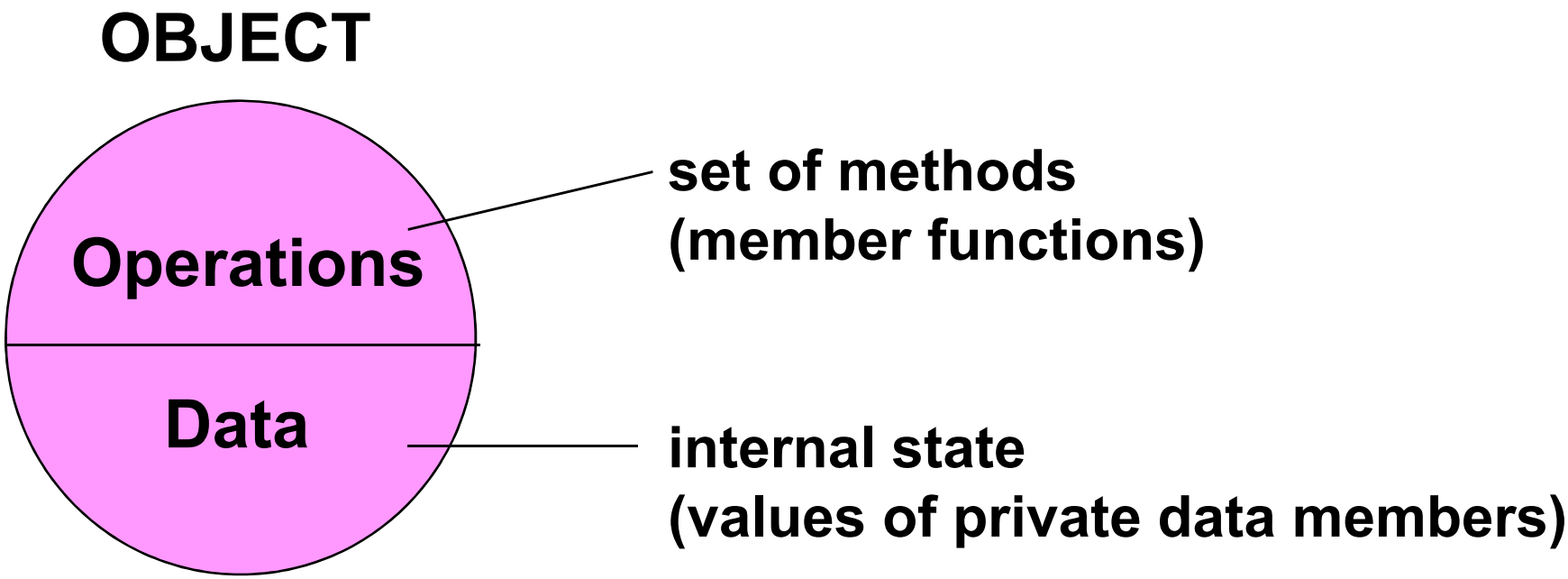
Member Functions

- **const member function**
  - declaration
    - *return_type func_name* (*para_list*) const;
  - definition
    - *return_type func_name* (*para_list*) const { … }
    - *return_type class_name* :: *func_name* (*para_list*) const { … }
  - Makes no modification about the data members (safe function)
  - It is illegal for a const member function to modify a class data member

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 by the member functions, and friends of this class
  - protected
    - acts as public for derived classes
    - behaves as private for the rest of the program

What is an Object?



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;
}
```

Declaration of an Object

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

r1 is statically allocated

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

r1

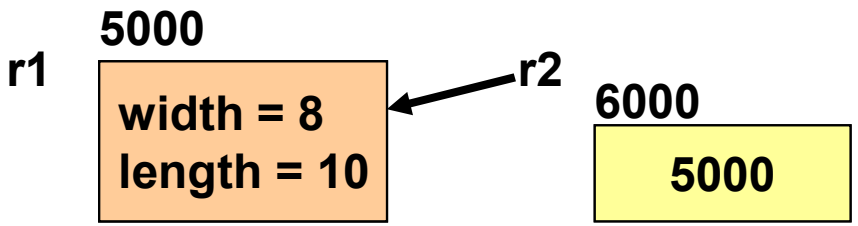
width = 5
length = 8

Declaration of an Object

```
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
}
```

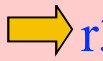


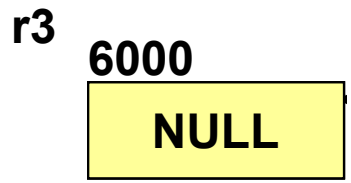
Declaration of an Object

```
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;
}
```





Object Initialization

```
#include <iostream.h>

class circle
{
  public:
    double radius;
};
```

1. By Assignment

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

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

}
```

Object Initialization

```
#include <iostream.h>

class circle
{
  private:
    double radius;

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

};
```

2. By Public Member Functions

```
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
}
```

Object Initialization

```
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

Class Constructor

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

When a class is declared with no constructors, the compiler automatically assumes default constructor and copy constructor for it.

- Default constructor

```
Rectangle :: Rectangle() { };
```

- Copy constructor

```
Rectangle :: Rectangle (const
    Rectangle & r)
{
    width = r.width;  length = r.length;
};
```

Cleanup of An Object

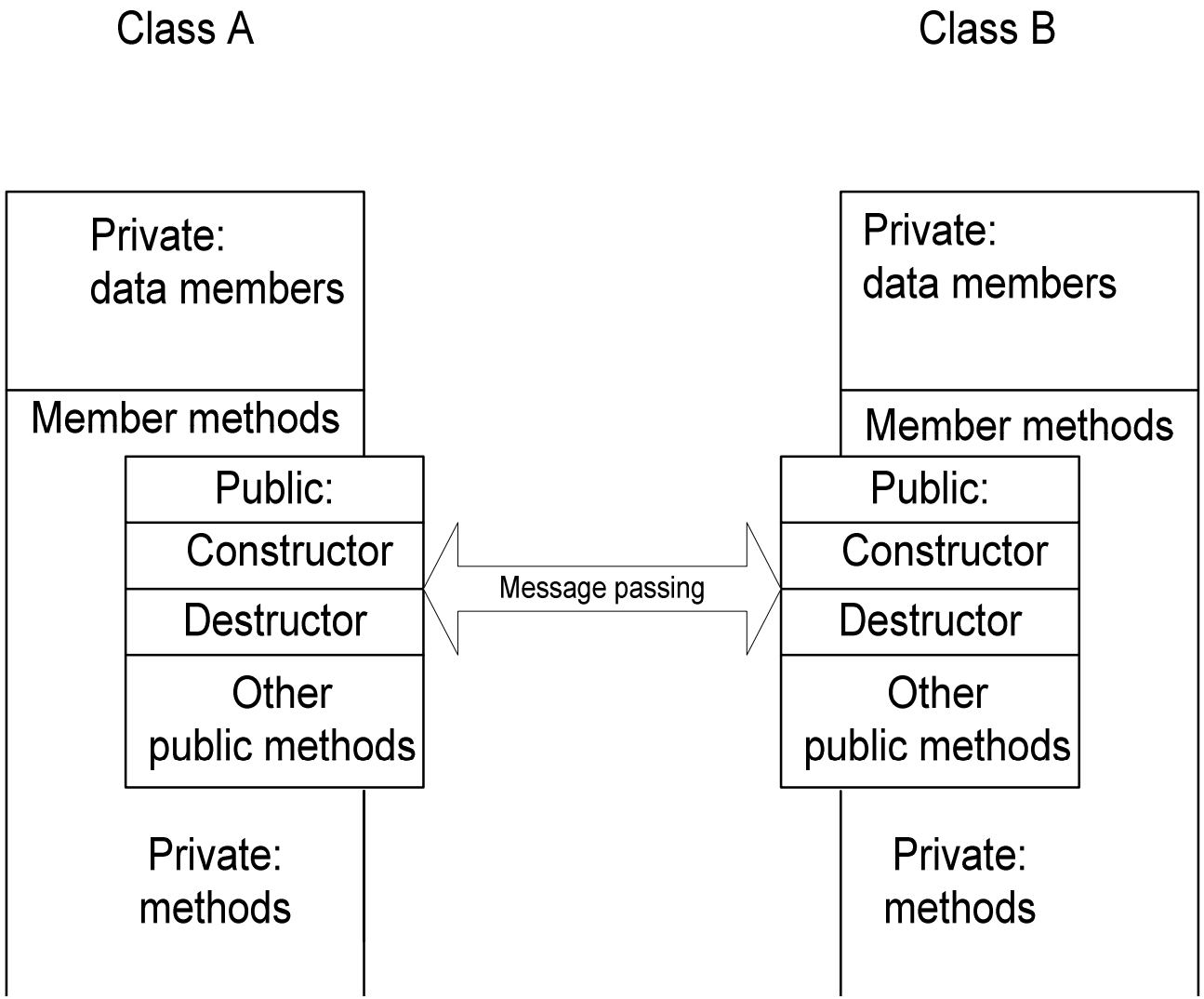
```
class Account
{
    private:
      char *name;
      double balance;
      unsigned int id;  //unique
    public:
      Account();
      Account(const Account &a);
      Account(const char *person);
      ~Account();
}
```

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

Interacting Objects



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 file 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) ) ;
 }
        . . .
```

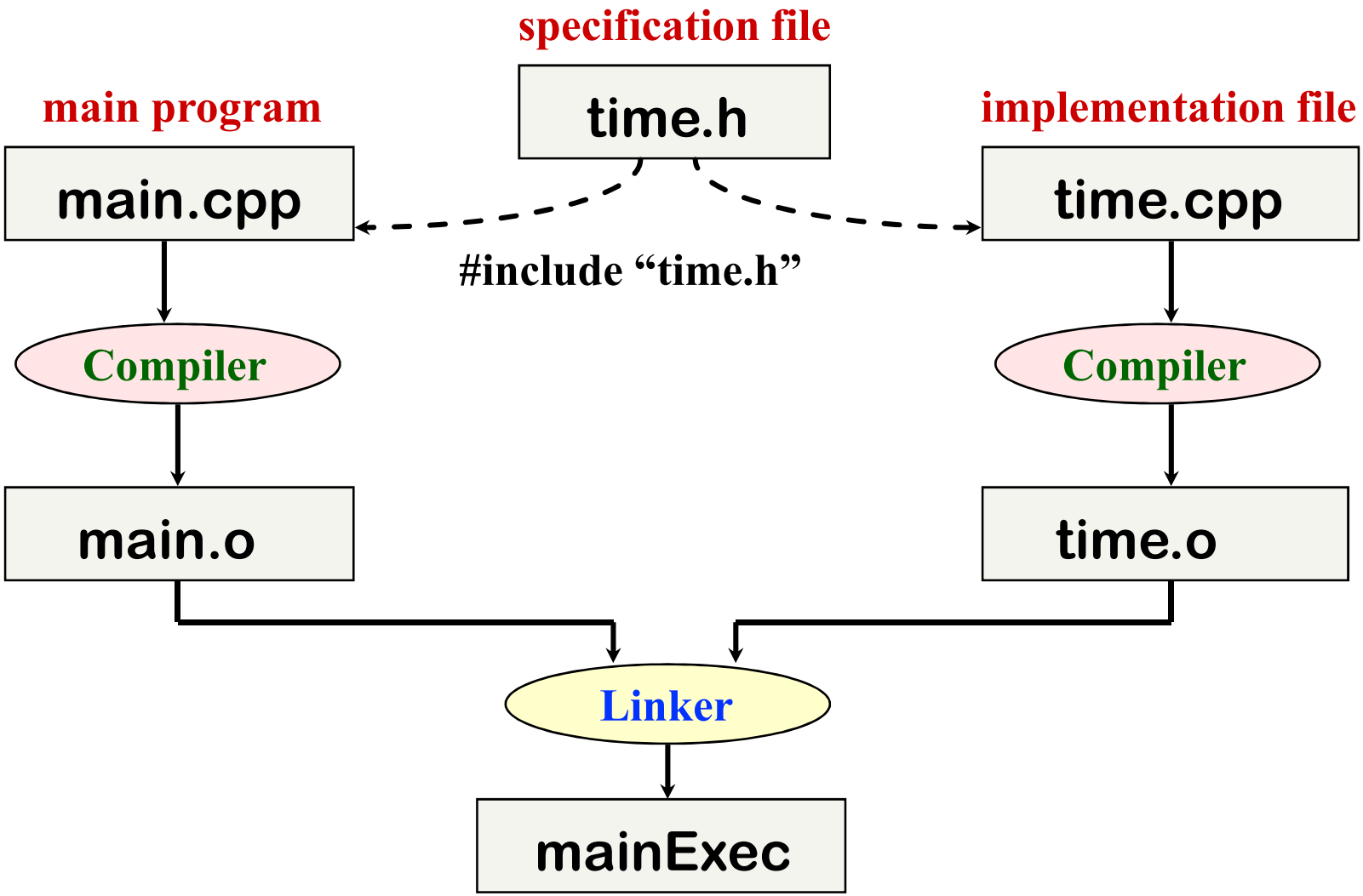
Example : main.cpp

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

// other functions, if any

int main()
{
   … …
}
```

Compile and Run

g++ -o mainExec *main.cpp time.cpp*

Separate Compilation and Linking of Files



[Lab – Practice #1]

- **Blackjack game**
  - a change of your project