ICT1002 – LAB – WEEK 11

Dynamic Memory Allocation

1 OBJECTIVES

1. To understand and use dynamic memory allocation with `malloc()` and `free()`.
2. To understand and use linked lists.

2 EXERCISES FOR WEEK 11 LAB

WEEK_11_LAB_EXE_1: 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 of 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.)

WEEK_11_LAB_EXE_2: MEMORY LEAKS

Each of the following fragments of code contains a memory leak. Identify the leak and correct it by inserting appropriate calls to `free()`.

```
a)
   int *p = (int *)malloc(sizeof(int) * N);
   /* do something */

b)
   GRADE_NODE_PTR g1 = (GRADE_NODE_PTR)malloc(sizeof(GRADE_NODE));
   GRADE_NODE_PTR g2 = (GRADE_NODE_PTR)malloc(sizeof(GRADE_NODE));
   g1->next = g2;
   g2->next = NULL;

   /* do something */

   free(g1);
```

3. WEEK_11_LAB_ASSIGNMENT

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, so that a 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 of the memory that is has created and terminate.

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

```
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
```

Note the following:

- Use `#define` and comments as usual.
- Check for memory allocation failures and report an error if they occur, but continue to execute the program.