Lab 3

Due Jan 24 by 6:30pm **Points** 1

Lab 3: Dynamic Memory

Due date: Friday 24 January before 6:30pm. Submissions made after the deadline will not be accepted.

Introduction

The purpose of this lab is to practice using pointers and dynamic memory allocation and to learn to use the valgrind tool.

To start, login to MarkUs and navigate to the `lab3` assignment. Like the previous lab, this triggers the starter code for this lab to be committed to your repository.

We have provided a checker program for each of the programs that you will write. There are two checks per program: one checks whether the program complies without errors/warnings, and the other checks a single test case. The purpose of the checker is to help you verify that your program's output is correctly formatted. You can run the checks yourself up to once per hour on MarkUs. After the deadline, we will run a larger set of tests on your submission for marking, so you should test your code thoroughly and not rely solely on the checker.

Fibonacci

Your first program will build a dynamic array containing the first `n` elements of the fibonacci sequence. Write the `fibonacci` function in the `fibonacci.c` file. Do not change the `main` function.

Split Arrays

In the `split_array.c` program, you will write the bodies of two functions. You will need to fill in the arguments to correctly call the two functions so that your program behaves as shown in the following examples.

Here are a couple of things to notice:

- `$` is just the shell prompt
- `argv[0]` is *not* included in the result arrays.

```
$ ./split_array 1 2 3
Original array:
1 2 3
result[0]:
1 3
result[1]:
2
$ ./split_array 1
Original array:
1
result[0]:
1
result[1]:

$ ./split_array
Original array:

result[0]:

result[1]:

$ ./split_array 10 234 6 5 33 44
Original array:
10 234 6 5 33 44
```

```
result[0]:
10 6 33
result[1]:
234 5 44
```

Using Valgrind

The program `valgrind` is a tool that is used for detecting memory and other errors in programs. In particular, it can detect when memory is not allocated or freed correctly.

Running `valgrind` on the solution to `split_array` produces the following output:

```
$ valgrind split_array 1 2 3
==52893== Memcheck, a memory error detector
==52893== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==52893== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==52893== Command: split_array 1 2 3
==52893==
Original array:
1 2 3
result[0]:
1 3
result[1]:
2
==52893==
==52893== HEAP SUMMARY:
==52893==     in use at exit: 0 bytes in 0 blocks
==52893==   total heap usage: 5 allocs, 5 frees, 1,064 bytes allocated
==52893==
==52893== All heap blocks were freed -- no leaks are possible
==52893==
==52893== For counts of detected and suppressed errors, rerun with: -v
==52893== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Try running `valgrind` on your solution to `split_array` on **teach.cs**. You should be able to get similar output. The "HEAP SUMMARY" should show that no bytes are still in use at exit, and the "ERROR SUMMARY" should show 0 errors.

If you didn't have any errors to fix and have more time, comment out one of the `free` statements in `split_array` and rerun `valgrind` to see the results. Don't forget to put it back before you commit your final solution.

Submission

Use git to submit your final `fibonacci.c` and `split_array.c` files. Do *NOT* add or commit executables to your repository. We will build executables by compiling your code as part of testing it.