CSE 6010
Workshop 1

Due Date: 11:59pm on Thursday, September 7
Submit to Gradescope through Canvas
48-hour grace period applies to all deadlines.

There are several purposes for the workshop assignments:

- To provide you with some low-stakes programming practice in C where you can get real-time help from the TAs (if you attend the class in person);
- To cover small programming-related subjects that don't have a logical place among the course topics; and
- To give you the opportunity to work together with other students, which can expose you to new ways of thinking about and writing C code.
- To provide the opportunity for extra credit (if you submit more than 4 workshop assignments).

For this assignment, you may work in groups of 2-3 students. If you prefer, you also may choose to work independently.

If you choose to work as a team, you are expected to **work together for the full assignment**. A divide-and-conquer approach, where each team member works independently on some part of the assignment and the team only interacts when they combine their separate codes to submit, is not in the spirit of this assignment. If you don't want to work with anyone, you should submit independently.

The workshop is designed so that we expect most of you will be able to finish most of the assignment during the **75-minute class period**, possibly with a little extra time offline for cleanup and submission. Nevertheless, we know that different individuals and groups will have different levels of comfort/proficiency with C, and that programming does not always follow a timeline. Thus, it is not strictly required that every submission necessarily will include all topics. As part of your submission, please write a few sentences about how things went. This is your opportunity to let us know if you got bogged down in a particular area and how you went about resolving any problems you may have encountered.

Contributions

We ask you to identify how each team member participated by filling in and including the contrib.txt file provided. For each problem, please assess the contribution level for each category according to the following scale.

- 3 = >50% (student did the majority of the work for that category – so there can only be a maximum of one "3" per category + problem combination)
- 2 = 20-50% (solid contribution)
- 1 = 1-20% (minor contribution)
- 0 = no contribution

The categories are:

- Ideas / planning
- Detailed code design
- Writing code / implementation
- Debugging / testing
- Documentation / comments

You can find the template 'contrib.txt' file on Canvas under Workshop 1.

When submitting to Gradescope, make sure your filename is '**contrib.txt**'.

Your assignment

1. ***Returning information three ways.*** Because a C function can only return a single value, alternatives are required when you want to retain information after calling a function. Here, you will create an array of *100* nonnegative double values and write three different functions that each use a different approach to return two pieces of information: the maximum value stored in the array and the index of the maximum value. You may include all three functions in one file and call them from the `main()` function in succession, and you should output to the screen the maximum value and the corresponding index found using each method (they should be the same). The array values should be set to `1 + sin(i)`, where `i` corresponds to the array index, and the array should be allocated statically (you do not need to call `malloc`). Note that to calculate `sin()` you likely will need to include `math.h` and in your code, and your compilation command likely will need to include `-lm` at the end.
    - **Method 1**: return both the maximum value and the corresponding index by passing pointers to the function and updating the values at those addresses.
    - **Method 2**: return the maximum value as the function return value; save the corresponding index by passing a pointer and updating the value at that address.
    - **Method 3**: define a struct that includes both the value and the index, and return that struct (multiple pieces of information in a single return variable!).

    When submitting to Gradescope, make sure your filename is '**return.c**'.

2. ***Dynamic memory allocation and memory leaks.*** Dynamic memory allocation is important in C but takes some get used to. In particular, good programming practice includes making sure that you have freed all allocated memory, so you should make such a check a habit. A useful tool for checking whether all memory is freed is "valgrind". In this assignment, you will set up a simple linked list, print out the values stored in the list, and free all memory. You should test whether you have memory leaks using "valgrind", as described below.

    Create a linked list of *10* items. Each item should be a struct that includes an integer as well as a next field that points to the next item on the list. Set up a `struct` with `typedef` to store a list item. Initialize the list by creating a dummy item whose value is `-1`, then use a for loop to add *10* items to the list; the integer stored for each item should be the loop index (starts from *0*). Add items to the front of the list so that you do not need to traverse the whole list every time you add an item. After you have added the items, print out the integers in the order they appear in the list (you should find they are in descending order). Finally, free all memory.

    When submitting to Gradescope, make sure your filename is '**linkedlist.c**'.

Valgrind

To test whether all memory has been freed, you can use valgrind, which is a tool to find memory errors. To do so, add the flag -g to your code compilation step, then run your code as

```
valgrind --leak-check=full ./mycode
```

(Assuming your executable is named `mycode`). If you have freed all memory, valgrind should provide output indicating that no leaks are possible. Otherwise, it will provide some information about how much memory has been lost because it has not been freed. Any memory that is "definitely lost" or "indirectly lost" constitutes a memory leak that should be fixed. Make sure that every `malloc()` call is paired with a corresponding `free()` call. If you free all your memory the first time, we recommend you comment out your free function call in your program temporarily just to become familiar with how valgrind shows memory leaks.

Valgrind can do other things, such as find when conditionals are used with uninitialized variables. For more information on Valgrind, please see online resources like https://valgrind.org/docs/manual/quick-start.html

Group submissions

Submit your code files named "**return.c**", "**linkedlist.c**" and "**contrib.txt**" on **Gradescope** under **Workshop 1**. If you don't know how to submit on Gradescope as a group, you can follow this tutorial on YouTube.