A program design document connects the program requirements to design choices and functional characteristics of a program. Every professional that builds any non-trivial artifact, whether physical or digital, starts by thinking carefully about the task at hand and planning the solution. Engineers, architects, programmers, teachers, and carpenters create and use design documents. In fact, carpenters coined a phrase to emphasize the necessity of planning before building, "measure twice, cut once".

In the software industry, design documents are almost always peer reviewed, before any coding starts, for every design decision and significant feature. Learning to write design documents now will make the design and review processes easier in the future. The programs we'll write in this course are assigned to reinforce the programming concept we are studying and won't be all that complicated (compared to "real" software). Therefore this is the perfect time to begin creating design documents and honing your ability to communicate a design.

Your design document must include the sections shown as boldface in the template description on the next page. Each section heading has a description to guide you in understanding the type of content for each portion of the document. There is no minimum or maximum length for each section and some design documents will be longer than others. The design document should completely and unambiguously describe all of the program requirements, inputs, outputs, the solution steps in the form of a step-by-step narrative, a flowchart of your algorithm, as well as a test plan. Note that in order to claim your solution satisfies all the program requirements you must have at least one test case for each requirement. Overall, your design document is finished if someone who is not familiar with the problem or your solution can read the document and carry out your solution without help.

Some students falsely believe that the design document should be created after they've finished the program (obviously the design is easier to create after the program, but without a design the program was harder to create and may not be correct). Creating the design document first is critical to building high quality software and a design will be required in industry before any code is written. The design document is a plan or blueprint of what we want to build and how we plan to build it. Plans often change during construction and that is OK. The design document's real value is that it is a formal process for thinking carefully about the problem and the solution in sufficient detail to implement it. Developing the design document is about understanding the problem completely and carefully reasoning through the solution. This process occurs before any coding takes place.

Put your name, course number and section, the program assignment number, and due date on the top of each page of your design document. Number the pages on the bottom starting at one.

Program Requirements:
This section describes what the program must do to be considered a success. Usually in CS 155, we can take this directly from the assignment. Later, requirements will come from the customer.

Program Inputs:
This is the data that will be input into the program. Be specific about the kind of data and acceptable values (type and range).

Program Outputs:
This is the information produced by the program. Again, be specific about the type of information and how it will be presented. The meaning (interpretation) and presentation of all outputs should be specified.

Test Plan:
This section describes how you will test your solution to determine if it is correct and satisfies the requirments. What test cases will you use, is data validation necessary, etc. For each test case, show the expected output for the given input. For most CS 155 projects, a test case will be sample input with the corresponding output.

Algorithm:
Before writing your program in C++, which requires strict adherence to C++ syntax and grammar rules, it is helpful to write your solution in human language. This representation of your solution should read like an algorithm, a step-by-step set of instructions to solve a specific task. Each instruction will be written in English. Each step should be detailed, concise, and unambiguous. This section should be detailed enough to be translated to code, although it should not be actual code. You are describing steps to be carried out in any formal language, not just C++. That means, if three different humans followed your instructions, given the same input, they would all produce the same result.

Flowchart:
This section is a flowchart depicting the design you've chosen for your solution. It will necessarily have some overlap with the previous sections. The big difference is that the algorithm narrative connects your design choices to the requirements. The flowchart is a high level, graphical representation of the solution.