Lab 3: Stacks

ELEC 278: Fundamentals of Information Structures

Learning Goals

- Implement and explain fundamental stack operations such as push, pop, and peek, and understand how these operations affect the stack's state.
- Learn to manage stack memory efficiently, including detecting and handling stack overflow and underflow conditions during stack operations.

Instructions

Download the zipped folder `lab3.zip` from OnQ and unzip it. Next, open the lab3 folder that you extracted in your code editor. You will complete the following exercises by completing the functions in the `stack.c` and `main.c` files.

To receive full marks, your code must be able to correctly handle corner cases. Comment your code and be ready to discuss how your code handles corner cases with your TA. Generate a list a cases that you will test your code against with your TA.

Grading (fraction of exercise marks)

0 No code or lack of understanding of the code.

1/2 Functioning code, but fails corner cases, if any, or not readable/not well-explained.

1 Functioning code that handles corner cases

Exercise 1. (0.5 mark) Stacks can be implemented with as an array or (a more memory efficient) linked list. Given that in the last lab, you gained practice with linked lists, this lab will focus on the array implementation.

For exercises 1-6, you will complete the functions in `stack.c`.

You are given the following `C`-struct for the stack, which assumes that the maximum stack size you will need is 100. You still need to handle cases of stack under- and over-flow.

```
#define MAX 100 // Maximum size of the stack

typedef struct Stack {
    int top;
    char array[MAX];
} Stack;
```

Use `top` to index the top position of the stack.

Write a function `Stack* createStack()` to create a new stack.

Hint: You will dynamically allocate memory for the stack.

Exercise 2. (0.5 mark) Write a function `bool isFull(Stack* stack)` to check if the stack is full.

Exercise 3. (0.5 mark) Write a function `bool isEmpty(Stack* stack)` to check if the stack is empty.

Exercise 4. (0.5 mark) Write a function `void push(Stack* stack, int item)` to add a new item to the stack if space is available on the stack; otherwise, print a statement notifying the user of over-flow and do not push the item to the stack.

Exercise 5. (0.5 mark) Write a function `int pop(Stack* stack)` to remove and return the top item from the stack if it is available; otherwise, print a statement notifying the user of under-flow.

Exercise 6. (0.5 mark) Write a function `int peek(Stack* stack)` to (look) return the top item from the stack if it is available.

Exercise 7. (1 mark) For this exercise, complete the function in `main.c`.

You are designing a fighting video-game with the following score-keeping rules.

At the beginning of a game round, you start with an empty score stack.

You are given a list of actions for the round, where `actions[i]` is the ith action and each action has a category `action.name` (out of those listed below) and a value `action.x`:

PUNCH	: You have punched your opponent. You get one point.
KICK	: You have kicked your opponent. You get two points.
WILD	: You have received a wild `action.x` points.
DOUBLE	: You have doubled your last points. Replace your last points with its double.
TRIPLE	: You have tripled your last points. Replace your last points with its triple.
HIT	: You have been hit. You have lost your last `action.x` point collections.

When the round is over, your score is the sum of all points on the stack.

Complete the function `int calPoints(Action actions[], int actionSize)`

For example, if the actions are

```
Action actions[] = {
    {PUNCH, 0},
    {KICK, 0},
    {DOUBLE, 0},
    {TRIPLE, 0},
    {WILD, 10},
    {KICK, 0},
    {KICK, 0},
    {HIT, 3},
    {DOUBLE, 0}
};
int actionSize = 9;
```

1. Add 1 to the stack, your stack is now [1]
2. Add 2 to the stack, your stack is now [1,2]
3. Double your last points, your stack is now [1,4]
4. Triple your last points, your stack is now [1,12]
5. Add a wild 10 points, your stack is now [1,12,10]
6. Add 2 to the stack, your stack is now [1,12,10,2]
7. Add 2 to the stack, your stack is now [1,12,10,2,2]
8. You've been hit hit. You lost your last 3 points collection. your stack is now [1,12].
9. Double your last points, your stack is now [1,24]

Your round score is 25.