CMSC 15200
Introduction to Computer Science II

Department of Computer Science
University of Chicago

Summer 2007 Quarter

Dates: July 23 through August 24, 2007
Lectures: MWF 1:30-3:20 in Ryerson 251
Labs: W 4:00-6:00 in Maclab (A-level of Regenstein)
Website: http://www.classes.cs.uchicago.edu/current/15200-91/

Lecturer: Borja Sotomayor
E-mail: borja@cs.uchicago.edu
Office: Ryerson 257–C
Office hours: Open door policy (see page 7)

Lab Instructor: Same as Lecturer

TAs: None.

Contents of this Document

Course description ........................................ 2
Course goals ............................................ 2
Course organization .................................... 3
Course contents ........................................ 4
Books ..................................................... 6
Grading .................................................... 6
Policy on academic honesty ............................ 7
Asking questions ........................................ 7
How to hand in assignments ......................... 8
Practical information ................................. 9
Course description

This course provides an introduction to computer programming and to software development. The first portion of the course introduces students to computer programming, using the C/C++ language, and covers fundamental topics such as flow of control, function definition, data structures, and object-oriented design and programming. The second portion of the course provides a holistic view of software development and introduces students to tools, libraries, and additional languages that programmers should be proficient in to become effective software developers. Topics in this portion include using build tools, third-party libraries, scripting languages, and a brief introduction to other programming languages.

Prerequisites

Students who are majoring in Computer Science are required to take CMSC 15100 before this course. Nonetheless, the Summer version of CMSC 15200 is designed as a standalone introduction to computer programming, and can be taken by students from any background who need to develop programming skills for their work (projects, thesis, etc.). In this case, CMSC 15100 or some programming experience will be helpful but is not strictly required.

Course goals

This course has the following goals:

1. To introduce students to the fundamental concepts of the imperative and object–oriented paradigms of computer programming, including data structures and algorithms.

2. To teach students how to apply these fundamental concepts by writing computer programs using C/C++, a widely–used object–oriented language.

3. To make students appreciate the importance of practical knowledge in becoming a proficient programmer. Learning a programming language is similar, in some respects, to learning how to speak a new language. To become a fluent speaker, it is rarely enough to learn the syntax, grammar, and vocabulary of the language, even if we can memorize them letter by letter. It is generally necessary to converse and interact frequently with native speakers to pick up all the nuances of the language. Similarly, the best way to learn computer programmer is by writing programs.

4. To make students appreciate that computer programming generally involves much more than using a single programming language, and that programmers typically handle several programming languages, tools, and technologies to accomplish their job. In this line, the course also aims to teach students how to compare and contrast different languages and technologies, and choose the best one to solve a given problem.

5. To teach students how to learn programming. Once a student knows how to use a programming language, the learning curve for languages of the same style (e.g., imperative, object–oriented, etc.) tends to decrease. Students will be encouraged to not constrain themselves to
the language that will be used for most of the course (C/C++), and to not be intimidated by the prospect of learning new languages.

By the end of the course, students should be able to:

— Understand the fundamental concepts of imperative and object–oriented programming, and have a basic understanding of the theory of data structures and algorithms.
— Write, compile, and run C/C++ programs on a computer.
— Be familiar with programming tools and technologies.
— Learn new programming languages on their own.

## Course organization

The class meets three times a week for lectures, where fundamental concepts will be explained, and once a week in a computer lab where each student will be able to apply these concepts in front of a computer. However, since this course emphasizes the practical nature of computer programming, students will be required to work with computers more often than the weekly labs allow. All homeworks will involve writing programs on a computer (in the computer lab or at home) and the final ‘exam’ will be a mini–project that the student will be able to develop throughout the quarter. Students taking the course for a letter grade will also have to read about material not covered in class in order to complete their homework assignments. There will be no ‘paper–and–pencil’ work of any kind.

### Homework

There will be two weekly homework assignments: a short assignment handed out in class on Wednesday and due on Friday before 5pm, and a longer one handed out on Friday and due on Wednesday before 1:30pm (before Wednesday’s lecture starts). These assignments involve solving programming problems related to the topics covered in class so far.

### Labs

There will be five lab sessions to complement the class lectures. Attendance to the lab sessions is mandatory, and students will be expected to hand in the results of an exercise at the end of the lab. Conditional on sufficient enrollment, some of the labs will involve working in groups.

### Project

At the end of the quarter each student will submit a small individual programming project. This project must meet the following requirements:

- Written in C/C++ or Python. Other languages are subject to instructor approval.
- Uses at least one form of data representation (flat files, XML, or databases)
• [For a perfect grade] Uses a third–party library not covered in class.

• Is adequately documented with in–code comments and a brief user manual.

The instructor will propose several ideas for projects at the beginning of the course. By the middle of the third week, each student will meet individually with the instructor to explain (1) the project they will be doing, and (2) their plan for the project (what languages they will use, etc.). Students are welcome to propose their own ideas for projects (subject to instructor approval). In particular, students who are taking this course to develop programming skills for thesis work, research, etc. are specially encouraged to choose a project that directly relates to their work.

Course contents

The course will cover the following topics in computer programming:

The Basics. Introduction to programming with the C/C++ language [weeks #1 and #2]

Build Tools. Using make and Eclipse to build complex programs [week #2]

Data Structures and Algorithms. [weeks #2 and #3]

OO programming. Introduction to object–oriented concepts with C/C++ [week #3]

Python. Introduction to scripting languages with the Python programming language [week #4]

Data representation. Flat text files, XML, and SQL databases [weeks #4 and #5]

Other programming languages. A brief overview of other programming languages not covered in this course [week #5]

The week–by–week schedule is:

Week 1

— Introduction to computer programming

— Variables, data types, expressions, assignment

— Operators, basic I/O, invoking functions, writing functions

— Branching structures, looping structures

— Pointers, arrays, memory allocation

— Lab #1: Using gcc. Simple programs.
Week II
— Complex data types
— File I/O
— Separate compilation
— Introduction to data structures
— Build tools: make and Eclipse
— Reading: Data structures
— Lab #2: Debugging with Eclipse.

Week III
— Introduction to object-oriented programming
— Classes, constructors, encapsulation, inheritance, polymorphism
— Introduction to algorithms
— Third-party libraries
— Reading: Sorting and searching algorithms
— Lab #3: Installing and using a third-party library

Week IV
— Introduction to scripting languages
— The Python programming language
— Data representation: Flat files and XML
— Lab #4: Python

Week V
— Introduction to databases
— The SQLite database engine
— Python and SQLite with pysqlite
— Other programming languages: Java
— Lab #5: SQL
Books

The text for this course is Absolute C++, 3rd edition, by Walter Savitch, published by Addison-Wesley. The book will be available for purchase from the Seminary Co-op Bookstore (5757 South University Avenue).

Grading

Grading will be based on homework (40%), labs (20%), and the final project (40%).

Types of grades

Students may take this course for a “quality” grade (a letter grade), a pass/fail grade, or as an auditor. Students must declare, on the first day of class, what type of grade they are pursuing.

Letter grades. Students pursuing a letter grade can switch to taking a pass/fail grade right up until the end of the last lecture, but not after they have handed in their final project.

Pass/fail. Students taking the course for a pass/fail grade are not required to do the extra readings required for some of the homeworks and cannot switch to a quality grade.

Withdrawal. Students can opt to take a W grade in the course right up until the end of the last lecture, but not after they have handed in their final project.

Incomplete. Students who do not intend to complete the course during the Summer Quarter, but wish to continue working towards meeting the requirements of the course, must speak with the instructor to discuss requesting an I (Incomplete) grade.

Auditing. Students are welcome to audit this course. A student who is auditing the course cannot switch to a letter grade or a pass/fail grade, and will receive an R (Registered) grade.

Note: Students taking this course to meet general education requirements must take the course for a letter grade.

Late assignments

Late submissions (homeworks or labs) will have a 50% point penalty if submitted no later than 24 hours after the deadline. After 24 hours, the assignment will not be graded. Nonetheless, you are allowed two 24-hour extensions, without any penalty, to be used at your discretion on any of the homeworks or labs. You must indicate in your submission that you are using an extensions, or the assignment will be graded as late.

For avoidance of doubt, handing in any work after the deadline automatically flags your entire submission as late. For example, you are not allowed to hand in a portion of your work before the deadline, and the remaining work after the deadline, to have only part of your submission penalized.

Four late submissions (not counting the extensions) will result in a failing grade for the course.
Policy on academic honesty

The University of Chicago has a formal policy on academic honesty which you are expected to adhere to:

http://www.uchicago.edu/docs/studentmanual/academic_honesty.shtml

In brief, academic dishonesty (handing in someone else’s work as your own, taking existing code and not citing its origin, etc.) will not be tolerated in this course. Depending on the severity of the offense, you risk getting a hefty point penalty or being dismissed altogether from the course.

Even so, collaboration between students is certainly allowed (and encouraged) as long as you don’t hand someone else’s work as your own. If you have discussed parts of an assignment, then make sure to say so.

As for consulting other sources, this can be a thorny issue, as programmers are always encouraged to not reinvent the wheel and reuse as much code as possible. However, in this course you are also required to show that you are capable of effectively coding a solution to a given programming problem. So, in no case will you be allowed to hand in an existing solution to a problem (e.g. taken from a website) even if you do cite its origin. In general, you are allowed to look at existing code in search of inspiration, as long as you cite the sources you consulted. For large programming problems (which will turn up later in the course), you will be allowed (and encouraged) to directly use code taken from other sources (web sites, programming books, etc.) as long as this code is used to solve a particular subproblem and not the entire problem.

If you have any questions regarding what would or would not be considered academic dishonesty in this course, please don’t hesitate to ask the instructor.

Asking questions

This course has an open door policy for asking questions. Instead of setting fixed office hours, you are welcome to consult with the instructor at any time. Nonetheless, you should try to give the instructor, whenever possible, some advance warning of your visit (by e-mail) to make sure that he will be in the office at that time.

The preferred form of support for this course is though the course mailing list, which can be used to ask questions and share useful information with your classmates. In fact, we encourage that all questions about homework assignments, labs, and programming in general be sent to the mailing list, and not directly to the instructor. The reason for this is that, this way, all your classmates will be able to benefit from the reply to your question. Furthermore, the instructor will usually be able to reply to your e-mail faster than it would take you to walk to his office and back. In some cases, some of your classmates might even pitch in to provide their insights into questions or issues discussed in the mailing list.

You can subscribe to the mailing list in the following web page:

http://mailman.cs.uchicago.edu/mailman/listinfo/cmsc15200
**How to hand in assignments**

For each assignment, you will be expected to hand in the following items:

**Grading sheet.** Each homework/lab handout includes a grading sheet where you will need to write down your name and student ID. The instructor will write down your grade in this sheet and return it to you. You can keep the rest of the lab handout.

**Source code and documentation.** All the code you’ve written, and any accompanying documentation (if required by the assignment). You will hand this in using the `hwsubmit` command (described below)

**Printout of source code** [Optional] If you provide a printout of your code and documentation, the instructor will return it with annotations and comments, along with the grading sheet. If you do not, you will only get back your grading sheet (which will make it difficult for you to see what you did right and what you did wrong)

**hwsubmit**

This command is available if you log into any of the Linux machines in the Maclab. Make sure all the files you want to hand in are inside a directory, and then run hwsubmit like this (where `dir_name` refers to the directory you want to submit):

```
hwsubmit cmsc15200 dir_name
```

For example, assuming that you are currently inside your home directory, and that you placed all the files for a particular assignment in directory `/home/myusername/hw01`, you would run hwsubmit like this:

```
hwsubmit cmsc15200 hw01
```

**Printing code**

An easy way to print code from a UNIX system is using the enscript command. This command will automatically format the code for you, and can handle most programming languages. For example, to print out C/C++ code, you could run enscript like this:

```
enscript hello.sh -P printer_name -Ecpp
```

Where `printer_name` is the printer you want to send the code to. See the `enscript` man page for more details on using this command, and for a list of languages that `enscript` can handle.

**Handing in printed matter**

You can hand in your grading sheet, documentation, and optional code printout in the instructor’s office.
Practical information

There are certain things you will need to do before you can start using the Maclab computers, work on homeworks and labs, etc.

Obtain a CS account

Lab sessions will take place in the Linux section of the Maclab. Before using those machines, you need to request a CS account. This account will allow you to access certain computing resources in the Department of Computer Science, most notably the Linux machines in the Maclab. You can claim your CS account here:

https://www.cs.uchicago.edu/info/services/account_request

Knowing your way around a UNIX system

Although you will have the option of doing your homeworks and labs on UNIX or Mac in the Maclab, all lab instructions will be given from a UNIX system. If you are completely new to UNIX, we encourage you to use the KDE desktop, which provides a graphical interface very similar to the ones found on Windows and Mac systems. However, at a certain point, you will need to perform certain actions from the UNIX command line interface (or “console”). The first lab in the course will provide a basic introduction to the UNIX console. However, if you want a more complete introduction, you can take a look at the following tutorials:

http://support.uchicago.edu/docs/misc/unix/general/feet.html
http://support.uchicago.edu/docs/misc/unix/tutorial/

Working from home

Although the Maclab provides an excellent work environment, with all the software you need to complete the lab exercises, you are certainly free to work from home. However, take into account that you will need to hand your homework in using the `hwsubmit` command described above. Since this command is only available in your CS account, you will need to log in remotely to a CS machine using SSH. Instructions on how to do this are available here:

http://www.cs.uchicago.edu/info/services/new_users_guide

Also, take into account that there are two ways of working from home:

1. If you have a UNIX system at home (such as a computer with GNU/Linux installed in it), you can do your homework assignment entirely in your machine, using whatever tools you prefer. Once your assignment is complete, you simply have to copy your files to your CS account home directory and submit them using hwsubmit. Just in case, you might want to make sure that your code works fine in your CS account, as it will be graded in one of the CS Linux machines.
2. Regardless of having a UNIX system or not at home, you can do all your work using your CS account. To do this, you will have to log into your account using SSH and then do the assignment using the tools and commands available in your CS account.