Nout Reusken

Professor Dunne

CSE 13S

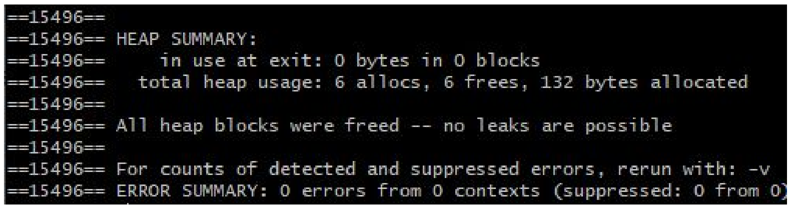
1 November 2020

Assignment 3 Write Up

In this assignment, I created a program that plays the game, *Tower of Hanoi*, using two different implementations. One is using the recursive method and the other one uses the implementation of stacks. Both implementations were very successful and performed the way I expected them to.

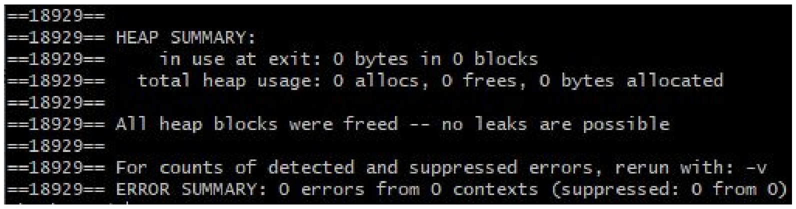
From the assignment, I got a deeper understanding of the game, *Tower of Hanoi*, and how either implementation worked with the issue of following the rules of the game, *Tower of Hanoi*. There were two observations that I made concerning the two implementations when putting them side to side.

First the memory complexity with the implementations. In the recursive implementation, there is barely any memory usage and the only real variable used and that change frequently were *num_disks* and *moves*. These only changed by increments of one and didn't cause the program to allocate a certain amount of space. While comparing this to the stack implementation this is a whole different story. Just by looking at the *stack.c* where the functions for a stack are defined, you can see how many variables are used causing it to be way more than the recursive implementation. Also when running Valgrind to see the *HEAP SUMMARY* and see if there were any leaks while running, you can also see how many times space was allocated and how many bytes were allocated. When running just running the program using the stack implementation with only 5 disks, you can see that the



```
==15496==
==15496== HEAP SUMMARY:
==15496==     in use at exit: 0 bytes in 0 blocks
==15496==   total heap usage: 6 allocs, 6 frees, 132 bytes allocated
==15496==
==15496== All heap blocks were freed -- no leaks are possible
==15496==
==15496== For counts of detected and suppressed errors, rerun with: -v
==15496== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

program had allocated 6 times and freed 6 times, and allocated a total of 132 bytes, which is not bad but when comparing this to when using the recursive implementation there is a pretty big difference. The recursive implementation doesn't have to allocate any space at all. This makes the recursive implementation way more efficient compared to the stack implementation.



```
==18929==
==18929== HEAP SUMMARY:
==18929==     in use at exit: 0 bytes in 0 blocks
==18929==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==18929==
==18929== All heap blocks were freed -- no leaks are possible
==18929==
==18929== For counts of detected and suppressed errors, rerun with: -v
==18929== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Another observation made is that the design complexity is way more complex for the stack implementation than the recursive. The stack implementation needs a full understanding of the stack functions and seeing the pattern in either the even or odd-numbered disks is hard to see until someone points it out. The recursive function also is way shorter than the stack implementation. The stack implementation really requires you to know how the function uses the stacks since if it *ending_peg* never gets filled it causes an infinite loop, which causes an added complexity when debugging.

In the end, the observations made by comparing the recursive implementation and the stack implementation of the game, *Tower of Hanoi*, in this assignment. Even though, when handling data and managing larger amounts of data using stacks is a good way to handle it, but when looking at the problem of this game, it isn't really beneficial since there is the memory that needs to be allocated often making the program slower. Also, the added complexity of the design makes it harder to work on.