Foundation of Computer Science: Class

Kafi Rahman

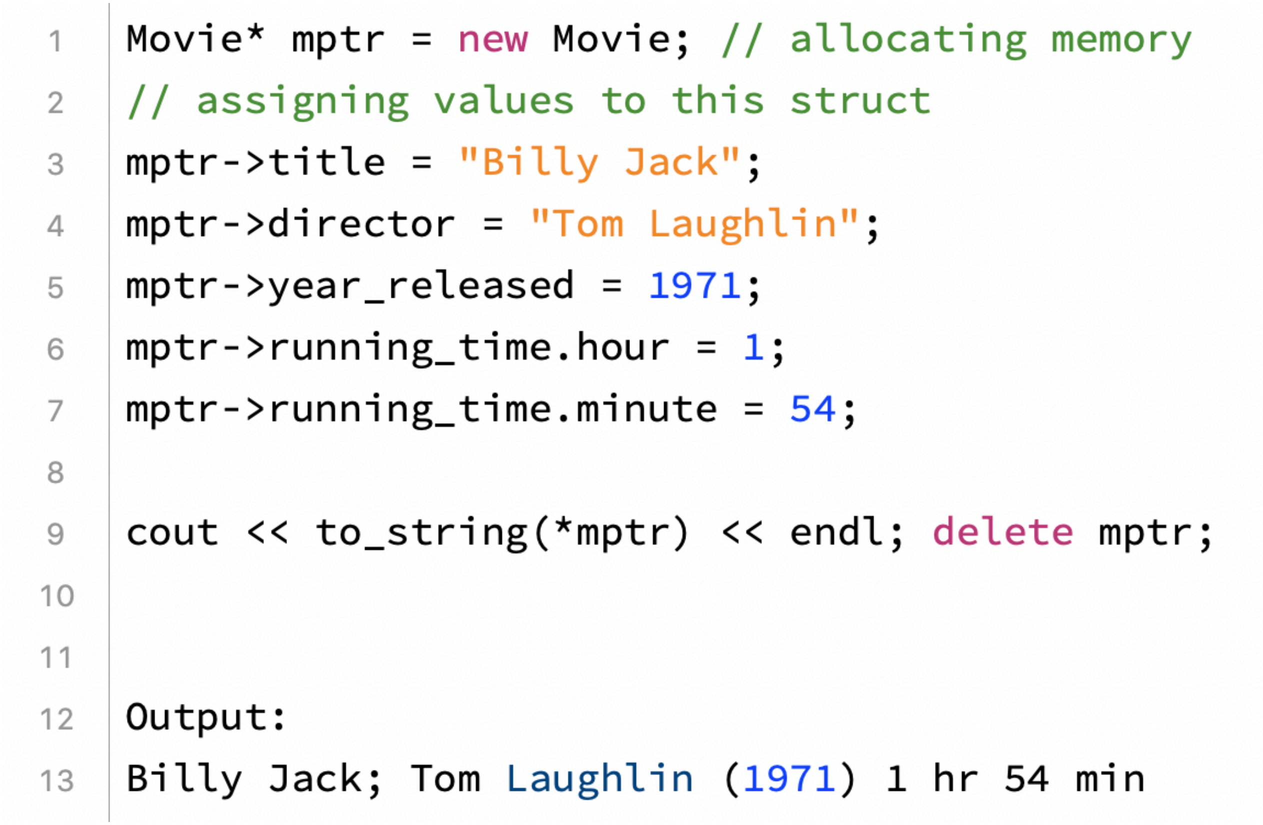
Assistant Professor

Computer Science

Truman State University

Dynamically Allocating Structures

- with the ability to have pointers to structure variables, we can dynamically allocate them
- this is essential in C, rarely done in C++ until CS310



```
Movie* mptr = new Movie; // allocating memory
// assigning values to this struct
mptr->title = "Billy Jack";
mptr->director = "Tom Laughlin";
mptr->year_released = 1971;
mptr->running_time.hour = 1;
mptr->running_time.minute = 54;

cout << to_string(*mptr) << endl; delete mptr;



Output:
Billy Jack; Tom Laughlin (1971) 1 hr 54 min
```

Overloading

- a topic from 6.14 that we skipped at the time program 6-27, on page 360, defines two functions with the same name
- the name of the function, square, is overloaded
- both functions have the same purpose they operate on arguments of different types, and return different types

```
int square(int number);
double square(double number);
```

Signatures

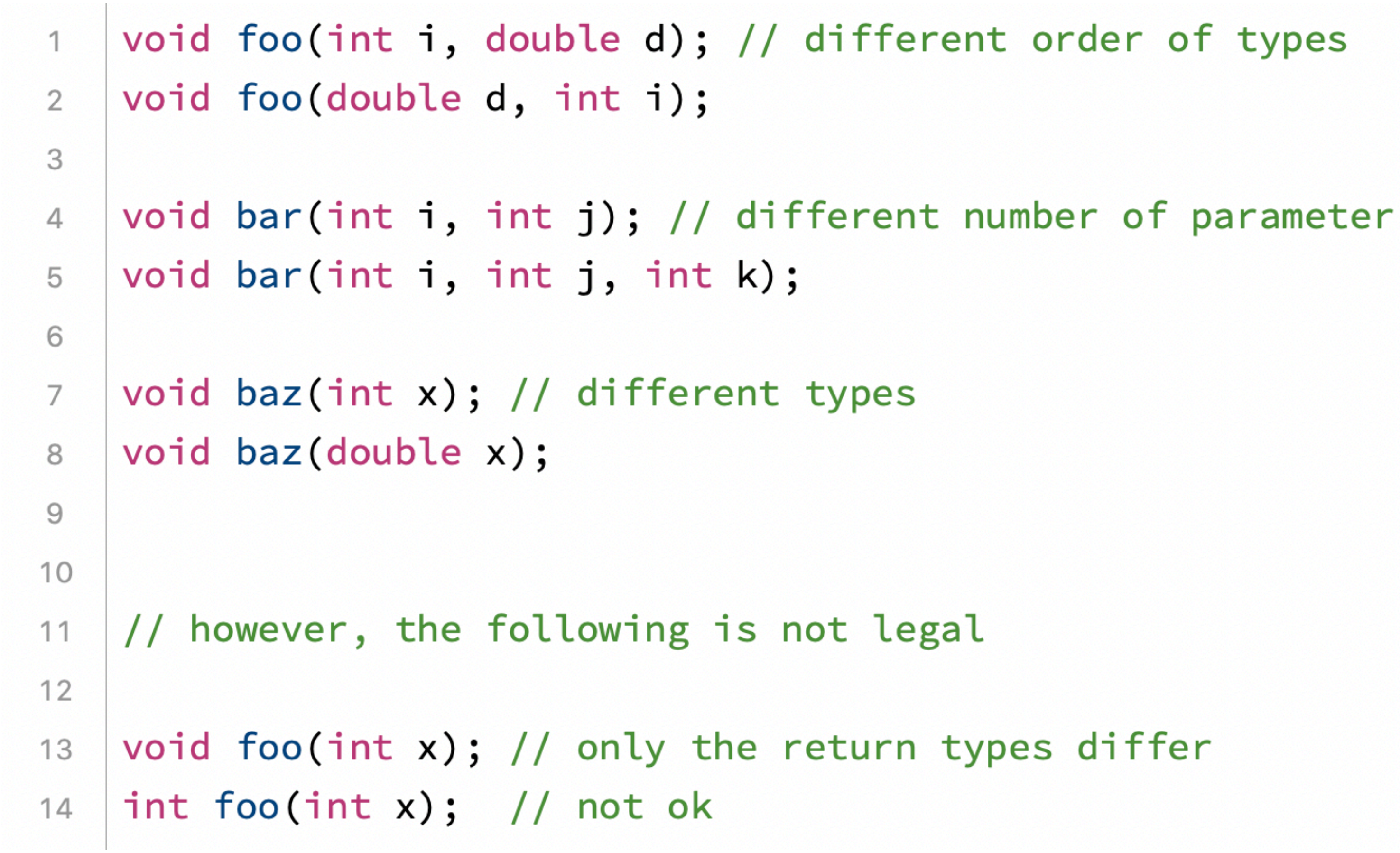
- in C++, every function has a signature the signature consists of
  - the function' s name
  - the data types of the function' s parameters, in order

- this is the information that is contained in the function prototype

- a function name can be overloaded if the types in the parameter list in the function signatures are different
  - different number or arrangement of types

Overloading - Examples

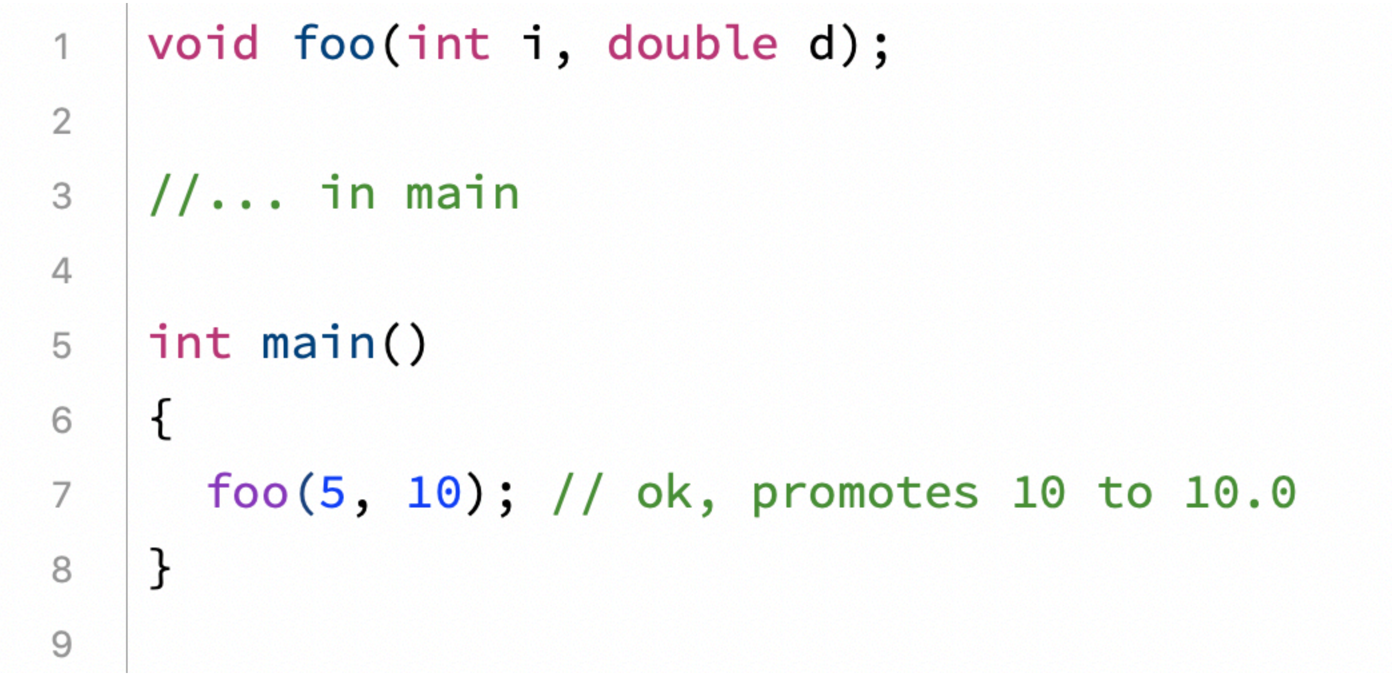
- all the following are legal examples of overloading



```
void foo(int i, double d); // different order of types
void foo(double d, int i);

void bar(int i, int j); // different number of parameter
void bar(int i, int j, int k);

void baz(int x); // different types
void baz(double x);


// however, the following is not legal

void foo(int x); // only the return types differ
int foo(int x);  // not ok
```

Overloading - Ambiguous

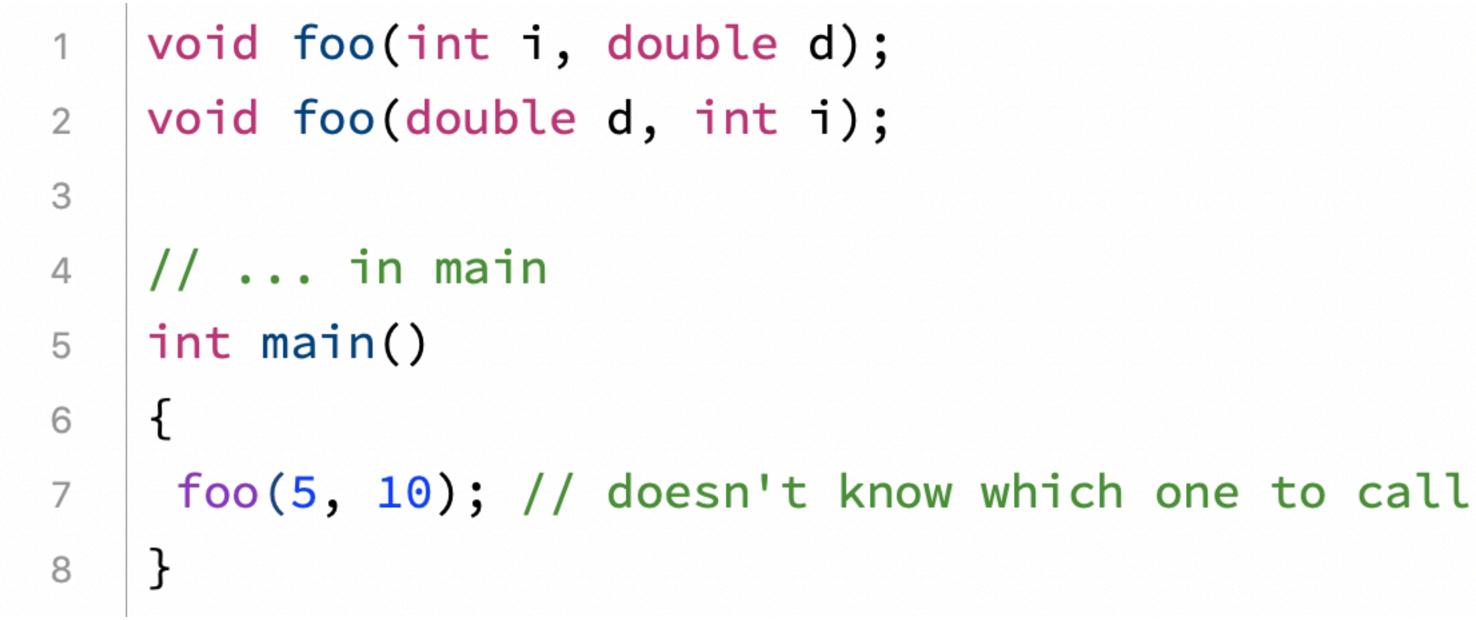
- What about the following function call?



```
void foo(int i, double d);

//... in main

int main()
{
  foo(5, 10); // ok, promotes 10 to 10.0
}

```

Overloading - Ambiguous

- however, the following won't compile due to ambiguity:



```
void foo(int i, double d);
void foo(double d, int i);

// ... in main
int main()
{
 foo(5, 10); // doesn't know which one to call
}
```

Object-Oriented Programming Terminology

- class: a class is a user defined data-type which has data members and member functions.
  - class is a grouping if variables and functions in a single entity
  - class = struct + functions

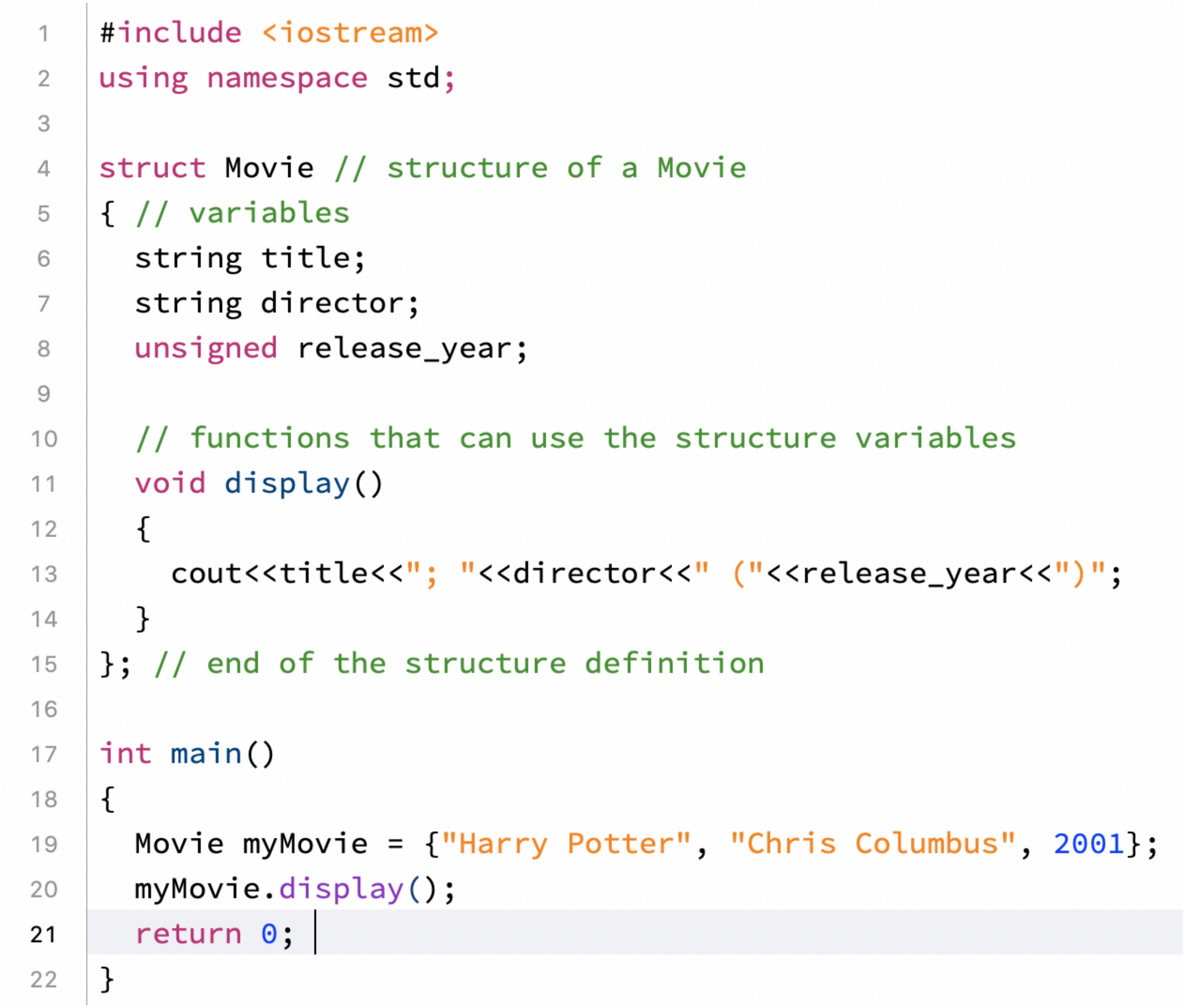
- object: is a variable (or an instance) of a class

Object-Oriented Programming Terminology

- attributes: members of a class

- methods or behaviors: member functions of a class
  - what the object of a class can do

Class= Structure + Functions



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

struct Movie // structure of a Movie
{ // variables
  string title;
  string director;
  unsigned release_year;

  // functions that can use the structure variables
  void display()
  {
    cout<<title<<"; "<<director<<" ("<<release_year<<")";
  }
}; // end of the structure definition

int main()
{
  Movie myMovie = {"Harry Potter", "Chris Columbus", 2001};
  myMovie.display();
  return 0;
}
```

Features of a Class

- enables data hiding: restricting access to certain members of an object

- provides public interface: members of a class that are available outside of the class.

    - This allows the object of the class to provide access to some data and functions

        - This mechanism provides protection from data corruption.

13.2

Introduction to Classes

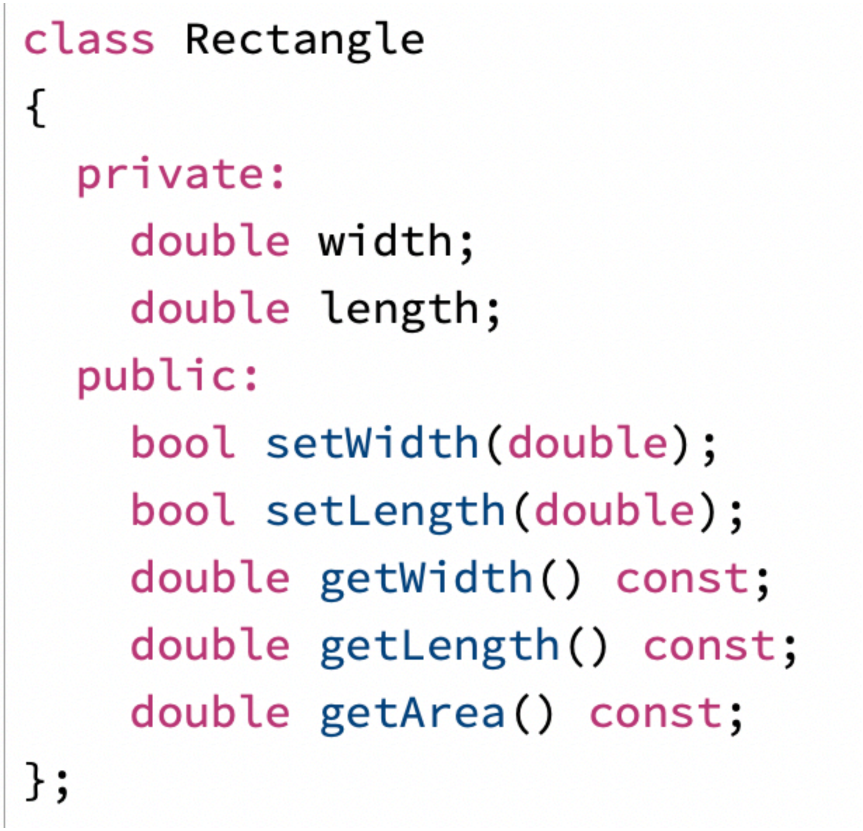
Introduction to Classes

- Format:

```
class ClassName
{
     // variables declaration;
     // functions declaration;
};
```

- Objects are variables of a class

Class Example

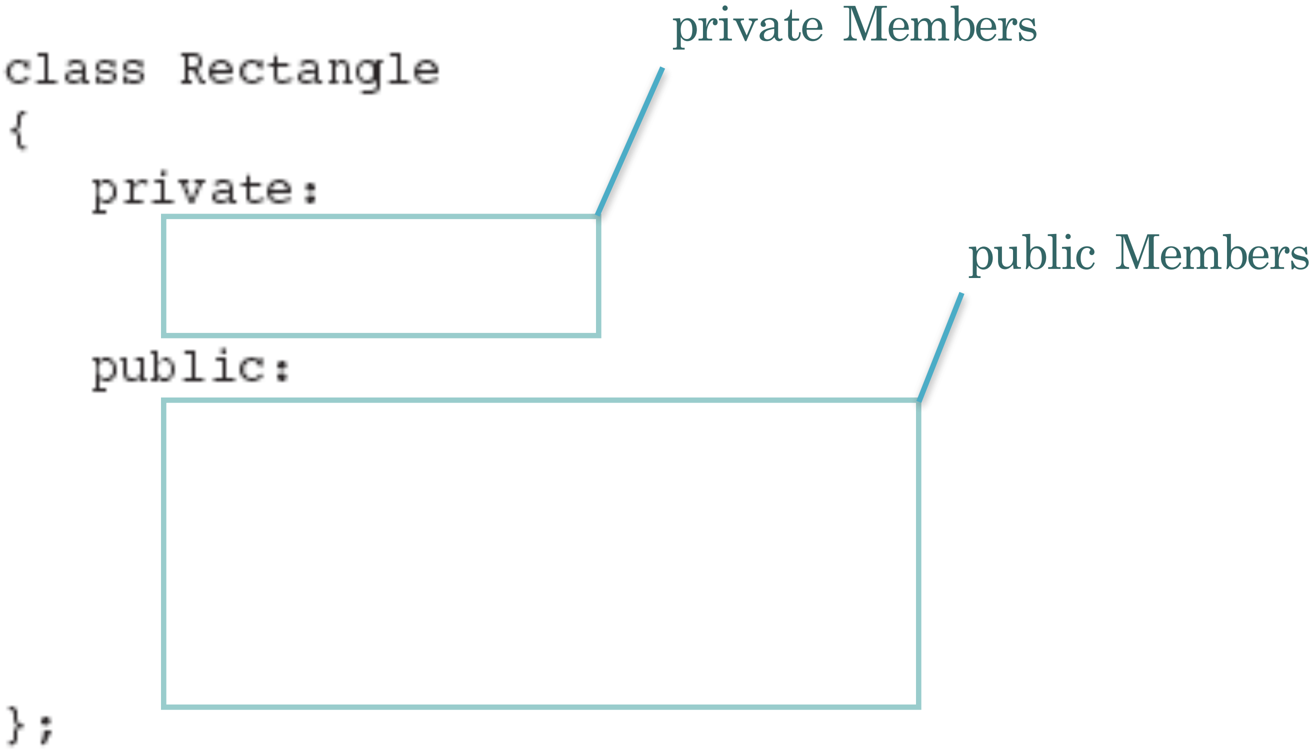


```
class Rectangle
{
  private:
    double width;
    double length;
  public:
    bool setWidth(double);
    bool setLength(double);
    double getWidth() const;
    double getLength() const;
    double getArea() const;
};
```

Access Specifiers

- access specifiers are used to control access to members of the class. They can be
  - public: these members can be accessed in the program from outside of the class
  - private: these members can only be called by or accessed by functions that are members of the class

Class Example



```
class Rectangle
{
   private:

   public:

};
```

private Members

public Members

More on Access Specifiers

- Can be listed in any order in a class
- Can appear multiple times in a class
- If not specified, the default is private

Code Example

access specifier