**ENGR 102**

**Lab # 8A – to be done as a team [100 points]**

**Activity #1: An Engineering Design Problem [30 points]**

This activity is meant to help you practice top-down design, outside the context of programming. We’ll look at an engineering challenge, and your team is to use a top-down design approach to describe how you would go about approaching the challenge. This is meant to be very broad and open-ended, and you are not expected to know the details of how individual pieces of this might work. The idea here is to get you to think of the **problems** that would need to be solved in order to meet this larger challenge. The goal is for you to be able to take a large, broad problem, and decompose it into smaller problems, repeatedly, until you have reached problems that seem small enough that they have a more straightforward solution.

Your team should pick **one** of these 3 options (each of these exists but the assumption is that you are not familiar with them already):

* An autonomous personal movement device – essentially an autonomous wheelchair. The device should operate with the same capabilities as a standard motorized wheelchair, but be able to operate autonomously, rather than being controlled directly by the person in the chair.
* A system for automatically inspecting rigid parts coming off of an assembly line, to examine whether they match some shape design. The process should not require human intervention, and you can assume pieces are coming off on a conveyor belt or some similar device.
* A system for retrieving devices that are at the bottom of the ocean. You can assume that the devices are emitting some sort of signal of your choice.

As a team, come up with a hierarchy stating the problems to be solved in creating such a system. Your base in this case should be something like “Autonomous Wheelchair”, with the next level of the hierarchy being the major problems that you would need to address in creating the device. Each of those problems should be further decomposed into smaller problems and so forth.

Your hierarchy should have a minimum of 5 levels, and at most 7 levels (not every branch of the hierarchy needs to go to the maximum depth, but many should). Because this is a very broad problem, in theory you could go into far more detail; that is not needed here. However, the “leaves” of your tree should be reasonably straightforward and less open-ended problems to be solved (e.g. things that have been done before, might have an off-the-shelf solution, or might still be new things, but at a level that it’s reasonable to expect a small team of people to be able to address them).

This should be done as a team, and the hierarchy should be written in a document that you submit in PDF form. You may use any office software (e.g. Word, Powerpoint) that you find most useful in creating your hierarchy.

*Activity 1 Rubric : [30 points]*

[8 points] – Hierarchical description of at least 5 levels, and at most 7 levels.

* Subtract points if the levels are very incomplete; there should be at least 30 “leaf” nodes in a reasonable breakdown (and there could be many more).

[10 points] – All leaf nodes of hierarchy are straightforward problems that could be reasonably solved.

* Subtract 2 per node if the node topic is far from a straightforward statement (something that has been previously solved, or that it’s reasonable that a small team of people could solve).

[12 points] – Hierarchical descriptions breakdowns from level to level are reasonable (are clear decomposition of earlier level, and address all the main challenges from the previous level).

* Judgment call: Subtract 1-3 points for each node that is not broken down in a reasonable way into subnodes.

**Activity #2: Top-Down Design of a Program [70 points]**

Following the process described in the lecture, you should, as a team, perform a top-down design for a program. Then, construct code for the program.

You want to create a program that first lets a user enter arbitrary data points (as x, y values), and then will provide a linear interpolation/extrapolation from those points to determine a value at any point a user gives. That is, the user should be able to enter **any** x value and get a y value back, such that the y value is the best estimate (using a linear estimation) from the nearest x value(s).

You are to create a document that includes parts a-c and g.

1. First, as a team, develop a top-down design for the program. Develop a hierarchy for the individual pieces, breaking each one down into as small of a piece so that the code for that portion of the program is “obvious”. You should put this hierarchy into a document (that you will submit as a PDF file later).
2. Next, as a team, determine what variables you will use for the main sections of your code. You should decide on the main data that you will need to keep that will be used in more than one “node” of the design. Write a list of these, along with a very brief (one sentence or less) description of what that variable will store. Note that you do not need to decide on variables that will be used only within one “node” of a program (e.g. you don’t need to describe a loop iterator if it is not going to be used outside the loop). Again, this should go into your initial document.
3. As a team, discuss briefly the test cases you believe you will need to allow for. Write down a list of exceptional/edge cases that you need to be able to handle.
4. As a team, create one file with the comments/outline of the code. Share this file among all four team members.
5. Next, divide the coding tasks among your team members, so that each person has a different section of the program.
   1. Divide the “leaf” nodes among the team members so that each person has approximately the same number of items to implement.
   2. If you have done a good job with the top-down design, and in specifying the variables that will carry over from one section to the next, then people should be able to write code for just their own section without seeing the other sections of the code!
6. Once each person has written their own separate code, bring the files together as a team, and have one person combine all the code into one program.
   1. You will likely need to thoroughly test and debug your code together as a team at this point.
7. As a team, write a short summary (a few sentences – about 1 paragraph for each)
   1. Describing the difficulty with which your team was able to combine the code at the end. Did this provide your team any insight into how the design itself might have been specified more clearly?
   2. Describing any benefits and drawbacks you saw into dividing the coding like this. Can you see reasons why this might be a good idea? Can you see reasons why this might be a bad idea?

Zip as Activity 2:

1. Your document for parts a, b, c, and g as a single PDF file,
2. The individual files you produced in part e, and
3. The single debugged file from part f.

*Activity 2: Rubric [70 points]*

[15 points] – Top-down hierarchy specified. Similar to Activity 1, it should clearly divide the program into parts, and the final leaves should be straightforward tasks to implement

[8 points] – Variables clearly specified in design document

[8 points] – Full set of test cases specified in design document

[10 points] – Program is broken into 4 different parts (or fewer if teams of less than 4), implemented by different people, reflecting the design (variable names, etc.).

[15 points] – Combined program is generated, and works on test cases

[7 points] – Combined program includes comments reflecting the hierarchical breakdown

[7 points] – Reflection questions answered in meaningful ways, demonstrating thought being put into reflection on process. Should be a paragraph on review of what was done, and another on benefits/drawbacks.

**Zip Activity 1 & Activity 2 and Submit**