

Lecture 16: Classes – Overloading and Files



PIC 10A
Todd Wittman

Overloading

- Last class we mentioned you can define a function with several different parameter choices. This is called overloading a function.

```
void drawMap ( );
```

```
void drawMap (Point TheShire);
```

```
void drawMap (Point TheShire, Point Moria);
```

```
void drawMap (string mapName);
```

- When we call the function drawMap, it picks the function with matching number of parameters and parameter types.

Operator Overloading

- We can also overload an operator, which defines operations like `+*/><` for a class.
- Recall we had an `is_better_than` function for the `Product` class.
- Remember this compared the “bang-for-the-buck” ratios of the the `Products` `next` and `best`.

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

- We used it like this:

```
if (next.is_better_than (best) )
```

Operator Overloading

- Similarly, we could define the operator `>` for the `Product` class.

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

- Then we could call with:

```
if (next > best)
```
- To be thorough, you could also define `<`, `>=`, `<=`.
- We saw operator overloading with `+` for the `string` class.

```
string name = “Frodo “ + string2;
```

Sec 5.7: Accessing Data Fields

- Only the member functions can access the private fields of a class.
- A member function just uses the variable directly, it's already declared in the class private section. These are called implicit variables.
- Recall the Point class has private variables x, y and a move function.
- The explicit variables dx, dy have to be passed to the function.

```
class Point {
```

```
    public: void move (double dx, double dy) const;
```

```
    private: double x, y;
```

```
};
```

```
void Point::move (double dx, double dy) const {
```

```
    x = x + dx;
```

```
    y = y + dy;
```

```
    return;
```

```
}
```

dx, dy are explicit variables -- to be used they have to be passed to the function

x, y are implicit variables -- they can be accessed directly by all member functions

Sec 5.8: Member vs. Non-member Functions

- To prevent a member function from changing an implicit parameter, add the word **const** to the function.

```
void Point::move (double dx, double dy)
```

Can change the implicit parameters x, y

```
void Point::move (double dx, double dy) const
```

Can't make changes to x, y

- For non-member functions, we have to add an **&** to change the value of an explicit parameter.

```
void move (double dx, double dy)
```

Can't change explicit value parameters dx, dy

```
void move (double& dx, double& dy)
```

Can change explicit reference parameters dx, dy

Member vs. Non-member Functions



	Explicit Parameter	Implicit Parameter
Value Parameter (cannot change)	Default: Changes to a passed parameter are not sent back	Use const after function to prevent changes
Reference Parameter (can change)	Pass with a & to send back any changes	Default: Private variables can be changed by any member function

Operator Overloading

- As another example, let's define the `<=` sign to compare two Rectangles based on their area.



`r1 <= r2`

- The declaration should look like:

```
bool operator<= (Rectangle r2) const;
```

File Layout

- Last class, we saw how to declare classes all in one file:
 - Included header files
 - Constants
 - Global variables
 - Class declarations
 - Class member functions
 - Functions
 - Main routine
- Note: This is how you're expected to set up your file for HW 5.
- But this makes your cpp file rather long and complicated.
- And it would be nice to re-use the classes you made in other programs.
- We can put the class information in a separate header file.



Sec 5.9: Separate Compilation

- The source file (.cpp) contains the basic program.
 - Definitions of global variables
 - Non-member functions
 - The main routine
- The header file (.h) contains the definition of the classes we use.
- The header file sets up and supports your program. Some useful things you might want to separate from the source file:
 - Definition of constants
 - Declaration of classes
 - Definition of member functions
 - Declaration of non-member functions
 - Declaration of global variables
- Then this header file can be used for other programs.
- For example, you've used "ccc_win.h" in your graphics programs.

Using Header Files

- The source file includes the header file at the top:

```
#include "filename.h"
```

- The basic header file looks like:

```
#ifndef FILENAME_H
```

```
#define FILENAME_H
```

```
#include <libraries>
```

```
**YOUR CLASSES HERE**
```

```
#endif
```

- The **ifndef** statements prevent multiple inclusion.
- So even if this header file is included by more than one program, it will still only be compiled once.

Typical File Setup

- Put the main routine and its non-member functions in the main source cpp file.
- Put Class declarations in Class.h (*or whatever class name is*).
- Put Class member function definitions in Class.cpp (*same name*).

main.cpp (*e.g. hw6.cpp*)

```
#include "Class.h"

void fun() {
...
}

int main() {
....
}
```

Class.h

```
#ifndef CLASS_H
#define CLASS_H

class Class {
...
};
#endif
```

Class.cpp

```
#include "Class.h"
Class::Class () {
...
}

void Class::fun () {
...
}
```

- Note both main.cpp and Class.cpp #include the Class header file.
- Let's take a quick peek at your ccc graphics files...

Header Files for Declarations

- Even if we don't use classes, it's common programming practice to declare functions in a separate header file.
- So in hw5.h, we might have:

```
void DrawMap( );  
double distance2points (Point x, Point y);  
double distancePointLine (Line xy, Point a);
```
- For your HW5, you could create such a file with just these 3 lines. *(Don't do this! I'm just saying that you theoretically could do it.)*
- Then in the source file hw5.cpp, we define the functions as normal. Remember to add to the top:

```
#include "hw5.h"
```
- A curious programmer can just look at your header file to see what functions it contains and whether it would be of any use to her.

Really Global Variables

- Suppose we want a variable to be available to all source files using that library.
- For example, your graphics programs all used the **cwin** object for drawing.
- Declare it as an external variable in your .h file.

```
extern GraphicWindow cwin;
```
- The associated source file has the definition:

```
GraphicWindow cwin;
```
- The variable **cwin** can be used by all the functions in all the files. *One variable to rule them all!*

Art vs. Science

- Do you have to use header files to declare functions? No, but it's a little more aesthetically pleasing than throwing everything into one file.
- The last thing in Chapter 6 is a very philosophical question: *Is computer programming an art or a science?*
- It's a little both. Your homework program has to compile, but you also get graded on style.
- You should be proud of your artwork. You can e-mail your executable files to your family and friends. Let your parents know what their tuition checks are paying for.