COSC2325 Lecture Notes

Release SU17

Roie R. Black

May 29, 2017

CONTENTS:

CHAPTER

ONE

ARE YOU READY TO CODE?

By now, you have had a few programming courses. For a few of you, you have been writing programs for quite a while. My question is simple: are you ready to code?

Silly questiton! Of course you are ready to code!

But are you really?

1.1 How Do You Code Now?

The answer for many of you is also simple: "In my favorite IDE, which runs on Windows."

Sorry! You are not ready to code.

1.2 What Is Wrong With My IDE?

Nothing really, if you expect to spend a lot of time working in that environment. The problem is that you may have picked the wrong IDE. When you get your first real job, you will discover what the right IDE is (for your employer). You will need to drop what you love now, and learn to love a new tool. You will also have to learn other new tools and procedures that dictate how you will write code for that employer.

Oh yeah! You will also learn that your code is going to be scrutinized by your programming peers. You will face something called the "Peer-Review" where your code is basically shredded by others, and you get to defend every line you wrote (Shudder!)

1.3 Where Can You Code?

An even bigger problem lurks out there, ready to bite you! You may not end up working exclusively on a Windows machine. You might have to work on a Mac, or on a system running Unix/Linux. Are you ready for that?

1.4 A Real Example

In my email the other day, I got a summary of the results from the ACM-ICPC (*International Collegiate Programming Contest*) World Finals, a major competition to see who has the best programmers in the world. (Hint, the top-placing US team came in 13th, not a good sign!)

Here is where you would be programming in this competition:

- Hardware:
  - **Intel i7 NUC (Model NUC5i7RYHR)**
    - CPU - 5th Generation Intel Core i7-5557U (3.1+ GHz)
    - RAM - 16 GB
    - Disk - 128 GB SSD
    - Graphics - Intel Iris Graphics 6100
  - External screen: Lenovo ThinkVision L2321x LCD Monitor Model 4014-HB6 (23in wide, 1920x1080 Native Resolution)
  - External keyboard: Lenovo 41A5100, sub-model KU-0225, USB
  - External mouse: Lenovo 41U3013, USB
- **Software:**
  - **OS:**
    - Ubuntu 16.04.1 LTS Linux (64-bit)
  - **Desktop:**
    - GNOME
  - **Editors**
    - vi/vim
    - gvim
    - emacs
    - gedit
    - geany
  - **Languages:**
    - **Java**
      - OpenJDK version 1.8.0_91
      - OpenJDK Runtime Environment (build 1.8.0_91-8u91-b14-3ubuntu1~16.04.1-b14)
      - OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode)
    - **C**

            - gcc (Ubuntu 5.4.0-6ubuntu1~16.04.2) 5.4.0 20160609
        - **C++**
            - g++ (Ubuntu 5.4.0-6ubuntu1~16.04.2) 5.4.0 20160609
        - **Python 2**
            - Python 2.7.10 (implemented using PyPy 5.1.2). A list of the installed modules can be found here
        - **Python 3**
            - CPython 3.5.2. A list of the installed modules can be found here
    - **IDEs:**
        - Java - Eclipse 4.6 (Neon)
        - C/C++ - CDT 9.0.1 under Eclipse 4.6
        - **Python**
            - PyDev 5.1.2 under Eclipse 4.6
            - Pycharm Community Edition 2016.2.3

The problems teams faced in this years competition are interesting. Here is this year's problem set:

- `/files/icpc2017.pdf`

If you were lucky enough to reach this competition, you walked into a room with nothing but your warm body, and faced the specified working environment. The competition is timed, and teams work through the problems and submit their code for testing. Points are awarded for the entries that actually work!

Since this competition is pretty fast paced, I do not see that they considered program style in their scoring. I certainly would have required a look at that, perhaps as a last step when awarding points.

1.5 Secondary Goal of this Course

This brings me to the secondary goal of this course. I want to turn you into professional programmers. Well, actually, I want to start you on the path to becoming one. To do that, we are going to have to un-learn a few bad habits, and learn some new good-habits. We will also need to learn some new tools, and we will learn to work in a new environment.

Specifically, here are the tools you will be using in this class (note: no IDEs allowed!):

- Hardware:
    - Hopefully your personal 64-bit laptop, or a school lab machine
- Software:
    - Virtual Machine: VMware or VirtualBox
    - OS: Ubuntu linux
    - **Languages:**

    - GNU C/C++
    - Assembly language (several)
  - Editor: Vim (or one of its variants)
  - Source Code Control System: Git and Github
  - Build tool: GNU Make
  - Documentation system: Python Sphinx or LaTeX

1.6 DON'T PANIC!

I do not expect you to become wizards in all of this, for most of you, you have never heard of many of these items. We will introduce each one slowly, and get you working with them as quickly as we can. I do recommend that you construct "cheat-sheets" to help you learn how each tool is used.

All of these tools have been in use in this course for many years. A few students have trouble with some of them, but I am here to help you through that. I have had quite a few old students contact me years later, thanking me for pushing them in this direction. Once they got into "real jobs" they discovered that being a professional programmer is not as easy as they thought it would be! Having a good start is important, and *NOW* is the time to get that start!

1.7 Meet Guido

Although it is not politically correct, I like to introduce a character in my courses. His name is Guido, thanks to my growing up watching old Mafia TV shows. With aplogies to every Guido out there, this Guido has to maintain your code, and he has a very short fuse! He is not someone you want to cross! If you write code that Guido is going to have a hard time maintaining, he will hunt you down! You do NOT want that to happen, so think about Guido as you craft your code.

Guido know his stuff, and he expects you to write nice, clean, well documented (and maintainable) code that *works*. Nothing else will do!

1.8 What is a Working Program Worth?

If you write a program that solves a problem I pose in this class, and it works, it is worth a "C".

What? Why is that?

I expect your code to work. That is what I expect from all programmers I have ever worked with, or who worked for me. You have to do better than that to get that "A" you thought you should get.

To get the "A", your code has to be neat (follow accepted style guidelines), clear (good choices in names you choose), it has to compile and run error free (even warnings are a bad thing). It has to be documented (not just with embedded comments). It has to be tested (one sample run is not enough).

Oh! And Guido has to like what you wrote!

You will not get to this point without putting in some effort. I will get progressively more picky about all of this as the course progresses. For now, just think about how you currently work, and what habits you might need to break.

1.9 Why?

There is a point to all of this picky stuff. I want you to enjoy what you do, and end up working in a job where you are contributing in a real way to the development of something cool.

You will not get to that place by being sloppy at what you do. That "A" you are convinced is essential to getting you that job will do no such thing. It may get you standing in front of a potential employer, but the "A" will get you no further. It is what you can demonstrate that you *know* and *can do* that will get you that job. Most employers will run you through a testing process to prove you can do the job. (I once faced a grueling six hour interview with a team of folks who wanted to make sure I could run the entire IT department for a major city. I aced that test because I practice what I preach!)

Programming can and should be enjoyable, and you should be proud of what you can do! Otherwise, you are headed in the wrong direction.

You will spend an enormous amount of your life working at something. You better enjoy that! In my over 50 years of working, I have not had one job I would not gladly go back to and work there again. Maybe I was lucky, but I believe that having the right attitude, and intentionally developing your skills will get you a career like that as well.

CHAPTER

TWO

WORKING ON THE COMMAND LINE

In this class, you will not spend much of your time working with the mouse in one hand. Instead, we will learn how to manage your system and development activities using just the keyboard.

Why?

Modern software is not developed solely on your friendly Windows platform, and not everything can be done using the mouse. For some of you the mere thought of letting go of that friendly object strikes you with fear.

Fear not! I actually had the opportunity to put my hands on the first mouse which came with the first machine that used this critter for anything. The machine was a Xerox Alto, which I saw at Carnegie Mellon University in the 1970's.


Xerox Alto
Diablo

Note: The normal screen was paper-white (what else would you expect from Xerox?) and the cursor was a little guy on skis. People would grab the mouse and make him ski down the screen. It was entertaining!

Then Steve and Bill got a look at this machine, and the rest is history.

Before the Alto, all we used to control the computer, or write software, was a keyboard. We can still do that, but we do so using a special program called the "shell" or "terminal" on Linux and Mac, and the "command-prompt" window on Windows.

This is an important skill for a developer. There will be times when the system you need to work on simply does not have a graphical interface, and your only way to deal with that machine will be using the command line.

Note: The term "command line" comes from a line where you type in a "command" to the operating system. That command directs the system to do something, and usually results in some program running on the system. What that program is depends on what you need at the moment.

CHAPTER

THREE

CREATING A PROJECT

CHAPTER

FOUR

THE DEVELOPMENT TIMELINE

CHAPTER

FIVE

DOCUMENTING YOUR PROJECT

CHAPTER

SIX

GIT INTRODUCTION

What?

You just spent all day making changes to a major project that was working just fine, and now nothing works! You need to go back to yesterday and start over.

Can you do that?

Odds are the answer for many of you is: No! It will take another day to recover from this disaster!

Linus Torvalds manages over 1000 programmers who contribute to the **Linux_** kernel code, the hert of **Linux_**. He certainly cannot afford to let any one of those programmers break the project (which consists of hyndreds of directories, and thousands of files, BTW!)

WHoe there were many tools available to help with this problem, he was not satisfied with any of them. So, likle any good programmer, he write his own.

The tool he developed is called Git, and today Git is one of the most popular *source code control system* tools around. BEst of all, it is free!

We will be using Git to manage all of the code you create for this class. With any luck, you will never write another line of code that is not being managed by Git, ever again!

Note: Well, things do change. Maybe there will be a cooler tool that replaces Git some day!

The **Gnu_** project has developed a collection of incredible free software development and system management tools. If you do not know about **'Richard Stallman'_** and his **'Free Software Foundation'_**, you should look into it if you indent to develop software seriously. Although most of these tools are now associated with the **Linux_** operating system, Richard initially set out to produce a completely free version of the Unix operating system. Today, you can find these tools on just about any platform.

We will be using the *Gnu Compiler collection (GCC_)* to develop projects for this class. Specifically, we will use these major tools:

- **gcc** - the Gnu "C" compiler
- **g++** - The Gnu "C++" compiler
- **make** - the Gnu build tool"

CHAPTER

SEVEN

THE VIM PROGRAMMER'S EDITOR

You absolutely must learn how to use a good cross-platform editor, not just the one you currently use that only works on Windows. In my over 50 years of programming, the one tool I have been able to use on just about any computer system I have ever touched is a version on Vim, which is commonly installed on all Unix/Linux machines. Vim is readily available for Windows ans Mac users as well.

Picking an editor is a religious affair. One developer might love editor X, and another, editory Y. They will go head to head debating the relative merits of each.

When you leave this class, you pick the editor you want to use. It is you wou will ahve to use this thing, and you should learn it well. My rules for picking one is based on the simple fact that I work on a huge variety of systems, and Vim has always been there for me. YMMV!

I will ask you to use Vim in this class, just to give your the experience in using it. DOn't panic, it is not that hard to learn. I does have one significant quirk, though. It is called a "modal" editor, because at any given moment in time, Vim is operating in one of two major modfes. More on that later.

7.1 Installing Vim

Before we can do much with this editor, we need to install it. I will show how to install it on all three major platforms. After that, we will do a short introduction int it's use.

7.1.1 Installing on Windows

7.1.2 Installing on Mac

7.1.3 Installing on Ubuntu Linux

This is pretty simple.

7.2 Vim Configuration

Vim uses a control file to let the user customize how it works. This is an important aspect of a good programmer's editor. The name and location of the control file varies with the system.

7.2.1 Windows Configuration

CHAPTER

EIGHT

THE MAKE BUILT TOOL

CHAPTER

NINE

THE GNU COMPILER COLLECTION

The **Gnu_** project has developed a collection of incredible free software development and system management tools. If you do not know about Richard Stallman and his Free Software Foundation, you should look into it if you indent to develop software seriously. Although most of these tools are now associated with the **Linux_** operating system, Richard initially set out to produce a completely free version of the Unix operating system. Today, you can find these tools on just about any platform.

We will be using the *Gnu Compiler collection (GCC_)* to develop projects for this class. Specifically, we will use these major tools:

- **gcc** - the Gnu "C" compiler
- **g++** - The Gnu "C++" compiler
- **make** - the Gnu build tool"

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search