Lebanese University
Faculty of Sciences 1

I2204 E – INFO216 E

Imperative Programming

Final 2019-2020
Wed. Feb. 12th am
Duration: 2 hours

Exercise I: Swapping Nodes in a DLL (25 pts)

Write the function **swap** which, given a doubly linked list (DLL) and two integer indices, swaps the nodes at those indices. The function should swap the nodes and not the data contained in the nodes because the nodes may be pointed to by other pointers. Therefore, we want to manipulate the pointers in the linked list in order to rearrange the nodes. In case the given indices were out of bound, no swap should occur. Moreover, care should be taken, when given indices are consecutive numbers, in order not to make loops inside the DLL. We give below an example of application of the swap function. We recall the type:

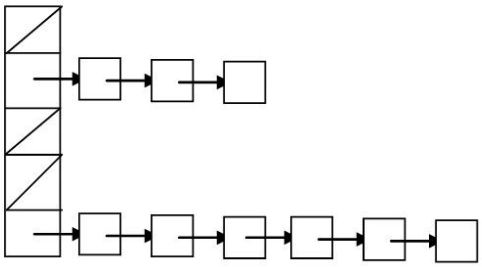
```
typedef struct node{
    int data;
    struct node *prev, *next;
}node;
```

Example of application:

```
List: 10 ⇌ 11 ⇌ 12 ⇌ 13 ⇌ 14
//swap(&head, 2, 1);
List: 10 ⇌ 12 ⇌ 11 ⇌ 13 ⇌ 14
//swap(&head, 0, 3);
List: 13 ⇌ 12 ⇌ 11 ⇌ 10 ⇌ 14
```

Exercise II: Hashtables (25 pts = 10 +15)

Arrays, by opposition to linked lists, have the advantage of the quick access to the elements and the disadvantage of the expensive structure alteration (insert and delete operations). Hashtables take the best from each world. Hashtables are arrays of linked lists called buckets. Below is an example of a hash table, and the data types used.



a hashtable

```
#define size 5

typedef struct element{ //...
} element;

typedef struct node{
    element data;
    struct node *next;
} node;

typedef struct hashtable{
    node* array[size];
} hashtable;
```

used data types

1- Write the **avgBucketLen** function which calculates the average number of elements in the non-empty buckets of the hash table. Example: considering the hash table in the figure above; we have five buckets, two of which are non-empty containing nine elements in total, hence the average bucket length is 4.5.

In order to add a given element to a hashtable, first calculate the element's hash code by calling the hash() function. Then, calculate the code modulo the array size to obtain an entry index. Finally push the element at the head of the linked list located at that index.

```
int hash(element elt);
```

2- Suppose already defined the function hash(). Write the function **add** which adds a given element in a given hashtable as described.

Exercise III: File & String Manipulation (20 pts)

A simple way to store an address book or telephone directory is in a text file with one line per entry. We have a file containing a collection of names, offices, and phone numbers. Here are a few sample lines from the file testfile.txt.

testfile.txt
```
Ima EE Professor#EE B 735#3-1415
Fearless Leader#EE1 116#3-6515
E. Fudd#EEGAD 548#2-4784
B. Bunny#EE 037#254-5512
```

Each line contains the name of an individual, the office location (building and room number), and phone number. There is a single hash (#) character between consecutive fields.
For some reason, the EE building has been denoted in several ways over the years: 'EE', 'EE B' (with one space between 'EE' and 'B'), 'EE1', 'EE 1' (with a space), and 'EEB'. We would like to produce a clean copy of this file by replacing the five names of the EE building with 'EEB'. For example, when run on the above input, the output shoud should be the following:

standard output
```
Ima EE Professor#EEB 735#3-1415
Fearless Leader#EEB 116#3-6515
E. Fudd#EEGAD 548#2-4784
B. Bunny#EEB 037#254-5512
```

For this problem, write in C the function **normalize** which, given the name of a file in the described format, copies its content to the standard output, changing all references to the EE building to 'EEB' as described above. Otherwise, the function must not modify the content.
You may assume the data is clean: each line contains exactly three fields separated by '#' characters, there are no extra spaces before or after each '#' character, there is one space between the building name and room number, and so forth.

Good Luck!