Assignment 2: mysort

Date: February 13th 2019
Deadline: February 20th 2019 23:59

Objectives

You must implement a list API and a number sorting program with several command-line options.

Requirements

Your sorting program must be named `mysort` and its basic operation is as follows:

- it reads integer numbers from its standard input until an `EOF` is encountered.
- it prints the same numbers in ascending order on its standard output, one number per line.

Numbers in the input can be separated by any type of whitespace, including newlines, but must be terminated by `EOF`. You can generated this manually when giving your program input [1] or you can use bash input redirection to send your program a whole file [2], which will always include an `EOF`.

Your program must perform sorting by maintaining a linked list in memory while reading the input and ensuring the list is sorted after all input has been read. It should implement a version of the insertion sort [3] algorithm to do this, but *some careful consideration of the problem might help you avoid implementing the complete algorithm on your linked list*.

Your program should take several command-line options that modify the sorted output in some way, the complete list of these options can be found under the **Grading** section. The code to parse the command-line arguments uses the `getopt` function and has already been provided in `main.c`. For the complete description of `getopt` you should consult the *man pages* using the command `man 3 getopt` in the terminal. Other man pages that will probably be useful for this assignment include `fgets` and `strtok`.

You must submit your work as a tarball. The command `make tarball` will create a a tarball called `insertion_sort_submit.tar.gz` containing the relevant files.

Getting started

1. Read all the function prototypes and descriptions in `list.h`. Decide on what type of *linked list* you will create and draw representations of your intended *list* and *node* structs on paper.
2. Implement the function prototypes from `list.h` in `list.c` in their given order and run `make check` after completing every function. If you think you have implemented enough functions to pass a test and the test still fails, fix the bug before moving on the the next set of functions.
3. Start with processing a simplified version of the input description in `main.c`, so only parse numbers separated by newlines, and don't consider multiple numbers on the same line yet. Sort the inputted numbers using your linked list implementation and print them in specified output format.
4. Add the code to parse the complete input description and combine this with your existing sorting code. Run `make check` and ensure that your code passes all sorting tests.
5. Reread all the functions descriptions in `list.h` and verify you implemented all these functions correctly. Carefully consider all edge case inputs for your functions and make sure they are handled by the implementation (i.e. they would not crash or break the program).
6. Start adding the extra option flags described in the grading section. If the flags were added, the related variable in the `config` struct will be set to `1` and you should modify the program output accordingly.

Grading

Your grade starts from 0, and the following tests determine your grade:

- +1pt if you have submitted an archive in the right format, your source code builds without errors and you have modified `list.c` and `main.c` in any way.
- +2pt if your list handles the basic operations `head`, `tail`, `next`, `prev`, `add_front`, `add_back`, `unlink` and `cleanup` correctly.
- +2pt if your `mysort` correctly processes any input meeting the decribed requirements and produces output in sorted order according to the specified format.
- -1pt if your code produces any warnings using the flags `-Wpedantic -Wall -Wextra` when compiling.

And the following features are not included in the provided tests at all. You will have to validate the correctness of these yourself by writing your own tests.

- +2pt if your list correctly implements all described functions in `list.h` and also handles invalid operations such as invalid inserts, invalid unlinks, etc. correctly.
- +0.5pt if your `mysort` can use the option `-u` which causes the program to ignore duplicate input values.
- +0.5pt if your `mysort` can use the option `-d` which causes the values in the list to be sorted in descending order.
- +1pt if your `mysort` can use the option `-i` which causes N-1 intermediate values to be computed and inserted into the sorted list when N values are given as input. The intermediate values should be halfway between two adjacent values, rounded to the nearest integer (with 0.5 being rounded up), and inserted between the original adjacent values in the list.
- +1pt if your `mysort` can use the option `-z` which causes the sorted list to be cut into two equal halves (with the first half being longer in case of odd length) and then zips the 2 halves back together starting with the first element of the first half and then alternating between elements from the second and first half until all elemens have been joined back into the single list.
- -1pt if enabling the `address sanitizer` [4] or running `valgrind` [5] reports errors while running your converter. *Note that you cannot test both of these at the same time, so disable ASAN in the Makefile when testing with valgrind.*

Your program should also be able to handle combinations of these options, which should be handled in the same order they are described here, e.g. *-i* always happens before *-z*. With any combination of options, the output should still be produced only once, after all the options have been applied, according to the specified output format.

---

1 https://askubuntu.com/questions/724990/what-is-eof-and-how-to-trigger-it
2 https://www.tldp.org/LDP/abs/html/io-redirection.html
3 https://en.wikipedia.org/wiki/Insertion_sort
4 https://github.com/google/sanitizers/wiki/AddressSanitizer
5 http://valgrind.org/docs/manual/quick-start.html#quick-start.intro