Classes Methods

Chapter 4 Revisited

Member Functions

- A member function is a function specific to the class and can only be accessed using the dot operator

- Member function implicitly has access to all data members of the class.

Defining a Function Member: Method 1

Prototype in Class Definition

Member Function Defined outside Class Definition

```
class SomeClass
{
    public:
    datatype funcName(datatype var1, ..., datatype varN);
};

datatype SomeClass::funcName(datatype var1, ..., datatype varN)
{
    ...;
     return ...;
}
```

Defining a Function Member: Method 1

Example

```
class Rectangle
{
   public:
   int width, height;
   int area();
};

int Rectangle::area()
{
   return width * height;
}
```

Defining a Function Member: Method 2

No Prototype

Member Function Defined inside Class Definition

```
class SomeClass
{
    public:
    datatype funcName(datatype var1, ..., datatype varN) {
        ...;
        return ...;
    }
};
```

Defining a Function Member: Method 2

Example

```
class Rectangle
{
    public:
    int width, height;
    int area()
    {
         return width * height;
    }
};
```

Comparison of Methods

- Method 1
  - More conducive to multi-file programming
  - Typically class definition is in the header file, and function members are defined in the source file

- Method 2
  - Requires less coding
  - Intuitively shows which class the function is in
  - In large classes, the class definition becomes muddled

- For the remainder I will use Method 1 since it is the most commonly used.

Back to the Example

```
class Rectangle
{
    public:
    int width, height;
    int area();
};
```

Specifies the class the function member belongs to.

Why is this needed?

```
int Rectangle::area()
{
     return width * height;
}
```

Specify Class of Member Function

```
class Rectangle {
   public:
   int width, height;
   int area();
};

class Circle {
 public:
    int radius;
    int area();
};

int Rectangle::area() {
    return width * height;
}
```

Without specifying the class area( ) would be ambiguous between the Rectangle and Circle classes

Back to the Example

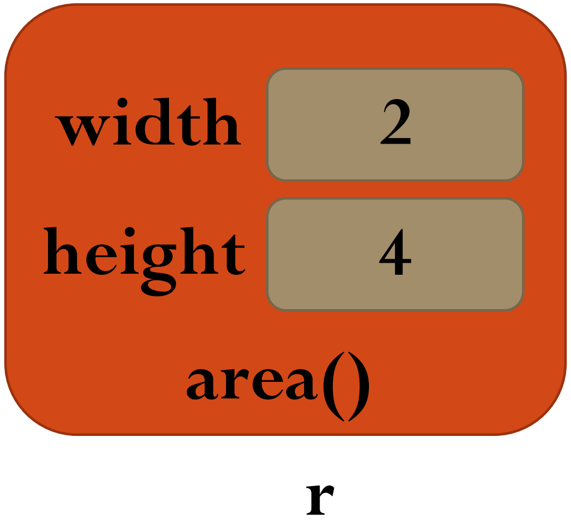
```
class Rectangle
{
    public:
    int width, height;
    int area();
};


int Rectangle::area()
{
     return width * height;
}
```

Since area( ) is a member of Rectangle it has implicit access to the width and height variables

Instantiating Rectangle and Calling Area ( )

```
int main()
{
   Rectangle r;
   r.width = 2;
   r.height = 4;
   int a = r.area();
   cout << "Area is " << a << endl;
   return 0;
}
```



width 2

height 4

area()

r

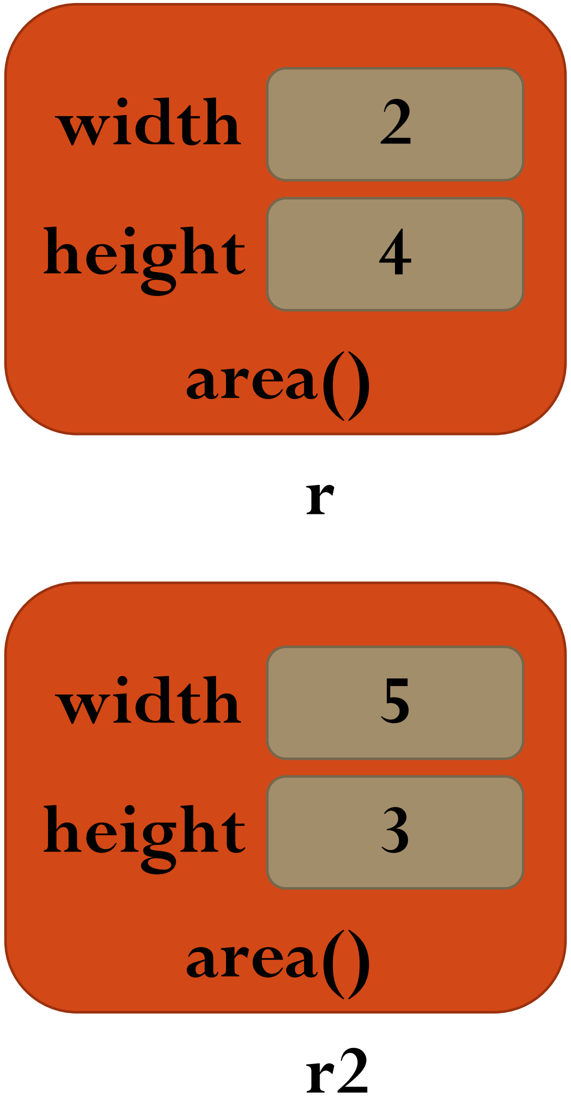
Instantiating Multiple Rectangles

```
int main()
{
   Rectangle r, r2;
   r.width = 2;
   r.height = 4;
   int a = r.area();  →  8

   r2.width = 5;
   r2.height = 3;
   int b = r2.area();  →  15

   cout << "Area is " << a << endl;
   cout << "Area is " << b << endl;

   return 0;
}
```



width 2

height 4

area()

r

width 5

height 3

area()

r2

Method with Parameters

```
class Rectangle {
   public:
   int width, height;
   int area();
   void set(int width, int height);
};

void Rectangle::set(int w, int h)
{
   width = w;
   height = h;
}
```

Set method overwrites the values of width and height to the object to which it is applied

Calling Method with Params

```
int main()
{
   Rectangle r;
   r.set(2, 4);
   cout << r.width << "," << r.height << endl;
   return 0;
}
```

Here width and height or *r* is set to 2 and 4 respectively.
Thus “2, 4” is printed to the console

Constructors

- A constructor is always called once and only once on instantiation of an object
    - If no constructor is provided, an empty one is provided by default

- A constructor with no parameters is called the default constructor

- A constructor may also have parameters

- Purpose: Initialize the object to a set of values

Constructor Syntax

```
class Class_Name {
   public:
   Class_Name(datatype var1, ..., datatype varN);
};

Class_Name::Class_Name(datatype var1, ..., datatype varN)
{
   ...
}
```

Constructor Syntax

No return type, not even void

Constructor is a member function which is named the same as the Class

```
Class_Name::Class_Name(datatype var1, ..., datatype varN)
{
    ...
}
```

Default Constructor Example

```
class Rectangle {
public:
   int width, height;
   Rectangle();
};

Rectangle::Rectangle() {
    height = 0;
    width = 0;
}
```

Calling Default Constructor

```
int main()
{
    Rectangle r;
    cout << r.width << "," << r.height << endl;
    return 0;
}
```

Implicit Call

Calling Default Constructor

```
int main()
{
   Rectangle r;
   cout << r.width << "," << r.height << endl;
   return 0;
}
```

For both cases the width and height is set to zero, and therefore 0,0 is printed to the console

Non-Default Constructor

```
class Rectangle {
public:
   int width, height;
   Rectangle();
   Rectangle(int w, int h);
};
```

Parameters for initializing internals of the class

```
Rectangle::Rectangle(int w, int h)
{
   width = w;
   height = h;
}
```

Calling Non-Default Constructor

Explicit call and passing parameters. Parameters maybe constants, variables, or expressions

```
int main()
{
    Rectangle r(3, 5);
    cout << r.width << "," << r.height << endl;
    return 0;
}
```

Here: 3,5 is printed to the console

Deconstructor

- Classes also have a deconstructor which is called anytime the object is destroyed.

- Deconstructors are very useful in preventing memory leaks when dynamic memory is used.

- Since we have not introduced dynamic memory, the deconstructor will be covered at a later point

Private vs Public

- Public members are accessable by non-member functions (such as main) and member functions

- Private members can only be accessed by member functions

Example of Private

```
class Rectangle {
public:
    Rectangle(int w, int h);
    int area();
private:
    int width, height;
};
```

Private Members

```
Rectangle::Rectangle(int w, int h){
    width = w;
    height = h;
}

int Rectangle::area() {
   return width * height;
}
```

Accessible by member functions

Example of Private

```
int main()
{
   Rectangle r(3, 5);
   cout << r.area() << endl;
   cout << r.width << "," << r.height << endl;
   return 0;
}
```

Legal, since accessing public members

Illegal, since width and height are private and main is not a member of rectangle

Will Cause a syntax error

Why use private?

- Some variables inside a class can only change states under a strict set of circumstances, otherwise the object can "break"
  - Giving access to functions outside the class risks incorrectly setting those variables

- Example: Imagine there is a class that represents a car, and inside the state of the transmission is kept.
  - Let us say the transmission state is public
  - Inside the main the user changes the transmission state from *Drive* to *Reverse* while the car is going at 80 mph in the simulation
  - This not only doesn't model the car properly, but also may cause an issue in the simulation

Summary

- Classes organize data and functions into single concepts

- A member function has access to all internal variables used.

- A constructor is used to initialized the variables internal to the object
  - The default constructor is implicitly called if the object is instantiated without the ( ) at the end of the name.

- Some internals of classes should be kept private to protect the state of the class for outside functions