Week 3 - C/C++ Toolchain

Hongzheng Chen Yucheng Chen

Nov 29, 2019

Week 3's seminar is partly based on Cornell ECE 2400: Computer Systems Programming lectured by Christopher Batten in Fall 2019.

1 C/C++ Introduction

As discussed in seminar, we use C and C++ as introductory languages for the following reasons:

- High-performance: C/C++ is extremely fast and is widely used in HPC. OpenMP and MPI are two most commonly used parallel libraries based on C, which will be covered in your *Parallel and Distributed Computing* course in Grade 2, *Distributed Systems* in Grade 3, and *High-Performance Computing* in Grade 3. Specific accelerators use C-like languages to program. For example, GPU uses CUDA to program (*Parallel and Distributed Computing*), and FPGA uses Verilog to program (*Digital Circuits* in Grade 1 and *Computer Organization* in Grade 2), both of which are C language's extension.
- Fundamental: C/C++ is very close to the hardware, thus makes you better understand how the computer works. *Operating Systems*, *Computer Networking*, and *Embedded Systems* in Grade 2 all require you have a good understanding of C. Otherwise, you can hardly make it work with assembly and correctly run on hardware.
- Influential: C/C++ influences many programming languages, including Java and C#, as depicted in Programming Languages Influence Network. Thus, if you learn C/C++ well, you will easily understand the primitives of other languages.

Two introductory books about C and C++ are:

- Brian Kernighan, Dennis Ritchie, *The C Programming Language (2nd ed.)*, Prentice-Hall, 1988 (Chinese version)
- Stanley B. Lippman, Jose Lajoie, Barbara E. Moo, *C++ Primer (5th ed.)*, Objectwrite, 2012 (Chinese version)

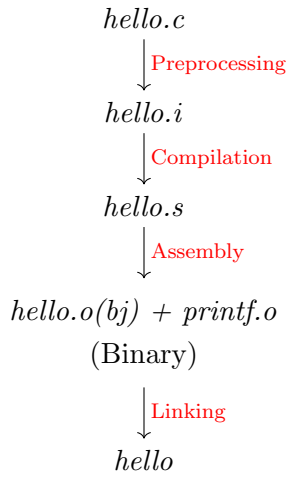
Please learn the theory of C/C++ carefully, since they may greatly influence your future courses. About C language's basic theory, you can see Topic 1: Introduction to C of *Cornell ECE 2400*, or refer to my Github notes.

Also, pay more attention to the difference between C and C++. C++ is NOT C plus STL.

C++ is more than that, with support to Object-Oriented Programming (OOP), templates and generic programming, and other modern language features. All of these can be found in the *C++ Primer* book.

2 Stages of Compilation

C language's compilation mainly consists of the four stages listed below. You have to clearly know what these stages do, which will give you better understandings when you read papers or do projects about compilers or systems in the future. I highly recommend you to read the *Computer Systems: A Programmer's Perspective (CS:APP)* book, Chapter 1 of which gives the overview of computer system stack.



Some experiments on these four stages can be found in this webpage, and also in Section 2: Compiling and Running C Programs of Cornell ECE 2400.

Common compilation options can be found here. About the difference between static libraries and dynamic libraries, you can refer to LearnCpp.

3 Auto Building

About Makefile's usage, there are lots of resources on the Internet can be referenced to. Following lists a few:

- My blog in Chinese
- CSDN Tutorial of Makefile in Chinese
- Section 3: C Build and Test Frameworks of ECE 2400 (also covers CMake)
- A Simple Makefile Tutorial
- GNU Manual

4 Debugging & Testing

Please refer to Section 4: C Testing and Debugging of ECE 2400, or my manual of gdb debugging in Chinese.

ECE 2400 provides a basic framework UTEST for testing. But nowadays the most commonly used testing framework may be Google's gtest. For English tutorial, you can refer to Unit Testing C++ with Google Test, or you can see Chinese one on Zhihu.

5 Profiling

The simplest profiling way is to measure the running time of a program. In C we can use `<time.h>` to time the program. In C++ we use `<chrono>` in C++11 standard. For the timing framework, please refer to this blog written by me. Also, we can use Linux's `time` command to time something. For example, we can directly type `time ./avg` to obtain the running time of the `avg` program. The output consists of three time — real time, user time, and system time, where we mostly care about the real time that is the wall clock time from start to finish the call. For more differences, please see Stack Overflow.

Other profiling tools on CPU and memory are `perf`, `pcm`, and `valgrind`. You can find them in Section 5: C Profiling of ECE 2400, and my blogs on pcm and Valgrind.

6 Assignments

6.1 Auto Building

In this assignment, you will learn how to build a small project and make each component function correctly.

The project demonstrates the basic usage of the Computer Graphics (CG) library OpenGL, which is commonly used in nowadays PC games. It reads in polygon mesh and displays the model on the screen, as shown in Fig. 1. Typing `w`, `a`, `s`, and `d`, the model will correspondingly move up, left, down, and right. Typing `r`, the model rotates clockwise; Typing `R`, it rotates anticlockwise.

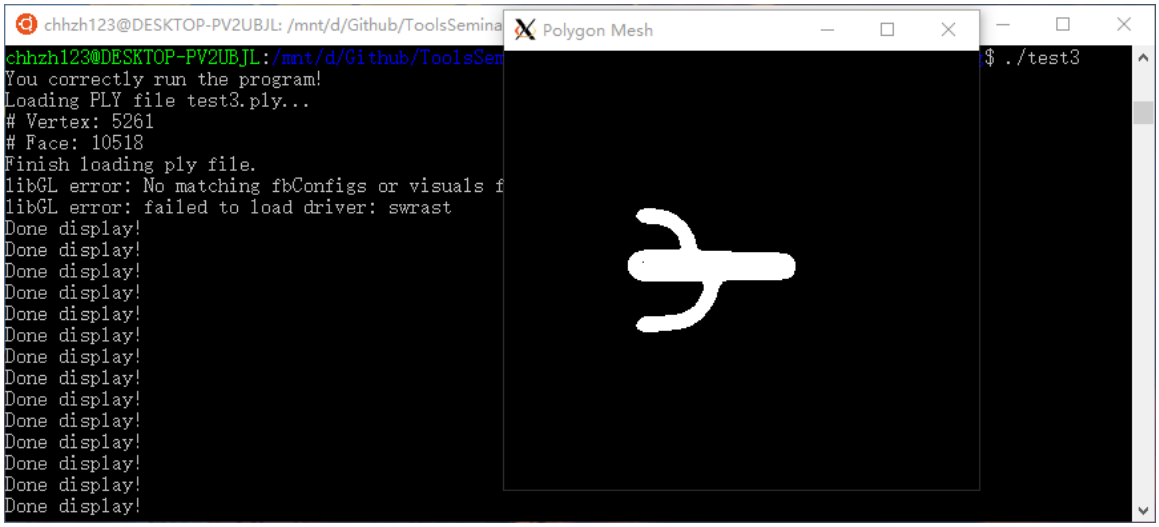


Figure 1: Expected output: A cactus

Several source program files are provided in `Assignments/CppToolChain-AutoBuilding` folder, please `cd` to that folder and have a look.
What you need to do is to **write a *Makefile* or *CMake* file** to build the project and generate the expected results as shown above. That is to say, you need NOT know how the program is working or how to use OpenGL. You only need to glue these files together.

Firstly you should install some dependency libraries, which provide minimum support for graphical display. Type the following commands on your Linux system.

```
sudo apt-get update
sudo apt-get install freeglut3 freeglut3-dev
```

After installing the libraries above, you can begin your experiments. To generate the output, you should follow the compilation and linking flow shown in Fig. 2. Since OOP and template are used, please use `g++` to compile.

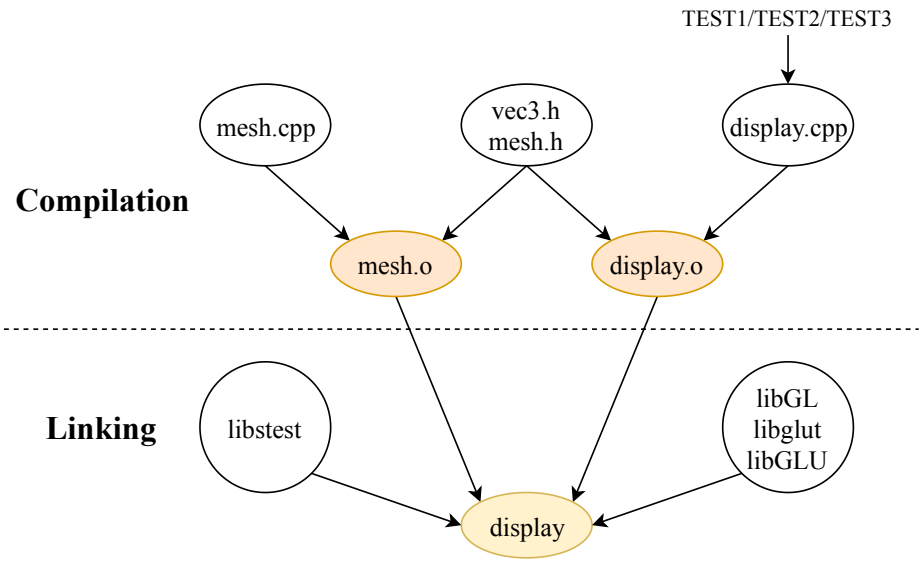


Figure 2: Compilation and linking flow

`mesh.cpp` and `display.cpp` are in the current folder. `vec3.h` and `mesh.h` are in `include/`. `libstest.a` is provided in `lib/` folder. `libGL`, `libglut`, and `libGLU` has been installed by `freeglut3` you just typed. `test1.in`, `test2.in`, and `test3.in` are three input files for the executable program. Thus, you only need to generate the colored three files in Fig. 2. Detailed descriptions as follow:

1. Compile `mesh.cpp` (which has included the `mesh.h` and `vec3.h` header) to `mesh.o`.
   (Hint: `-c` flag enables `g++` to compile the source file to the object file.)
2. Compile `display.cpp` (which also includes the `mesh.h` and `vec3.h` header) to `display.o`. Notice three different macros (`TEST1`/`TEST2`/`TEST3`) should be defined respectively, which can be viewed in Line 190-196 in `display.cpp`.
   (Hint: `-D` flag enables `g++` to pass predefined macros to preprocessors.)
3. Link the generated `mesh.o` and `display.o` with three CG libraries called `libGL`, `libglut`, and `libGLU`. Also, a test static library provided in `lib/libstest.a` should also be linked.
   (Hint: To link these static libraries, the first three characters `lib` can be omitted. For example, to link libGL, you only need to write `-lGL`.)

Basically, you will have three different `display.o` files respectively for test1, test2, and test3, and one `mesh.o` for all these tests. For example, for test1, you define `TEST1` when compiling `display.cpp`, generate `display1.o`, and obtain `display1` executable, which reads in `test1.obj` and outputs results.

If you correctly compile and link the files, you can run the generated `display` programs

and see *three* different things displayed on the screen. If you see that, cheers! You have done it!

I believe you will meet lots of problems in this project, so I list a few below:

Q1: Compile time error

A1: Maybe your `g++` is out-of-date (use `g++ --version` to check your version), please use the following commands to update your `g++` (my Ubuntu system uses `g++` 7.4.0 version) and change the default `g++` version.

```
sudo apt-get update
sudo apt-get install gcc-7 g++-7
sudo update-alternatives --config g++
```

Q2:

```
display.cpp:10:10: fatal error: mesh.h: No such file or directory
#include <mesh.h>
```

A2: Please check whether you correctly add include file search path in your command. (Hint: Use `-I` command.)

Q3:

```
display.o: In function 'init()':
display.cpp:(.text+0x1d): undefined reference to 'glClearColor'
display.cpp:(.text+0x707): undefined reference to 'gluPerspective'
display.o: In function 'main':
display.cpp:(.text+0x944): undefined reference to 'libTest()'
display.cpp:(.text+0x979): undefined reference to 'glutInit'
collect2: error: ld returned 1 exit status
```

A3: Please check if you link the three CG libraries (libGL, libglut, libGLU) and the test library (libstest) correctly.

Q4:

```
/usr/bin/ld: cannot find -lstest
collect2: error: ld returned 1 exit status
```

A4: Check if you add `lib` to the link file search path. (Hint: Use `-L` command.)

Q5:

```
Error: You do not load any models!
```

A5: Please check if you define any one of the macros (`TEST1`, `TEST2`, `TEST3`) when compilation.

Q6:

```
Loading PLY file test3.ply...
# Vertex: 5261
# Face: 10518
Finish loading ply file.
You correctly run the program!
freeglut (./test3): failed to open display ”
```

A6: This is good! You have done this project! This may result from your Linux system having no GUI. If you use WSL and insist to see the output figure, you can follow this page to obtain your results.

Q7: For any other questions, please Google or directly consult me.

Moreover, about compiling and linking files in this tree-like structure, you can find more guidance on ECE 2400's webpage.

To submit your homework, you need to `git push` the whole `CppToolchain-AutoBuilding` folder. But remember to remove unnecessary files like `mesh.o` and `display.o` before you commit.

6.2 Debugging

Use `gdb` or other debugging tools to find the bugs in the `Assignments\CppToolchain-Debugging\mergesort.c` program.

For details about the merge sort, you can refer to GeeksforGeeks.