1. INTRODUCTION

The aim of this experiment is to introduce you to pointer basics and a simple data structure; stack. You are expected to implement a game, in which soldiers from two sides challenge each other for a battle. Soldiers will have different properties, which determine the outcome of a battle.

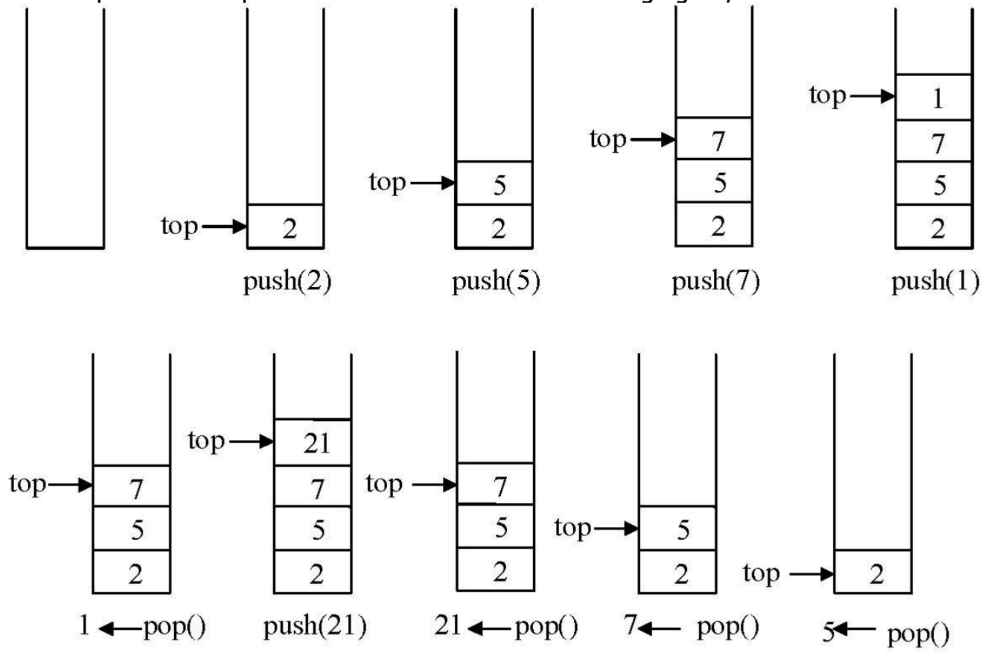
2. BACKGROUND INFORMATION

Stack is a basic data structure in which an element is added on the top of other elements (push) and elements are removed from the top of the stack (pop). Real life example of a stack can be seen in any cafeteria where the trays are stored on top of each other. In a tray stack you can not put your tray under all of the trays because of all the effort, therefore trays are always put on the top and taken also from the top. In the same way, some computer science problems require such data structures to remember its former condition or other scenarios. Stack is a last in, first out (LIFO) data structure. Each stack implementation basically is formed by following functions;

push(x)	Inserts an element as the top element of the stack
pop()	Returns and removes the top element of the stack
top()	Returns the top element without removing

Stack is a dynamic structure, it may be empty or have several elements according to the situation. Therefore a fully implementation must manage memory allocations dynamically.

An example of stack operations can be seen in the following figure;



3. PROBLEM

In this game, there will be two sides and their soldiers. Soldiers will be selected from each sides and fight with each other, turn based. After several hits when one of the soldiers is dead, another soldier will be selected from defeated side to stand against the other.

3.1. SOLDIER

A soldier have two properties; health and strength. Health varies between 0 and 100, when a soldier's health reaches zero he is dead. Strength can be between 0 and 999, it is used to calculate a soldier's damage to opposite side. Damage can be calculated by using the following formula;

$$Damage = (Strength1 - Strength2) \times 0.05 + 50$$

Soldier's properties can be stored in an integer value. Therefore stacks would store an integer value representing a soldier. A sample soldier with 89 health and 718 strength can be stored as 89718.

Simple scenario;
A soldier(X) with 32 health and 892 strength is selected from the first side. Soldier(Y) from the second side has 68 health and 573 strength. Soldier from the first side always have the priority. Damage of X can calculated as;

$$DamageX = (892 - 573) \times 0.05 + 50 = 65$$

As the outcome of the hit, soldier y will have a health of 3 (68 – 65). When soldier Y hits X, damage would be;

$$DamageY = (573 - 892) \times 0.05 + 50 = 34$$

Resulting the death of soldier X (32 – 34 < 0). Then a new soldier from the first side will be selected and selected soldier from first side will hit consequently.

4. SPECIFICATIONS

You have to store a stack for each side. Soldiers of both sides will be stored in their corresponding stacks. On these stacks you have to implement several operations. All commands will be given from the standard input. It is forbidden to use static array structure as stack and your assignment will not be evaluated.

4.1. ADD SOLDIERS

Add soldiers to the specified side. Each soldier will be separated with a semicolon “;”, health and strength values are separated with a comma “,”.

Input	Output	Explanation

`A 1 100,59;22,987;75,647`	`add soldiers to side 1`
	`S- H:100 S:59`
	`S- H:22 S:987`
	`S- H:75 S:647`
		Adds three soldiers to first side. First soldier with 100 health, 59 strength, second soldier with 22 health, 987 strength, third soldier with 75 health, 647 strength.

4.2. FIGHT

When a fight command is entered, a soldier will be selected from the stack of the side by top(). When a soldier dies it will be popped from the stack.

Input	Output	Explanation

`F`
`F`
`F`
	`1 hit 65 damage`
	`2 hit 34 damage`
	`1 has a casualty`
		Runs a turn. Several outputs are shown. Each is an output of executed F command.

4.3. CALL REINFORCEMENTS

A soldier with random properties will be pushed to the stack.

Input	Output	Explanation

`R 1`	`Called reinforcements to side 1`
	`S- H:100 S:59`
		Adds a random soldier to given side. Health and strength are randomly assigned between specified limits.

4.4. CRITICAL SHOT

A soldier will be popped from the stack, acts like a fight command but instantly kills the opposite soldier. Turn changes.

Input	Output	Explanation

`C`	`Critical shot`
	`1 has a casualty`
		Kills the soldier of the opposite side. It decides the side depending on the whose turn is it.