

Classes

In Object Oriented Programming (OOP) the first development is the object: it's state and behaviors. The state and behaviors are combined together into what C++ calls a class.

Here is an example of a rectangle class.

```
class Rectangle
{
public:
    // Constructors
    Rectangle()           // Default constructor
    Rectangle(double l, double w);

    // Accessors
    double Area() const;
    double Perimeter() const;

    // Mutators
    void SetLength(double l);
    void SetWidth(double w);

private:
    double Diagonal() const;    // private member function
    double myLength;           // State of class
    double myWidth;            // State of class
};
```

The state of the above class are the two double variables *myLength* and *myWidth*. The state stores the current information about the object, in this case, the length and width of the current rectangle.

The behaviors are the functions *Rectangle*, *Rectangle(int l, int w)*, *Area*, *Perimeter*, *SetLength*, and *SetWidth*. The behaviors return information about the state, or can change the state of the object.

There are two sections in a class: the public and the private section. The public section is the interface to the client program. These are the functions or variables that the client program can use. For example:

```
#include "rectangle.h"

int main ()
{
    Rectangle r;

    r.SetWidth(4);
    r.SetLength(5);
    cout << "The area of the rectangle is " << r.Area() << endl;
```

The *include* statement brings in the definitions of the rectangle class so that they can be used in the client program. The header file loaded in would be the class definition listed above along with some additional assembler directives (which we will talk about later).

The statement *Rectangle r;* is called instantiating an object. This creates the object and calls the corresponding constructor. In this case the first constructor *Rectangle();* is called and executed. We have not seen the code for the constructor yet, but this constructor is a normal C++ function. Lets assume that the constructor set the state, *myLength* to 1, and *myWidth* to 1.

The next statement in the client program *r.SetWidth(4)* calls the function *void SetWidth(int w);* that changes the state of the rectangle. In this case the code would change the value of *MyWidth* to 4. The statement *r.SetLength(5)* does the same as *r.SetWidth(4)*.

The last reference to the rectangle object is calling *r.Area()* in the cout statement. This calls the function *double Area() const;* which computes the area of the rectangle object and returns this value.

All of these functions can be called from the client program because they declared in the public section of the class. The private section is hidden from the client program. Hiding data and information from the client is called encapsulation. The following lines of code will produce compiler errors:

```
r.myLength = 5;
r.myWidth = 7;
cout << r.Diagonal() << endl;
```

Here is the complete header file for the class.

First lets call this file *rectangle.h*.

```
#ifndef _RECTANGLE_H // Determines if this header file as been defined before
#define _RECTANGLE_H // If not define it

class Rectangle
{
public:
    // Constructors
    Rectangle() // Default constructor
    Rectangle(double l, double w);

    // Accessors
    double Area() const;
    double Perimeter() const;

    // Mutators
    void SetLength(double l);
    void SetWidth(double w);

private:
    double Diagonal() const; // private member function
    double myLength; // State of class
    double myWidth; // State of class
};
```

```
#endif // end of _RECTANGLE_H definition
```

Here is the implementation file for the class. Lets call this file rectangle.cpp

```
#include "rectangle.h"
#include <math.h>          // Needed for pow and sqrt functions

Rectangle::Rectangle()    // Default constructor
{
    myLength = 1;
    myWidth = 1;
}

Rectangle::Rectangle(double l, double w)
{
    if (l > 0)            // Check for valid input
        myLength = l;
    else
        myLength = 1; // Set the state to something value
    if (w > 0)
        myWidth = w;
    else
        myWidth = 1;
}

double Rectangle::Area() const
{
    return myLength * myWidth;
}

double Rectangle::Perimeter() const
{
    return 2*myLength + 2*myWidth;
}

void Rectangle::SetWidth(double w)
{
    if (w > 0)            // Make sure we keep the private data valid
        myWidth = w;
}

void Rectangle::SetLength(double l)
{
    if (l > 0)            // Make sure we keep the private data valid
        myLength = l;
}

double Rectangle::Diagonal() const
{
    return sqrt(pow(myLength,2) + pow(myWidth,2));
}
```

The :: operator between the class name and the class function (Rectangle::Area) is called the scope resolution operator. The left side of the :: indicates the class that the function, which is on the right side of the ::, can be found.

The accessor functions all have the reserved word const at the end of the function. By adding const at the end of the function definition, it indicates that the private data fields cannot be changed, and if that function tries to change the private fields it will produce a compilation error.

We need the definitions from the header file, so at the beginning of this file we need to include the header file.

Finally we have the client program that uses the class.

```
#include <iostream.h>
#include "rectangle.h"

int main ()
{
    Rectangle rect1;
    Rectangle rect2(3, 5);

    rect1.SetLength(5.5);
    rect1.SetWidth(.5);

    cout << "Area of first rectangle is " << rect1.Area() << endl;
    cout << "Perimeter of second rectangle is " << rect2.Perimeter() << endl;

    return 0;
}
```

The client program needs to include the header file for the class. When the client builds a project they will also have to include the rectangle.cpp file along with the client .cpp file.