

Final Project Guidelines

General Information

Below are a few suggestions for final projects. You can choose one of these projects (or some variation), or you can propose your 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.

The project must meet the following requirements:

- Written in C/C++ or Python.
- Uses data structures (can be something we talked about in class, or something from the STL).
- Uses at least one form of data representation (flat files, XML, or databases).
- Is adequately documented with in-code comments and a brief user manual.

Meeting with the Instructor 3rd week

By 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.). Please email the instructor to set up a meeting time. The earlier in the week the better, so you can get started on your project.

Final Presentation

On the last day of class, Friday August 28th, each student will give a short presentation (10 min) where they show the class what they did for their project. You are allowed to use a projector, just the blackboard, or any other prop you would like.

Due Date

The final project is due in class on Friday August 28th. You should turn in your code via email, together with the user manual, and makefiles if necessary. You should also turn in a paper copy of the user manual in class on Friday August 28th.

Grading

Grades will be assigned as follows:

- 15% — Coding style:
 - Using multiple files.
 - Documentation: comments, and a brief user manual.
- 15% — Final Presentation.
- 30% — Planning:
 - Use of data structures.
 - Use of user defined classes and/or structs.
 - What functionality your project has.
- 40% — Correctness:
 - Does your code work?

Project Ideas:

Here are a few ideas. You don't have to stick to them if you have other ideas, and even if you do choose them, you can take them in another direction and do something somewhat different. For the first idea, I wrote down what I expect from the project, and the corresponding grade. Corresponding degrees of difficulty should correspond to the same grade for different projects.

1) Connect4 or other games

You will write a connect 4 program. Here is what I would expect and the corresponding grade:

- C:
 - A simple text based interface
 - Can play one player against another
 - Stores the score in a file, and is able to retrieve it next time the same two players play (e.g. by using a code, or the players' names)
- B:
 - All the functionality above, and
 1. Implements a simple AI (simply looks at the board and decides what to do based on that)
 2. or implements a GUI interface for the game so it looks more like a real game (not just text)
- A:
 3. Either do both B1 and B2 or
 4. B.1 and the AI is more complicated: it looks several moves ahead, and then does a static evaluation (i.e. same as B.1).
- A+
 - A.3 and A.4 or
 - A.4 but the AI is even more advanced: it keeps track of games it has played and decides what moves to make based on that.

You can do the same project for any other game you would like, for example, chess, go, so on... If you choose a game like chess, where even implementing basic moves will be more complicated, you don't have to add quite as much functionality to get the same grade (about a ½ grade more for the same functionality).

2) Implement a Data Structure

Implement a library for a (sufficiently complicated) data structure. You will have to define a class (a template class if you want) for your data structure. What the operations that your data structure will be able to perform are will depend on the specific data structure you choose, and will be approved by the instructor. Here are two examples, but other data structures are allowed:

- Red Black trees
 - This is a specific implementation of balanced binary search trees
- Union Find
 - This is a data structure that is used in many applications, including algorithms to find connected components, or minimum spanning trees in graphs.
 - The union-find data structure keeps track of sets of objects, and what set each object belongs to

3) Graph Algorithms Package

Implement a library that performs various graph algorithms. You will have to decide how to store the graphs (you might want to use different representations depending on whether your graph is sparse or dense), and whether you want to allow for directed graphs as well as undirected graphs. Algorithms that your package could implement include (but are not limited to):

- Finding the connected components (strongly connected and weakly connected if the graph is directed)
- Breadth First Search
- Depth First Search
- Dijkstra's shortest path algorithm
- Single source shortest path
- Recognize if your graph is a tree
- Find all the cycles in the graph
- ...

4) WordSquish

This is the final project from last year's class. Here is a description from last year's class' website:

<http://people.cs.uchicago.edu/~wax/courses/cm15200/project.html>

It is essentially an implementation of the online game wordsquish, which can be found here:

<http://www.WordSandwich.com>