# FINAL EXAM – SPRING 2020

## DUE DATE: Saturday May 16th$^{h}$ , before Midnight.

### DESCRIPTION:

Ok, this is the last one. Just like before, 4 problems, no EC, and all meant to be done by you and you alone. Same rules as the midterm too: Read, Ask if you need something to be clarified, you can ask for code checks, I can nudge in the right direction, and you have to submit on time.

Don't leave the final exam for the last 2 or 3 days, trust me, you are going to struggle if you do. **Read it, there are many instructions on what I am asking you to do.** If something is confusing and you need further explanation on the problem post your question in the discussion group or email me and I can share it with the class if needed.

**This is a Final, so all the work should be done by yourself, no tutors, no friends, no other classmates, this should be your work. If you use any of those resources, you are not fooling me, you are just fooling yourself. If you reallly want to be a programmer start by testing what you know and work on the topics that you need to get better at.**

Also Remember!

- Do not change the name of the functions
- Do not change the name of the parameters
- **The pts in the exam are to total 100pts which becomes 15 pts of your total grade, no EC pts this time.**
- At this point in the semester you should be using functions to comparmentalize your code. Long main functions that do everything will cause you to lose points.

### PROBLEM #1: VECTOR FULL OF LETTERS – 20 PTS

**Description:**

Lets start with some fundamentals with strings and vectors ( section 8.3 ) :D

For this program we are going to calculate the weight of a word. The way we are going to calculate this is by taking the ASCII value for each character that forms the word and add it up. Use two vectors, one for odd characters and one for even, save the characters into the vector they belong to.

At the end, print the word, one character with its ASCII value under neath, and then print the odd vector of characters and the total sum for that vector, and then the even vector, with the total sum for that vector.

Example of prorgram running:

```
  --> Enter word you want to find the value to: classroom
        c       l       a       s       s       r       o       o       m

        99      108     97      115     115     114     111     111     109

Total Weight of the word: 979
        c       a       s       s       o       o       m
Total odd value: 757
        l       r
Total even value: 222
```

Another example:

```
  --> Enter word you want to find the value to: multiplication
       m       u       l       t       i       p       l       i       c       a       t       i       o       n

       109     117     108     116     105     112     108     105     99      97      116     105     111     110

Total Weight of the word: 1518
       m       u       i       i       c       a       i       o
Total odd value: 848
       l       t       p       l       t       n
Total even value: 670
```

## REQUIREMENTS:

- You must use Vectors to store the characters for even/odd values.
- Use functions, when you calculate the total weight of a vector or when you print a vector, don't repeat the code, use functions.
- **Only half grade max will be given with implementations without vectors**

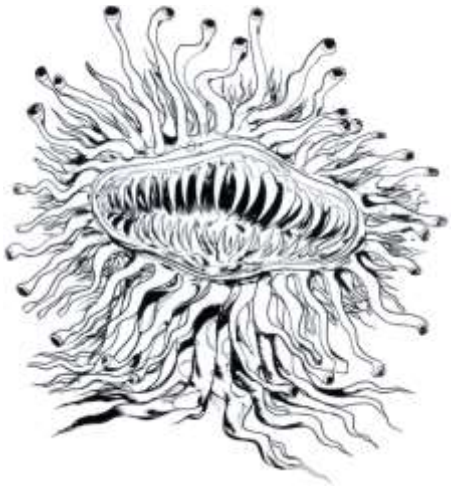
## WHAT DO YOU NEED TO SUBMIT?

- Create a file with your student id and name it **[your_student_id]_problem1.cpp**
- Your file should include:
  - A main( ) function solving this problem.
  - Any functions you might have written

## PROBLEM #2 : THE MARLBORO MONSTER! 40PTS

**Description:**

I am not talking about cigarretes, or some other type of plant. I am actually talking a very foul monster that if you have ever played any Final Fantasy game, you have encountered this creature for sure. This is a lovely picture for a reference:



Lets say that you want to grow this lovely plant monster in your backyard, the problem is the rate at which these creatures reproduce. These monster plants can be grown in single lines of pots. The problem, and I am taking artistic freedom here so don't sue me, is that this plants reproduce to adjacent pots in a day if they are available. If another monster plant has already duplicated to it, then the plant doesn't duplicate in that direction. Since we are using a row of pots to grow little baby monster plants, then one planted monster can duplicate to the left or to the right if there is space to do it.

In order to grow this animals in a safe way so that there are enough for all FF games, you need to make sure that you know how many days you need in order for your row of pots to be filled, and you also need to know the age of each plant.

**The idea is simple, we start with a row of empty pots ( n pots). The farmer can select between 1 pot, boring, to any large number, 32000 pots if you want to really stress test the system.**

Each pot contains the following information:

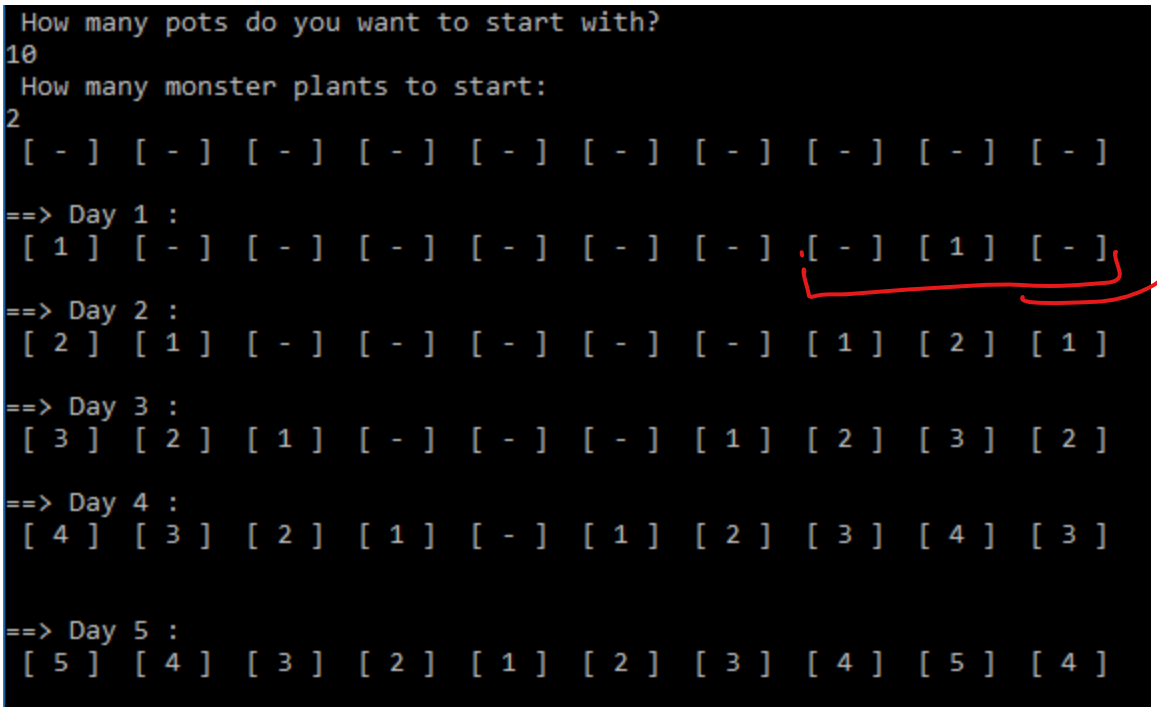
- **plant_days:** int value that tells you how many days since a monster plant moved in.
- **is_planted:** boolean value that tells you whether the pot has a monster plant or not.

At the beginning of the simulation we need to ask the user 2 pieces of information:

1) number of pots to use
2) how many monster plants to start the simulation with.

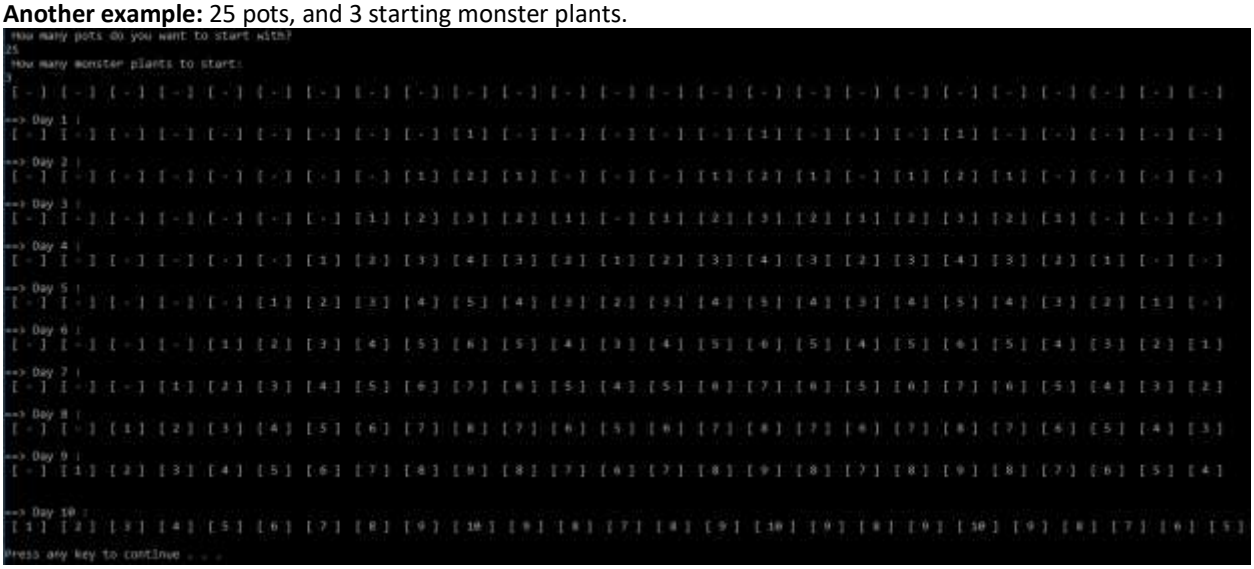
The program then runs until all the pots have a little tiny monster plant that one day will probably kill somebody looking for a crystal. **Your goal is to produce a model that shows how the monster plants are duplicating on the pots, and tells how many days each of the plants is.**

**For example:**




- The row of pots starts by showing all pots empty.
- **Day 1:** 2 monster plants get placed in random pots
- **Day 2:** The 2 monster plants have duplicated to where an open pot is, and the days since planted is show for all monster plants.
- **Day 3:** The 5 monster plants now again duplicate to where there is space. The days since planted are shown.
- **By Day 5: All pots are filled.**

**Another example:** 25 pots, and 3 starting monster plants.



## REQUIREMENTS

- Look beyond the weird concept and you see references to many things you need to use.
- **You need to use dynamic memory for the array of pots.** Failing to use dynamic memory will result in a max of half points for this problem.
- **Remember to free the memory when the program is done.**

## SUGGESTIONS

- Build your code little by little, start with simple questions of information from the user, creating a dynamic array, initializing it, printint it.
- Remember structures, you can use classes on this one too, but structures might make it simple.
- The update that you have to do each day is the toughest part of this problem.
    - Get the current location of each plant in the row of pots
    - Then use that location to look left and right to see if there is a pot available, and plant whenever you can

## WHAT DO YOU NEED TO SUBMIT?

- Create a file with your student id and name it **[your_student_id]_problem2.cpp**
- Your file should include:
    - A main( ) function solving this problem.
    - Any functions and structs you might have written

## PROBLEM #3: DISNEY++ ( 40 PTS )

Ok, last problem. This one is for you to implement this application, not with super detailed technical guidelines, but instead, as the software developer that you are. I will provide a general description of the problem and it will be up to you to make it work using everything that you have learned throughout the semester.


**Description:**

If you have a Smart TV, Roku, Fire Stick, or a PC you have probably used any of the many streaming services that exist out there. The systems work in very similar way:

- You have a list of movies/shows
- You watch one
- You can search
- You can load your preferences, and you can save them

What we are going to implement is a prototype for one of these services that focuses on movies.
Lets call it : **Disney ++** ( we will probably are going to get sued for the name, but who cares ). The system propototype is going to work as follows:

The user can:

- **Add a movie to the list:** The user can enter manually the movies they want to watch or keep in their viewing list.
- **Remove a movie from the list:** The user can remove one of the movies from the viewing list by entering the name.
- **Search a movie in the list:** Searches the list for a movie using the name of the movie.
- **Display List:** user picks whether to display the viewing list **by name** or by **favorite value**. ( both sorted )
- **Save Data:** Saves the current viewing list to a file saving each movie information one line at a time. ( look below for the information you have to save and how to save it )
- **Exit the system**

**The system should run until the user selects the option to exit the system. Each of the top 5 options above are 8 pts each.**

The information you need to store for the movie is:

- **Name:** string for the name of the movie
- **Length:** int value for the number of minutes the movie plays for
- **Favorite:** int value between 0 and 10. **Everytime a movie is searched this number gets incremented by one**

And when you save the data to the file,  you should save it as:

Movie #1 Name
Movie #1 Length
Movie #1 Favorite

Movie #2 Name
Movie #2 Length
Movie #2 Favorite

Movie #3 Name
Movie #3 Length
Movie #3 Favorite

…

## SUGGESTIONS

- Don't start coding right away, think about the system and all the different pieces that you need to make it work. Planning is very important part of any software development cycle, so do this step first.
- You can use any of the programming tools you have learned so far, make sure you use them.
- Think about the system and your own experience with similar systems, and then with the limitations of the console, how would you make it work for people to use it.
- Ask questions if something is not clear about the system you are developing. Asking your client, in this case your instructor, is always part of developing the software.

## WHAT DO YOU NEED TO SUBMIT?

- Create a file with your student id and name it **[your_student_id]_problem3.cpp**
- Your file should include:
    - A main( ) function solving this problem.
    - Any functions you might have written