Université de Strasbourg
UFR de mathématique et d'informatique

L2 informatique
quinquennal 2018-2023

Structures de données et algorithmes

TP Listes en chaînage double et itérateurs

Introduction

L'objectif de ce TP est de maîtriser parfaitement les listes doublement chaînées et circulaires en limitant la dépendance à l'implantation aux opérations du noyau.

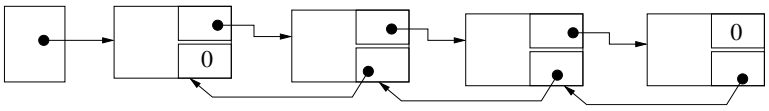
Double chaînage

1. Expliquez pourquoi il n'est pas efficace de parcourir en sens rétrograde une liste implantée avec un simple chaînage.

   On utilise un double chaînage en définissant la structure de données suivante :

```
typedef struct sldc {
                        S v;              // valeur
                        struct sldc *s;   // suivant
                        struct sldc *p;   // précédent
                       } StrListe, *Liste ;
```

   pour implanter des objets de ce genre :



2. Expliquez l'intérêt de rendre circulaire une liste implantée avec un double chaînage.
3. Implantez les opérations standard définissant le type Liste : liste vide, accès, ajout et suppression en tête, longueur, test de vacuité et d'appartenance.
4. Implantez les opérations en i-ème position sur le type Liste : accès, insertion, suppression et changement.
5. Faites un makefile et écrivez un programme de test avec trois listes différentes en utilisant toutes les opérations : vide, adjonction en tête, insertion, suppression ....

Itérateur

Lorsqu'on veut faire un parcours efficace avec cette structure de données, il faut obligatoirement garder un pointeur pt sur une cellule et passer au suivant ou au précédent en faisant pt = pt->s ou pt = pt->p. La visibilité des structures de données n'est pas souhaitable si ultérieurement nous souhaitons en changer. Pour cacher ceci, nous proposons d'utiliser un nouveau type de données :

```
typedef struct {
                    Liste first; // premier élément de la liste
                    Liste cur;   // élément courant
                    Bool fin;    // vaut vrai si cur est sur le dernier élément
} *Iter;
```

Ainsi, avec les opérateurs ci-dessous :

```
Iter init(Liste L);
Bool end(Iter i);
Iter next(Iter i); // mise à jour du Bool fin en cas de fin
Iter prev(iter i);
S value(Iter i);
```

On pourra ainsi écrire des choses du genre :

```
Iter i;
for(i = init(L1); !end(i); next(i)).......
```

1. Programmez les fonctions indiquées sur la structure de données Iter.
2. En les utilisant, programmez une fonction qui cherche un élément dans une liste et testez-la.
3. Tester si la macro suivante :

   ```
   #define foreach(V,L) Iter _I = init(L); for( V = value(_I) ; ! end(_I);  next(_I), V = value(_I))
   ```

   permet d'écrire des choses comme :

   ```
   foreach(v, t) printf("%d",v);
   ```