Série 13.1 : Les listes chainées

Exercice 1

Ecrire un programme qui permet de gérer une liste chaînée.

1) Il faut insérer en début de liste les nombres saisis par l'utilisateur, ceci tant que le nombre entré est différent de 0. Il s'agit de créer un nouveau noeud pour chaque nombre. Les nœuds sont toujours insérés en début de liste !
2) Afficher la liste
3) Parcourir la liste et augmenter le contenu de chaque nœud de 1.
4) Effacer un à un les noeuds en début de liste de la liste et liberer la mémoire.

Utilisez un pointeur sur le premier nœud : **`struct noeud *premier=NULL ;`**

Version I (sans fonction)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int         data;
    struct node* next;
} NodeTy;

typedef NodeTy* PtrNodeTy;

int main()
{
    PtrNodeTy first=NULL;   // pointeur qui a TOUJOURS l'adresse
    du premier maillon
    int value;

    // 1. Let the user INSERT new nodes
    do
    {
        printf("Entrez une valeur: (0=fin)\n");
        scanf("%d", &value);

        if (value !=0) // ADD AT BEGINNING
        {
            PtrNodeTy node = (PtrNodeTy) malloc(sizeof(NodeTy));
            node->data=value;
            node->next=first;
            first=node;
        }
    }
    while(value!=0);
```

```
    //2. Display the Linked List
    PtrNodeTy current=first;
    while(current!=NULL)
    {
        printf("%d --> ",current->data);
        current=current->next;
    }
    printf("NULL\n"); // pour faire joli

    // 3. Increment all the values of the list
    printf("On parcourt la liste et incremente la valeur de tous ses maillons de 1\n");
    current=first;
    while(current!=NULL)
    {
        current->data += 1;
        printf("%d --> ",current->data);
        current=current->next;
    }
    printf("NULL\n"); // pour faire joli

    // 4. Supress nodes (free memory)
    {
        PtrNodeTy current = NULL;
        while (first!=NULL)
        {
            current=first;
            first= first->next;
            free(current);
        }
    }

    system("PAUSE");
    return 0;
}
```

Version II avec des fonctions et un typedef

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int          data;
    struct node* next;
} NodeTy; // permet de remplacer « struct nœud » par « NodeTy »

 void displayList(NodeTy* first)
 {
     NodeTy* current=first;
     printf("List:");
     while(current!=NULL)
     {
         printf("%d ",current->data);
         current=current->next;
     }
     printf("\n");
 }

void removeNodes(NodeTy** first)
{
    NodeTy* current = *first;

    while (current!=NULL)
    {
        current=*first;
        *first= (*first)->next;
        free(current);
        current=*first;
    }
}
```

```
void addNodeAtBegin(NodeTy** first, int value)
{
    NodeTy* node = (NodeTy*)malloc(sizeof(NodeTy));
    node->data=value;

    if (*first==NULL)
    {
        node->next=NULL;
        *first=node;
    }
    else
    {
        node->next=*first;
        *first=node;
    }
}

int main()
{
    NodeTy* first=NULL; // first est un pointeur sur
                        // une structure de noeud
    int value;

    do
    {
        printf("Entrez une valeur: (0=fin)\n");
        scanf("%d", &value);

        addNodeAtBegin(&first,value);
        displayList(first);

    }while(value!=0);

    displayList(first);
    removeNodes(&first);
    displayList(first);

    return 0;
}
```

Exercice 2 [BONUS]

Inserer les nombres positifs en début de liste et les nombres négatifs en fin de liste.

- Indication : Utilisez un second pointeur qui pointe sur le dernier nœud et mettez ces deux pointeurs dans une structure qui correponde à une liste (`ListTy`).

```
typedef struct list{
    NodeTy*      first;
    struct node* last;
} ListTy;
```

- Allouez dynamiquement une liste avec la fonction `newList();`

```
ListTy* mylist=NULL;
mylist=newList();
```

Programme

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node{
    int         data;
    struct node* next;
} NodeTy;

typedef struct list{
    NodeTy*      first;
    struct node* last;
} ListTy;


ListTy* newList()
{
    ListTy *l= (ListTy*)malloc(sizeof(ListTy));
    l->first = NULL;
    l->last  = NULL;
    return l;
}

void removeNodes(ListTy* list)
{
    NodeTy* current = list->first;

    while (current!=NULL)
    {
        current     = list->first;
        list->first = list->first->next;
        free(current);
        current     = list->first;
    }
}

void freeList(ListTy** list)
{
    removeNodes(*list);
    free(*list);
    *list=NULL;
}
```

```
void displayList(ListTy* list)
 {
     NodeTy* current=list->first;
     while(current!=NULL)
     {
         printf("%d ",current->data);
         current=current->next;
     }
 }

void addNodeAtBegin(ListTy* list, int value)
{
    NodeTy* node = (NodeTy*)malloc(sizeof(NodeTy));
    node->data=value;

    if (list->first==NULL)
    {
        node->next = NULL;
        list->first = node;
        list->last  = node;
    }
    else
    {
        node->next  = list->first;
        list->first = node;
    }
}


void addNodeAtEnd(ListTy* list, int value)
{
    NodeTy* node = (NodeTy*)malloc(sizeof(NodeTy));
    node->data=value;
    node->next=NULL;

    if (list->first==NULL)
    {
        node->next = NULL;
        list->first = node;
        list->last  = node;
    }
    else
    {
        list->last->next = node;
        list->last       = node;
    }
}
```

```
int main()
{
    ListTy* mylist=NULL;
    mylist=newList();

    int value;
    do
    {
        printf("Entrez une valeur: (0=fin)\n");
        scanf("%d", &value);

        if (value>0)
            addNodeAtBegin(mylist,value);
        else if (value<0)
            addNodeAtEnd(mylist, value);

        displayList(mylist);printf("\n");

    }while(value!=0);

    printf(" Final list  : ");
    displayList(mylist); printf("\n");

    removeNodes(mylist);

    printf(" After remove: ");
    displayList(mylist); printf("\n");

    freeList(&mylist);

    return 0;
}
```