

# Structures de données

---

Semestre 4



---

# Avant-propos

---

Suite du module d’algorithmique et programmation, accent sur les structures de données

- Pile
- File
- Arbre
- ...

## Heures

- 24h de CTDI
- 26 de TDM

## Notation

**Contrôle intermédiaire** 30%

**Contrôle terminal** 50%

**TP** 20%

**TP Noté** 50%

**Devoir écrit** 25%

**Devoir TP** 25 %

---

# Table des matières

---

<b>2</b>	<b>Structures de données classiques</b>	<b>9</b>
2.1	Pile . . . . .	9
2.2	File . . . . .	11
2.3	File avec priorité . . . . .	14
2.4	Liste avec priorité . . . . .	14
<b>3</b>	<b>Parcourir une collection</b>	<b>16</b>
3.1	Itérateur sur la liste doublement chaînée . . . . .	16
<b>A</b>	<b>Cours sur les pointeurs en C</b>	<b>16</b>
A.1	Syntaxe . . . . .	16
A.2	Opérateur autorisés sur les pointeurs . . . . .	17
A.3	Pointeur sur fonction . . . . .	18
<b>B</b>	<b>Liste des codes sources</b>	<b>20</b>
<b>C</b>	<b>Table des figures</b>	<b>21</b>
<b>D</b>	<b>Exercices</b>	<b>22</b>
D.1	Pointeurs . . . . .	22

# Parcourir une collection

Un itérateur est une structure de données qui permet de parcourir une collection d'objets.

## Ex

```
for(i=N-1; i >= 0; ++i);
```

Ici *i* joue le rôle d'un itérateur, il permet de parcourir une collection de *N* éléments du début à la fin ou de la fin au début.

Un itérateur est lié à une collection, on peut

- se placer en début/fin de la collection
- Passer à l'élément suivant/précédent de la collection
- savoir quand on est arrivé à la fin/début de la collection

En utilisant un langage pseudo objet, cela nous donnerai l'algorithme suivant :

```
i = creerIterateur(c); //c étant la collection
for(i = debut(i); !videSuivant(i) ; i = suivant(i)) {
    //...
}
```

## 3.1 Itérateur sur la liste doublement chaînée

Création d'un itérateur sur liste doublement chaînée. Écrire les fonctions suivantes :

**creerIte** Création d'un itérateur sur une LDC et place l'itérateur en début de liste

**next, previous** Déplace l'itérateur sur le suivant/précédent et renvoie la valeur courant d'avant le déplacement

**hasNext, hasPrevious** Renvoie 1 si l'élément suivant/précédent existe

**begin, end** Place l'itérateur sur le début/fin de la liste.

```
1 typedef struct etIte* Iterateur;
2
3 Iterateur creerIterateur(LDC);
4 Element next(Iterateur itereur);
5 Element previous(Iterateur itereur);
6 int hasNext(Iterateur itereur);
7 int hasPrevious(Iterateur itereur);
```

Listing 3.1 – Itérateur sur Liste – Header

```
1 #include "iterateur.h"
2
3 typedef struct etIte {
```

```
4   LDC l;  
5   Cell* cour;  
6 } etIterateur;  
7  
8  
9 Iterateur creerIterateur(LDC liste) {  
10    Iterateur ite;  
11    ite = (Iterateur) malloc(sizeof(etIterateur));  
12    assert(ite != NULL);  
13    ite->l = liste;  
14    ite->cour = liste->debut;  
15  
16    return ite;  
17 }  
18  
19 Element next(Iterateur iterateur) {  
20    Element ret = iterateur->cour->val;  
21    assert(hasNext);  
22    iterateur = iterateur->cour->suivant;  
23  
24    return ret;  
25 }  
26  
27 Element previous(Iterateur iterateur) {  
28    Element ret = iterateur->cour->val;  
29    assert(hasPrevious);  
30    iterateur->cour = iterateur->cour->precedent;  
31  
32    return ret;  
33 }  
34 int hasNext(Iterateur iterateur) {  
35    return (iterateur->cour != iterateur->l->fin);  
36 }  
37 int hasPrevious(Iterateur iterateur) {  
38    return (iterateur->cour != iterateur->l->debut);  
39 }
```

Listing 3.2 – Itérateur sur Liste – Implémentation

---

# Liste des codes sources

---

2.1	File – Axiones . . . . .	12
2.2	Element – Prototype <code>comparer</code> . . . . .	14
3.1	Iterateur sur Liste – Header . . . . .	16
3.2	Iterateur sur Liste – Implémentation . . . . .	16
A.1	Syntaxe de déclaration d'un pointeur . . . . .	16
A.2	Exemple de déclaration . . . . .	16
A.3	Syntaxe utilisation d'un pointeur . . . . .	16
A.4	Exemple d'utilisation d'un pointeur . . . . .	16
A.5	Exemple d'utilisation de la constante <code>NULL</code> . . . . .	16
A.6	Syntaxe d'allocation dynamique . . . . .	17
A.7	Exemple d'allocation dynamique . . . . .	17
A.8	Syntaxe de libération de mémoire . . . . .	18
A.9	Déclaration d'un pointeur de fonction . . . . .	18
A.10	Utilisation d'un pointeur de fonction . . . . .	18
A.11	Exemple d'utilisation d'un pointeur de fonction . . . . .	18
D.1	Pointeurs – Exercice 1 . . . . .	22
D.2	Pointeurs – Exercice 2 . . . . .	22
D.3	Pointeurs – Exercice 3 . . . . .	23
D.4	pointeurs – Exercice 4 . . . . .	23

---

## Table des figures

---

2.1	Pile avec une liste simplement chaînée . . . . .	10
2.2	Pile avec une liste doublement chaînée . . . . .	11
2.3	File avec une liste simplement chaînée . . . . .	12
2.4	File avec une liste doublement chaînée . . . . .	13
2.5	Liste doublement chaînée . . . . .	15