MIT EECS 6.815/6.865: Assignment 0:

Introduction to C++ and the 6.815/6.865 Image Class

Due Wednesday September 16 at 9pm

1 Summary

- Learning C++
- Compiling, Debugging and Submitting Code
- C++ (Source/Header, Static Typing, Text I/O, Classes)
- The Image Class

2 Installation

Throughout this course, you will develop a C++ library for computational photography that you can use in future projects.

Like Java, C++ is an object-oriented programming language. Its syntax is pretty similar to both C and Java. For those familiar with Java, http://www.cprogramming.com/java/c-and-c++-for-java-programmers.html can be a great resource. We also suggest using http://www.cplusplus.com/.

C++ is one of the most widely used programming languages in the world. Therefore, if you come across an error it is very likely that someone else has already encountered this problem. Just Googling your errors can often be the best first step in debugging.

C++ is a compiled language, which means C++ code must be compiled before it can be executed. In this section, we will go over how to compile and submit your code. Regardless of which compiler you choose to use, your code must compile on our online submission system.

2.1 Compilers

The code you write should be portable enough to compile with any compiler. The submission system uses the GNU `g++` compiler on Linux. You can use this compiler either your own machine running Linux or on the machines in the Athena cluster to compile your code (to install on Ubuntu: sudo apt-get install build-essential). You can use the same setup on Mac OS by installing command line tools either through Xcode (Preferences → Downloads) or the terminal (http://osxdaily.com/2014/02/12/install-command-line-tools-mac-os-x/). The process may be different for different versions of OS X; similar tutorials can easily be found online. On Windows, you can use Cygwin to create a unix-like shell and use the same compilation setup (https://www.cygwin.com/).

There are 4 `.cpp` source files in the assignment folder: `a0_main.cpp`, `a0.cpp`, `Image.cpp` and `lodepng.cpp`. Normally, you would compile this collection of files by executing the following command:

```
g++ -Wall -o a0 a0_main.cpp a0.cpp Image.cpp lodepng.cpp -I.
```

This compiles the four `.cpp` files and generates the output (`-o`) `a0`. The `-I.` flag tells `g++` to look for the headers (or include files) in the current directory (`.`). The assignment folder contains three headers: `a0.h`, `Image.h`, `lodepng.h`. See 3.1 for more details on header and source files. The `-Wall` flag (for 'warn all') tells the compiler to display all warning messages emitted during compilation.

If you have `g++` installed, compile the code by going to the code directory and typing the above command. Check that an executable named `a0` is created. You can run the executable with the following command:

```
./a0
```

If you are successful, you should get a message that says
`Congratulations, you have compiled the starter code!`

2.2 Makefiles

Typing the compile command directly each time you want to re-compile can become cumbersome, especially if you have many `.cpp` files that have complicated dependencies. For example, if you are making changes to only one `.cpp` file, recompiling all of them every time (which is what the previous command does) is time consuming and inefficient. `Makefiles` are a simple way to organize code compilation. A `Makefile` contains a list of commands to compile a series of files as well as their dependencies. While in a directory containing this `Makefile`, you can type `make` and the commands in the `Makefile` will be executed. Open the `Makefile` in a text editor and see how compiler commands and source file dependencies are specified. Now, compile your code by going to the code directory and typing

```
make
```

on the command line. This will also create an executable called `a0`. You can run this by typing

```
make run
```

Alternatively you could use an IDE such as Visual Studio (https://www.dreamspark.com/Student/Default.aspx) on Windows.

We strongly recommend that you stick to the `Makefile`/`g++` approach for this class. We will try to help you with compilation problems during office

hours.

1) Make sure you can compile the starter code without problems. There is nothing to turn in for this problem, but if you can't compile the code, you are unlikely to complete the remainder of the assignment.

2.3 Submission System

The online submission system compiles your code on our servers, executes it and then displays the output. All text written to the standard output and standard error streams is printed on screen. Any `.png` images written to the `asst/Output` directory is displayed by the submission system, so you can verify that your code is working properly through the system. Keep all your code in the `asst` directory at the root of the `.zip` archive you submit. The submission system will ignore files outside this directory.

2.4 Debugging

You are welcome to use a debugger, such as `gdb` (http://www.gnu.org/software/gdb/) or the one that comes with your IDE. However, we also suggest you use `assert` statements or print statements to make sure conditions you think are true are actually true.

For example, if you are performing a division, you may want to assert that the divisor is not zero.

```
float safeDivision(float dividend, float divisor) {
  assert(divisor!=0, "Divisor is zero");
  return dividend/divisor;
}
```

If the divisor is zero, the program will abort and tell you which assertion failed. Otherwise, the returned value would be `Inf` or `NaN`, which might be undesirable.

2.5 Image Input/Output

Our `Image` class supports reading `.png` files only. All the sample images we give you will be in the PNG format and you can use one of a handful of tools to convert your own images to this format (e.g. http://image.online-convert.com/convert-to-png). In addition, the `Image` class can only write to `.png` files. You can use your favorite image viewer to view the images.

3 C++

In this section, we introduce a few C++ language features that are different from previous languages you may have used. C++ is most similar to Java. The big differences are that C++ has explicit memory management, distinguishes

between references and pointers and organizes code into header and source files. You can find more information in this C++ tutorial for Java programmers (http://www.cprogramming.com/java/c-and-c++-for-java-programmers.html) and this reference website (http://www.cplusplus.com/).

3.1 Headers and Source Files

C++ programs are usually organized into header and source files. Actual executable code is written in source `.cpp` files, while function and class declarations live in `.h` header files. Open the attached `a0.h` and `a0.cpp` files in a text editor or IDE of your choice. The function

```
void helloworld(float a, float b);
```

is in both the header file (`a0.h`) and the source file (`a0.cpp`). `j` While the `.h` file has only a *declaration* of the function's name and signature, the `.cpp` files provides it with an actual body of code: its *definition*.

Throughout the assignments, we will give you key function declarations in the header file, and you will implement them in the source file.

When you compile and execute your code, the program runs all commands in the `main` function located in `a0_main.cpp`. This allows you to execute your own functions and verify that your code is correct.

We will not grade the content of `a0_main.cpp` as we will replace it with our own unit tests.

3.2 Static Typing

All C++ variables must have a type. In 6.815/6.865, we use IEEE single-precision floats to represent the value of a pixel.

2a) Return a floating point variable `c` that is the sum of `a` and `b` in the function `void helloworld(float a, float b)`.

3.3 Text Input/Output

You may find it useful to print values to the screen for to debug your program or get information about what you are working on. You can do this using the syntax

```
cout << "Hello World!" << endl;
```

You can also use `cout` to display variables using the same syntax.

2b) Write statements in `void helloworld(float a, float b)` that prints the following

```
Hello World!
The value of a is _.
The value of b is _.
The sum of a and b is _.
```

where the underscores are replaced by the actual numbers that make the sentences true.

3.4 Classes and Functions

We have provided the specification for a class to represent Images (`Image.h`). The specification contains a number of methods and variables that belong to each instantiation of the Image class. Most of these methods are implemented in the source file (`Image.cpp`).

In a later part of this pset, you will implement some of the definitions in the source file `Image.cpp`. For now, just look over the header file and look for the two constructors on lines 18 and 21 of `Image.h`. They are

```
Image(int width_, int height_ = 0, int channels_ = 0,
      const std::string &name="");
Image(const std::string & filename);
```

You can create an instance of the Image class using either constructor. The first creates a blank image of dimensions `width_` $\times$ `height_` $\times$ `channels_`. You can use the first constructor to create an Image variable `my_im` that is 100×100 pixels with three color channels by typing

```
Image my_im(100,100,3);
```

3) Implement the function `Image readAnImage(const std::string &filename)` in `a0.cpp`, which returns an Image created using the second constructor taking `filename` as an input.

4 Submission

Turn in your files to the online submission system (link is on Stellar) and make sure all your files are in the `asst` directory under the root of the zip file. If your code compiles on the submission system, it is organized correctly. The

submission system will run code in your main function, but we will not use this code for grading.

In the submission system, there will be a form in which you should answer the following questions:

- How long did the assignment take?
- Potential issues with your solution and explanation of partial completion (for partial credit)
- Any extra credit you may have implemented
- Collaboration acknowledgment (you must write your own code)
- What was most unclear/difficult?
- What was most exciting?