PIC 10A

Lecture 16: Classes II

Point class

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

class Point
{
public:
   Point();
   Point(double new_x, double new_y);
   void move(double dx, double dy);
   double get_x() const;
   double get_y() const;

private:
   double x;
   double y;
};
```

Important concepts:

Public, Private
Member functions,
Variables
Accessors, Mutators
Constructors
Syntax

Constructor definitions

```
Point::Point()
{
   x=0.0;
   y=0.0;
}
```

- This is a default constructor
- Constructors have the same name as the class.
- There is no return type
- Private variables are “global” to the class

```
Point::Point(double new_x, double new_y)
{
   x=new_x;
   y=new_y;
}
```

- This is a constructor with parameters
- Parameters cannot have same name as private variables.
- We can have multiple constructors with same names! How does computer know which one to use?

Member function definitions

```
void Point::move(double dx, double dy)
{
   x+=dx;
   y+=dy;
}

double Point::get_x() const
{
   return x;
}

double Point::get_y() const
{
   return y;
}
```

Point class main

```
int main()
{
    Point P1;
    Point P2(1.0,3.0);

    cout << "x coordinate of P1 is: " << P1.get_x()
         << ". The y coordinate of P1 is: " << P1.get_y() << "\n";

    cout << "x coordinate of P1 is: " << P2.get_x()
         << ". The y coordinate of P2 is: " << P2.get_y() << "\n";

    cout << "Moving P2..."  << "\n";

    P2.move(3.0,2.0);

    cout << "x coordinate of P1 is: " << P2.get_x()
         << ". The y coordinate of P2 is: " << P2.get_y() << "\n";

    return 0;
}
```

Warning about const member functions

When declaring member functions as const you have to make sure that your member function does not change the private variables!

This means that your const member function cannot call non-const member functions.

Even if the function you are calling from inside a const function does not change the private variables, the computer thinks that there is a possibility that the function you are calling changes the private variables. This would mean that your const function might be indirectly changing the private variables!

Recall

If you don't write any constructors the computer creates a default constructor for you! This constructor does not take any arguments.

You can create an instance of your class:
Point P;

This default constructor creates the object but does not initialize the variables.

You can have several constructors for your class. One constructor could be the default constructor and the second one that takes arguments.

If you write a constructor that takes arguments you no longer have the computer created default constructor. So you should write a default constructor yourself.

Product class example

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

class Product
{
public:
    Product();
    void read();
    bool is_better_than(Product b) const;
    void print() const;
private:
    string name;
    double price;
    int score;
};
```

Product class example

```
Product::Product()
{
    name = "";
    price = 1;
    score = 0;
}
```

Product class example

```
void Product::read()
{
   cout << "Please enter the model name: ";
   getline(cin, name);
   cout << "Please enter the price: ";
   cin >> price;
   cout << "Please enter the score: ";
   cin >> score;
   string remainder; // Read remainder of line
   getline(cin, remainder);
}
```

Product class example

```
bool Product::is_better_than(Product b) const
{
    if (price == 0) return true;
    if (b.price == 0) return false;
    return score / price > b.score / b.price;
}
```

- We compute score/price to see which product gives you better “bang for the buck”.
- We return true if the current product has better ratio than product b.
- Be sure to avoid division by zero!
- To call this function for Products a and b, use something like:

```
if ( a.is_better_than(b) ) cout<<“Buy a!”;
```

- We can access the price of a because we are in the instance of a, however it is slightly strange that we can access the price of b just by writing b.price. Recall that non-member functions can’t get to price directly. We might expect that we would have to do b.get_price(). This is not necessary in this special case because b is also of type Product.

Product class example

```
void Product::print() const
{
    cout << name
         << " Price: " << price
         << " Score: " << score;
}
```

Product class example

```
int main()
{
   Product best;
   bool more = true;
   while (more)
   {
      Product next;
      next.read();
      if (next.is_better_than(best))
         best = next;
      cout << "More data? (y/n) ";
      string answer;
      getline(cin, answer);
      if (answer != "y")
         more = false;
   }
   cout << "The best value is ";
   best.print();

   return 0;
}
```

What our program files look like now

Structure of the program looks like this:

- #include <libraries>
- using namespace std;
- class declarations { };
- class member functions
- function declarations
- program functions
- main routine

Rectangle class

Rectangle is defined by its height and width.

We want to be able to draw a rectangle at a specific location, so we also need a point.

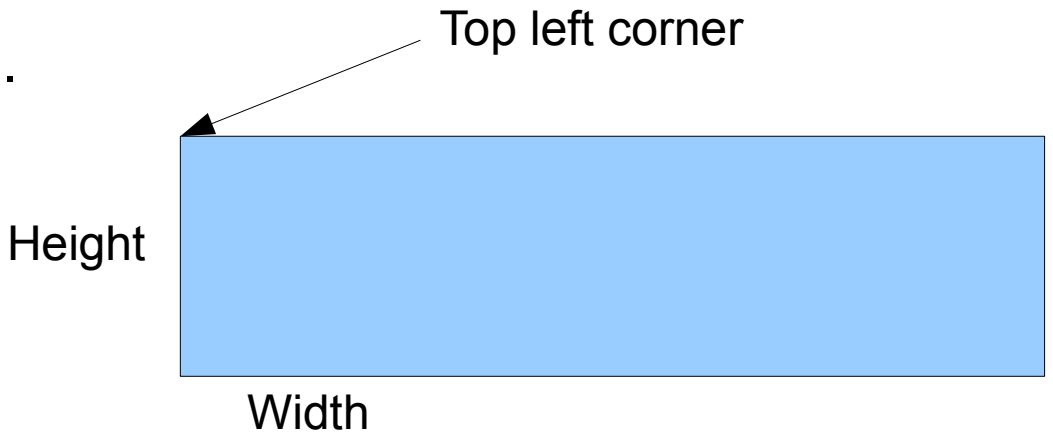
Rectangles location will be determined by the top left corner point.

We should have a member function that draws the rectangle.

We will also write a member function to move the rectangle.

We also need constructors.

Ok lets write it...



Example

```
#include "ccc_win.h"

class Rectangle {
public:
    Rectangle();
    Rectangle(Point new_topLeftCorner, double height, double
width);
    void draw() const;
    void move(double dx, double dy);
private:
    Point topLeftCorner;
    double height;
    double width;
};
```

Example

```
// Default constructor.

Rectangle::Rectangle() {
      topLeftCorner = Point(0,0);
      height = 1.0;
      width = 1.0;
}
// Constructor that takes Point, height, width as input.
Rectangle::Rectangle(Point new_topLeftCorner, double new_height, double new_width) {
      topLeftCorner = new_topLeftCorner;
      height = new_height;
      width = new_width;
}
// Draws the Rectangle on the screen using cwin<<Line
void Rectangle::draw() const {
      Point topRight = Point(topLeftCorner.get_x()+width,topLeftCorner.get_y());
      Point bottomLeft = Point(topLeftCorner.get_x(),topLeftCorner.get_y()-height);
      Point bottomRight = Point(topLeftCorner.get_x()+width,topLeftCorner.get_y()-height);
      cwin << Line(topLeftCorner,topRight)
            << Line(bottomLeft,bottomRight)
            << Line(topLeftCorner,bottomLeft)
            << Line(topRight,bottomRight);\
      return;
}
// Moves rectangle horizontally by dx, vertically by dy.  Uses the Point move function.

void Rectangle::move(double dx, double dy) {
      topLeftCorner.move(dx,dy);
      return;
}
```

Example

```
int ccc_win_main() {
    Rectangle r1;
    Rectangle r2(Point(-3,5),3,5);
    r1 = Rectangle(Point(2,-3),6,6);
    r1.draw();
    r2.draw();
    r2.move(-3,-2);
    r2.draw();

    return 0;
}
```

Function overloading

Having multiple constructors with same name is example of something called function overloading.

You are allowed to have functions with same names provided that:

1) They have different number of arguments

```
int foo(int a, int b);
int foo(int a);
```

2) The types of arguments are different.

```
int foo(double a);
int foo(int a);
```

It is not enough for return types to be different:

```
void foo();
int foo();  // Illegal
```

Function overloading

```
double average(double x1, double x2)
{
return (x1+x2)/2;
}

double average(double x1, double x2, double x3)
{
return (x1+x2+x3)/3;
}
```

Now from main I can call either function by:

```
x = average(a,b);
```

or

```
x = average(a,b,c);
```

Operators are functions too!

Operators such as + - % == are nothing but functions that are used with differe syntax from normal functions.

We don't write +(a,b) we write a+b.

Just as you can overload regular functions, you can overload operators.

Next time we will talk about how to define operators for classes.