HW1 – Case Study: Stacks

Version 1

Q1: Before printing, we popped one element, thus the elements left in the stack are {x, y}, and top = 2. However, the array still has all the elements {x, y, z} since we didn't necessarily deleted any values from the memory.

```
afgde
```

Q2: We first pushed 5 elements to the stack, {a,b,c,d,e}. By popping 4 times, we only now get {a} as the element of the stack. Then, pushing f and g gives us {a,f,g} as the elements of the stack. However, we are technically replacing the values in the array when we push {f, g} because of the stack implementation. Printing the Store array would give us {a,f,g,d,e}.

Version 2

```
Popped: y
```

Since we changed the value of top to 2, we treat the element at index 2 as the next "free" space for the stack, which also means that 'z', which is located at index 2, becomes a garbage value. This also means that 'y' is now the top-most element in the stack, hence we get 'y' when we pop from the stack.

Version 3

```
4   void application(){
5       struct stack s0, s1;
6       initstack(&s0);
7       initstack(&s1);
8
9       push(&s0, 'x');
10      push(&s1, 'y');
11      push(&s1, 'z');
12      push(&s0, 'a');
13
14      s1.Store[1] = 'd';
15
16      char ans = pop(&s1);
17      printf("Popped from s1: %c \n", ans); // this outputs 'd'
18  }
```

```
Popped from s1: d
```

One weakness is that we can directly access the variables of the stack struct (as demonstrated at line 14), which could potentially be a security issue.

Version 4
We are only limited to whatever the value of MAX_STACKS is for the number of stacks we can have, which would not be controlled by the programmer of `app.c`.

```
 9      for (int i = 0; i <= 10; i++) {
10          push(i, 'a'); // out at i=10: Invalid stack number 10
```

```
Invalid stack number 10
Popped from stack 0: a
```

Version 5
Reason 1: Simplicity
A programmer would prefer **Version 1** if they are just trying to comprehend the basic theory of a stack, excluding any programming techniques that can come with its implementation.

Reason 2: Use case
A programmer would prefer Version 4 if they have a specific use case that warrants the usage of limiting the number of stacks available. Developing a game can be an example; there could only be a constant number of stacks that will be used for the mechanics of the game (ex. 10 stacks for 10 players holding card hands).