Objectives and Outcomes

This course seeks to advance students’ programming knowledge and experience in multiple directions: a new language and paradigm; increased programming problem size and complexity; and more rigorous testing and validation. This focus on programming advancement intrinsically involves a laboratory environment, apprenticeship-style training, and increased adherence to process and form. Long after the course concludes, my hope is that you will be able to:

1. Design, compile, and test programs within a command-line environment.
2. Correctly solve medium-sized programming problems.
3. Follow academic and technical best practices throughout the course.

Prerequisites/Prior Background

The prerequisite course is CMSI 185 or its equivalent. Students who have not taken (specifically) CMSI 185 require prior approval of the instructor.

Materials and Texts

There are no required texts; however, the following are recommended (latest editions are not necessary; sufficiently recent editions will work):

- Assorted handouts, articles, and sample code to be distributed throughout the semester

In addition, do not hesitate to look for further information regarding the concepts, techniques, tools, and paradigms that we will discuss.

Course Work and Grading

This course uses standards-based grading: your proficiency in each course objective is directly evaluated according to the outcomes shown on page 4 of this syllabus. Proficiency is measured according to the following key:

<table>
<thead>
<tr>
<th>+</th>
<th>Advanced proficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Appropriate proficiency</td>
</tr>
<tr>
<td>/</td>
<td>Approaching appropriate proficiency</td>
</tr>
<tr>
<td>-</td>
<td>Needs practice and support</td>
</tr>
<tr>
<td>O</td>
<td>No basis for evaluation</td>
</tr>
</tbody>
</table>

Your submitted work is used to evaluate these outcomes (see below). Letter grades are then assigned as follows:

<table>
<thead>
<tr>
<th>+</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>many</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>B</td>
<td>many</td>
<td>few</td>
<td>none</td>
</tr>
<tr>
<td>C</td>
<td>some</td>
<td>few</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>some</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>many</td>
<td></td>
</tr>
</tbody>
</table>

A–, B+, B–, C+, and C– grades are assigned for outlier combinations between the above thresholds. Qualitative considerations (e.g., degree of difficulty, effort, class participation, time constraints, overall attitude) may improve proficiency measures. To resolve close calls, a quantitative calculation with 4, 3, 2, and 1 standing in for +, |, /, and – respectively will be used. You will receive feedback and proficiency updates after every assignment.

Cumulative proficiencies for individual outcomes are based on their statistical mean, using the numeric mapping given in the previous paragraph.
Programming Assignments
A new programming problem will be assigned approximately every few weeks. For each one, we will discuss several possible algorithms, finally singling out one of them for implementation. Sketches of key program structures will then be provided, and your mission will be to complete the program.
You will do a lot of programming during the workshop itself; the remainder is to be done as homework. Because the emphasis of this course is the development of individual programming skill, collaboration with other classmates should be kept to a minimum.

- You must be in front of a workstation during class, whether your own or the Keck Lab’s.
- You may ask a classmate for help in diagnosing problems or discuss the assignment in general terms, but unless the assignment explicitly calls for pair programming, you may not share any code.

As programs increase in difficulty, their proficiencies will weigh more heavily. For simplicity, this weighting is reflected by counting that proficiency twice or more on the record.

Responding to Feedback in Subsequent Work
The size of the programs in this course does not lend itself well to resubmission and regrading. Instead, make sure that feedback given in earlier assignments factors into your work in later ones. The uniform structure of the course and its grading system provide ample opportunity to make up for past pitfalls. Problems only arise when you make a habit of not proactively improving upon what has been called out before.

Version Control
Version control is an indispensable part of today’s computer science landscape in industry, the academy, and the open source community. We introduce a form of version control to you in this course: make sure that you get the hang of it.
You will be asked to commit your work at the beginning of every class, then commit it again at the end. When doing homework, make sure to commit your code as you finish up units of work (e.g., methods, classes, tests, etc.).

Workload Expectations
In line with LMU’s Credit Hour Policy, the workload expectation for this course is that for every one (1) hour of classroom instruction (50 scheduled minutes), you will complete at least two (2) hours of out-of-class work each week. This is a 3-unit course with 3 hours of instruction per week, so you are expected to complete $3 \times 2 = 6$ hours of weekly work outside of class.

Attendance
Attendance at all sessions is expected, but not absolutely required. If you must miss class, it is your responsibility to keep up with the course. The last day to add or drop a class without a grade of W is January 16. The withdrawal or credit/no-credit deadline is March 20.

Academic Honesty
Academic dishonesty will be treated as an extremely serious matter, with serious consequences that can range from receiving no credit to expulsion. It is never permissible to turn in work that has been copied from another student or copied from a source (including the Internet) without properly acknowledging the source. It is your responsibility to make sure that your work meets the standard of academic honesty set forth in the LMU Honor Code and Process.

Special Accommodations
Students with special needs who require reasonable modifications or special assistance in this course should promptly direct their request to the Disability Support Services (DSS) Office. Any student who currently has a documented disability (ADHD, autism spectrum, learning, physical, or psychiatric) needing academic accommodations should contact DSS (Daum 224, x84216) as early in the semester as possible. All requests and discussions will remain confidential. Please visit http://www.lmu.edu/dss for additional information.
**Topics and Important Dates**

Correlated outcomes are shown for each topic. Specifics may change as the course progresses. University dates (italicized) are less likely to change.

- **January**
  - Introduction to the Java programming language (*1a–1c, 3a–3f*)
  - **January 16**
    - Last day to add or drop a class without a grade of *W*

- **February**
  - Java transition exercises (*1a–1c, 3a–3f*); board/parlor game logic programs (*all outcomes, 1a–3f, from this point forward*)
  - **February 18–20**
    - Spring break; no class

- **March**
  - Discrete event simulation programs; implementation of arithmetic and geometry routines
  - **March 20**
    - Withdraw/credit/no-credit deadline
  - **March 30–April 3**
    - Easter break; no class

- **April–May**
  - Dynamic programming; backtracking search

You can view my class calendar and office hour schedule in any iCalendar-savvy client. Its subscription link can be found on the course web site (it's too long to provide in writing).

If necessary, this syllabus and its contents are subject to revision. Students are responsible for any changes or modifications announced in class.

**Tentative Nature of the Syllabus**

If necessary, this syllabus and its contents are subject to revision; students are responsible for any changes or modifications distributed in class or posted to the course web site.
## Course Outcomes

1. **Design, compile, and test programs within a command-line environment.**

   - **1a** Set up and maintain an operational, productive programming environment.
     - These outcomes represent first-principle programming fundamentals. Developer tools abound for improving the efficiency and productivity of experiences programmers…emphasis on experienced. Such tools presume pre-existing knowledge and understanding—these very outcomes.

   - **1b** Edit and manage source code effectively.

   - **1c** Issue correct commands for common programming tasks.

2. **Correctly solve medium-sized programming problems.**

   - **2a** Adhere to problem specifications.
     - As the types of programs that you write increase in complexity and size, discipline and good habits factor more greatly into your success. These outcomes seek to reflect this expansion.

   - **2b** Gracefully handle errors and edge cases.

   - **2c** Pass automated unit and functional tests.

3. **Follow academic and technical best practices throughout the course.**

   - **3a** Write syntactically correct, functional code.
     - Code has to compile. Code has to work. No errors, no bugs. Use unit tests as much as possible.

   - **3b** Use coding best practices, demonstrating principles such as DRY, proper separation of concerns, correct scoping of variables and functions, etc.
     - This is the basis of good software design. It makes software easier to maintain, improve, and extend.
     - In this course, these practices will be introduced as the class progresses. Heed feedback well.

   - **3c** Write code that is easily understood by programmers other than yourself.
     - This outcome involves all aspects of code readability and clarity for human beings, including but not limited to spacing & indentation, proper naming, presenting code in a manner that is consistent with its structure, documentation & comments when appropriate, and adherence to conventions or standards.

   - **3d** Use available resources and documentation to find required information.
     - The need to look things up never goes away. Remember also that the course instructor counts as an “available resource,” so this outcome includes asking questions and using office hours.

   - **3e** Show steady progress by using version control as described by the protocol given in the course.
     - This course introduces version control in a specific way, to acquaint you with its principles and uses without information overload. Industry-standard use comes in later courses.

   - **3f** Meet all designated deadlines.
Sample Standards Achievement Report
Based on these proficiencies, the student is a qualitative call between a B– and a C+.

1 Design, compile, and test programs within a command-line environment.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Set up and maintain an operational, productive programming environment.</td>
</tr>
<tr>
<td>1b</td>
<td>Edit and manage source code effectively.</td>
</tr>
<tr>
<td>1c</td>
<td>Issue correct commands for common programming tasks.</td>
</tr>
</tbody>
</table>

2 Correctly solve medium-sized programming problems.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2a</td>
<td>Adhere to problem specifications.</td>
</tr>
<tr>
<td>2b</td>
<td>Gracefully handle errors and edge cases.</td>
</tr>
<tr>
<td>2c</td>
<td>Pass automated unit and functional tests.</td>
</tr>
</tbody>
</table>

3 Follow academic and technical best practices throughout the course.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3a</td>
<td>Write syntactically correct, functional code.</td>
</tr>
<tr>
<td>3b</td>
<td>Use coding best practices, demonstrating principles such as DRY, proper separation of concerns, correct scoping of variables and functions, etc.</td>
</tr>
<tr>
<td>3c</td>
<td>Write code that is easily understood by programmers other than yourself.</td>
</tr>
<tr>
<td>3d</td>
<td>Use available resources and documentation to find required information.</td>
</tr>
<tr>
<td>3e</td>
<td>Show steady progress by using version control as described by the protocol given in the course.</td>
</tr>
<tr>
<td>3f</td>
<td>Meet all designated deadlines.</td>
</tr>
</tbody>
</table>

This student excelled in technology and tool use, reaching advanced proficiency in the outcomes that exemplify that area (1a–1c, 3d–3f). However, he or she was a “my way or the highway” type, consistently imposing their own assumptions on a particular programming problem and not paying adequate attention to problem instructions and details. As a result, expected functionalities and use cases were missed, thus failing to reach proficiency in outcomes 2a, 2c, 3a, and 3b. With 12 total outcomes, getting 4 /s borders on “some” and not “few.”

The student had sufficient technical skill to address issues when they set their minds to it (2b) and knew their way around an editor enough to present their code decently (3c). But again, insistence on preconceived notions and, importantly, a habit of disregarding feedback held those two outcomes back.

The student should have avoided the /s in order to guarantee a B-level grade or higher—more 1s tend toward a B while more +s tend toward an A. If the student had not been very proficient at technology use and gotten lower proficiencies at those 6 outcomes, the grade would have been at a C-level or lower for sure.