Task 1:

In this task, I created essential functions for stack functionality with the cstk.h header file. These functions included basic, necessary functionality, such as pop, push, peek, checking if empty or full, and more. The stack can also be copied to a new stack or destroyed.

Task 2:

I created the cstk.c file, which contains the implementations of the functions defined in the cstk.h header file

Task 3:

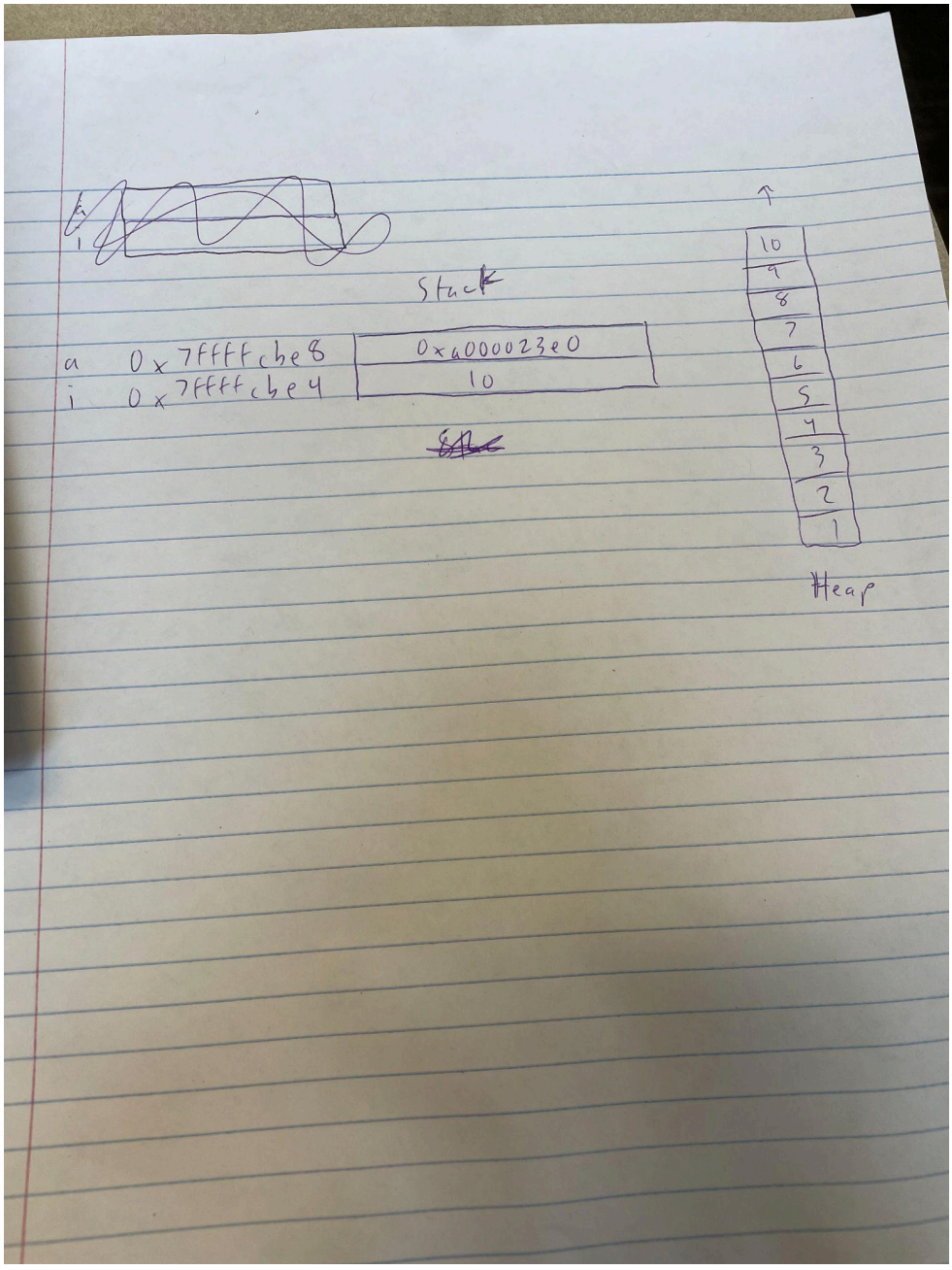
```
All non-display tests passed!
You should now see the numbers from 0 to 9:
Printing stack in original order:
0 1 2 3 4 5 6 7 8 9
Now you should see the numbers from 9 to 0:
Printing stack in reverse order:
9 8 7 6 5 4 3 2 1 0
```

Task 4:

```
Mark 1: Stack and Heap Memory
------------------------------
Stack Pointer (s): 0xa00000450
Heap Memory (Stack Data)
------------------------------
Address: 0xa00002330
Top Pointer: 0xa00002330
Capacity: 0xa00000460

a is at 0x7ffffcbe8
i is at 0x7ffffcbf4
array starts at 0xa000023e0
The original list: Printing stack in original order:
1 2 3 4 5 6 7 8 9 10
The reversed list: Printing stack in reverse order:
10 9 8 7 6 5 4 3 2 1

Mark 2: Stack and Heap Memory After Destruction
------------------------------------------
Stack Pointer (s): 0xa00000450
Heap Memory (Stack Data) Freed
```



In the images above: ‘a’ is a pointer to the stack stored in the memory heap. Additionally, when the function “stk_destroy” is encountered in the code, allocated memory is freed. This is illustrated in the video I provided in my .zip. By freeing both p->top and p, we are ensuring that no memory leaks occur.