Lab 2

27 January 2023

This week's lab is about practicing two things in C: structs, and pointers (including heap allocation).

Black box spec

Your program will read a sequence of commands from the user. Each will be either "`push N`" where N is an integer within the two-byte standard signed integer range (even if on a system that has four-byte integers!), or "`pop`", or "`end`". A response to a push will be "`ok`" on a line by itself, and will store the integer in a stack internally; a pop command will respond with the single popped number on a line by itself; and ending will print the entire remaining contents of the stack, one per line, before ending the program. Any other input should get the response "`error`".

Example

```
push 4
ok
push 6
ok
pop
6
push 8
ok
blah
error
push 10
ok
end
10
8
4
```

Internal and other requirements

Your stack should be maintained with a linked list, each node of which is a two-element struct.

All dynamically-allocated memory should be released when the program is done with it (including at the end of the program, not just relying on the OS to do so).

All string manipulation should be protected from buffer overruns and malicious input.

You should not simply put the entire program in one enormous `main` function. Exact design is up to you, but you should use functions appropriately.

All the reading and printing should be in `main`, though.

Error input should trigger output of `error` (as shown above) but some programs may print `error` multiple times and that's ok, as long as after the (possibly multiple) `error` lines it gets back to reading and responding to stack instructions.

As before, your handin should include a readme documenting how to compile, run, and test your program. You may use a makefile; you should not rely on `compile`. (If you wish to use unci, i.e. `.u` files, to test any functions you write, you can use `uncic` directly to compile the `.u` files into `.o` files. See me for details if you need help.)

Programs that do not compile will get few or zero points.

Use test cases to show me what works.

Duedate and handin

The lab is due Friday at 4pm.

Hand it in using the handin script:

```
handin cmsc242 lab2 dir_of_stuff/
```

Rubric (tentative)

RUBRIC

General and design

1 compile instructions
1 testing files or instructions
1 functions used
1 good function breakdown and overall design (e.g. no globals)
1 follows I/O spec on what to print, including ok and error
1 appropriate header files included
1 `main` declared and compiles

Main loop and I/O

1 main loop
1 read word into valid buffer
1 ... limited to buffer length (or otherwise guarded)
1 read number
1 parses any and all valid input
1 prints error message on bad input, continues
1 checks commands against "`push`", "`pop`", "`end`"
1 ... with valid comparison
1 prints anything
1 prints a number

Data design

1 declared a valid struct
1 ... with at least one field
1 correct number of fields, can hold data
1 includes field to support linked list
1 declares a pointer-to-struct variable in code
1 accesses a field of a struct value
1 inits empty stack appropriately
1 allocates heap memory somewhere

Push

1 reads num after push and only then
1 push stores num
1 allocates heap memory for linked list node (correct size)
1 struct gets assigned values after allocation
1 new node connects to linked list (no extra nodes, all new nodes connected)
1 main stack variable gets updated

Pop

1 pop on empty gives error
1 stores values in temp variables during deallocation
1 reassigns pointer(s) to remove from linked list, update stack variable
1 deallocates node memory (and doesn't allocate more)
1 prints number that was popped

End

1 end loops through remaining stack
1 prints each number in stack in order
1 deallocates all remaining nodes exactly once (no double-deallocate)
1 program ends after printing and cleanup