CHAPTER 13

INTRODUCTION TO CLASSES

PROGRAM 5

- **Program 5 on Canvas; due in 2 weeks (4/8)**
- **Midterm exam this Friday**
- **We will review for midterm this Wednesday in class**
  - ***Study guide*** on Canvas
  - ***What you need to know*** on Canvas document
  - Take Quiz 4 on Canvas (Anytime between 12am – 11:59pm)

13.1 PROCEDURAL VS. OBJECT-ORIENTED PROGRAMMING

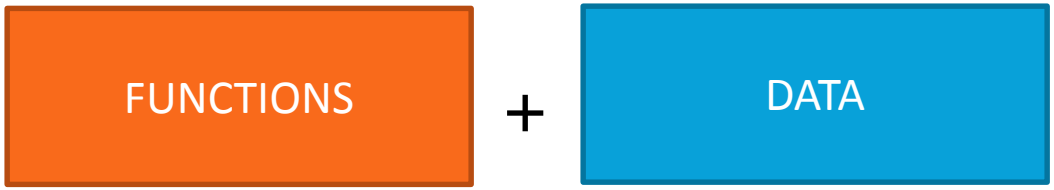
PROCEDURAL VS. OBJECT-ORIENTED PROGRAMMING

Procedural programming (CS2010, so far in this class too)

Also called…

- structured programming
- modular programming
- top-down
- bottom-up

Dividing problem into sub-problems usually implemented as functions and operating on data passed to functions as parameters



LIMITATIONS OF PROCEDURAL PROGRAMMING

If the data structures change, many functions must also be changed (prototype, header, call statements)

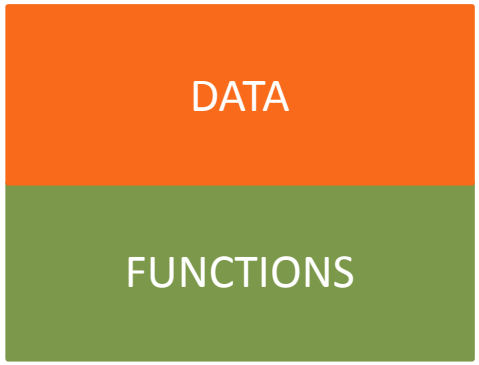
Programs that are based on complex function hierarchies are:

- difficult to understand and maintain
- difficult to modify and extend
- easy to break

PROCEDURAL VS. OBJECT-ORIENTED PROGRAMMING

Object-Oriented programming

Combines data and operations that can be performed on the data into ***a single unit***…



Encapsulation: binding together the data and the functions that manipulate the data.

OBJECT-ORIENTED PROGRAMMING TERMINOLOGY

Class definition: similar to a `struct` definition

Object: an instance of a `class`, in the same way that a variable can be an instance of a `struct`

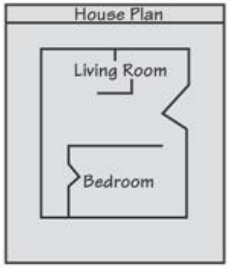
```
struct StructName
{
    // struct members
};

// using structure
StructName myVariable;
```

```
class ClassName
{
    // class members
};

// using class
ClassName myObject;
```

CLASSES VS. OBJECTS

A Class is like a blueprint and objects are like houses built from the blueprint

Blueprint that describes a house.



CLASS

Instances of the house described by the blueprint.

OBJECTS

CLASS MEMBERS

Three categories of class members exist and are designated by access specifiers:

Access specifiers:

public

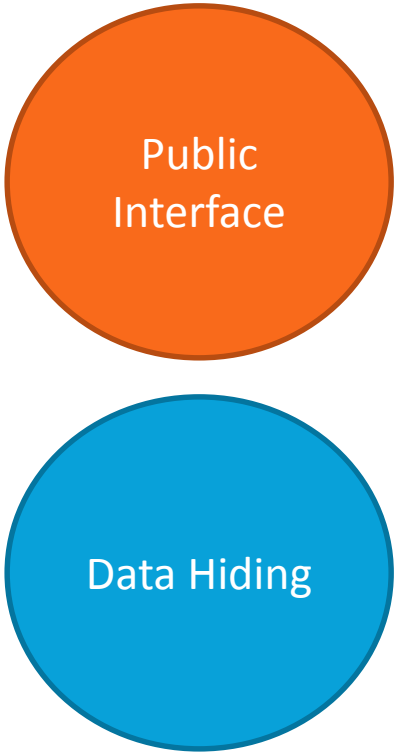
- accessible to everyone

private

- is the *default* category
- accessible only to member functions

protected

- limited access… more on this later



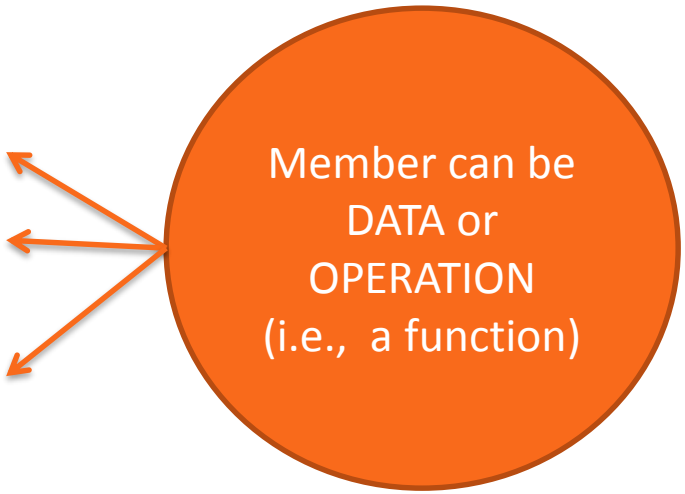
13.2 Introduction to Classes

INTRODUCTION TO CLASSES

Objects are created from a `class` definition…

Syntax:

```
class ClassName
{
public:
      member definition;
      member definition;
private:
      member definition;
};
```



CLASS EXAMPLE

Suppose we want to define a class to implement the **time of day** in a program… To represent time we will need three integer variables (hr, min, sec).

We will also want to perform operations on the time…

set the time, given hr, min, sec

retrieve the current time

print the time to the console window

increment hours by 1

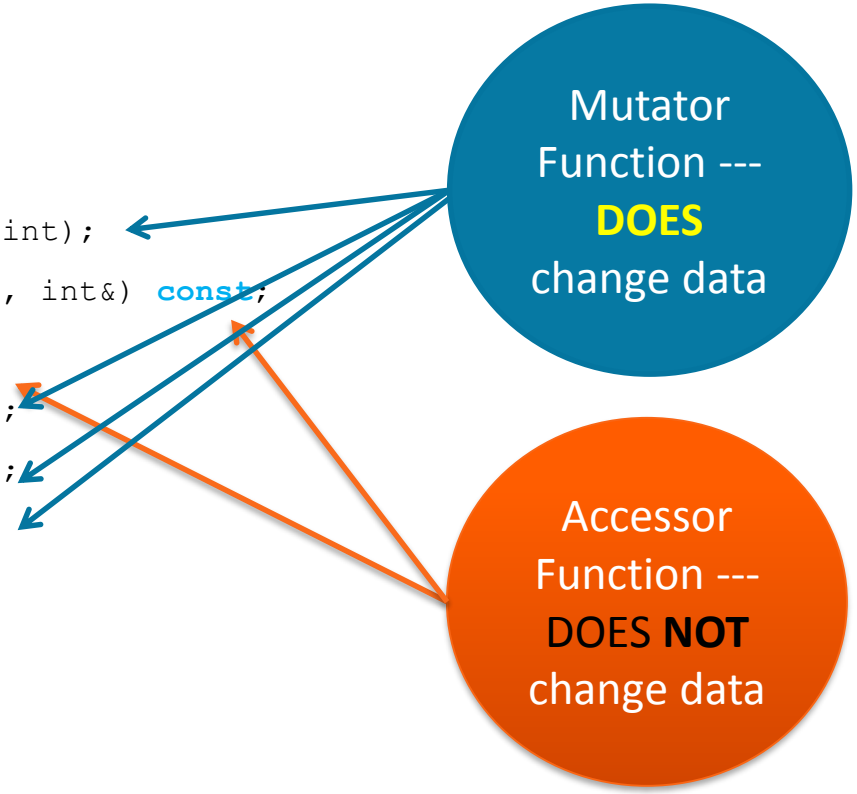
increment minutes by 1

increment seconds by 1

CLASS EXAMPLE

Class definition:

```
class ClockType
{
public:
    void setTime(int, int, int);
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
private:
    int hr;
    int min;
    int sec;
};
```



DEFINING A MEMBER FUNCTION

When defining a member function (a.k.a. *method*):

- Create a prototypes in class definition
- In function definition use class name and *scope resolution operator* (**::**)
- Place definition where other functions normally are defined ***or*** in a separate class implementation file

```
void ClockType::setTime(int h, int m, int s)
{
     hr = h;
     min = m;
     sec = s;
}

void ClockType::getTime(int &h, int &m, int &s) const
{
     h = hr;
     m = min;
     s = sec;
}
```

INSTANTIATING AN OBJECT

```
int main()
{
   ClockType watch;          // static memory allocated
                             // object instantiated

   watch.setTime(10,23,10);     // calling a mutator
   watch.printTime();           // calling an accessor

   return 0;
}
```

IN CLASS EXERCISE

A program will need to deal with rectangles. It will be useful to declare a class to keep both data and operations on the data in one neat container.

The way to describe a rectangle is to specify its width and length.

Operations on a rectangle object include:

- setting its width
- setting its length
- getting its width
- getting its length
- computing its area

IN CLASS EXERCISE

Define a class called Rectangle

Make sure it has the necessary members (both ***private*** and ***public***)

(1) data

(2) operations (both ***mutators*** and ***accessors***), i.e., methods

IN CLASS EXERCISE

Analyze the needed class…

1. What data will it need to operate on? (*variables*)

   width, length – both integers

2. What operations will need to be performed on the data? (*functions*)

   set width, set length, get width, get length, get area

3. What members should be private?

   width, length variables

4. What members should be public?

   functions to set/get width/length and get area

5. What operations (i.e., functions) WILL modify the data? (*mutators*)

   set width, set length

6. What operations (i.e., functions) will NOT modify the data? (*accessors*)

   get width, get length, get area

IN CLASS EXERCISE

```
class Rectangle
{
private:
      int width;
      int length;

public:
      // accessors
      int getWidth() const;
      int getLength() const;
      int getArea() const;

      // mutators
      void setWidth(int);
      void setLength(int);
};
```

Private Members

Public Members

IN CLASS EXERCISE

Let's use our Rectangle class

Declare a variable, an instance of the Rectangle class, call it dining_room.

```
Rectangle dining_room;
```

IN CLASS EXERCISE

Let's use our Rectangle class

Declare a variable, an instance of the Rectangle class, call it dining_room.

```
Rectangle dining_room;
```

Set width of the dining room to 12 and length to 16.

```
dining_room.width = 12;        // cannot directly access private
                               // data members (data hiding)

dining_room.setWidth(12);
dining_room.setLength(16);
```

IN CLASS EXERCISE

Let's use our Rectangle class

Declare a variable, an instance of the Rectangle class, call it dining_room.

```
Rectangle dining_room;
```

Set width of the dining room to 12 and length to 16.

```
dining_room.width = 12;      // width is a private member!

dining_room.setWidth(12);
dining_room.setLength(16);
```

Write a cout statement that will show dining room area.

```
cout << "Area of the room is " << dining_room.getArea() << endl;
```

OUT OF CLASS EXERCISE

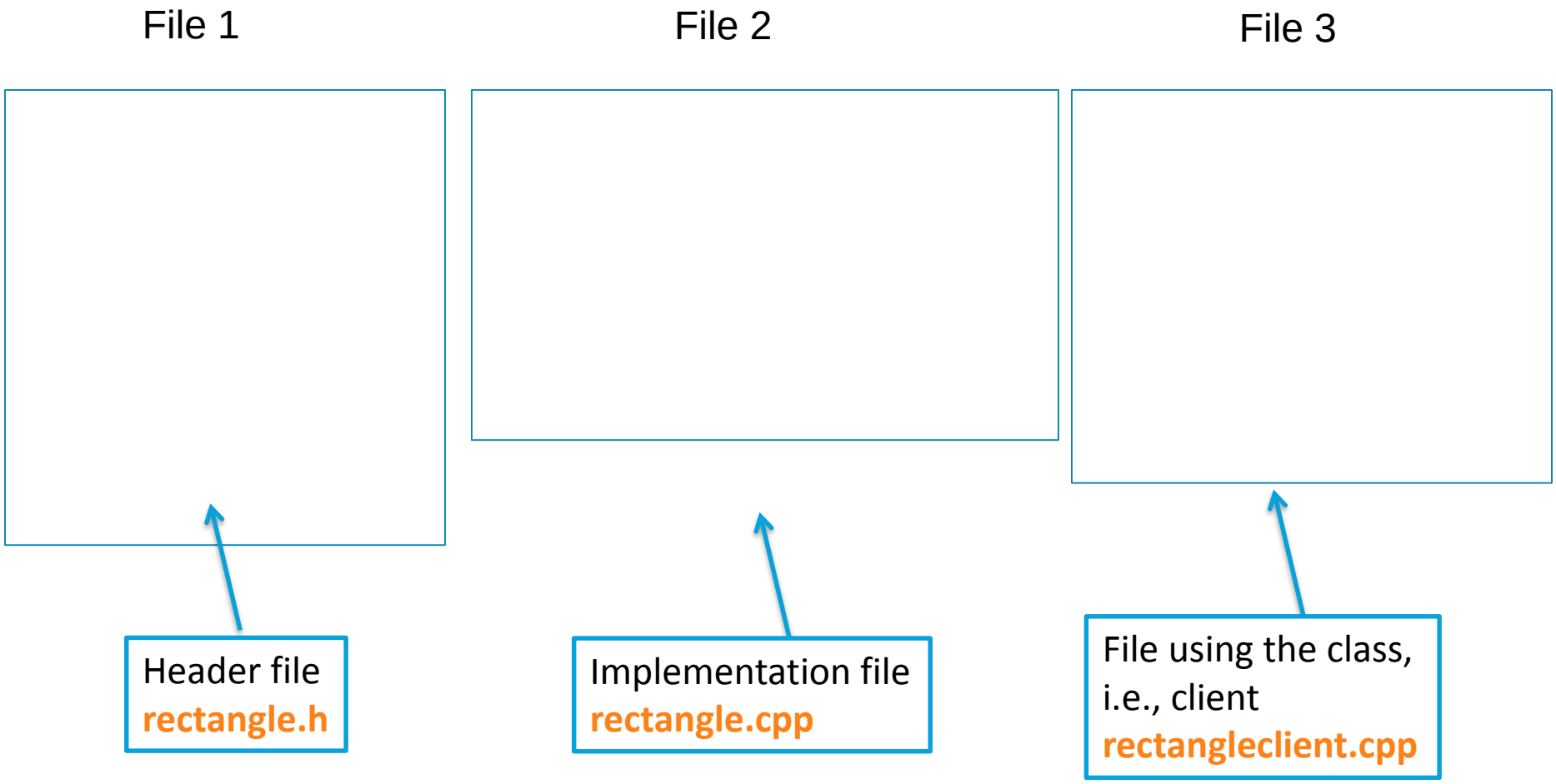
```
// Write function definitions for all
// member functions for class Rectangle

int Rectangle::getaArea() const
{
   return width * length;
}
```

13.5 SEPARATING SPECIFICATION FROM IMPLEMENTATION

SO FAR…

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

class Rectangle {};        // class definition


void someFunction(int);   // prototype

int main()
{
    return 0;
}

void someFunction(int x) // function definition
{
}

// class method definitions
int Rectangle::getArea() const
{
}
```

SEPARATING SPECIFICATION FROM IMPLEMENTATION



SEPARATING SPECIFICATION FROM IMPLEMENTATION

File 1

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

// CLASS DEFINITION
// (SPECIFICATION)

class Rectangle
{
public:
   void setWidth(int);
};


#endif
```

Header file
rectangle.h

File 2

Implementation file
rectangle.cpp

File 3

File using the class, i.e., client
rectangleclient.cpp

SEPARATING SPECIFICATION FROM IMPLEMENTATION

File 1

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

// CLASS DEFINITION
// (SPECIFICATION)

class Rectangle
{
public:
   void setWidth(int);
};


#endif
```

Header file
rectangle.h

File 2

```
#include "rectangle.h"


// CLASS IMPLEMENTATION

void Rectangle::setWidth(int w)
{
}
```

Implementation file
rectangle.cpp

File 3

File using the class, i.e., client
rectangleclient.cpp

SEPARATING SPECIFICATION FROM IMPLEMENTATION

File 1

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

// CLASS DEFINITION
// (SPECIFICATION)

class Rectangle
{
public:
   void setWidth(int);
};


#endif
```

Header file
rectangle.h

File 2

```
#include "rectangle.h"


// CLASS IMPLEMENTATION

void Rectangle::setWidth(int w)
{
}
```

Implementation file
rectangle.cpp

File 3

```
#include <iostream>
#include "rectangle.h"
using namespace std;

int main()
{
    Rectangle room;
    room.setWidth(10);

    return 0;
}
```

File using the class,
i.e., client
rectangleclient.cpp

SEPARATING SPECIFICATION FROM IMPLEMENTATION

- Place class definition in a header file that serves as the class specification file. Name the file *ClassName*.h, for example, rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

// CLASS DEFINITION
// (SPECIFICATION)

class Rectangle
{
public:
   void setWidth(int);
};


#endif
```

SEPARATING SPECIFICATION FROM IMPLEMENTATION

- Place member function definitions in *ClassName*`.cpp`, for example, `rectangle.cpp` File should `#include` the class specification file (i.e., the header file rectangle.h)

```
#include "rectangle.h"


// CLASS IMPLEMENTATION

void Rectangle::setWidth(int w)
{
}
```

SEPARATING SPECIFICATION FROM IMPLEMENTATION

- Programs that use the class must `#include` the class specification file, and be compiled and linked with the member function definitions

```
#include <iostream>
#include "rectangle.h"
using namespace std;

int main()
{
    Rectangle room;
    room.setWidth(10);

    return 0;
}
```

Example:

g++ rectangleclient.cpp rectangle.cpp

g++ lab6.cpp lab6client.cpp