Université Toulouse III – Paul sabatier
L2 Informatique

Structures de données

Semestre 4

Toulouse, le 20 février 2013

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 ecrit 25%

Devoir TP 25 %

Table des matières

Chapitre 3

Parcourir une collection

Un itérateur est une structure de donnés 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 chainée. Écrire les fonctions suivantes :

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

`next` Déplace l'itérateur sur le suivant renvoie la valeur courant d'avant le déplacement

`previous` Déplace l'itérateur sur le précédent et renvoie la valeur d'avant le courant

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

`begin, end` Plate l'itérateur sur le début/fin de la liste.

```
1 typedef struct etIte* Iterateur;
2
3 Iterateur creerIterateur(LDC);
4 Element next(Iterateur iterateur);
5 Element previous(Iterateur iterateur);
6 void begin(Iterateur iterateur);
7 void end(Iterateur iterateur);
8 int hasNext(Iterateur iterateur);
9 int hasPrevious(Iterateur iterateur);
```

Listing 3.1 – `Iterateur` sur `Liste` – Header

```c
#include "iterateur.h"

typedef struct etIte {
  LDC l;
  Cell* cour;
} etIterateur;


Iterateur creerIterateur(LDC liste) {
  Iterateur ite;
  ite = (Iterateur) malloc(sizeof(etIterateur));
  assert(ite != NULL);
  ite->l = liste;
  ite->cour = liste->debut;

  return ite;
}

Element next(Iterateur iterateur) {
  Element ret = iterateur->cour->val;
  assert(hasNext);
  iterateur = iterateur->cour->suivant;

  return ret;
}

Element previous(Iterateur iterateur) {
  Element ret;
  assert(hasPrevious);
  if(iterateur->cour != NULL)
    iterateur->cour = iterateur->cour->precedent;
  else
    iterateur->cour = iterateur->l->fin;

  ret = iterateur->cour->val;

  return ret;
}
void begin(Iterateur iterateur) {
  iterateur->cour = iterateur->l->debut;
}
void end(Iterateur iterateur) {
  iterateur->cour = NULL;
}
int hasNext(Iterateur iterateur) {
  return (iterateur->cour != NULL);
}
int hasPrevious(Iterateur iterateur) {
  return (iterateur->cour != iterateur->l->debut);
}
```

Listing 3.2 – `Iterateur` sur `Liste` – Implémentation

Annexe B

Liste des codes sources

Annexe C

Table des figures