Correction of the practical worksheet N°3

2) Example 1: Pointer manipulation

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.

Answer:

Please explain to the students the role of each instruction.

```
#include <stdio.h>
int main() {
  int x, y; float z; int* p;
  x = 5;
  printf("x=%d\n", x);
  p = &x;      //p receives the address of x
  printf("p=%d\n", p);
  y = *p - 2;   //*p reads "the content of p", which is the value of x (since p points to x). Thus, y takes the value of x minus 2 = 3
  printf("y=%d\n", y);
  p++;  //The content of p is incremented (by 4 bytes); p now points to y.
  printf("p=%d\n", p);
  *p = *p * *(p - 1);   //y takes the value of y * x = 15
  printf("y=%d\n", y);
  p = &z;      //In principle, this is incorrect because p is a pointer to an integer, so it cannot take the address of a real number.
  printf("p=%d\n", p);
  p = malloc(4 * 2);   //Allocates 8 bytes, which is the space needed to store 2 integers (array of 2 elements), and puts the address in p
  printf("p=%d\n", p);
  *p = 1;      //The first element of the array takes the value 1
  printf("p[0]=%d\n", *p);
  *(p + 1) = 2;  //The second element of the array takes the value 2
  printf("p[1]=%d\n", *(p + 1));
  free(p);     //Frees the previously allocated space, but the address of this space is still contained in p
  printf("p=%d\n", p);
  return 0;
}
```

3) Example 2: Linked List Manipulation

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

Answer:

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

3. Can you justify the obtained result?

   Answer:

   We constate that the numbers from 1 to 10 are displayed in reverse order. This is in fact due to the fact that in the **`for`** loop, numbers from 1 to 10 are inserted at the head of the list using the **`insertHead`** function. The **`insertHead`** function always inserts the new element at the beginning of the list, making it the new head. As a result, the numbers are inserted in reverse order, with 10 being the first element and 1 being the last.

   Make the necessary modifications for the display to be as expected.

   Answer:

   If you want to display the numbers in the original order (1 to 10), Two solutions are possible:

   - Inserting the numbers at the tail of the list rather than the head. This way, the last number inserted will be the last element in the linked list.

- Modify the **`for`** loop to start from 10 to 1.

We will adopt the second solution, which is the simpler solution:

```
main(){
    int i;List L;
    L = NULL;
    for(i=10; i>=1; i--)
        L = insertHead(L,i);
    displayList(L);
}
```

4. Finally, replace the **`insertHead`** function with a procedure.

Answer:

In the procedure, the head should be passed by variable because it will be modified.

During the call, don't forget to pass the address of the head and not its value.

```
void insertHead(List* L,int v){
    List p = malloc(sizeof(node));
    p->val = v;
    p->next = *L;
    *L = p;
}
...
main(){
    int i;List L;
    L = NULL;
    for(i=10; i>=1; i--)
        insertHead(&L, i);
    displayList(L);
}
```

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.

   Answer:

   To be able to test this procedure, values inserted into the list should be entered via the keyboard.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* List;
typedef struct node {
    int val;
    List next;
} node;
List insertHead(List L, int v) {
    List p = malloc(sizeof(node));
    p->val = v;
    p->next = L;
```

```
        L = p;
        return L;
    }
    void displayList(List L) {
        List p;
        if (L == NULL)
            printf("The list is empty");
        else {
            p = L;
            while (p != NULL) {
                printf("%d ", p->val);
                p = p->next;
            }
        }
    }
    void sum(List L, int* sPos, int* sNeg) {
        List p;
        *sPos = 0;
        *sNeg = 0;
        p = L;
        while (p != NULL) {
            if (p->val >= 0)
                *sPos = *sPos + p->val;
            else
                *sNeg = *sNeg + p->val;
            p = p->next;
        }
    }
    int main() {
        int i, v, sPos, sNeg;
        List L;
        L = NULL;
        for (i = 1; i <= 10; i++) {
            printf("Enter value %d: ", i);
            scanf("%d", &v);
            L = insertHead(L, v);
        }
        printf("List: ");
        displayList(L);
        sum(L, &sPos, &sNeg);
        printf("\nSum of positive elements: %d", sPos);
        printf("\nSum of negative elements: %d\n", sNeg);
        return 0;
    }
```

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.

Answer:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node* List;
typedef struct node {
    int val;
    List next;
} node;
List insertTail(List L, int v) {
    List p, q;
    p = malloc(sizeof(node));
    p->val = v;
    p->next = NULL;
    if (L == NULL)
        L = p;
    else {
        q = L;
        while (q->next != NULL)
            q = q->next;
        q->next = p;
    }
    return L;
}
void displayList(List L) {
    List p;
    if (L == NULL)
        printf("The list is empty");
    else {
        p = L;
        while (p != NULL) {
            printf("%d ", p->val);
            p = p->next;
        }
    }
    printf("\n");
}
List exists(List L, int v) {
    List p;
    if (L == NULL)
```

```
        return NULL;
    else if (L->val == v) //Special case: if the searched value is
in the head
        return L;//return the head, and consider this case in the
separate procedure
    else {
        p = L;
        while (p->next != NULL && p->next->val != v)
            p = p->next;
        if (p->next == NULL)
            return NULL;
        else
            return p;
    }
}
void separate(List* L, List* L1, List* L2, int v) {
    List p;
    *L1 = NULL;
    *L2 = NULL;
    p = exists(*L, v);
    if (p != NULL) {
       if (p->val == v) //if the searched value is in the head
            *L2 = *L;  //L2 starts from L, and L1 remains empty
       else {
            *L2 = p->next;
            p->next = NULL;
            *L1 = *L;
       }
       *L = NULL;
    }
}
int main() {
    int n, i, v;
    List L, L1, L2;
    L = NULL;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
       printf("Enter value %d: ", i);
       scanf("%d", &v);
       L = insertTail(L, v);
    }
    printf("Initial list L: ");
    displayList(L);
    printf("Enter the separation value: ");
    scanf("%d", &v);
    separate(&L, &L1, &L2, v);
    printf("List L after separation: ");
```

```
    displayList(L);
    printf("List L1: ");
    displayList(L1);
    printf("List L2: ");
    displayList(L2);
    return 0;
}
```

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.

Answer:

```
#include <stdio.h>
typedef struct node* List;
typedef struct node {
    float val;
    List next;
} node;
List insertSorted(List L, float v) {
    List p, q, r;
    p = malloc(sizeof(node));
    p->val = v;
    p->next = NULL;
    if (L == NULL || L->val > v) {
        p->next = L;
        L = p;
    } else {
        q = L;
        r = L->next;
        while (r != NULL && r->val < v) {
            q = r;
            r = r->next;
        }
        q->next = p;
        p->next = r;
    }
    return L;
}
void displayList(List L) {
    List p;
    if (L == NULL)
        printf("The list is empty");
    else {
        p = L;
```

```
        while (p != NULL) {
            printf("%.2f ", p->val);
            p = p->next;
        }
    }
    printf("\n");
}
int main() {
    int i;
    float v;
    List L;
    L = NULL;
    for (i = 1; i <= 10; i++) {
        printf("Enter value %d: ", i);
        scanf("%f", &v);
        L = insertSorted(L, v);
    }
    printf("List L: ");
    displayList(L);
    return 0;
}
```