Hacettepe University
Computer Science and Engineering Department

Name and Surname	**:** İdil CAN
Identity Number	**:** 21727011
Course	**:** BBM104
Experiment	**:** Assignment 1
Subject	**:** Structs and Dynamic Arrays
Data Due	**:** 14th March 2018
Advisors	**:** Özge Yalçınkaya
e-mail	**:** idil.can@hacettepe.edu.tr

INDEX OF THE REPORT

1. SOFTWARE USING DOCUMENTATION

1.1. Software Usage

This program takes two input files from command line. One that includes the characters and their attributes. Other one has the commands that code's going to execute. Program needs no inputs from keyboard while execution time.

The output is printed to the screen, not to a file. It prints out the reaction to the commands. Not all commands need an output, but "SHOW", "ATTACK" and "MOVE" function needs something to print. "SHOW" command shows the situation of the given type. In "MAP" mode, it shows the map. In a character's mode, it shows the characters' status as hearth points (and experience points if it exists). "ATTACK" command prints out that the type declared before has attacked. "MOVE" function, in the other hand, prints out that if the character can't move other than "ATTACK"s executing declaration. The program also prints out the extinct type if there is one.

1.2. Provided Possibilities

The program holds the position data in two different struct (map and character's attribute) so when a program needs it, it can reach it from the easiest and the shortest path. It gives the programmer to being able to manipulate the data while accessing it. Programmer needs to be careful when using it but when it's done properly it makes updating easier: like adding more functions.

1.3. Error Messages

The program doesn't give error messages most part of the code. Only "MOVE" function has an error message programmed. "MOVE" command checks if the declared character is alive, if the new position in the map that "LOADMAP" function created before or if the new position is free. When one of these checks is wrong, it prints out the wrong one with saying that the command cannot be executed.

2. SOFTWARE DESİGN NOTES

2.1. Description of the program

This program is a combat simulator between two types of creatures: Heroes and Monsters. According to basic commands (“LOADMAP”, “PUT”, “SHOW”, “ATTACK”, “MOVE”) it simulates the battle between Heroes and Monsters.

2.1.1. Problem

This program has a complicated structure as having some objects. While C language is not a object-oriented language, it was hard to create a structure.

Since we needed to hold a map that’s assigning its size, we needed to have a multi-dimensional dynamic array.

2.1.2. Solution

Despite the fact that C programming language is a non-object-oriented language we needed to use the struct function and a struct list to hold the created structs in the creating order.

For map we need dynamic memory management but there is no dynamic array in C as a built-in function. Luckily, C has allocation functions. They provide the programmer to provide a memory area, increase or decrease a memory area at the execution time. These are handy in the character types’ implementation.

2.2. Main Data Structure

The code as three data structured that created by the “struct” function. Two for the character initialization (Hero and Monster) and one for map’s cells(Point). Hero and Monster takes name (char array), place (integer row and integer column), health point(integer) and damage(integer). Hero takes an integer as experience point as a difference from Monster and that’s the reason I have two structures for it but not one. Point struct holds two values: one integer for its occupancy (0: Blank, 1: Filled with hero, 2: Filled with Monster) and if it’s occupied name (char array).

I hold heroes and monsters in different lists. “locHero” (name stands for: list made by allocation for heroes) is an Hero array. It reallocates when a new hero comes in. This is the same in “locMonst”.

For map iteration I create an empty area to use as a list with “malloc”. And in a loop, I reallocate the memory area in order to make it two dimensional. The map list is a Point array.

2.3. Algorithm

The code takes the arguments from the command line and opens them. It starts to read the first file (characters) and adds the character to the proper structure. When the character file ends, it has two lists: one for monsters' list in the creation order and one for the heroes'.

When the second part of the program comes gets the commands from the command file line by line and it catches the commands (Ex: "ATTACK") and by using this information looks to the next words of that line. By that it performs the command.

In "LOADMAP" command, it creates the map as I explained above (*2.2. Main Data Structure*) . "PUT" command takes the coordinates and it loads the data to the map's that point's occupancy and name. Then it loads this data into the character's struct. The cross reference that I used here is not a necessity but I wanted to double-check it so I could reach the information on the shorter way without mistake.

"SHOW" command takes one more command that declares the type. Type has three options: "MAP", "HERO", "MONSTER". If this command calls in the "MAP" mode it prints the map by going through all the lists inside the map array and all the points in the lists. The program checks the point's inside(integer) value. If the point is empty it prints "*" as that point. If point's taken, it prints out the name value's first character that is being held by the point struct. "MONSTER" mode goes through the locMonst list and prints out the element's attributes as name and hearth point which are being kept in the Monster struct. "HERO" mode is same as the "MONSTER" mode with one difference: "HERO" mode prints out the experience point information too (Monsters don't have experience point).

When "ATTACK" command comes the code calls a function looks to the neighbor cells as what we've been told in the "assignment1.pdf". It returns if there is an enemy or not. If there is an enemy the enemy's heart point decreases as much as attacker's damage. If heart point becomes 0 or less this program equals the heart point to 0 and removes it from the map.

"MOVE" command checks if the character is alive or dead by looking its hp. If it's dead, it passes to the next character. If not, it checks if the given point is inside of the map. When it's not, it passes. If the character provides all those above, the program checks if the area is full. If it's not, program changes the place of the character in the struct and in the map so it keeps both data updated.

Lastly, the program checks if all the character (Hero or Monster) are dead or not. If one of the types' has no alive member, the game ends. If not, the program takes the next line. When the commands end it ends the program with to extension.