Profs. Jingke Li, Classes: TR 14:00-15:50, KMC 390, Office Hours: TR 1300-1400 (in-person @FAB 120-06).

Assignment 3 (X86-64 Assembly Programming)

(Due Thursday 2/27/25 @11:59pm)

In this assignment, you'll practice x86-64 assembly language programming by implementing a set of tasks. Clarity and correctness are our primary goals. In particular, we're not looking to demonstrate mastery of the full instruction set and we're not going to worry about performance or code size. This assignment carries a total of 20 points.

The Tasks

In each of these tasks, the goal is to write a fragment of x86-64 code that takes the address of an array of (32 bit/4 byte) integers as its input in the `rdi` register and returns a result in the `eax` register. In each case, we will assume that the elements of the array are stored in successive locations in memory, each 4 bytes wide, with the first entry at the address that is provided in `rdi`. We will also assume that the array is terminated with a zero and that it contains at least one element before that terminating zero is reached.

- *Task 1* [2 pts] — Return the length of the input array in `eax`.
- *Task 2* [2 pts] — Return the sum of the numbers in the array in `eax`.
- *Task 3* [2 pts] — Return the average value of the numbers in the array using integer division and ignoring any remainder.
- *Task 4* [2 pts] — Return the largest number from the array in `eax`.
- *Task 5* [2 pts] — Return the position of the largest number in the array in `eax`.
- *Task 6* [5 pts] — Reverse the order of the elements in the array, without using any additional storage.
- *Task 7* [5 pts] — Sort the elements in the array into increasing numerical order, without using any additional storage. (No algorithmic sophistication should be expected here!)

Each of the tasks should be written in a separate file, e.g. `task1.s`, `task2.s`, and ect. A `template.s` file is provided; you should copy it to each of the task files, to use it as the starting point.

Requirements

Add a comment to every line of your code. The comment must be based on array-level references, not instruction-level references. For instance, for the following instruction:

```
movl    (%rdx), %eax
```

a comment `"read in next array element"` is acceptable, while `"move data at address held in rdx to register eax"` is not. In this assignment's grading, comments carry the same weight as program code.

Hints

The first five tasks are relatively simple. The last two require some algorithmic design.

- For Task 6, you may want to search to the end of the array to find the address of the last array element; with that, you can swap the values of the first and the last elements. Update both addresses and repeat the process.
- For Task 7, you need to choose an in-place sorting algorithm, such as bubble sort or selection sort. Undertand how the algorithm works first, then implement it.

Testing

For testing purposes, a `main.c` program is provided that runs the assembly language code on three arrays and also shows the contents of the array before and after the call. The latter is particularly useful for Tasks 5 and 6 that modify the array but don't return a useful result.

You can compile an task program either using `gcc` directly, or with the provided `Makefile`:

```
linux> gcc -o taskN main.c taskN.s   or
linux> make taskN
```

and then run the program using the following line:

```
./taskN
```

Make sure you test your programs on the CS linuxlab system, where they will be graded.

Submission

Zip the task program files `task*.s` into a single zip file and upload it through the upload it through the "File Upload" tab in the "Assignment 3" folder. (You need to press the "Start Assignment" button to see the submission options.) *Important:* Keep a copy of your submission file in your folder, and do not touch it. In case there is a glitch in Canvas submission system, the time-stamp on this file will serve as a proof of your original submission time.