Lecture 17: Classes III, overloading
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

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

2) The types of arguments are different.

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

It is not enough for return types to be different:

```c
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 different 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.

Specifically we want to overload operators for classes!
Comparing classes

Wouldn't it be nice if we could compare classes like we compare variables?

For example we wrote earlier for products a and b:

```java
if (a.is_better_than(b))
{
    // Do stuff
}
```

Instead we would like to write:

```java
if (a > b)
{
    // Do stuff
}
```
We need to define for the computer what it means to compare two classes. We do this by overloading the > operator.

In our class declaration we make a small change:

class Product
{
public:
    Product();
    void read();
    // bool is_better_than(Product b) const;
    bool operator > (Product b) const;
    void print() const;
private:
    string name;
    double price;
    int score;
};
bool Product::operator>(Product b) const
{
    if (price == 0) return true;
    if (b.price == 0) return false;
    return score / price > b.score / b.price;
}
while (more) {
    Product next;
    next.read();
    // if (next.is_better_than(best))
    if (next > best)
        best = next;
    
    cout << "More data? (y/n) " ;
    string answer;
    getline(cin, answer);
    if (answer != "y")
        more = false;
}
We can overload other operators as well. Without knowing we have already been using one other overloaded class operator:

```c++
string name = "Homer " + "Simpson";
```

+ is overloaded for the string class.

We could overload >=, ==, *, +, - even %.
Rectangle class example

Let's overload == for our Rectangle class.

```cpp
class Rectangle
{
    public:
        Rectangle();
        Rectangle(Point new_LeftCorner, double new_height, double new_width);
        void move(double dx, double dy);
        void draw();
        bool operator == (Rectangle r);

    private:
        Point LeftCorner;
        double width;
        double height;
};
```
When should we consider a Rectangle R1 equal to Rectangle R2?

There are many ways one could do it.

We will compare their areas.

```cpp
bool Rectangle::operator == (Rectangle r) {
    if (height * width == r.height * r.width)
        return true;
    else return false;
}
```
if (R1 == R2)
{
    // Do stuff
}
Passing classes to functions

When an instance of a class is passed to a function by default it is passed by value just as any ordinary variable.

If you want your functions to change your object you need to pass it by reference.

```cpp
void foo (Point &P)
{
    P.move(1.0, 3.9);
}
```

There are other reasons to pass your object by reference. Making a copy of a class costs a lot of overhead.
Separate compilation

Structure of our programs looks like this:

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

Help! Its getting crowded in this .cpp file!
Reasons to split your code into files

• Make your files shorter

• Improve organization

• Make your files easier to read

• When the compiler runs it only recompiles the files that have changed

• When working in a team each person can be responsible for their own file
What your header file should contain

- Your class declarations
- Declarations of nonmember functions (prototypes)
- Declarations of global variables
- Declaration of constants (global)

In general we put declarations of all shared “stuff”.
The source file contains

- Definitions of member functions
- Definitions of non-member functions
- Definitions of global variables

In general the source file implements everything that the header file declared.

Note: This source file does not contain main!!
We will break our product class into three separate files.

**product.h**
This will contain our product class declarations

**product.cpp**
This will contain all the member functions

**product_main.cpp**
This file will contain the main routine
#ifndef PRODUCT_H
#define PRODUCT_H

#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;
};

#endif
#include <iostream>
#include "product.h"
using namespace std;

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

/*
   And all the other member functions...
*/

Main thing to note here is that we have to include product.h here.
```cpp
#include <iostream>
#include "product.h"

int main()
{
    Product best;

    bool more = true;
    while (more)
    {
        Product next;
        next.read();

        // ETC.
    }

    Why don't we just write the main into the product.cpp file?
```

We of course need to include product.h in our main.cpp file to use the product class.
See separate compilation example for our rectangle class.