INF1002 Programming Fundamentals – C Lab4

OBJECTIVES:

1. To understand and use dynamic memory allocation and linked lists.

EXERCISE-1: USER-DEFINED DATA TYPES

Use typedef to create each of the following data types, using macros and struct where appropriate:

a) A type, *INTL_MONEY_VALUE*, which can store a floating-point number to represent a money value plus a 3-character string (e.g. SGD, USD, etc) to represent its currency.
b) A type, *INTL_MONEY_VALUE_PTR*, being a pointer to the international money value type above.

TEST YOUR TYPES BY PUTTING THEM INTO A PROGRAM, DECLARING A FEW VARIABLES OF EACH TYPE, AND CHECKING THAT IT COMPILES.

EXERCISE-2: LINKED LISTS

Write a program that creates and manipulates a list of students according to the steps below. You may start with the following type declarations:

```
struct grade_node {
      char surname[20];
      double grade;
      struct grade_node *next;
};
typedef struct grade_node GRADE_NODE;
typedef GRADE_NODE *GRADE_NODE_PTR;
```

All the following manipulations may be done in *main()*.

1. Declare a pointer called head that will point to the start of the list. The list begins without any elements, so its initial value will be NULL.
2. Create a new node and fill it with the following data:
    - surname: “Adams”
    - grade: 85.0

   Place this node at the start of the list.
3. Create another node and fill it with the following data:
    - surname: “Pritchard”
    - grade: 66.5

   Place this node at the end of the list.
4. Create one more node and fill it with the following data:
    - surname: “Jones”
    - grade: 91.5

   Place this node between the “Adams” node and the “Pritchard” node. (The list should be in alphabetical order.)

Lab Assignment:

To help you better practice, you need to perform a set of tasks in one auto-grading system, Gradescope in LMS/xSiTe. Gradescope will provide you immediate feedback of your program, such that you will know the issue of your program.

In this lab, you need to finish one task in Gradescope system. Gradescope system will provide you immediate feedback about the correctness of your program. You can view the feedback of your system by clicking the failed case. Gradescope adopts a test cases-based approach to check your program. Your score is depending on the number of test cases that you program can pass. Note that to train you to have a good programming practice, you must write your program strictly according to the requirement of the tasks, including your input and output format. If there is any difference (even one space), your program will fail on the test cases. SO, TRAIN YOURSELF TO BE AN EXACT THINKER!

To help you practice, you are allowed to do multiple attempts for each task. Enjoy your learning!

Task: INSERTION SORT

Task Description:

One way to sort a collection of items is called insertion sort. The idea is to start with an empty list, then insert items one at a time into it, placing them in their correct position in the order each time. Your task is to implement this algorithm using a linked list in a program *insertionSort*, so that the program can sort an arbitrary number of words entered by the user.

Every node in the list should store one word, composed entirely of lower-case characters. The word may also contain apostrophes and hyphens, but not spaces, quotes, or any other characters that do not normally appear in the middle of English words. The word may be up to 32 characters long.

The program should repeatedly ask the user to enter a word. The program should automatically convert upper-case letters into lower-case ones, but reject words containing characters other than letters, apostrophes, and hyphens.

Each new word should be inserted into the list into its correct position in alphabetical order. For example, if the list currently contains the words "cat", "dog", and "monkey", and the user enters the word "elephant", the new word should be inserted between "dog" and "monkey". You can use *strcmp()* to determine whether a word comes before or after another in the alphabet (hyphens and apostrophes will be sorted according to their ASCII values).

The program stops asking for words when the user enters the special text "***". The program should then print out the words, in order, one per line.

Finally, the program should de-allocate all the memory that is has created and terminate.
Note:

1. Use *#define* and comments as usual.
2. Check for memory allocation failures and report an error if they occur but continue to execute the program.
3. There is no white space in the print after the colons (:).

4. Don't use *sys.argv[]* for user inputs. Use other functions such as *scanf()*, *fgets()*, *fgetc()*, etc.,

Some sample output is shown below, with the user input shown in red:

Example – 1:

```
Please enter a word:
cat
Please enter a word:
dog
Please enter a word:
monkey
Please enter a word:
elephant
Please enter a word:
***
All the entered words in order:
cat
dog
elephant
monkey
```

Example – 2:

```
Please enter a word:
hello
Please enter a word:
good-bye
Please enter a word:
it's
Please enter a word:
invalid word
Invalid word
Please enter a word:
valid
Please enter a word:
"quote"
Invalid word.
Please enter a word:
another
Please enter a word:
***
All the entered words in order:
another
good-bye
hello
it's
valid
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file “`insertionSort.c`” from the “C\Labs\Lab4” xSiTe folder.
2. Do not change file name or function names.
3. Add your code to the function “*main()*” and other required functions.
4. Don’t use `sys.argv[]` for user inputs. Use other functions such as *scanf(), fgets(), fgetc(),* etc.,
5. Drag and drop your locally tested code to Gradescope. You can submit and auto graded any number of times. Last submission counts.
6. There are 5 test cases and maximum score for this task is 5.
7. Autograder will be timed out after 10 minutes.