< C >

Les listes

23 | MAR | 2020

Dorian.H Mekni

Les listes

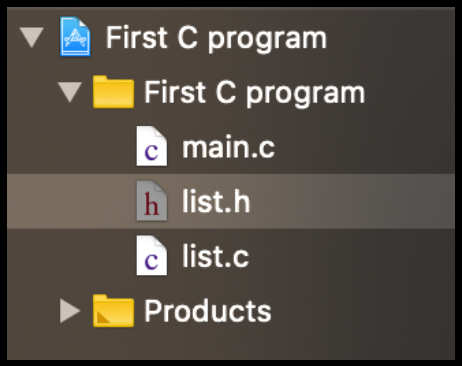
- La liste est une structure de donnée
- Pas d’ordre dans l’insertion des données
- On peut ajouter une donnée à la fin au debut ou au milieu de cette liste.

Les fonctions

- push_front()
- push_back()
- pop_front()
- pop_back()

Architecture Administrative

- 3 fichiers -> .h .c .c



- 1 main.c -> application et manipulation des fonctions avant compilation
- list.c -> construction de nos fonctions
- list.h -> repertoire des prototypes de nos fonctions

list.h -> listing des prototypes

```c
#ifndef list_h
#define list_h

#include <stdio.h>

// Définition du type Booléen
typedef enum
{
    false,
    true
}Bool;

// Def d'une liste
typedef struct listObject
{
    int value;
    struct listObject *next;

}ListObject, *List;

// Prototypes
List new_list(void);
Bool is_empty_list(List li);
int list_length(List li);
void print_list(List li);
List insert_back(List li, int x);
List insert_front(List li, int x);
List delete_back(List li);
List delete_front(List li);
List clear_list(List li);

#endif /* list_h */

```

type Bool

```
// Définition du type Booléen
typedef enum
{
    false,
    true
}Bool;
```

struct -> list

Sans paramètres puisque la liste n’es pas aussi réglementée quant à ses insertions.

struct ->

```c
// Def d'une liste
typedef struct listObject
{
    int value;
    struct listObject *next;

}ListObject, *List;
```

new_list()

cette fonction permet de créer une nouvelle liste

```
List new_list(void)
{
    return NULL;

}
```

is_empty_list()

Celle-ci nous permet de savoir si notre list est vide ou pas

```
/*-------------------------------------------*/

Bool is_empty_list(List li)
{
    if(li == NULL)
        return true;

    return false;
}

/*-------------------------------------------*/
```

list_length()

Elle calcule la longueur d'une liste : Combien d'elements contient elle ?

```c
int list_length(List li)
{
    int size = 0;

    if(is_empty_list(li))
        return size;

    while(li != NULL)
    {
        ++size;
        li = li->next;
    }

    return size;
}
```

print_list()

Elle permet d'afficher les valeurs de chaque donnée dans la liste

```c
void print_list(List li)
{
  if(is_empty_list(li))
  {
      printf("nothing to print, the list is empty.\n");
      return;
  }

    while(li !=NULL)
    {
        printf("[%d] ", li->value);
        li = li->next;

    }

    printf("\n");
}
```

Test basique de compilation

```c
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main(void)
{
    List mylist = new_list();

    if(is_empty_list(mylist))
        printf("Empty list.\n");
    else
        printf("The list contains elements.\n");

    printf("The size of the list is %d\n", list_length(mylist));
    return 0;
}

```

```
Empty list.
The size of the list is 0
Program ended with exit code: 0
```

insert_front()

insert_front() permet l'insertion d'une donnée à l'avant d'une liste:

```
/*-------------------------------------------------*/

List insert_front(List li, int x)
{
    ListObject *element;

    element = malloc(sizeof(*element));

    if(element == NULL)
    {
       fprintf(stderr, "error : dynamic allocation issue.\n");
        exit(EXIT_FAILURE);
    }

    if(is_empty_list(li))
        element->next = NULL;
    else
        element->next = li;

    return element;
}

/*-------------------------------------------------*/
```

insert_back()

Insertion de donnée en fin de liste-> reprend la 1er partie de insert_front() quand il s'agit de créer la donner en premier lieu, la suite prend un autre cours:

```
List insert_back(List li, int x)
{
    // Création d'un élément: Début->
    ListObject *element;

    element = malloc(sizeof(*element));

    if(element == NULL)
    {
       fprintf(stderr, "error : dynamic allocation issue.\n");
        exit(EXIT_FAILURE);
    }
    // Création d'un élément: Fin<-

    element->value = x;
    element->next = NULL;

    if(is_empty_list(li))
        return element;

    ListObject *temp;
    temp = li;

    while(temp->next != NULL)
        temp = temp->next;

    temp->next = element;

    return li;

}
```

delete_back()

Fonction permettant l'elimination de la donnée en fin de liste

```
List delete_back(List li)
{
    if(is_empty_list(li))
        return new_list();

    if(li->next == NULL)
    {
        free(li);
        li = NULL;

        return new_list();
    }
    // En utilisant des pointeurs, on a impacté la liste li
    // Ces pointeurs sont comme des références.
    ListObject *temp = li;
    ListObject *before = li;

    while(temp->next != NULL)
    {
        before = temp;
        temp = temp->next;
    }

    before->next = NULL;
    free(temp);
    temp = NULL;

    return li;

}
```

delete_front()

```
List delete_front(List li)
{
    if(is_empty_list(li))
    return li;

    ListObject *element;

    element = malloc(sizeof(*element));

    if(element == NULL)
    {
       fprintf(stderr, "error : dynamic allocation issue.\n");
        exit(EXIT_FAILURE);
    }

    element = li->next;

    free(li);
    li = NULL;

    return element;

}
```

Compilation [1]

1er Test de compilation sur nos fonctions basiques

```c
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main(void)
{
    List mylist = new_list();
    print_list(mylist);

    mylist= insert_back(mylist, 12);
    print_list(mylist);

    return 0;
}
```

```
nothing to print, the list is empty.
[12]
Program ended with exit code: 0
```

Compilation [2]

Utilisation de toutes nos fonctions:

```c
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main(void)
{
    List myList = new_list();
    print_list(myList);

    myList= insert_back(myList, 12);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    myList= insert_front(myList, 87);
    myList= insert_back(myList, 14);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    myList= delete_front(myList);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    myList= delete_back(myList);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    myList= delete_back(myList);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    return 0;
}
```

Compilation | Console [2]

```
nothing to print, the list is empty.
[12]
Number of elements in the list : 1.
[87] [12] [14]
Number of elements in the list : 3.
[12] [14]
Number of elements in the list : 2.
[12]
Number of elements in the list : 1.
nothing to print, the list is empty.
Number of elements in the list : 0.
Program ended with exit code: 0
```

clear_list()

Elle nettoie une liste de ses données.

```
/*-----------------------------------------------*/

List clear_list(List li)
{
    if(is_empty_list(li))
        return new_list();

      else
          return li = NULL;
}

/*-----------------------------------------------*/
```

Compilation[3]

Utilisation de la fonction clear_list()

```c
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main(void)
{
    List myList = new_list();
    print_list(myList);

    myList= insert_back(myList, 12);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    myList= insert_front(myList, 87);
    myList= insert_back(myList, 14);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    myList= clear_list(myList);
    print_list(myList);
    printf("Number of elements in the list : %d.\n", list_length(myList));

    return 0;
}
```

Compilation | Console [3]

```
nothing to print, the list is empty.
[12]
Number of elements in the list : 1.
[87] [12] [14]
Number of elements in the list : 3.
nothing to print, the list is empty.
Number of elements in the list : 0.
Program ended with exit code: 0
```