< C >

Listes [doublement] chaînées

Dorian.H Mekni

24 | MAR | 2020

Liste [doublement] chaînée

- Une alternative au tableaux pour stocker des données
- Une liste chaînée est une structure comportant des champs de données et un pointeur vers une structure de même type.
- Une liste doublement chaînée est une liste dont chaque élément peut accéder à l'aide de pointeurs aux éléments positionnés immédiatement avant et après lui dans la liste.
- Le chaînage s’opère dans les deux sens, ce qui rend possible le parcours de la liste en avant mais aussi en arrière
- Cette option n’est pas faisable dans la liste simple.

Struct [1]

- Une structure auto référentielle (aussi appelée structure récursive) correspond à une structure dont au moins un des champs contient un pointeur vers une structure de même type.
- De cette façon on crée des éléments (appelés parfois noeuds ou liens) contenant des données, mais, contrairement à un tableau, celles-ci peuvent être dispersées en mémoire et reliées entre elles par des liens logiques (des pointeurs), cad un ou plusieurs champs dans chaque structure contenant l'adresse d'une ou plusieurs structures de même type.
- Lorsque la structure contient des données et un pointeur vers la structure suivante on parle de liste chaînée.

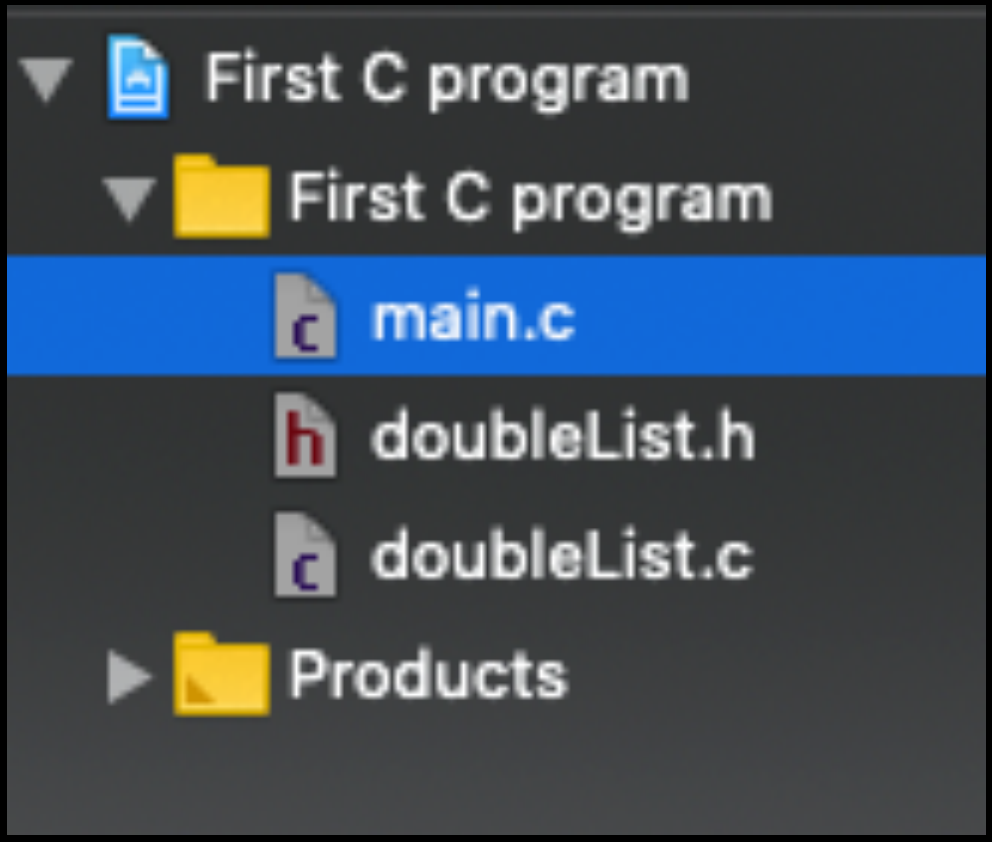
Struct [1]

- Si la structure contient des données, un pointeur vers une première structure suivante, et un pointeur vers une seconde, cela s'identifie en arbre binaire.

- Une liste doublement chaînée est basée sur le même concept que la liste chaînée. Elle diffère dans le sens où elle contient un pointeur vers le maillon suivant et également un pointeur vers le maillon précédent.

- Ce paramètre permet de cette façon le parcours de la liste dans les deux sens.

Classification de fichiers



doubleList.h->Fichier en tête

```c
#ifndef doubleList_h
#define doubleList_h
#include <stdio.h>

// Définir un Bool

typedef enum
{
    false,
    true
}Bool;

// Définir la chaîne d'une double liste

typedef struct DoubleListNode
{
    int value;
    struct DoubleListNode *back;
    struct DoubleListNode *next;

}DoubleListeNode;

// Définir une double liste

typedef struct DoubleList
{
    int length;
    struct DoubleListNode *begin;
    struct DoubleListNode *end;
}*DoubleList;

// Prototypes

DoubleList new_dlist(void);
DoubleList push_back_dlist(DoubleList li, int x);
DoubleList push_front_dlist(DoubleList li, int x);
DoubleList pop_back_dlist(DoubleList li);
DoubleList pop_front_dlist(DoubleList li);
DoubleList clear_dlist(DoubleList li);
Bool is_empty_dlist(DoubleList li);
void print_dlist(DoubleList li);
int DoubleList_length(DoubleList li);
int DoubleList_first(DoubleList li);
int DoubleList_last(DoubleList li);
#endif /* doubleList_h */
```

Le booléen

```
Bool is_empty_dlist(DoubleList li)
{
    if(li == NULL)
        return true;

    return false;
}
```

DoubleList new_dlist()

new_dlist permet la creation d'une double liste

```
DoubleList new_dlist(void)
{
    return NULL;
}
```

DoubleList_first()

DoubleList_first()-> se réfère au premier élément de la liste.

```
int DoubleList_first(DoubleList li)
{
    if(is_empty_dlist(li))
        exit(1);

    return li->begin->value;
}
```

DoubleList_last()

DoubleList_last()-> se réfère au dernier élément de la liste.

```
int DoubleList_last(DoubleList li)
{
    if(is_empty_dlist(li))
        exit(1);

    return li->end->value;
}
```

push_back_dlist()

push_back_dlist() est une fonction qui ajoute un élément en fin de liste.

```
DoubleList push_back_dlist(DoubleList li, int x)
{
    // Allocation d'espace necessaire pour le DoubleListeNode.
    DoubleListeNode *element;

    element = malloc(sizeof(*element));

    if(element == NULL)
    {
        fprintf(stderr, "Fatal Error, dynamic allocation issue!.\n");
        exit(EXIT_FAILURE);
    }
    // On crée l'élément de base sans aucune liason
    element->value = x;
    element->next = NULL; //Rien ne la suit
    element->back = NULL; // Rien ne la précède

    if(is_empty_dlist(li))
    {
        li = malloc(sizeof(*li));

        if(li == NULL)
        {
            fprintf(stderr, "Fatal Error, dynamic allocation issue!.\n");
            exit(EXIT_FAILURE);
        }
        li->length = 0;
        li->begin = element;
        li->end = element;

     } else {

            li->end->next = element;
            element->back = li->end;
            li->end = element;

        }
                    li->length++;
                    return li;

}
```

push_front_dlist()

push_front_dlist() permet l'ajout de donnée en tête de liste.

```c
DoubleList push_front_dlist(DoubleList li, int x)
{
    // Allocation d'espace necessaire pour le DoubleListeNode.
    DoubleListeNode *element;

    element = malloc(sizeof(*element));

    if(element == NULL)
    {
        fprintf(stderr, "Fatal Error, dynamic allocation issue!.\n");
        exit(EXIT_FAILURE);
    }
    // On crée l'élément de base sans aucune liason
    element->value = x;
    element->next = NULL; //Rien ne la suit
    element->back = NULL; // Rien ne la précède

    if(is_empty_dlist(li))
    {
        li = malloc(sizeof(*li));

        if(li == NULL)
        {
            fprintf(stderr, "Fatal Error, dynamic allocation issue!.\n");
            exit(EXIT_FAILURE);
        }

        li->length = 0;
        li->begin = element;
        li->end = element;

     } else {

            li->begin->back = element;
            element->next = li->begin;
            li->begin = element;

        }
                    li->length++;
                    return li;

}
```

pop_back_dlist()

pop_back_dlist() permet le retrait de donnée en fin de liste.

```
DoubleList pop_back_dlist(DoubleList li)
{
    if(is_empty_dlist(li))
    {
        printf("Nothing to delete, the Double list is empty.\n");
        return new_dlist();
    }

    if(li->begin == li->end)
    {
        free(li);
        li = NULL;

        return new_dlist();
    }

    DoubleListeNode *temp = li->end;

    li->end = li->end->back;
    li->end->next = NULL;
    temp->next = NULL;
    temp->back = NULL;

    free(temp);
    temp = NULL;

    li->length--;

    return li;
}
```

pop_front_dlist()

pop_front_dlist() retire la donnée en tête de liste.

```
DoubleList pop_front_dlist(DoubleList li)
{
    if(is_empty_dlist(li))
       {
           printf("Nothing to delete, the Double list is empty.\n");
           return new_dlist();
       }

       if(li->begin == li->end)
       {
           free(li);
           li = NULL;

           return new_dlist();
       }

       DoubleListeNode *temp = li->begin;

       li->begin = li->begin->next;
       li->begin->back = NULL;
       temp->next = NULL;
       temp->back = NULL;

       free(temp);
       temp = NULL;

       li->length--;

       return li;
}
```

DoubleList_length()

DoubleList_length() calcule combien de donnée une liste contient.

```
int DoubleList_length(DoubleList li)
{
    if(is_empty_dlist(li))
        return 0;

    return li->length;
}
```

print_dlist()

print_dlist() se traduit en affichage des données d'une liste.

```c
void print_dlist(DoubleList li)
{
    if(is_empty_dlist(li))
    {
        printf("Nothing to be show.\n");
        return;
    }

    DoubleListeNode *temp = li->begin;

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

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

clear_dlist()

clear_dlist() nettoie une liste de toutes ses données.

```
DoubleList clear_dlist(DoubleList li)
{
    if(!is_empty_dlist(li))
        return NULL;

    else
        return li;
}
```

Compilation Test 1

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

int main(void)
{
    DoubleList myDlist = new_dlist();

    if(is_empty_dlist(myDlist))
        printf("The double list is empty.\n");
    else
        printf("\nshe contains data.\n");

    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");
    return 0;
}
```

```
The double list is empty.

The size of the double list is 0.

Program ended with exit code: 0
```

Compilation Test 2

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

int main(void)
{
    DoubleList myDlist = new_dlist();
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");

    myDlist = push_back_dlist(myDlist, 12);
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");

    myDlist = push_back_dlist(myDlist, 5);
    myDlist = push_back_dlist(myDlist, 12);
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");

    myDlist = pop_front_dlist(myDlist);
    myDlist = pop_front_dlist(myDlist);
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");

    myDlist = pop_front_dlist(myDlist);
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");


    return 0;
}

```

Compilation Test 2 sur Console

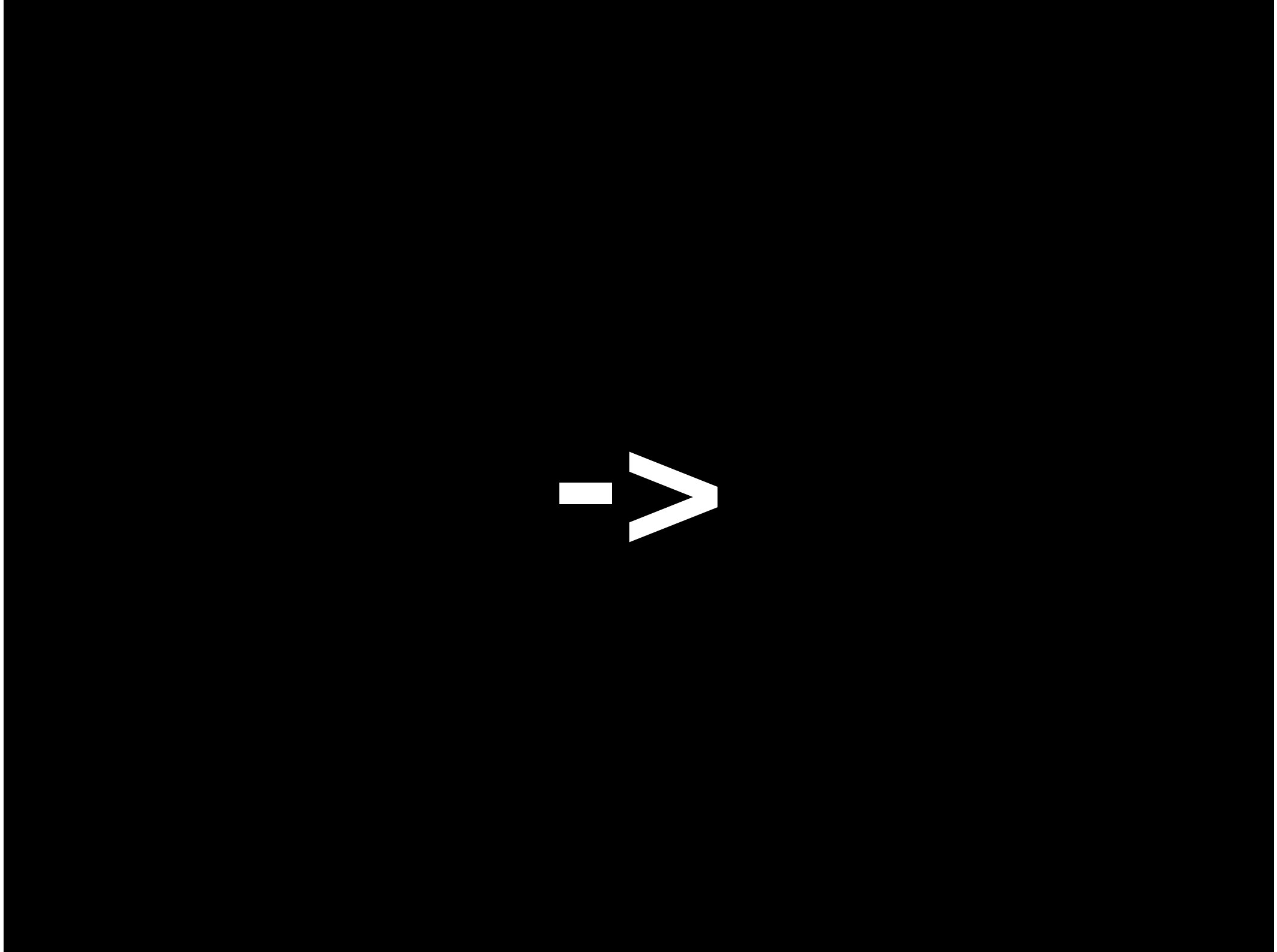
```
Nothing to be show.

The size of the double list is 0.

[12}

The size of the double list is 1.

[12} [5} [12}

The size of the double list is 3.

[12}

The size of the double list is 1.

Nothing to be show.

The size of the double list is 0.

Program ended with exit code: 0
```

Compilation Test 3

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

int main(void)
{
    DoubleList myDlist = new_dlist();
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");

    myDlist = push_back_dlist(myDlist, 12);
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");

    myDlist = push_back_dlist(myDlist, 5);
    myDlist = push_back_dlist(myDlist, 12);
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");

    myDlist =clear_dlist(myDlist);
    print_dlist(myDlist);
    printf("\nThe size of the double list is %d.\n", DoubleList_length(myDlist));
    printf("\n");


    return 0;
}
```

Compilation Test 3 sur Console

```
Nothing to be show.

The size of the double list is 0.

[12}

The size of the double list is 1.

[12} [5} [12}

The size of the double list is 3.

Nothing to be show.

The size of the double list is 0.

Program ended with exit code: 0
```


->
->
->