Lab 5.1

Out: Wednesday, March 2nd, 2005

What you'll learn.

In this lab, you will write a C++ class complete with a Constructor, Destructor, member variables, and overloaded operators.

How you'll do it.

1 Task 1: Creating a Header File

In C, the header file was used to declare the function prototypes and sometimes define structs that would be used by the functions. In C++, there is generally a header file (H, h, hpp) and a code file (C, c, cpp, etc.) for each class.

Our class will be called "`Image`." Using simple names for classes is encouraged, but there is the potential for a naming conflict. In C++, two classes are not allowed to have the same name, however, the language does provide for the concept of namespaces. With namespaces, only classes in the same namespace need unique names. Two classes could share a name, but only if they are in different namespaces. We will be adding this class to a special namespace called "cs34."

To make a class part of a namespace, you must do the following when you declare the class:

```
namespace MyNamespace {
  class MyClass {
  public:
    MyClass();
    ~MyClass();
  };
}
```

One other new C++ feature that needs some explanation is iostreams. In C++, a concept called streams replaces the standard printf()/scanf() duo. There will be more information in a later section of this lab about streams, for now just know that they are in the namespace std::, and you must `#include <iostream>` in order to use them.

Your header should declare the following struct:

```
struct Color {
  unsigned char r;
```

```
  unsigned char g;
  unsigned char b;
};
```

As well as the following member functions in the `Image` class:

```
Image(std::istream& input);            //constructor
~Image();                              //destructor
Image(const Image& other);             //copy constructor
Image& operator=(const Image& other); //assignment operator
void write(std::ostream& output);      //output the image
Color getPixel(int row, int col);      //get a particular pixel
int getWidth();                        //get the width of the image
int getHeight();                       //get the height of the image
```

Task: Write a header file for a class called `Image` and a struct called Color, and make them part of the cs34 namespace.
In order to implement these methods, you will probably need to declare some member variables as well, but we'll let you decide exactly what is needed.

2 Task 2: Creating the C file and compiling

Now that you've filled in the header file, you can write the actual implementation of the `Image` class. The first step is to create a skeleton C file which does nothing except compile.

For each method that appears in your header file, you will need to define a method in the C file with the exact signature. For example, this is how you would define the the `read()` member function:

```
void
cs34::Image::read(FILE* Image)
{
  //method body
}
```

It can become tedious to type `cs34::Image` every time. Instead, we can simply add the cs34 namespace to this C file by typing the following line (preferably right after you include `Image.H`):

```
using namespace cs34;
```

Once you have that line, you can omit the `cs34::` everywhere you see `Image`. You could also include the "std" namespace here so as to avoid having to type `std::string` every time.

Since some of the member functions of the `Image` class have return values, you will need to temporarily have them return dummy values so that they will compile.

> **Task:** Create the skeleton C file for the `Image` class.
>
> Create another file called main.C which has an empty main function. At the top of this file, include `Image.H`
>
> Now, create a Makefile to compile the skeleton. You need to use g++ instead of gcc to compile C++ code. Try the compiling the program.

3 Task 4: Implement the `Image` class and test it

Now that everything is set up, all that remains is to fill in the bodies of the member functions. The entirety of the method bodies will take roughly 50 lines of code to write. You'll notice that there's a lot more overhead in C++ than there is in C. Sometimes, that's the price you pay for the "nice" features in C++ (most of which you'll see in a later lab).

Before you can write the constructor there are a few things you'll need to know about streams. First, a little motivation.

One of the most annoying things about C is the dealing with strings. Anytime you wanted to make a copy of string you first had to find out its size, then declare a char*, then malloc the char* to be the size of the string you want to copy (plus 1!), and finally, you had to copy the old char* into the new char*. C++ provides functionality to make dealing with strings much easier using std::string, and the iostream library.

The iostream library is very general and powerful. Using it to its full potential requires a lot of practice. However, we can do some simple things with streams very easily. For example, consider the following small program:

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

int main() {
  cout << "Hello, World!" << endl;  //same as printf("Hello, World!\n");
  return 0;
}
```

This is a simple "Hello, World" program. `cout` (actually `std::cout`) is a special kind of stream called an `ostream` (output stream). A stream is really just a buffer for characters, but it is "smart" in that you never have to worry about filling up the buffer or deallocating space for the buffer when you take data out of it. The stream handles those operations for you. An ostream is `special` because you are only allowed to push data into the stream, not take it out. Furthermore, `cout` is a very special instance of an ostream that happens to print out to the terminal whatever you push into it. The double angle brackets are called a stream operator. **The angle brackets always point in the direction that data is flowing!**

Here's another simple program:

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

int main() {
  string user_input;
  cin >> user_input;
  cout << user_input;
  return 0;
}
```

Notice that the angle brackets after `cin` are now facing the other way. That is because in this case, we are reading input from the terminal. The data is flowing from the terminal, to the string called user_input. This program would actually only read from the terminal until it encountered whitespace or a newline character. The stream operator is smart in that it can be used for formatted data input/extraction. Depending on what type of variable is on the right hand side of the stream, a different overloaded version of the stream operator will be called. For example, having a char on the right hand side will read will take only one char from the stream no matter what. Putting an `int` there would cause the stream to read only if the next thing it finds in the stream (ignoring whitespace) is an integer.

File input output is handled in exactly the same way, except using a specialized kind of stream which is called `fstream`. You must `#include <fstream>` in order to use those kinds of streams. For more information about streams, you can browse the man pages or take a look at this handy web page: `http://www.cplusplus.com/ref/`

Now that you know streams, you should have no problem writing the constructor for `Image` which takes a reference to an istream, and treats it as data from a PPM file. Note that streams are almost always passed by reference.[1] Don't ever pass a stream by value unless you can think of a really good reason for doing so. Before moving on to the other member functions, trying doing steps 1-3 of the task below.

Once you've got the constructor working, the rest of the member functions should be no problem. Take a look at the lecture notes from this week for information on how to write the destructor, copy constructor, and operator= methods. The getPixel() function should take a row and a column and return the `Color` located at that location in the `Image`. The write function takes an ostream reference as an arguments, and should essentially do the reverse of what the constructor did. One important note about the write function: use the get function to find the `Colors` instead of accessing the `Colors` directly with the member

[1] Passing by value means a local copy of the object is made. On the other hand, passing by reference will allow you to modify the original object.

variable. You'll see why in the next lab.

Task: Fill in all the methods in `Image.C`

In main.C write the following test sequence:
1. Use `argv[]` to get the name of a file to load.
2. Create an `ifstream` (input file stream) using the filename
3. Make an instance of the `Image` class, passing the `ifstream` to the constructor
4. Use the '=' operator to make a copy of the `Image`
5. Create an `ofstream` (output file stream) using a different filename
6. call the write() method of the Image copy, passing the `ofstream` as an argument.

If everything is working correctly, the output image should be the same as the input image.

4 You're Done!