Experiment
Linked List

Purpose: The objective of this experiment is to demonstrate how linked lists are utilized in computer programs in order to organize program data.

Introduction

Data structures are arguably the most important subject that you will learn throughout this lecture. Without any exaggeration, the entire memory management system is built upon these structures. A very important member of data structures is the linked list. Linked lists allow dynamic allocation of memory blocks by chaining the blocks with pointers. In this manner, memory can be allocated in distinct physical locations on hardware, giving(most of the times to the operating system) the ability to use all the storage capacity in a flexible way. The ”malloc” is a common function that you might be familiar with that uses linked list structures behind the curtains to dynamically allocate data on your computer RAM. A linked list is composed of data blocks called nodes, where each of these nodes consists a pointer called a link to the next element in the list. Inserting a new node in a linked list is as easy as changing the link pointer of the head node to point out to the new node. You can see how the linear linked list structure is organized from the figure below.

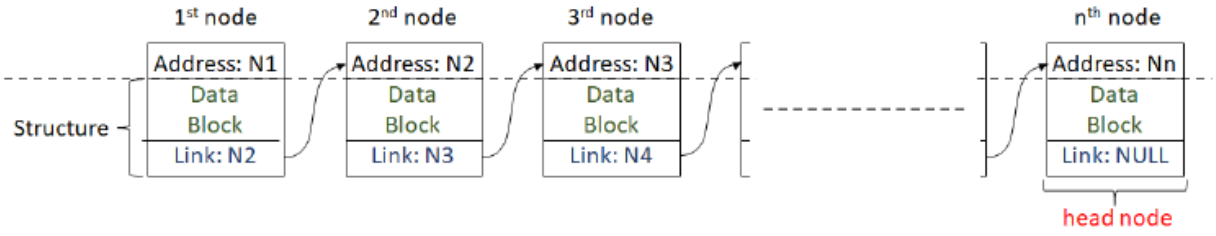

Figure 1: Graphical representation of the nodes in a linear linked list

Keep in mind that the main drawback of a linked list is that you cannot randomly access an intermediate node. You have to pass over the links starting from the head of the linked list to access the intermediate node of your choice. You can think of a linked list as an elevator in a tall building. Imagine that you need to take a package from the sixth floor and you have just arrived and entered the building. In this case, you would take the elevator and pass through the first five floors to reach the sixth one to get your package. In this analogy, the nodes are the floors of the building with the head node being the first floor, the data in the node is the package that you are trying to reach, and the elevator is the linked list that takes you through the nodes.

A circular linked list has the same structure as a linear linked list except that the “last” node contains a pointer back to the “first” node as shown below.

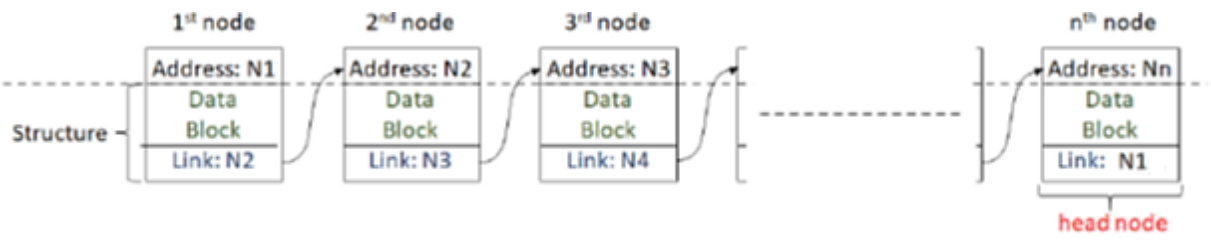

Figure 2: Graphical representation of the nodes in a circular linked list

Problem Statement

Assume that you want to create **two** linear letter lists where the first one is the list of the uppercase letters and the second one is the list of the lowercase letters.

Uppercase	Lowercase		Combined
A	a		A
B	b		a
C	c		B
D	x	→	b
E	y		C
F	z		c

The order of the same letters in uppercase and lowercase lists can be equal as shown above or not equal as shown below.

Uppercase	Lowercase		Combined
B	c		B
D	z		b
F	y		A
A	b	→	a
E	a		C
C	x		c

After creating linear uppercase and lowercase lists, by **only** using them, create **another** linear letter list that contains the **combined** letters. For an uppercase letter in uppercase list find its lowercase version, if any, in lowercase list. Keep finding process for each uppercase letter and adding the uppercase letter from uppercase list and the found lowercase version from lowercase list to your combined list, until the last element of the uppercase list. Combined list **should** be in the order of the uppercase letters list.

You need to write a program that create linear linked lists to lowercase, uppercase and combined letters. Assume that the lowercase and uppercase letters are available in main function (See Section 6 in Lab Procedure) and you need to choose a proper letter among them to add to your uppercase and lowercase lists.

Your program should run **6 times to create each list (uppercase and lowercase)** and in each loop ask the letter you want to add at the front of the uppercase list or the lowercase list. You can fill the lists, asynchronously. Then apply “combine” process and finally print the lists as shown in Figure 3.

Lab Procedure

1. Write the necessary beginning lines and create the structure to build your nodes using the following code snippet.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 30

const char *low[N]; //contains the lowercase letter in main
const char *up[N]; //contains the uppercase letter in main

struct Node {
  char letter[N];
  struct Node *next;
};
struct Node *Node_combined = NULL; //contains the combined list
```

2. Write a function that inserts an item at the front of the list using the prototype shown below.**(15p)**

```
struct Node * insert_front(struct Node *Node_Head, int l_select, const char *low_up[N]){
// const char *low_up[N]) → low[] or up[]
// l_select → selected letter's number
}
```

3. Write a function that prints all the items in the list using the prototype shown below.**(10p)**

```
void print_list(struct Node *Node_Head) {
}
```

4. Write a function that gives warning and return "0" if the selected item is already inserted in the list. Use the prototype shown below. (See Figure 3)**(20p)**

```
int letter_check(struct Node *Node_Head,int l_select,const char *low_up[N]){
// const char *low_up[N]) → low[l_select] or up[l_select]
// l_select → selected letter's number
//You should use strcmp(char *str1,char *str2) function to compare inputs and it returns "0" if the inputs are equal
}
```

5. Write a function that finds the same letters in the uppercase and the lowercase lists and adds the found letters to the "Node_combined" list by following the letter order of the uppercase list. **(40p -> See Section 7)**

- **Hint:** Consider making your lowercase list circular here → easier in nested loops
- You should use "strlwr" function to find the lowercase version of an uppercase letter. **Note that,** once you use "strlwr" function it changes the defined letter to lowercase completely. So, remember to use "srtupr" function in proper lines to change it to its original uppercase version.
- You should use "strcmp" function here as well to compare the letters.

```
void combine(struct Node *Node_Head_l, struct Node *Node_Head_u){
// Node_Head_l is for lowercase list and Node_Head_u is for uppercase list
}
```

6. You should use the definitions given for lowercase/uppercase letters and the corresponding linked list initializations in main function and write other necessary lines to run your code.**(15p)**

```
int main(){

  low[0]="a";                    up[0]="A";
  low[1]="b";                    up[1]="B";
  low[2]="c";                    up[2]="C";
  low[3]="x";                    up[3]="D";
  low[4]="y";                    up[4]="E";
  low[5]="z";                    up[5]="F";

 struct Node *Node_lowerCase = NULL;//list for lowercase
 struct Node *Node_upperCase = NULL;//list for uppercase

 // Necessary lines

 return 0;
 }
```

7. To get full credit from Section 5, test different combinations of "low[ ]" by changing it in main function and try to compare your outputs with the example outputs below. **(%75 of the full credit of Section 5)**

Ex1:

low[0]="b";
low[1]="c";
low[2]="d";
low[3]="f";
low[4]="x";
low[5]="z";

```
Lowercase letters: z d f b c x
Uppercase letters: C F A B D E
Combined letters: C c F f B b D d
--------------------------------
```

Ex2:

low[0]="a";
low[1]="b";
low[2]="d";
low[3]="e";
low[4]="y";
low[5]="z";

```
Lowercase letters: a d y b z e
Uppercase letters: B D C F A E
Combined letters: B b D d A a E e
---------------------------------
```

Ex3:

low[0]="b";
low[1]="x";
low[2]="y";
low[3]="z";
low[4]="k";
low[5]="m";

```
Lowercase letters: y b x m z k
Uppercase letters: D C B F E A
Combined letters: B b
------------------------------
```

```
Select a Uppercase Letter: 1.A   2.B   3.C   4.D   5.E   6.F
Number of letter: 5

Uppercase letters: E
Select a Uppercase Letter: 1.A   2.B   3.C   4.D   5.E   6.F
Number of letter: 2

Uppercase letters: B E
Select a Uppercase Letter: 1.A   2.B   3.C   4.D   5.E   6.F
Number of letter: 2
The letter is already inserted!

Select a Uppercase Letter: 1.A   2.B   3.C   4.D   5.E   6.F
Number of letter: 1

Uppercase letters: A B E
Select a Uppercase Letter: 1.A   2.B   3.C   4.D   5.E   6.F
Number of letter: 6

Uppercase letters: F A B E
Select a Uppercase Letter: 1.A   2.B   3.C   4.D   5.E   6.F
Number of letter: 4

Uppercase letters: D F A B E
Select a Uppercase Letter: 1.A   2.B   3.C   4.D   5.E   6.F
Number of letter: 3

Uppercase letters: C D F A B E


Select a Lowercase Letter: 1.a   2.b   3.c   4.x   5.y   6.z
Number of letter: 5

Lowercase letters: y
Select a Lowercase Letter: 1.a   2.b   3.c   4.x   5.y   6.z
Number of letter: 1

Lowercase letters: a y
Select a Lowercase Letter: 1.a   2.b   3.c   4.x   5.y   6.z
Number of letter: 1
The letter is already inserted!

Select a Lowercase Letter: 1.a   2.b   3.c   4.x   5.y   6.z
Number of letter: 4

Lowercase letters: x a y
Select a Lowercase Letter: 1.a   2.b   3.c   4.x   5.y   6.z
Number of letter: 2

Lowercase letters: b x a y
Select a Lowercase Letter: 1.a   2.b   3.c   4.x   5.y   6.z
Number of letter: 3

Lowercase letters: c b x a y
Select a Lowercase Letter: 1.a   2.b   3.c   4.x   5.y   6.z
Number of letter: 6

Lowercase letters: z c b x a y
Uppercase letters: C D F A B E
Combined letters: C c A a B b
--------------------------------
Process exited after 58.03 seconds with return value 0
```

Figure 3: Screenshot of running program