Les listes chaînées

Définition

Structure de données allouée dynamiquement composée d'éléments liés entre eux soit par un pointeur avant (liste chaînée simple) soit par un pointeur avant et un pointeur arrière (liste chaînée double).

Avantages

L'insertion d'un nouvel élément en milieu de liste se fait efficacement (comparée à un tableau).
De même, la destruction d'un élément en milieu de liste se fait aussi efficacement.

Schéma

Supposons les données suivantes: 5, 3, 9, 2

A) En ordre FIFO (First In First Out)

Schéma utilisé dans l'implémentation des files d'attente.

Liste =====> 5 >=======>3 >======>9 >========> 2 X

B) En ordre LIFO (Last In Last Out)

Schéma utilisé dans l'implémentation d'une pile.

Liste =====> 2 >=======>9 >======>3 >========>5 X

Déclaration d'une liste chaînée

```
typedef struct elem* PtrElem;
typedef struct elem
{
    int         valeur;
    PtrElem     suivant;
} Elem;

PtrElem    liste;
```

Création d'une liste chaînée

A) En ordre FIFO (First In First Out)

```
PtrElem CreationFIFO (FILE* fLect)
{
    PtrElem cour, prec;
    PtrElem tete = NULL;

    while (!feof (fLect))
    {
        cour = (PtrElem) malloc (sizeof (Elem));
        fscanf (fLect, "%d", &cour->valeur);
        if (!tete) tete = cour;
        else prec->suivant = cour;
        prec = cour;
    }
    if (tete) prec->suivant = NULL;
    return tete;
}
```

B) En ordre LIFO (Last In Last Out)

```
PtrElem CreationLIFO (FILE* fLect)
{
    PtrElem cour, dernier = NULL;

    while (!feof (fLect))
    {
        cour = (PtrElem) malloc (sizeof (Elem));
        fscanf (fLect, "%d", &cour->valeur);
        cour->suivant = dernier;
        dernier = cour;
    }
    return dernier;
}
```

Recherche d'un élément dans une liste chaînée

A) Pour une liste non triée

```
PtrElem ChercherNT (PtrElem liste, int val, PtrElem* prec)
{
    *prec = NULL;
    while (liste && liste->valeur != val)
    {
        *prec = liste;
        liste = liste->suivant;
    }
    return liste;
}
```

B) Pour une liste triée en ordre croissant

```
PtrElem ChercherT (PtrElem liste, int val, PtrElem* prec)
{
    *prec = NULL;
    while (liste && liste->valeur <= val)
    {
        if (liste->valeur == val) return liste;
        else
        {
            *prec = liste;
            liste = liste->suivant;
        }
    }
    return NULL;
}
```

Insertion d'un élément dans une liste chaînée

A) Pour une liste non triée

1. FIFO

```
PtrElem InsererFIFO (PtrElem* tete, int val, PtrElem prec)
{
    PtrElem cour = (PtrElem) malloc (sizeof (Elem));
    cour->valeur = Val;
    if (!prec)
    {
        cour->suivant = *tete;
        *tete = cour;
    }
    else
    {
        cour->suivant = prec->suivant;
        prec->suivant = cour;
    }
    return cour;
}

fscanf (..., "%d", &val);
if (! (cour = ChercherNT (tete, val, &prec))) cour = InsererFIFO (&tete, val, prec);
```

2. LIFO

```
PtrElem InsererLIFO (PtrElem* tete, int val)
{
    PtrElem cour = (PtrElem) malloc (sizeof (Elem));
    cour->valeur = val;
    cour->suivant = *tete;
    *tete = cour;
    return cour;
}

fscanf (..., "%d", &val);
if (! (cour = ChercherNT (tete, val, &prec))) cour = InsererLIFO (&tete, val);
```

B) Pour une liste triée

Même chose que la fonction InsererFIFO.

```
fscanf (..., "%d", &val);
if (! (cour = ChercherT (tete, val, &prec))) cour = Inserer (&tete, val, prec);
```

Destruction d'un élément dans une liste chaînée

```
void Detruire (PtrElem* tete, PtrElem prec, PtrElem cour)
{
    if (!prec) *tete = (*tete)->suivant;
    else prec->suivant = cour->suivant;
    free (cour);
}
```

A) Pour une liste non triée

1. FIFO

```
fscanf (...,"%d", &val);
if (cour = ChercherNT (tete, val, &prec)) Detruire (&tete, prec, cour);
```

2. LIFO

```
fscanf (...,"%d", &val);
if (cour = ChercherNT (tete, val, &prec)) Detruire (&tete, prec, cour);
```

B) Pour une liste triée

```
fscanf (...,"%d" , &val);
if (cour = ChercherT (tete, val, &prec)) Detruire (&tete, prec, cour);
```

Destruction d'une liste chaînée (tous les éléments)

A) Façon incorrecte

```
PtrElem DetruireListe (PtrElem tete)
{
    while (tete)
    {
        free (tete);
        tete = tete->suivant;
    }
    return NULL;
}
```

B) Façon correcte

```
PtrElem DetruireListe (PtrElem tete)
{
    PtrElem tempo;
    while (tete)
    {
        tempo = tete->suivant;
        free (tete);
        tete = tempo;
    }
    return NULL;
}
```