CSC-150 Computer Science I

The College of Idaho

Spring 2016

Computers are to computing as instruments are to music. Software is the score whose interpretation amplifies our reach and lifts our spirits. Leonardo da Vinci called music the shaping of the invisible, and his phrase is even more apt as a description of software.

Alan Kay (attributed)

Quick Reference

You will have frequent homework, submitted sometimes online and sometimes on paper. See below for more details. You are free to work together on problems, but all work you submit must be your own. We also have frequent quizzes and in-class activities that cannot be made up. It is important to come to class every day.

GRADING

Tier	Weight	Date
Attendance & participation	0.10	daily
Workshops	0.10	continual
Quizzes	0.15	1–2/week
Homework	0.25	2/week
Midterm	0.15	March 15
Final	0.25	May 17

Instructor: Dr. Dave Rosoff
Assistant: Sam Chandler

Boone Hall 102C

MW 1:00–2:30; T 9:10–10:10, or by appointment

drosoff@collegeofidaho.edu

`@daverosoff`

`https://cofi.instructure.com/`

Preface: Learning outcomes

This course is designed to provide certain experiences, called "learning outcomes", to students who successfully complete it. These outcomes are enumerated in the margin.[1] I explicitly include these outcomes in the syllabus so that it is clear why I have chosen the various course components (each of which is described below.) Each learning outcome is addressed by one or more components of the course: homework, quizzes, in-class workshops and reflection writing assignments, and exams. See the *Grading* section below for more information.

[1] LEARNING OUTCOMES:
1. Recognize and describe fundamental ideas from course content described below.
2. Illustrate these ideas with examples and translate them into everyday terminology.
3. Demonstrate the use of course ideas on specific computing problems.
4. Improve their programming skills.
5. Effectively discuss and solve computing problems in group settings.
6. Apply a range of algorithmic techniques appropriately.
7. Use valid C++ syntax to aid in debugging code.
8. Plan, organize, and combine algorithms to solve new problems, as appropriate.

Introduction

Welcome to CSC-150, Computer Science I. I am pleased to be teaching the course a third time. Like many of you, I have enjoyed using computers for all of my life (that I can remember, anyway), and especially enjoy using them to *solve problems*. Solving problems of one kind or another is what computer science is all about, even more than mathematics (a closely allied field).

Many people refer to the first computer science course as "Programming", or "Intro programming", or even more metonymically by the name of the language used (in our case, C++). This is unfortunate, because the principles we will develop are not so limited.[2] They are fundamental to the whole discipline of computer science. You may be surprised to learn that this discipline is not limited to the writing of computer programs. Programming is essential to computer science fundamentals, but there is more, much more, to the world of computer science than writing code. I say all this by way of encouraging you to remember that while most of what we'll do in class could be called "programming", that is a rather narrow view of the whole endeavor.

[2] Computer languages aren't so different as you might think, even though there are hundreds of them in use. There's even a technical sense in which they are all equivalent—that is, all equally powerful. There's no "best language" out there to learn, or which you should be learning. Rather, if you have a strong command of the fundamentals of computer science, you can learn any language and use it appropriately.

THE PROBLEM-SOLVING AND ALGORITHMIC TECHNIQUES you will learn in this course are general, even universal: all other programming languages of interest have most of the same structures[3] that you will use in C++, and many problems in computing and elsewhere are amenable to the same methods of reductive solution.[4] If you do not go on to a career in computer science or even take a second course, I hope very much that you will still be able to make use of these ideas, techniques, and skills elsewhere in your life.

[3] A profusion of structures does not add to the computational power of a language. Instead it adds to the expressive power, helping programmers to mold their code more exactly to their thinking. Availability of additional programming structures also can help to produce more elegant, simple, or convenient programs. (Usually, those three qualities go together.)

[4] Breaking a problem into smaller pieces to be solved individually.

Programming is also fun. It is fun because it is creative. You are creating something out of nothing. You are making order, structure, where before there was none. Some people find a joy in it similar to the joy of music or poetry. It's easy to go overboard with comparisons like that, so I will leave them alone and add that the structures designed by programmers are often of significant benefit to themselves and to other people. This is the nature of abstraction. If you can solve a problem in a general way, you only have to solve it once.

You will get to *play* when you work on homework for this class, in a way similar to how art or creative writing students are playing when they are involved in the creation of their media. Many, many people over the seven decades encompassing the history of modern computer programming[5] have found themselves unexpectedly staying up all night, coding—just because they were having fun doing it.

[5] Computer programming properly begins in at least the early 19th century, when Charles Babbage's Analytical Engine was programmed with punched cards. Such cards had been in use to program looms to weave different patterns for thirty years at that point. It is thought by some that musical automata developed by the medieval Muslim polymath al-Jazarī were also programmable: the movement of small pegs would change the programmed movements of the robot musicians. Al-Jazarī lived and worked in the 13th century, in what is now Turkey and Kurdistan.

The fun and excitement of programming can bring together the most different kinds of people. Anyone can do it and change their lives and minds for the better. The ideas in computer science are very powerful. They can rewire your mind from the inside out.

Catalog description

"An introduction to fundamental principles of computer science. A brief introduction to computers, including data representation and storage and digital computation. Program design and implementation skills are developed using a high-level language. Topics may include fundamental programming constructs (e.g., functions, branching, looping), algorithm design, data abstraction, recursion, and object-oriented programming."

Text

Our text is C++ *for Everyone* by Cay S. Horstmann, second edition. If you do not already have it, please order it immediately. You will not be able to complete the reading assignments without it.[6]

[6] You need the second edition.

Grading

Scores are computed as a weighted average, with the following weights: participation and attendance 0.10, in-class labs/workshops 0.10, homework 0.25, quizzes 0.15, midterm exam 0.15, and final exam 0.25. Observe that the weights sum to 1 = 100%.[7] The exact determination of letter grades from these scores depends on the final distribution of scores in the class, but you can expect a C for earning 75% of the points, a C+ for 78%, a B– for 80%, and so on.

[7] The exams are relatively lightly weighted, but homework, class participation, and in-class workshops each weigh in at 15 %–20 %. It is not possible to earn a B or better in this class unless you are steadily and persistently engaged, including attending class and actively participating! It is my hope that this reduces the incentives and payoffs to cram and realigns them in the appropriate direction, supporting steady incremental effort.

Workshops & Labs

This course functions better as an interactive course than as a pure lecture. It is easier to stay engaged. We use in-class activities (sometimes full-period labs) to explore ideas from the text, go into greater depth, and guide you through examples.[8] Instead of doing these at home, we will use class time to work on them, so that you can get help when you get stuck. Occasionally you may need to finish them outside of class, if we don't have enough time to finish. They have proved to be very valuable in helping students get main ideas before tackling bigger programming projects. The workshops and the homework are the heart of the course.[9]

[8] In-class activities address learning outcomes 1, 2, 3, and 5, and sometimes others.

[9] Workshops are often submitted on paper, the same day as they are completed. Labs are due in Canvas or as otherwise specified. You should have plenty of time in class to make sure you understand properly, but if you need more time, you are welcome to come to office hours to talk about them.

Homework

Homework in this class consists mostly of programming assignments (actual code, submitted online). We may occasionally have problem sets (batches of short questions you'll answer on paper).

1. Programming assignments[10] assigned approximately weekly. You will have several days to work on these. In a programming assignment, you will apply ideas and techniques from your reading, homework, and workshops to a specific and more involved problem. Solutions to programming assignments must be working C++ source code, submitted via Canvas. I strongly encourage you to do your programming in Sublime Text 3. Full-credit solutions match example runs exactly, function as specified, are properly laid out and indented, and are free of typographical errors and appropriately commented.

[10] Programming assignments address learning outcomes 3, 4, 6, 7, and 8.

2. Problem sets[11] assigned occasionally. You will have a day or two to work on these. Problem sets are to be completed on paper, probably with a pencil or pen, although you may type your solutions if the results are *professional*. Your write-ups should be clear, complete, and free of typographical errors. I am happy to talk about these problems in office hours with you.

[11] Problem sets address learning outcomes 1, 2, 3, and 7.

Exams

Two exams are given: one in-class midterm, and one final exam.[12] *I will consider make-ups only with compelling, documented reasons.* The final exam takes place at the indicated date and time. It cannot be rescheduled *for any reason*. Make your travel plans accordingly. If you miss an exam, the other exam will constitute the whole of your exam grade, but it is not possible to earn more than a C without taking the final exam regardless of what else happens.

[12] The date of the midterm is tentative.
Midterm exam: Tuesday, March 15
Final Exam: May 17, 8:30 am–11:30 am
Exams address learning outcomes 1, 2, 6, 7, and 8.

Software

Like most specialized tasks, computer programming requires specialized software. If you wish to use your own computer, you will probably need to install some things. At a minimum, you need a *text editor* (Sublime Text 3) and a C++ *compiler* (the MinGW package, on Windows).[13] If you use Windows, you probably need both; if you use a Mac or any Linux operating system, you just need to install Sublime Text 3. Macs and Linux both ship with the standard GNU g++ compiler. Windows users should install MinGW (follow all directions

[13] For text editors, there are many choices. Choices with a † are free.
All platforms: Sublime Text 3. This is the editor I would prefer you to use. It is very powerful and extensible.
Windows: †Notepad++
Mac: †TextWrangler, BBEdit
Linux: †gedit, †Emacs, †vim
It is extremely wrong and bad to use word processors (like Word) to write code. Don't do this.

carefully).

The computers we have for use in class have Sublime Text 3 installed. I use Sublime Text for many hours each week and have grown to love it, but it is not free (trial period, $70 for license). Editor choice is a very personal matter[14] and you may want to experiment with different editors on your own.

[14] See the Wikipedia entry for *editor war*.

The use of an IDE (integrated development environment) is not recommended for this class. If you choose to use one and have problems with it, I may not be able to assist you. I recommend that you stick to writing your code in Sublime Text.

Academic integrity

I encourage all students to form study groups and collaborate on homework; each student is of course individually responsible for their own work. Collaborators must be acknowledged.

Students are expected to complete all graded work in accordance with the College Honor Code. Plagiarism, cheating, or borrowing without proper credit will not be tolerated. Violations of academic honesty can result in loss of credit on an assignment, failure on an exam, or failure in the course. Referrals will be made to the Vice President for Academic Affairs for any party involved in academic dishonesty.

Special accommodations

Students who have documented disabilities as addressed by the Americans With Disabilities Act and who need any test or course materials to be furnished in an alternative format should notify me immediately (during the first two weeks of class). Reasonable efforts will be made to accommodate your needs.

GOOD LUCK THIS SEMESTER!