Travaux Dirigés d'algorithmique n°8

Cours d'Informatique de Deuxième Année

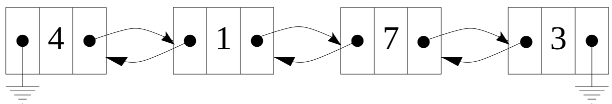
—DEUG Sciences S3-MI—

Complément sur les listes

L'objet de cette feuille d'exercices est l'extension de la notion de liste chaînée. On abordera notamment les listes doublement chaînées. Ce double chaînage signifie que l'on ne parle plus simplement du "suivant" d'une cellule mais également du "précédent".

▸ Exercice 1. *Liste doublement chaînée*

Une liste doublement chaînée est une liste dont chaque élément possède un pointeur vers l'élément suivant et un pointeur vers l'élément précédent. Par exemple :



Chaque élément de la liste est défini en C par :

```
struct cellule
       { int val;
         struct cellule * suivant;
         struct cellule * precedent; };
typedef struct cellule * Ptrcellule;
```

Définir en C, les fonctions suivantes :

- `Ptrcellule InitialiserListe(void)`*;*
- `Ptrcellule InsererDebut(int v, Ptrcellule ptrdebut)` *: insertion d'un élément de valeur* `v`*, en tête de liste référencée par* `ptrdebut`*;*
- `void InsererApres(int v, Ptrcellule ptrcell)` *: insertion d'un élément de valeur* `v`*, après l'élément positionné à l'adresse* `ptrcell`*;*
- `void SupprimerSuivant(Ptrcellule ptrcell)` *: supression de la cellule située après la cellulle référencée par* `ptrcell`*.*

Pour les trois dernières fonctions, on fera attention à conserver un chaînage cohérent pour toutes les cellules de la liste chaînée.

▸ Exercice 2. Manipulation de listes chaînées circulaires
Une liste chaînée circulaire est une liste dont le dernier élément pointe sur le premier. En utilisant la représentation chaînée suivante :

```
typedef struct cellule {
                  int val;
                  struct cellule *suivant;
                } Strcellule;

typedef Strcellule * PtrCellule;

typedef struct deux_pointeurs {
                  PtrCellule premier;  /* pointeur sur le premier element */
                  PtrCellule dernier;  /* pointeur sur le dernier element */
                } Liste;
```

écrire deux fonctions qui :

1. *insère un élément en tête de liste ;*
2. *retire lélément en tête de la liste et retourne cette valeur.*

▸ Exercice 3. Problème de Josephus
Supposons que n personnes soient assises en cercle et que l'on souhaite éliminer ces personnes selon un certain ordre. Cet ordre est établi en comptant m personnes (pour $m \leq n$) à partir d'une personne initiale, en retirant la m-ième personne ainsi désignée et en réitérant le processus à partir de ce point dans le cercle. L'ordre dans lequel les personnes sont ainsi éliminées définit ce que l'on appelle la permutation de Josephus (n, m) pour la suite ordonnée (1,2, ..., n). Par exemple, la permutation de Josephus (7, 3) est égale à (3, 6, 2, 7, 5, 1, 4). Ecrire un algorithme en $O(nm)$ qui donne la permutation de Josephus (n, m). Utiliser pour cela une liste circulaire chaînée comme celle indiquée plus haut.