ECE368 Programming Assignment #2

Due Friday, February 17, 2017, 11:59pm

Description

This programming assignment is to be completed on your own. Similar to assignment 1, you will implement Shell sort. The major differences here are that

1. You will use a linked-list to store the long integers, instead of using an array for storage. Consequently, your sorting will have to be performed on a linked-list.

2. You can choose to use insertion sort, selection sort, or bubble sort as the basic sorting routine for sorting a linked list.

For this assignment, you will use the same sequence as in assignment 1 for Shell sort. For each *k* in the sequence, there are *k* "subarrays" to be sorted. If you choose to implement each subarray as a linked-list, your Shell sort implementation will have to sort *k* linked-lists.

In this assignment, you will use the following user-defined type to store integers:

```
typedef struct _Node {
   long value;
   struct _Node *next;
} Node;
```

You have to define this structure as provided above in `sorting.h`

You may also use the following user-defined type to store a linked-list of linked-lists. To be exact, the following structure can be used to implement a linked-list of addresses pointing to the `Node` structure.

```
typedef struct _List {
   Node *node;
   struct _List *next;
} List;
```

This structure is probably useful for you to maintain *k* linked-lists, where *k* is a number in your sequence.

Functions you will have to write:

All mentioned functions must reside in the program `sorting.c`. You should also declare these functions and define the structure `Node` as given above in `sorting.h`. It is up to you whether you want to define the structure `List` in `sorting.h`. Any help functions for these mentioned functions should also reside in `sorting.c` but not declared in `sorting.h`.

The first two functions `Load_From_File` and `Save_To_File`, are not for sorting, but are needed to transfer the integers to be sorted to and from files.

```
Node *Load_From_File(char *Filename)
```

```
int Save_To_File(char *Filename, Node *list)
```

The input and output files are of the same format as in assignment 1. `Filename` in each of the two functions is the address of the `char` array containing the name of the input or output file.

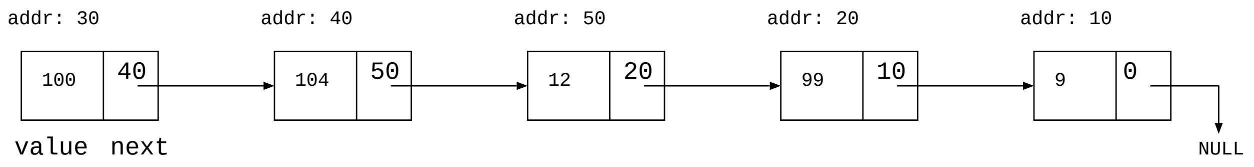
The load function should read all (`long`) integers in the input file into a linked-list and return the address pointing to the first node in the linked-list. The linked-list must contain as many `Nodes` as the number of `long` integers in the file.

The save function should write all (`long`) integers in a linked-list into the output file. This function returns the number of integers successfully written into the file.

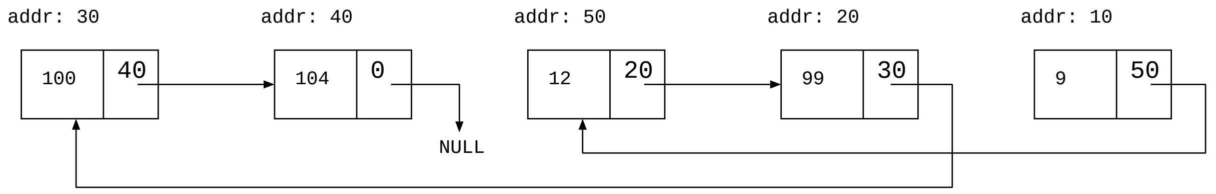
```
Node *Shell_Sort(Node *list)
```

This function takes in a `list` of long integers and sort them. To correctly apply Shell sort, you would have to know the number of elements in the `list` and find the sequence accordingly. The address pointing to the first node of the sorted list is returned by the function.

(a) Original list



(b) Sorting by manipulating addresses of Nodes



The `Shell_Sort` function must perform sorting by manipulating the `next` fields of the `Nodes`. Figure (a) shows an original list that is unsorted. Figure (b) shows how the list is sorted by storing the correct addresses in the `next` fields. The `long` integers stored in the `value` fields remain in the original `Nodes`. For example, the integer `99` is stored in a `Node` with an address `20` in the original list. The `field` of the same `Node` stores the address `10`, allowing it to point to the `Node` storing the value `9`.

After sorting, `99` is still stored in the `value` field of the `Node` with address `20`. However, the `next` field of the `Node` now stores `30`, allowing it to point to the `Node` storing the value `100`.

In other words, each `long` integer must reside in the same `Node` in the original list before and after sorting.

There should not be a need to use an array in the `sorting.c` file. If you have used an array to store the sequence generated in an array, you would now have to store that sequence in a linked-list. Any attempt to use an array in `sorting.c` file will result in a zero credit for this assignment.

`main` function you will have to write:

You have to write another file called `sorting_main.c` that would contain the `main` function to invoke the functions in `sorting.c`. You should be able to compile `sorting_main.c` and `sorting.c` with the following command (with an optimization flag -O3):

```
gcc -Werror -Wall -Wshadow -O3 sorting.c sorting_main.c -o proj2
```

When the following command is issued,

```
./proj2 input.b output.b
```

the program should load from `input.b` the integers to be sorted and store them into a linked-list, run Shell sort on the linked-list, and save the sorted integers in `output.b`. The program should also print to the standard output (i.e., a screen dump), the following information:

```
I/O time:  AAAA
Sorting time:  BBBB
```

where `AAAA` and `BBBB`, all in `%le` format, report the statistics you have collected in your program.

Although this assignment does not ask you to count the numbers of data comparisons, it is recommended that you keep them initially so that you can compare your new code against your code from assignment 1.

You may declare and define other help functions in `sorting.c` and `sorting_main.c`. However, these help functions should not be declared in `sorting.h`. It is best that these helper functions be declared as `static`. Do not name these help functions with a prefix of two underscores "__".

Report you will have to write:

You should write a (brief) report that contains the following items:

- A tabulation of the I/O run-times and sorting run-times obtained from running your code on some sample input files. You should comment on how the I/O run-times and sorting run-times grow as the problem size increases, i.e., the time complexity of your routines.
- The space complexity of your program. Do not include the space required by the linked-list to store the integers (and the addresses). However, if you have to create a linked-list of lists (or a linked list of addresses), you would have to account for the additional space for that linked list of lists (or pointers). If you have to create a linked-list to store the sequence, you also have to account for that additional space.
- A comparison of the I/O run-times and sorting run-times obtained in this assignment and in assignment 1 (for insertion sort only). Explain any significant differences that you observe.

Each report should not be longer than 1 page and should be in PDF, or plain text format (using the textbox in the submission window.) The report will account for 10% of the overall grade of this assignment.

Submission:

The assignments requires the submission (electronically) through Blackboard of the C-code `sorting.c` and `sorting_main.c` and the header file `sorting.h`. You also have to submit the report through blackboard. You should create a zip file called proj2.zip that contains all of the above and submit the zip file. If the zip file contains a `makefile`, we will use that file to make your executable.

Grading:

The grade depends on the correctness of your program, the efficiency of your program, the clarity of your program documentation and report.

The sorting function will account for at least 50the entire grade. The other functions and the report will account for the remainder of the grade.

It is important all the files that have been opened are closed and all the memory that have been allocated are freed before the program exits. Memory leak will result in at least 50% penalty.

Given:

This is the only file you are provided for this assignment. You may use the input files provided in assignment 1 to evaluate your implementation.

Continue sorting!