MIT EECS 6.815/6.865: Assignment 0:

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

Due Wednesday September 15 at 9pm

1 Summary

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

2 Installation and Environment Setup

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 on 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/`) or use the Windows Linux Subsystems feature (https://docs.microsoft.com/en-us/windows/wsl/install-win10).

There are four `.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 -std=c++11 -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. The `-std=c++11` flag tells the compiler to use the C++11 standard.

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. Please take a moment to read through the "tutorial" comments we provided inside the `Makefile` to get a better idea of how the file is 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. We provide a `Makefile` rule to prepare a submission zipfile. Run it using

```
make prepare
```

This `make` rule will execute the necessary sequence of commands to produce a submission zip file. You can inspect the commands specified inside the `Makefile`. You can also manually zip your submission code. If you do so, please make sure to keep all your code in the `asst` directory at the root of the `.zip` archive and make sure to exclude all build outputs, executables as well as the `Output` folder and its contents as these will bloat the size of your zip file and slow down your upload.

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.

Warning: Renaming a JPG file's extension to `.png` does not convert the file type! You might think this is a silly comment but we include this because people have tried and failed to do so. Instead, you may open the image in some image editor and change the format when saving to convert a JPG file.

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`). While the `.h` file has only a *declaration* of the function's name and signature, the `.cpp` file 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.

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

3.3 Text Input/Output

You may find it useful to print values to the screen 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.

> 2.b Write statements in `void helloworld(float a, float b)` that print 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 Simple Image Class

In `a0.h` we have provided a simple image class (`SimpleImage`). The aim of this class is to create a binary image (just zeros and ones) to demonstrate the basics of how data can be stored and manipulated. The specification of the class contains the declaration of various methods, all of which have dummy implementations in `a0.cpp`. Throughout this exercise, you will be asked to provide proper definitions.

Images are inherently two-dimensional (three if multiple channels are available), however, it is often desired to store them as a single-dimensional array and index into them appropriately (we'll see more details about this in future problem sets). For our `SimpleImage` class, we will store a `width` $\times$ `height` image as an array of `width` $*$ `height` elements in row-major order. That is, consecutive columns are tightly packed (no separation between them) while consecutive rows have a separation of `width` elements.

In general, we do not know the size of images ahead of time. As a result, our arrays must vary their sizes. Rather than handle memory directly, we will generally rely on the Standard Library's `vector` container (see line 53 of `a0.h`). The vector container provides various convenience methods for array-like operations where the size of the array can vary arbitrarily. See http://www.cplusplus.com/reference/vector/vector/ for the container's documentation.

Now that the mechanisms for storing data have been outlined, we can move on to constructing the class, line 33: declares the class constructor:

```
SimpleImage(int width, int height);
```

The constructor takes care of creating an instance of the object when needed. As we would expect, the constructor above takes two input integers, the width and height of the desired image. For example, typing

```
SimpleImage my_im(10,10);
```

would form an empty image of size 10×10.

> 3.a Implement the `SimpleImage` constructor:
> `SimpleImage(int width, int height)` in `a0.cpp`.
> Make sure to keep all relevant information and initialize the image to zero (avoid adding elements one-by-one if possible).

An empty image is not of much use so let's go ahead and add some detail by using the class method

```
    void make_rectangle();
```

to create a white rectangle (that is pixel value of 1.0) in the center of our image. Let's make the rectangle be 20% of the width (and height respectively). It is sometimes helpful to think of the image plane as a continuous space (observed at discrete locations) and we will adopt this convention in this example; as a result, the center of the image is allowed to be fractional and similarly, any pixel that partially overlaps with the rectangle should be denoted as white. That is, calling

```
    SimpleImage img2(11,11);
    img2.make_rectangle();
```

should produce a white rectangle spanning $[4, 4]$ to $[6, 6]$. *(Note: the indices are zero-indexed and ranges are inclusive.)*

To provide additional examples:

- Image of size 5×3 would have a rectangle of one pixel at $[2, 1]$
- Image of size 4×4 would have a rectangle spanning $[1, 1]$ to $[2, 2]$
- Image of size 3×7 would have a rectangle spanning $[1, 2]$ to $[1, 4]$
- Image of size 10×9 would have a rectangle spanning $[4, 3]$ to $[5, 5]$

> 3.b Implement the class method `void make_rectangle()` in `a0.cpp`.
> This method should set pixels to white (1.0) that are within 10% (inclusive) of the image center in each direction (hint: check your

boundary conditions to ensure the rectangle is symmetric around the image center).

Sometimes it is helpful to compute statistics across the entire image. A potentially useful statistic could be the average pixel intensity. Let's add that functionality to our binary image.

3.c Implement the class method `float compute_average()` in `a0.cpp`. Loop over all image pixels and return the computed average.

We have an image, but no way of visualizing it. We could print the pixel values directly to the terminal, but this could be tedious to parse. Instead, let's use characters, similar to ASCII art (https://en.wikipedia.org/wiki/ASCII_art). We only have two values in our images so let's use the characters `-` and `+` to encode zeros and ones respectively:

3.d Implement the class method `void print_to_terminal()` in `a0.cpp`. Loop over the image and print pixel values as characters instead of numbers. Map the values so that zeros appear as `-` and ones appear as `+` in a `width` $\times$ `height` format (hint: you can use case statements or the values to index into an array of characters).

3.5 A Better Image Class

We have provided the specification for a class that better represents 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 the next pset, you will implement some of the definitions in the source file `Image.cpp`. In the meantime, look over the header file and look for the two constructors on lines 34 and 39 of `Image.h`. They are

```
Image(int width_, int height_ = 1, int channels_ = 1,
      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 $\times$ 100 pixels with three color channels by typing

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

4 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 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. The submission system should also show you the image your code writes to the `./Output` directory

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

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