CSE3080-3: Data Structures (Spring 2020)
Midterm Project 1: Master of Linked Lists

Handed out: May 7, Due: May 18, 11:59PM (KST)

1. Problem Description

In this homework, we will practice handling a doubly linked list. We will write a program that reads a list of commands from an input file and execute them, while maintaining a double linked list as the core data structure. The doubly linked list must be managed so that the data of the nodes (integers) are sorted in the ascending order. Also, duplicate data is not allowed in the list.

There are four commands that need to be supported by the program:

(1) insert

- This command inserts an integer into the linked list. If the line reads “INSERT 3”, the program should insert a new linked list node with data = 3 to the list.

- If the linked list already has a node with data = 3, then we should not insert the new node because duplicates are not allowed.

- If a node with data = 3 does not exist in the list, then the new node should be inserted at the proper place within the list, so that the data is maintained sorted in the ascending order.

(2) delete

- This command deletes a node from the linked list. If the line reads “DELETE 3”, the program should search the list for a node with data = 3.

- If the node is found, the node should be deleted from the list.

- If the node is not found, nothing needs to be done.

(3) print data in ascending order

- This command prints the contents of the linked list in the forward direction. If the line reads “ASCEND”, then the program should call function **print_forward**. The implementation of function **print_forward** is given.

(4) print data in descending order

- This command prints the contents of the linked list in the reverse direction. If the line reads “DESCEND”, then the program should call function **print_reverse**. The implementation of function **print_reverse** is given.

2. Your task and requirements (Read Carefully!)

(1) You should write a single C program, named **mp1**.

(2) The program takes one command-line argument, which is the input file name. For example, the user will run the program like this:

```
./mp1 input.txt
```

The first argument is “input.txt”, which is the name of the input file.

(3) If the number of command-line arguments is not 1 (zero or more than 1), then you should print a usage string on the display and exit the program. The usage string looks like this:

```
usage: ./mp1 input_filename
```

(4) If the input file does not exist, your program should display an error message on the screen and terminate. The error message should be

```
The input file does not exist.
```

(5) The input file format is like the following. Each line contains one command.

```
INSERT 5
INSERT 8
INSERT 3
ASCEND
INSERT 10
DELETE 5
DESCEND
```

An example input file will be given for your reference. At the evaluation, the TA will use a different input file. You may assume that the format of the input file is always correct; You do not need error handling for the contents of the input file.

(6) [IMPORTANT] If the command is ASCEND, you must do nothing but call the function print_forward. This function is given to you (at the end of this document), and you must include this function in your program without modification. Your other parts of the code should be written so that this function print_forward compiles well and works well.

(7) [IMPORTANT] Similarly, if the command is DESCEND you must do nothing but call the function print_reverse. This function is given to you (at the end of this document), and you must include this function in your program without modification. Your other parts of the code should be written so that this function print_forward compiles well and works well.

(8) Similar to the other homeworks, you should write a Makefile. The TA will build your code by running "make". It should create the binary file, "mp1".

※ Test-run the given binary code and write your program so that it produces the same result as the given binary code.

3. Submission

You should submit your source codes and the Makefile. (You do not need to submit compiled binary files.) Combine your files into a zip file named cse3080_mp1_**20190001**.zip. The red numbers should be changed to **your student ID**. Submit your files on the cyber campus.

4. Evaluation Criteria

(1) Your program should compile well to produce the binary file.

(2) If the user does not give correct number of arguments, your program should print the usage message on the display and terminate.

(3) If the input file does not exist, your program should correctly print an error message on the display and terminate.

(4) When the user correctly executes the program with a command-line argument, your program should produce the result file "mp1_result.txt".

(5) The contents of your output file should be the same as the output file created from running the given binary file.

5. Notes

- You must write your own code. You may discuss ideas with other students but must not copy their work. Similarly, you can find ideas on the Internet, but must not copy the code from the sources obtained from the Internet.

- Our department uses a software that detects duplicate codes. Since the software uses an assembly code level detection, just changing the variable names will not bypass the software. Make sure you write your own code from the scratch. Then, you will have no problem.

- Explained earlier, but your program should compile and run on the **cspro** machine. Make sure you check before you submit your work. The TA will download your code, run “make” and execute your binary files to check if they produce correct outputs.

6. Late Policy

10% of the score is deducted for each day, up to **three days**. Submissions are accepted up to three days after the deadline.

Appendix

Function `print_forward`.

```
void print_forward(listPointer list) {
  listPointer curr;
  FILE *outfile;

  outfile = fopen("mp1_result.txt", "a");
  if(list) {
    curr = list;
    while(1) {
      fprintf(outfile, "%d ", curr->data);
      printf("%d ", curr->data);
      curr = curr->next;
      if(curr == list) break;
    }
  }
  fprintf(outfile, "\n");
  printf("\n");
  fclose(outfile);
}
```

Function `print_reverse`.

```
void print_reverse(listPointer list) {

  listPointer curr;
  FILE *outfile;

  outfile = fopen("mp1_result.txt", "a");
  if(list) {
    curr = list->prev;

    while(curr != list) {
      fprintf(outfile, "%d ", curr->data);
      printf("%d ", curr->data);
      curr = curr->prev;
    }
    fprintf(outfile, "%d ", curr->data);
    printf("%d ", curr->data);
  }
  fprintf(outfile, "\n");
  printf("\n");
  fclose(outfile);
}
```