CS 2630, Fall 2022
HW1: Basic C Skills
Assigned: Aug. 30

Due: Sept. 6 (HW1a), Sept. 12 (HW 1b), 11:59PM

1 Introduction

The purpose of this assignment is to become more familiar with C programs, especially passing by value and pointers. You'll do this by implementing the functions declared in the hw1a.h and h1b.h files. These functions are called by the main function in main.c, which is a test driver for the assignment. This assignment will take much longer than you expect.

2 Logistics

This is an individual project. All handins are electronic. You will submit the hw1a.c and hw1b.c files you create via the ICON assignment page. NOTE: these files must first be copied to a file with a modified name, which is done with the Makefile. See the Handin Instructions for details. Clarifications and corrections will be posted on the course ICON page.

3 Handout Instructions

Download the `hw1-handout.tar` file from the ICON HW1 assignment page and copy it to your cs2630 (protected) directory on a Linux machine in which you plan to do your work. Then execute the command

```
  unix> tar -xvf hw1-handout.tar.
```

This will cause a number of files to be unpacked in a new HW1-handout directory. You will create files `hw1a.c` and `hw1b.c` and turn in copies of those files (with names prefixed with your hawkid, as detailed in the Handin Instructions section).

The `hw1a.h` and `hw1b.h` files contain function declarations (aka prototypes) for each of the 18 functions you need to implement (6 functions for HW1a and 12 functions for HW1b).

4 HW1a

This section describes the functions you must define in `hw1a.c`, which is due Sept. 6 at 11:59PM. The function prototypes are declared in `hw1a.h`. You will submit a copy of the file `hw1a.c` for HW1a and no other files. Thus, your solution should not rely on changes to any other files.

Table 1 lists the function names, a brief description of what each does, and the points awarded for a correct implementation. The functions are listed in the general order of difficulty, and I suggest you implement them in that order. See the comments in `hw1a.h` for more details on the desired behavior of the functions. You may also refer to the test functions in `main.c`.

Name	Description	Points
`maxNumber()`	return the largest number in an array	2
`minNumber()`	return the smallest number in an array	2
`sum()`	return the sum of the numbers in an array	2
`sortIncreasingOrder()`	sort an array in increasing order	4
`sortDecreasingOrder()`	sort an array in decreasing order	4
`medianNumber()`	return the median of the numbers in an array	4

Table 1: HW1a functions. HW1a functions all have two input parameters: an array, and the length of the array. The sort functions sort the array in place.

Use the `Makefile` to build your program with the command `make hw1a`. The make rule hw1a compiles `hw1a.c` and `main.c`, and then links the `*.o` files into an executable file called `hw1`. The command `make clean` removes all `*.o` files and the executable file `hw1`.

When executed, `hw1` reads up to 15 integer numbers from the console and stores them in an array. Enter a `cntrl-d` on an empty line when you are done entering numbers. The function `main()` then calls the function `hw1a()`, which then calls all 6 methods declared in `hw1a.h`, passing the array of numbers read.

5 HW1b

This section describes the functions you must define in `hw1b.c`, which is due Sept. 12 at 11:59PM. The function prototypes are declared in `hw1b.h`. You will only submit a copy of the file `hw1b.c` for HW1b and no other files. Thus, your solution should not rely on changes to any other files.

Table 2 lists the function names, a brief description of what each does, and the points awarded for a correct implementation. The functions are listed in the general order of difficulty, and I suggest you implement them in that order. See the comments in `hw1b.h` for more details on the desired behavior of the functions. You may also refer to the test functions in `main.c`.

Use the `Makefile` to build your program with the command `make hw1b`, `make hw1`, or `make all`. The command `make hw1b` compiles `hw1b.c` and `main.c`, and then links the `*.o` files into an executable file called `hw1`. The command `make all` or `make hw1` compiles `hw1a.c`, `hw1b.c`, and `main.c`, and then links the `*.o` files into an executable file called `hw1`. The command `make clean` removes all `*.o`

Name	Description	Points
`createNode()`	return an initialized linked list node	2
`destroyNode()`	free the space allocated to a node	2
`appendList()`	append a node to the end of a linked list	3
`createList()`	create a linked list from an array	3
`destroyList()`	free the space allocated to a linked list	2
`listLength()`	return the length of a linked list	2
`maxElement()`	return the largest number in a linked list	2
`minElement()`	return the smallest number in a linked list	2
`sumElements()`	return the sum of the numbers in a linked list	2
`sortListIncreasingOrder()`	sort a linked list in increasing order	5
`sortListDecreasingOrder()`	sort a linked list in decreasing order	5
`medianElement()`	return the median of the numbers in a linked list	8

Table 2: HW1b functions. HW1b functions operate on linked lists. The sort functions return a sorted linked list. Note: there is no header reference for the linked lists. The first node is the start of the linked list.

files and the executable file `hw1`.

When executed, `hw1` reads up to 15 integer numbers from the console and stores them in an array. Enter a `cntrl-d` on an empty line when you are done entering numbers. If built only for hw1b, the function `main()` then calls the function `hw1b()`, which then calls all 12 methods declared in `hw1b.h`, passing the array of numbers read. If built with `make all` or `make hw1`, `main()` calls both `hw1a()` and `hw1b()`.

6 Handin Instructions

- You will be handing in two sets of files (with one file in each set):
  - Part A: `hw1a.c`.
  - Part B: `hw1b.c`.
- Make sure you have included your name and hawkID in a comment at the top of each of your handin files.
- To build your handin file for part X, go to your `HW1-handout` directory and type:

  ```
  unix>  make handin-partX
  ```

  where `X` is `a` or `b`. For example, to handin Part A:

  ```
  unix>  make handin-parta
  ```

- **Use the ICON assignment page to handin your file for part X, which is in your directory** `handin-partX`. For example, for Part A submit the file in `handin-parta`. This file will have the name $USER-hw1a.c, where $USER is your FastX username, which should be the same as your HawkID.

- Each part will be graded on FastX, so be sure your solution compiles and executes there. You will get zero Compile points if it fails to compile or you submit the file without the proper name.

7 Evaluation

HW1 has a maximum score of 70: 25 points for hw1a and 45 points for hw1b.

7.1 HW1a Evaluation

Your score will be computed out of a maximum of 25 points based on the following distribution:

18 Correctness points.

4 Style points.

3 Points for compiling without errors and the correct file name.

Correctness points. Each function is worth 2 or 4 points, as indicated in Table 1, such that their sum totals to 18. We will evaluate your functions using a set of input arrays. You will get full credit for a function if it passes all of the test input arrays, and no credit otherwise.

Style points. We've reserved 4 points for a subjective evaluation of the style of your solutions and your commenting. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

Compiles and doesn't crash points. Finally, we've reserved 3 points for correctly naming your `hw1a.c` file as `hawkid-hw1a.c`, e.g., `sgoddard-hw1a.c`) and submitting a file that compiles and doesn't crash the autograder. NOTE: the Makefile will copy your solution to a file with the proper name (as described above).

7.2 HW1b Evaluation

Your score will be computed out of a maximum of 45 points based on the following distribution:

38 Correctness points.

4 Style points.

3 Points for compiling without errors and the correct file name.

Correctness points. Each function is worth 2 to 7 points, as indicated in Table 2, such that their sum totals to 38. We will evaluate your functions using a set of input arrays. You will get full credit for a function if it passes all of the test input arrays, and no credit otherwise.

Style points. We've reserved 4 points for a subjective evaluation of the style of your solutions, commenting, and properly freeing all allocated blocks of memory. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

Compiles and doesn't crash points. Finally, we've reserved 3 points for correctly naming your `hw1b.c` file as `hawkid-hw1b.c`, e.g., `sgoddard-hw1b.c`) and submitting a file that compiles and doesn't crash the autograder. NOTE: the Makefile will copy your solution to a file with the proper name (as described above).

8 Advice

- Create a small test program to test individual functions with various parameters.
- Create a helper function that makes a copy of a linked list. This will be very helpful when finding the median value of a linked list. If you don't do this, you will likely lose part of your linked list.
- Feel free to find example algorithms for your functions on the Internet and implement those. Just be sure to cite where your algorithm/code came from. (The goal of this assignment is to gain experience programming in C, not in the development of algorithms.)
- Your program must have no memory leaks and no memory errors (e.g., invalid reads or writes). Run it through `valgrind` on the FastX linux machines to check for memory leaks. The TAs will run `valgrind` on your program. Here is a quick start tutorial:
  `https://valgrind.org/docs/manual/quick-start.html`.
  You can use Google to find more detailed tutorials if necessary.
- START EARLY!!! In the worst case, you are done early. The more likely scenario is that you encounter a pointer error, or some similarly insidious bug, that requires several hours of debugging and, for some, painful banging your head against the wall.
- Avoid banging your head against the wall. This is dangerous and can cause serious adverse medical conditions. To avoid the temptation to bang you head against the wall: START EARLY and ASK QUESTIONS!! Your TA and instructor are very happy to assist!