Lab Sheet No 3. Pointers and Linked Lists

1) Objectives

The objective of this lab is to learn, through a set of activities and practical exercises, the use of pointers and linked lists in the C language.

By the end of this lab, students should be familiar with pointers, dynamic memory management, and be capable of solving complex problems using dynamic data structures, specifically linked lists.

2) Example 1: Pointer Manipulation

Consider the following program:

```
#include <stdio.h>
int main() {
     int x,y;float z; int* p;
     x = 5;
     ........
     p = &x;
     ........
     y = *p-2;
     ........
     p++;
     ........
     *p = *p**(p-1);
     ........
     p = &z;
     ........
     p = malloc(4*2);
     ........
     *p = 1;
     ........
     *(p+1) = 2;
     ........
     free(p);
     ........
     return 0;
}
```

1. Create a new project and type the above code.
2. Complete the code to display the content of variables after each instruction, then compile and execute.

3) Example 2: Linked List Manipulation

The following program is supposed to insert numbers from 1 to 10 into a linked list of integers (insertion at the head) and then display them.

```
#include <stdio.h>
//Declaration of the data structure (the type describing a linked list)
typedef struct node* List;
typedef struct node{
               int val;
               List next;
     }node;
List L;
//Function allowing to insert a value  v  at the head of a linked list  L
List insertHead(List L,int v){
    List p = malloc(.......);
    p->val = v;
    p->next = L;
    .........
    return L;
}
//Function allowing to display the elements of a linked list  L
void displayList(List L){
  List p;
  if(L == NULL)
     printf("The list is empty");
  else{
     .......
     while(......){
          printf("%d ",p->val);
          .......
     }
  }
}
main(){
    int i;List L;
    L = NULL;
    for(i=1; i<=10; i++)
        L = insertHead(L,i);
    displayList(L);
}
```

1. Create a new project and type the above code.
2. Complete the code to achieve the desired processing, then compile and execute.

3. Can you justify the obtained result?
4. Make the necessary modifications for the display to be as expected.
5. Finally, replace the **`insertHead`** function with a procedure.

4) Application Exercises

1. Add to the program from the previous Example 2, the procedure **`sum(L,sPos,sNeg)`** that calculates and returns the sum of positive and negative elements of the linked list **`L`** passed as input.

   Test this procedure in the **`main`** function.

2. Let **`L`** be a list of integers without redundancies. We want to create a program that separates the list **`L`** into two lists based on a value entered by the user. To accomplish this:

   a) Write the function **`insertTail(L, v)`** that inserts an integer value **`v`** into the list **`L`**. (Covered in lecture).
   b) Write the procedure **`displayList(L)`** that displays the elements of the list **`L`**. (Covered in lecture).
   c) Write the function **`exists(L, v)`** that checks if a value **`v`** exists in the list **`L`**. This function should return the address of the element preceding the element containing **`v`** if **`v`** exists and return **`NULL`** otherwise.
   d) Write the procedure **`separate(L, L1, L2, v)`** that separates the list **`L`** into two lists: **`L1`** will contain the elements before the element containing the value **`v`**, and **`L2`** will start from that element. The original list **`L`** should be empty.
   e) Using the previous sub-programs, write the **`main`** function that allows the user to enter **`n`** integers (where **`n`** is entered by the user), puts them into a linked list, separates this list into two lists based on a value **`v`** also entered by the user, and displays the resulting two lists.

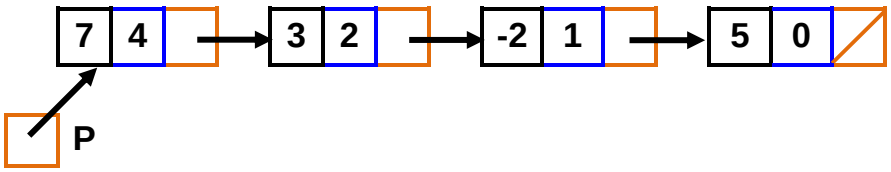
3. Consider a list of real numbers. (Covered in TW).

   a) Write the function **`insertSorted(L, v)`** that inserts an element into a linked list sorted in ascending order, ensuring that the list remains sorted after insertion.
   b) Write the procedure **`displayList(L)`** that displays the elements of the linked list **`L`**.
   c) Write the **`main`** function that asks the user to enter 10 real numbers and inserts them into a linked list in such a way that the resulting list is sorted in ascending order. Display the obtained list.

5) Additional Exercises

1. Polynomials can be represented by a linked list, where each term of the polynomial is stored in a node of the list containing the degree (degree) of the term and the corresponding coefficient (coeff).

   For example, the polynomial $7x^4 + 3x^2 - 2x + 5$ is represented as follows:



   a) Provide the data structure **`Poly`** allowing to represent a polynomial. (The algorithmic declaration was covered in TW).

   b) Write the function **`createMonomial(deg, coeff)`** to create a monomial, store its coefficient and degree, and return its address.

   c) Write the function **`addMonomial(P, deg, coeff)`** that adds a monomial (with degree **`deg`** and coefficient **`coeff`**) to the polynomial list **`P`**. Note: The polynomial should always be sorted in descending order of exponents.

   f) Write the function **`evaluate(P, x)`** to evaluate a polynomial **`P`** for a given value **`x`**. The polynomial and the value **`x`** are passed as arguments. (The algorithm was covered in TW).

   d) Write the **`main`** function that allows keyboard input of a polynomial, stores it in a linked list, reads a value **`x`**, and evaluates the polynomial for the given value **`x`**.

2. Consider a program that manages rental housing by storing them in a linked list. For each housing unit, we store its identifier (integer), category (economical, social, or normal), price (real), and location (string). We are interested in the following functionalities:

   a) Create a linked list **`L`** to represent the set of housing units.

   b) Display information for all housing units with the category "economical."

   c) Delete the element with the identifier **`id`** from the list **`L`**. The identifier **`id`** should be read from the keyboard.

   d) Create three sub-lists **`L_eco`**, **`L_soc`**, and **`L_norm`** (based on the **`category`** field) from the list **`L`** without allocating new spaces.

   e) Assuming that the three created sub-lists are sorted by housing identifier, add a housing unit with information entered through the keyboard.

   Use subprograms for these functionalities.