CS 202: Computer Science II
Assignment 4: The Battle of Crait

Incoming Transmission from Kylo Ren: https://www.youtube.com/watch?v=FV4kIep6_tg

The end of the resistance is within our grasps. The Supreme Leader Kylo Ren has corned the resistance to the planet of Crait. We are preparing to make our land invasion of the planet to wipe out the resistance once and for all and bring glory to the First Order.

But first we need your help to develop a strong plan to invade effectively and with the lowest casualties. To do so you must develop a 2D map of the land area so we can plan our troop mobilization plan and coordinate our assault. Do not disappoint us, Kylo Ren expects a fully operational program, or else you will have to deal with him, personally.

In this assignment you will develop your skills with pointers and dynamic memory allocation along with improving your usage of object-oriented programming to achieve your goals.

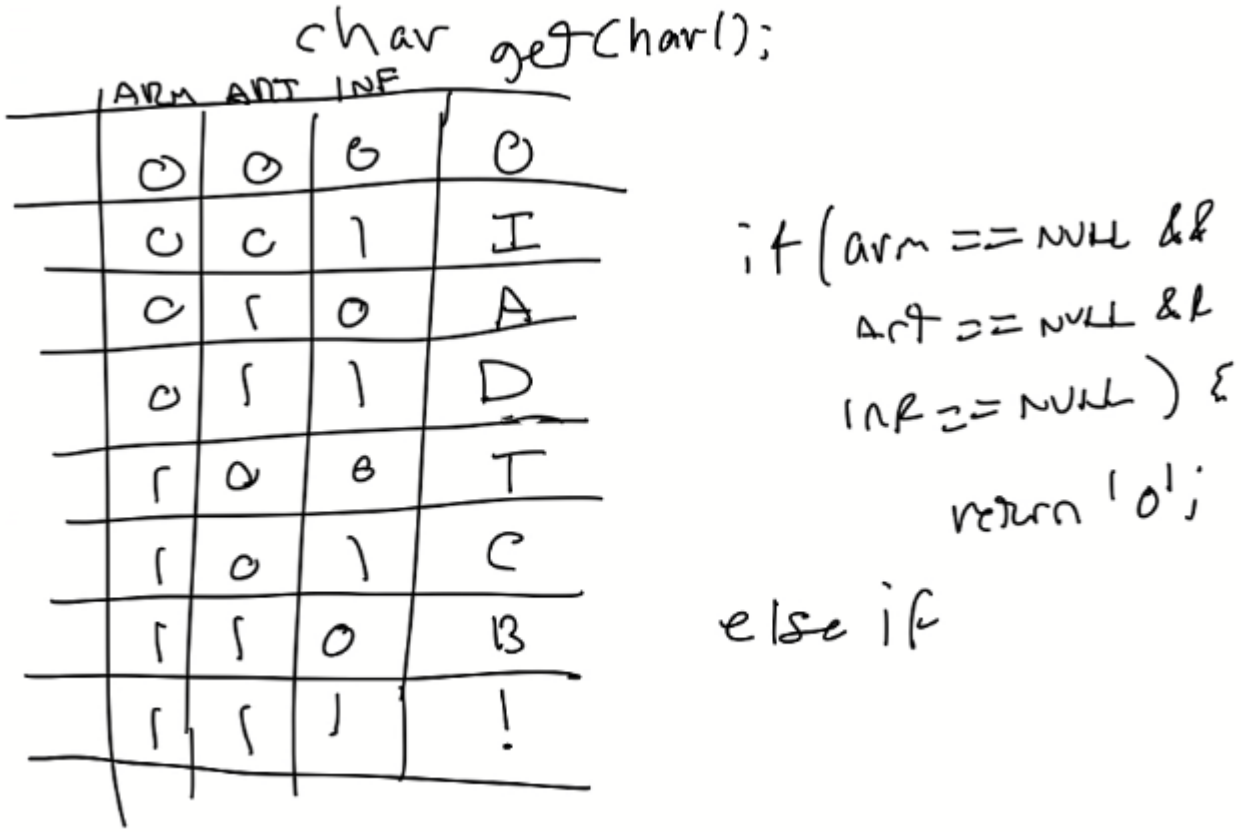
When complete, your program should display a 2-dimensional array representing X,Y coordinates in a map. Where each of the coordinates can display special characters representing the land units located in that point. There are three types of units. Armor, Artillery, and Infantry. For a given X,Y coordinate. There can only be one armor squad unit, one artillery squad unit, and an unlimited amount of infantry squad units. This is achieved at run time via dynamic memory allocation. You can create a dynamic array of infantry squads that can grow or shrink as needed be. Additionally. The units can move in the map since each point contains only 3 pointers to hold each of the given units so the actual units objects themselves can easily be pointed to by wherever they are moving to.

Rather than coding from scratch, you are given a skeleton code where you may only add code in the sections that say **//Your Code Here**. This is good practice of not just writing your own code but understanding other people's code and working with given limitations. Something that unfortunately happens more often than not.

Before we go any further let's take a look at the map you should be able to generate:

```
0   0   0   0   0   0   0   0   0   0
0   0   0   0   T   T   0   T   0   0
0   0   0   !12 0   0   I22 0   I13 0
0   0   0   0   0   0   0   0   0   0
0   0   T   0   0   0   0   0   A   0
0   0   0   I22 0   0   0   0   0   0
0   0   B   0   0   0   0   0   !11 0
0   0   0   0   0   0   0   0   D33 0
0   0   0   0   0   0   C25 0   0   0
```

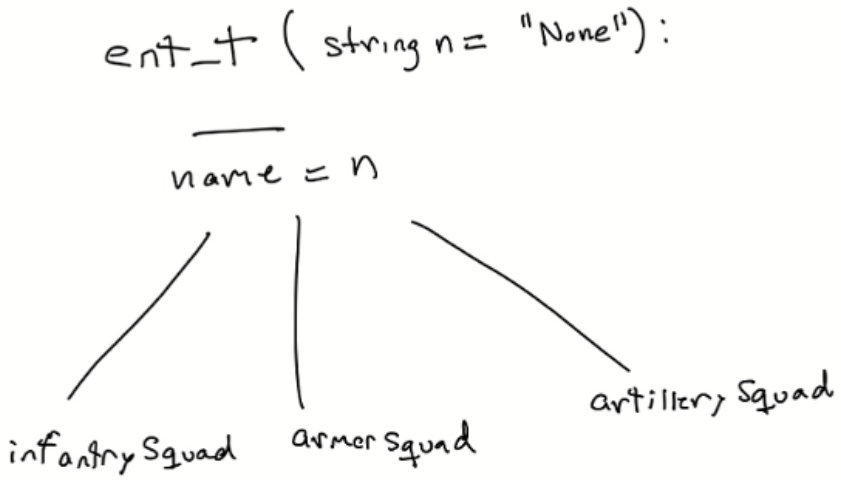
As one can see, the map is made up of ASCII characters representing what units are found in a given spot. The following table explains what each of the characters represents. It is from the video of the assignment.



In summary, a 0 in that table represents no unit present and a 1 represent a unit present. So if there is only armor and infantry units, but no artillery, then a C is displayed. If no units are present, then '0' is displayed. The numbers next to the characters are reserved for printing the number of infantry units in the array. So for example in the bottom right of the map shown at the top of this page we can see D33. This means that there is one artillery unit and 33 infantry units. Always display the number of infantry units in a given point and do not display anything (just 2 spaces) if there isn't any infantry there.

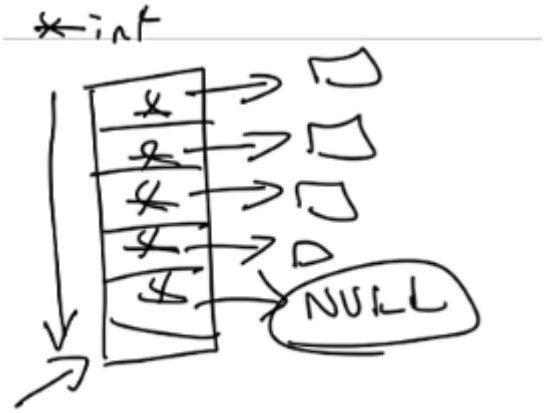
Let's go over the classes and functions that the program must contain. First there is a superclass called ent_t whose job is to provide a string variable called name. This is useful for keeping the name of each

unit. This variable should be dynamically allocated with new and delete. So you will need to add that to the constructor, destructor. The rest of the code for this class is already done for you. Reminder: Do not modify any part of the code. Only add to the code in the //Your Code Here sections. Next you have the three unit classes that are inheriting from ent_t: infantrySquad_t, armorSquad_t, and artillerySquad_t.



The code for these three should practically be identical. You are provided 2 out of the three already so just add the last one.

Next is the point_t class. This class is used to represent each point on the map and what it holds. As I already mentioned, it can hold up to 1 armor and artillery squad but can also hold a variable amount of infantry squads (dynamic array). These are 'held' by the class via pointers. See the bottom of the class in the skeleton code to see the pointer names arm, art, and inf (armor, artillery, and infantry respectively). As you can see inf is a double pointer because it holds an array rather than a single object. But it does not hold an array of objects but rather an array of pointers where the objects can live at. In other words, the array of pointers holds addresses of where the actual infantry objects live in memory. Furthermore, nowhere in the program do we keep track of how big this array is. Traditional programming in CS135 and 202 so far told you that you should keep track of how big your arrays are using some sort of size variable. Well, that's a luxury that you do not have in this program because you must learn how to handle situations where you don't have that. In the C days, strings worked this way and the way they found out how long they were was by adding the null character. Hence the term "null-terminated string". You will be doing something similar here by keeping the last entry of the infantry array with a null pointer (so address of 0). This is the way.



Doing this will allow you to iterate through the array until you see the null at which point you know when you have reached the end of the array. We will come back to this in a minute, but first let's discuss what the constructor and destructor for the point_t class shoul do. First the constructor should initialize the pointers to null and pass the coordinates so the point_t knows where it is in the map. This is already done for you, but it's important to understand it since we use null as the sign that nothing is prersent for that unit type. Next for the destructor it should de-allocate anything that the pointers are pointing to and then set them to null. This is trivial for the armor and artillery, but for the infantry you will have to iterate through the array, using the null trick to know when to stop, to be able to de-allocate all of the infantry objects that are pointed to by the array. Then at the end don't forget to de-allocate the arrray itself and set the inf pointer to null.

The print function's job is to print information on a point and can be seen here:

```
Point (3, 2) contains:
Some rusty tanks acting as decoys
Damaged Artillery
Decoy Resistance Soldiers1, Decoy Resistance Soldiers2, Decoy Resistance Soldiers3, Decoy Resistance Soldiers4, Decoy Resistance Soldier
s5, Decoy Resistance Soldiers6, Decoy Resistance Soldiers7, Decoy Resistance Soldiers8, Decoy Resistance Soldiers9, Decoy Resistance Sol
diers10, Decoy Resistance Soldiers11, Decoy Resistance Soldiers12,
```

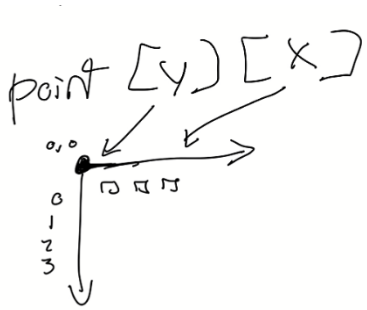
This information comes from the point's location and the name variable that the three unit types have. However again for the infantry you will need to iterate through the array until you hit the null pointer to print all of the unit names.

The getChar function returns the ASCII character to print for that point based on what the point contains at that given moment during run-time. Refer to the earlier table for how to do this along with the video.

The getInfantryCount() function returns a string with the number of infantry units in that point. To do so basically count the array and return the string version of the number. If you are caught storing the size somewhere instead of using this null-terminating method you will receive a 0 for the assignment.

Finally, the map_t class contains a 2D array of points. First let's discuss the constructor and destructor. I highly recommend that you complete the constructor of this class when you first start working on the assignment as this will allow the code to compile and run without crashing. The constructor should allocate the 2-dimensional array of pointers. The array should be dynamic but unlike the infantry array, the array itself is of actual point_t objects and not pointers. That is why it is only a double pointer, but a 2D array versus the infantry array that is a double pointer but a 1-dimensional array of pointers :)

Make sure you make the first dimension your Y and the second your X so point[y][x] is how it is organized. Else your entire map will be transposed compared to my solution. For the destructor you basically want to de-allocate the 2d Array. gg ez.

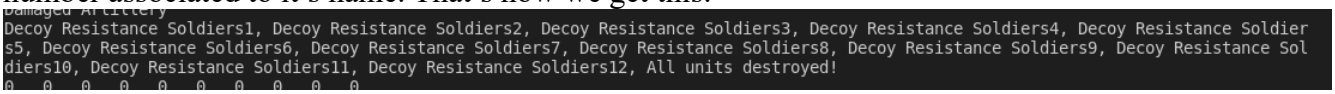


Next we have the draw function. This function's job is to call the getChar and getInfantryCount() functions to draw the correct character for each spot in the map printed to the terminal. Make sure the spacing matches mine. If done right the function should be pretty simple since the other 2 functions do all the work.

Next we have the two action functions. First is attackMove. This function's job is to move armor from one point (origin) to the destination (dest). If there is existing armor in the destination point's location then it will de-allocate it and print out that it has been Destroyed. See sample output and video for more details. Do note that there is no team designation in the simulation so friendly fire is fair game. Moving a unit is a simple as taking the arm pointer from one spot and copying that address to the new spot. Then setting the old's spot pointer to null so that you don't end up with 2 arm pointers pointing to the same unit. Don't go all Schrodinger's cat on me! The fireAt function on the other hand does not move any units but it does de-allocate and remove all units at the target point. Make sure you set them to null after you de-allocate them and print the according messages (see sample output and video). Be careful with this de-allocation, that's all I'll say there.

Finally, the last function to cover is addInfantry. It's also probably the longest. This function's job, as the name implies, is to add infantry to the point. Whereas adding armor and artillery is a simple new call. Because the infantry pointer is an array you have to do a bit more complex logic than a simple new call. The first thing to point out is that you can have two scenarios. One where there are currently no infantry units in that point and one where there already is an infantry array. In the first scenario all you have to do is allocate a new array. Remember you need space for the null pointer at the end so because you are adding the "amount" integer representing the number of infantrySquads to add, then allocate amount +1 and immediately set that pointer to NULL so you do not forget! Then go ahead and manually do the new call for each other spot in the array. For the second scenario where an array already exists, well this is where the fun begins…

So here you have to do what is known as a deep copy. In short you have to do the following: First count how big the existing array is using the same logic as before: iterate through the array until you hit the null and store that count in a local variable in the function. Once you have that count then take the amount parameter and add that to the count you just did, and finally add + 1 for the new null (make sure you don't end up with 2 nulls, only add 1 null. I warn you because I accidentally did this when I was writing the code :P) so once you finish your addition, that is going to be the size of the new array. So allocate an array of that size. Then go ahead and copy (using a loop!) the old array into the new one. That should be a 1:1 copy so the order will be maintained. In other words, we are appending at the end of the array the new squads. After you finish copying the old array, go ahead and allocate the new units added and again don't forget to have that NULL at the end! Once that is done then make sure you de-allocate the old array and make sure that this new array you made, which should currently be in some temporary pointer, copy that address into the inf pointer. One last thing to note is that when you create an infantry unit in either of the two scenarios for the constructor parameter pass something like this: new infantrySquad_t(n+to_string(i+1)); where **i** is the loop that runs from the beginning of the insertion. What this will do is make a number be part of the name so each of the soldiers has a unique number associated to it's name. That's how we get this:



As you can see the soldier names have a number right after it without a space. It's just so the soliders have unique names. It's cooler that way.

That's it. The main code is given to you already. DO NOT MODIFY IT. Be sure to watch the video and keep this open as you code for reference. Of course, feel free to reach out to me if you have questions that aren't already answered in either medium. May the force be with you.

Notes:
-Make sure your code passes all Codegrade tests if you want to receive full points.
-We may run your code with a different main that contains different values to make sure you didn't hardcode anything, so be sure to run extra tests on your own but do not submit a modified main.
-Make sure your output is EXACTLY formatted like the given sample outputs or Codegrade will eat your grade.
-Comment your source code appropriately according to the stipulations on the rubric.
-**Your program must have 0 errors from 0 contexts, and 0 bytes in 0 blocks definitely, indirectly, or possibly lost in valgrind.** That means **NO MEMORY LEAKS** allowed.

Sample Output to Terminal (I ran it with valgrind):

```
AST04$ g++ main.cpp
AST04$ valgrind ./a.out
==52663== Memcheck, a memory error detector
==52663== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==52663== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==52663== Command: ./a.out
==52663==
The First Order is the best order...
0   0   0   0   0   0   0   0   0   0
0   0   0   0   T   T   0   T   0   0
0   0   0   !12 0   0   I22 0   I13 0
0   0   0   0   0   0   0   0   0   0
0   0   T   0   0   0   0   0   A   0
0   0   0   I22 0   0   0   0   0   0
0   0   B   0   0   0   0   0   !11 0
0   0   0   0   0   0   0   0   D33 0
0   0   0   0   0   0   C25 0   0   0

No armor available.
Speeders moving to (5, 1)
Destroyed AT-ATs
AT-MA Fires at (3, 2)!!!
Point (3, 2) contains:
Some rusty tanks acting as decoys
Damaged Artillery
Decoy Resistance Soldiers1, Decoy Resistance Soldiers2, Decoy Resistance Soldiers3,
Decoy Resistance Soldiers4, Decoy Resistance Soldiers5, Decoy Resistance Soldiers6,
Decoy Resistance Soldiers7, Decoy Resistance Soldiers8, Decoy Resistance Soldiers9,
Decoy Resistance Soldiers10, Decoy Resistance Soldiers11, Decoy Resistance
Soldiers12, All units destroyed!
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   T   0   T   0   0
0   0   0   0   0   0   I22 0   I13 0
0   0   0   0   0   0   0   0   0   0
0   0   T   0   0   0   0   0   A   0
0   0   0   I22 0   0   0   0   0   0
0   0   B   0   0   0   0   0   !11 0
```

```
0   0   0   0   0   0   0   0   D33 0
0   0   0   0   0   0   C25 0   0   0

==52663==
==52663== HEAP SUMMARY:
==52663==     in use at exit: 0 bytes in 0 blocks
==52663==   total heap usage: 737 allocs, 737 frees, 96,803 bytes allocated
==52663==
==52663== All heap blocks were freed -- no leaks are possible
==52663==
==52663== For counts of detected and suppressed errors, rerun with: -v
==52663== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
AST04$
```