Assignment 1

CS/ECE 354 - Spring 2016

1. Collaboration Policy

For this assignment, you may work in pairs (2 people). All students (whether working in a pair or not) must individually turn in the assignment. Therefore, 2 copies of the assignment will be turned in for those working as a pair. The grader will choose to grade only one copy of a pair's work, and assign the same score to both students.

If one member of a pair does something unexpected with respect to turning in the assignment (for example: bad copy of the source code, late, or perhaps not turning in portions of the assignment), then *both* of the pair get the same penalty and the same score.

If working in a pair, the names and sections of *both* students must appear as comments at the top of all files of the turned in assignment.

In addition, students who work in a pair must turn in an extra file that identifies the pair. Details are given in the section on Handing In the Assignment.

2. Learning Goals

The purpose of this assignment is to quickly become good at writing C programs, gaining the experience of working in a non-object oriented language. By the end of this assignment you should be comfortable with arrays, command-line arguments, file I/O, pointers, and structures in C. Additionally you will learn how to implement a linked list and some of its basic operations such as list traversal, node addition, and sorting a linked list.

3. Specifications

In this assignment, you will write a C program named `list.c` that reads a list of integers from an input file, stores them in an array and linked list, then builds a sorted linked list and then writes the sorted output to a new file. Sections 3.2, 3.3 and 3.4 specify how you should write up the main method for this program and complete any functions outlined.

3.1. Generate integers and store it in a file

To begin with, you are given the program `generate.c`, which generates a random set of integers and prints out every integer in its own line to the console (stdout). The program takes two command line arguments:

1. `<total_nums>:` The number of integers to be generated
2. `<max_num>:` The upper bound on the highest value

To use this program to generate the input file for **`list.c`** do the following:

1. Create a folder named **`p1`** in your **`private/354`** directory that you created for assignment 0.
2. Copy the `generate.c` file into your p1 directory. The file is located in the directory `/p/course/cs354-common/public/spring16/code/p1`
3. Compile the program and store the executable in a file named `generate`.
4. Run the program using the following command:

   ```
   ./generate 20 100
   ```

   For the purpose of this program we will keep the number of integers(20 in this case) below 1000 which is defined as a symbolic constant `MAX_INTS` in `list.c`.
5. The program should have printed out a list of integers to stdout, each on its own line.
6. Alter the shell command such that the output sent to stdout is instead captured in a file. This is called a shell redirect.

   ```
   ./generate 20 100 > numbers.txt
   ```
7. The program should have created a file named `numbers.txt`, which contains a list of integers written as ASCII characters.
8. To try to see the results, you might try the shell command:

   ```
   cat numbers.txt
   ```

```
[gerald@francisco] (211)$ ls
generate*  generate.c  list.c
[gerald@francisco] (212)$ ./generate 20 100 > numbers.txt
[gerald@francisco] (213)$ cat numbers.txt
66
40
81
41
12
58
21
40
35
43
74
43
17
4
96
62
92
48
98
59
[gerald@francisco] (214)$
```

Figure 1: Output of generate.c

3.2. Read the integers from the file and build a linked list

For this and the next parts of the assignment you have been provided with the file `list.c` which contains skeleton code for the functions that you need to complete to make the program to work as expected. Copy the `list.c` file into your p1 directory. The file is located in the same directory as `generate.c`.

The program takes in a single command line argument `<input_file>` which is the name of the file containing the list of integers to sort.

1. Open the file whose name is given as the command line argument.

2. Read the integers one at a time from the file, and place each integer in an array of size `MAX_INTS`.
3. Print the array using the `print_array` function.
4. Create a linked list from the array using the `create_list` function, and print the list using the `print_list` function.

To create the list you will essentially need to complete the following functions:

- `struct node* create_list(int intarray[], int len)`
    - This function takes in an integer array and its size, traverses the array while using the function `add_item_at_start(struct node *head, int data)` to build a list. This function returns the head of the new linked list that was created.
- `struct node* add_item_at_start(struct node *head, int data)`
    - This function takes in a pointer to the head of the linked list, adds a new node with the data at the beginning of the list, updates and returns the new head.

NOTE:
A pointer to the first node in the list identifies where the list is. It is called the head of the list. If the head has the value NULL, then the list is empty. The structure of a node has already been provided for you in the code skeleton.

3.3. Search for integers

Next we will have the user input a number for which we will search both the array and the linked list to confirm its index and position in both of them.

1. Prompt the user for a number to search (as shown in Figure 2) and read the input from the keyboard (stdin).
2. Search for the number in the array to find its index. This can be done by completing the `search_array(int integers[], int numints, int element)` function. This function returns the index of the `element` if the `element` is found in the array, otherwise it returns -1. The array index is zero based, and its value goes from 0 to n-1, where n is the number of elements in the array. The first element in the array is at index 0 (zero).

3. Print the result of the search and the index of the number if it was found in the array. See Figure 2 for the output format.
4. Search the number in the linked list to find its position. This can be done by completing the `search_list(struct node *head, int element)` function. It returns the position of `element` if found in the list, otherwise -1. The position value ranges from 1 to n. The first node in the linked list is at position 1 (one).
5. Print out the result of the search and the position of the number if it was found in the linked list.
6. Repeat steps 1-5 until the user enters the character 'q', in which case stop prompting the user for an input and move on. You can assume that the user will only enter a valid integer or the character 'q'.

3.4. Sort the linked list and write the output to a file

Now we will sort the numbers by constructing a new list. Be careful not to destroy the original list that we created in section 3.2

1. Create a new sorted list by calling
   `create_sorted_list(struct node *head)`
   which takes in as input the head of our original list, constructs a new sorted list(ascending order), and returns the head of the sorted list. This function should use
   `add_item_sorted(struct node *sorted_head, int data)`
   The way this works is that for every node in the old list, you insert it into the new list in its correct position based on the sort order. This is also called insertion sort. For example, in the list shown in Figure 2, when we create the sorted linked list we would insert the first node with value 59 from the original linked list to the sorted linked list. Next, when we try to insert the second node from the original linked list (with a value of 98) into the sorted linked list, we need to make sure that this node with value 98 is inserted after the node with value 59. So, if you try to print the sorted linked list after inserting two nodes in the sorted order it would look like as shown below:

   SORTED LINKED LIST: head → |59| → |98| → NULL

   Print the sorted list using the `print_list` function.
2. Copy the sorted list to a new array using the function
   `copy_list_to_array(struct node *head, int *array)` which takes in a pointer to the head to a list and a pointer to the start of an array, and returns the number of integers copied to the array. Please make sure that you don't overwrite the contents of your original unsorted array.

3. Print the sorted array using the `print_array` function.
4. Print the original linked list again using the `print_list` function.
5. Print the original array again using the `print_array` function.
6. Open a new file named "`sorted_numbers.txt`".
7. Write the sorted integers from the sorted array to this file, line by line with each integer on its own line.
8. Close the file and print out the number of integers written to the file.

```
[gerald@francisco] (197)$ ls
generate*  generate.c  list*  list.c  numbers.txt
[gerald@francisco] (198)$ ./list numbers.txt
Reading integers from a file to an array...

ARRAY: | 66 | 40 | 81 | 41 | 12 | 58 | 21 | 40 | 35 | 43 | 74 | 43 | 17 | 4 | 96 | 62 | 92 | 48 | 98 | 59 |

LINKED LIST: head->|59|->|98|->|48|->|92|->|62|->|96|->|4|->|17|->|43|->|74|->|43|->|35|->|40|->|21|->|58|->|12|->|41|->|81|->|40|->|66|->NULL

Enter an element to be searched in the list and array:33

Element 33 not found in the array.

Element 33 not found in the linked list.

Enter an element to be searched in the list and array:12

Element 12 found in the array at index 4.

Element 12 found in the linked list at position 16.

Enter an element to be searched in the list and array:q

SORTED LINKED LIST: head->|4|->|12|->|17|->|21|->|35|->|40|->|40|->|41|->|43|->|43|->|48|->|58|->|59|->|62|->|66|->|74|->|81|->|92|->|96|->|98|->NULL

SORTED ARRAY: | 4 | 12 | 17 | 21 | 35 | 40 | 40 | 41 | 43 | 43 | 48 | 58 | 59 | 62 | 66 | 74 | 81 | 92 | 96 | 98 |

ORIGINAL LINKED LIST: head->|59|->|98|->|48|->|92|->|62|->|96|->|4|->|17|->|43|->|74|->|43|->|35|->|40|->|21|->|58|->|12|->|41|->|81|->|40|->|66|->NULL

ORIGINAL ARRAY: | 66 | 40 | 81 | 41 | 12 | 58 | 21 | 40 | 35 | 43 | 74 | 43 | 17 | 4 | 96 | 62 | 92 | 48 | 98 | 59 |

Writing integers from a sorted array to a file...

Number of integers written to the file = 20.
[gerald@francisco] (199)$ ls
generate*  generate.c  list*  list.c  numbers.txt  sorted_numbers.txt
```

Figure 2: Output of list.c

NOTE:

1. The final output should look similar to Figure 2.
2. The contents of the file `sorted_numbers.txt` should be as shown in Figure 3.

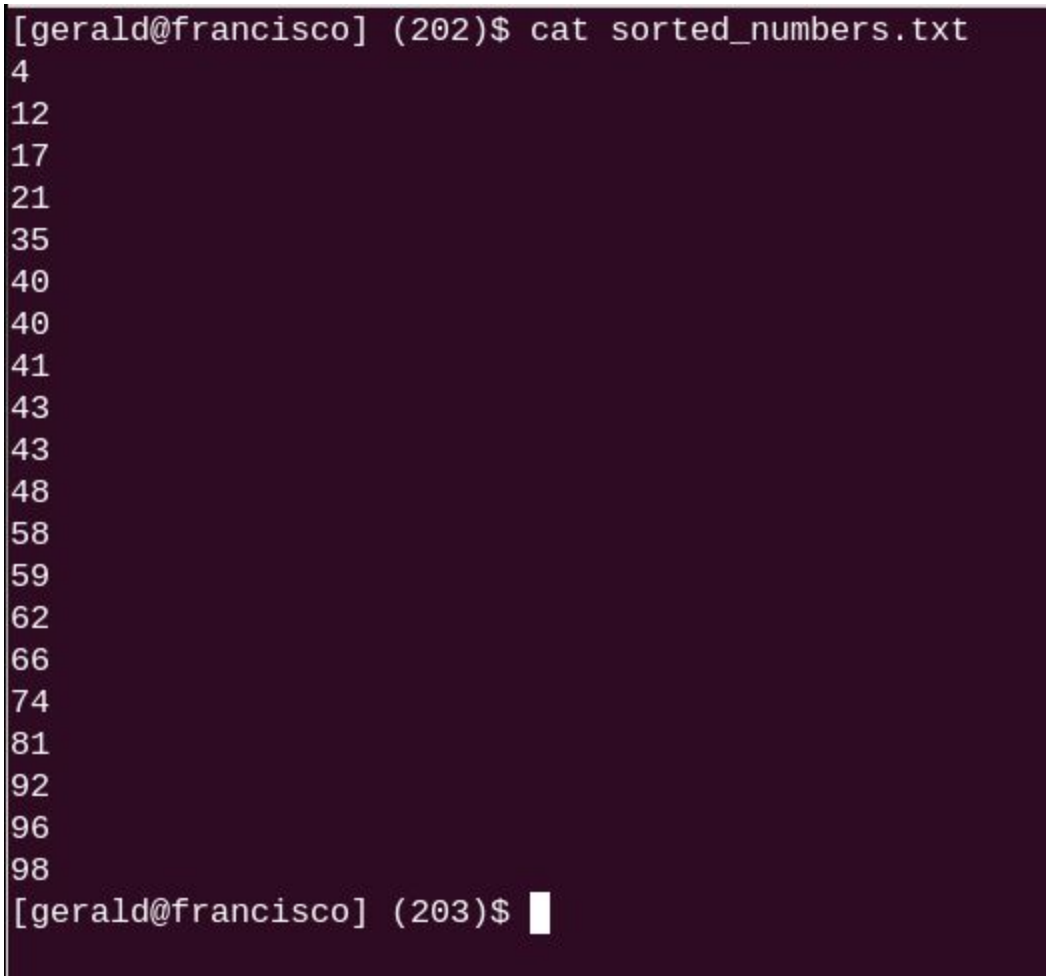

Figure 3: Contents of sorted_numbers.txt

4. Error Handling

- If the user invokes the `list` program incorrectly (for example, without an argument, or with two or more arguments), the program should print an error message and call

`exit(1)` as shown below.

```
[gerald@francisco] (218)$ ./list
Usage: ./list <input_file>
[gerald@francisco] (219)$ ./list numbers.txt randomfile
Usage: ./list <input_file>
[gerald@francisco] (220)$
```

- Be sure to always check the return value of library functions. For example, if a file cannot be opened, then the program should not read input. Instead it should print an error message as shown below. **Points will be deducted for forgetting to check return values from library functions**.

```
[gerald@francisco] (228)$ ls
generate*  generate.c  list*  list.c  numbers.txt
[gerald@francisco] (229)$ ./list random_numbers.txt
ERROR: Cannot open file for reading!
[gerald@francisco] (230)$
```

- These guidelines will apply to later programs as well!

5. Notes and Hints

- Using library functions is something you will do a lot when writing programs in C. Each library function is fully specified in a manual page. The `man` command is very useful for learning the parameters a library function takes, its return value, detailed description, etc. For example, to view the manual page for `fopen`, you would issue the command "`man fopen`". If you are having trouble using man, the same manual pages are also available online. You will use these library functions to write this program:
    - `fopen()` to open the file.
    - `malloc()` to allocate memory for a new linked list node.
    - `fgets()` to read each input from a file. fgets can be used to read input from the console as well, in which case the file is stdin, which does not need to be opened or closed. An issue you need to consider is the size of a the buffer. Choose a buffer that is reasonably large enough for the input.
    - `fclose()` to close the file when done.
    - `printf()` to display results to the screen. To view the manual page, use man 3 printf on a **galapagos** machine.

    - `fprintf()` to write the integers to a file.
    - `aoti()` to convert the input which is read in as a C string into an integer.
- When opening a file for reading using `fopen`, make sure you specify the correct mode (read or write) for which you are opening the file.
- Remember to #include the corresponding header files needed for all these library functions.

6. Requirements

1. Your program must follow style guidelines as given in Style Guidelines. The guide is from CS302 for Java, but most if it is applicable for C as well. The relevant sections are braces, whitespace and indentation.
2. Include a comment at the top of each source code file with your **name and section**. You must comment every function with a header comment. See the Commenting Guide, where applicable for C.
3. Your programs should operate exactly as the sample outputs shown above.
4. Use a Linux machine for this assignment!
5. We will compile each of your programs with

   ```
   gcc -Wall -m32 -std=gnu99
   ```

   on a galapagos Linux machine. So, your programs must compile there, and **without warnings or errors**. It is your responsibility to ensure that your programs compile on the department Linux machines, and **points will be deducted** for any warnings or errors.
6. Remember to do error handling in all your programs. See the instructions on error handling for more details.

7. Handing in the Assignment

Copy the file `list.c` into your handin directory:

`/p/course/cs354-common/public/spring16.handin/<yourloginID>/p1`

where `<yourloginID>` is the username of your CS account.

If you are working as part of a pair, you must turn in an extra file. This file will contain the names and sections of **both** students in the pair. As an example, if Lokesh worked with Urmish on this assignment, the contents of the extra file for both Lokesh and Urmish would be

Lokesh Jindal section 1
Urmish Thakker section 2

The name of this file is specialized to help the 354 automated grading tools identify who worked together. This file name is composed of the CS logins of the partners separated by a period. The file name is of the form `<login1>.<login2>`. Lokesh's login is `lokesh`, and Urmish's login is `uthakker`. The file name that both use will be `lokesh.uthakker` Please have both partners use the same file name. It does not matter which partner's name is first within this file name.

8. About Copying Code and Academic Misconduct

Don't cheat. Read this link carefully.
*Do **not** post your assignment solutions (or drafts) on any publicly accessible web sites. This specifically includes GitHub. It is academic misconduct to post your solution.*
For almost any C program that does something useful, someone has already written this program and further, has posted it for others to use. These programs do not do much that is useful, and are not likely posted anywhere. Still, it is academic misconduct for you to copy or use some or all of a program that has been written by someone else.
The penalty for academic misconduct on this assignment (and all CS/ECE 354 assignments) will be a failing grade in the course. This penalty is significantly more harsh than if you simply do not do the assignment. You will gain much more by doing the assignment than by copying, possibly modifying, and turning in someone else's effort.

Good luck with the assignment!