Listes linéaires chaînées

D.E ZEGOUR

École Supérieure d'Informatique

ESI

Listes linéaires chaînées

Exemple d'introduction

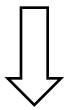
Trouver tous les nombres premiers de 1 a n et les stocker en mémoire

n : donnée à lire à l'exécution

Problème : choix de la structure d'accueil

Si tableau : pas possible de définir sa taille

Si tableau : pas possible de définir sa taille avec précision même si n est connu



Réservation de l'espace doit être dynamique.

Listes linéaires chaînées

Notions d'allocations statique et dynamique

Allocation statique

L'allocation de l'espace se fait tout à fait au début d'un traitement.

En termes techniques, on dit que l'espace est connu à la compilation.

C'est donc la notion de tableau.

Allocation dynamique

L'allocation de l'espace se fait au fur et à mesure de l'exécution du programme.

L'utilisateur doit disposer des deux opérations : allocation et libération de l'espace.

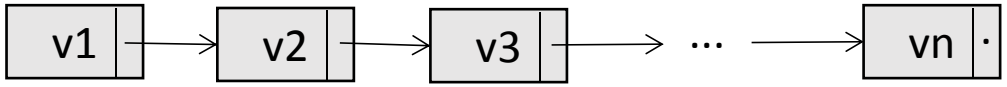
Si le langage offre ces possibilités, on les utilise directement

Sinon, les simuler, c'est à dire gérer l'espace soi-même dans un grand tableau.

Listes linéaires chaînées

Définition

Une liste linéaire chaînée (Llc) est un ensemble de maillons alloués dynamiquement chaînés entre eux



Tete de liste

Un élément d'une Llc est toujours une structure (objet composé) avec deux champs :
- Champ Valeur : contenant l'information
- Champ Adresse : donnant l'adresse du prochain maillon

A chaque maillon est associée une adresse

On introduit ainsi une nouvelle classe d'objet: le type POINTEUR

Une Llc est caractérisée par l'adresse de son premier élément.

NIL constitue l'adresse qui ne pointe aucun maillon.

Listes linéaires chaînées

Machine abstraite

Allouer(P)	Allocation d'un maillon. L'adresse est rendue dans la variable P
Libérer(P)	libération de l'espace pointé par P
Valeur(P)	Accès au champ Valeur du maillon d'adresse P
Suivant(P)	Accès au champ Adresse du maillon d'adresse P
Aff_Adr(P, Q)	Affecter l'adresse Q dans le champ Adresse du maillon d'adresse P
Aff_Val(P,Val)	Affecter la valeur Val dans le champ Valeur du maillon d'adresse P

Listes linéaires chaînées

Solution au problème d'introduction

On utilise les deux faits suivants :

Un nombre n'est premier que s'il n'est pas divisible par les nombres premiers qui le précèdent

Tous les nombres premiers sont de la forme 6m+1 ou 6m-1.

Listes linéaires chaînées

Solution au problème d'introduction

{ Création de maillons pour les nombres premiers 2 et 3 }

Allouer(T,P) ; Aff_Val(P,2) ; Tête := p
Allouer(T, Q)
Aff_Val(Q,3) ; Aff_Adr(Q, NIL)
Aff_Adr(P,Q)
P := Q
M := 1 ; Continue := VRAI ; Aig := VRAI

{ Création des maillons pour les nombres premiers supérieurs à 5}
TANTQUE Continue
    Gen_nombre (M, Aig, Nombre)
    Aig := NON Aig
    SI Aig : M := M + 1 FSI
        SI Nombre <= N
            SI Premier (Tête, Nombre)
                Allouer (T, Q); Aff_Val(Q, Nombre) ;
                Aff_Adr(Q, NIL) ; Aff_Adr(P, Q)
                P := Q
            FSI
        SINON
            Continue := FAUX
        FSI
FINTANTQUE

Listes linéaires chaînées

Solution au problème d'introduction

Module Gen_nombre(M, Aig, Nombre)

```
SI Aig : Nombre := 6M - 1
SINON Nombre := 6M + 1 FSI
```

Divisible (A, B)

```
Q := A / B {Division entière}
Divisible := Q.B = A
```

Premier (L, N)

```
P:= L ; Trouv := FAUX
TANTQUE P <> NIL ET NON Trouv :
      SI Divisible ( Nombre, Valeur(P) ) :
                  Trouv := VRAI
            SINON
                  P := Suivant(P)
      FSI
FINTANTQUE
Premier := NON Trouv
```

Listes linéaires chaînées

Algorithmes sur les listes

De même que sur les vecteurs, on peut classer les algorithmes sur les Llcs comme suit :

- Parcours :accès par valeur, accès par position

- Mise à jour : insertion, suppression

- Algorithmes sur plusieurs Llcs : fusion, interclassement, éclatement,...

- Tri sur les Llcs

Listes linéaires chaînées

Listes linéaires chaînées particulières

Liste bidirectionnelle : C'est une Llc que l'on peut parcourir dans les deux sens.

$M_{Llcb} = M_{Llc} - \{Aff\_Adr\} + \{ Aff\_Adrg, Aff\_Adrd, Précédent \}$

Le modèle des Llc est donc étendu par les opérations suivantes :

Précédent(P)	Accès au champ adresse gauche du maillon d'adresse P.
Aff_Adrg(P, Q)	Dans le champ Adresse1 (adresse gauche) du maillon d'adresse P, on range l'adresse Q.
Aff_Adrd(P, Q)	Dans le champ Adresse2 (adresse droite) du maillon d'adresse P, on range l'adresse Q

Listes linéaires chaînées

Listes linéaires chaînées particulières

Suppression d'un élément pointé par P dans une liste linéaire chaînée bidirectionnelle L.

```
SI P # NIL :
   SI Précédent(P) # NIL
         Aff_Adrd( Précédent(P), Suivant(P) )
      SINON
         L := Suivant (P)
   FSI
   SI Suivant(P) # NIL :
         Aff_Adrg( Suivant(P), Précédent(P) )
   FSI
   Libérer(P)
FSI
```

Listes linéaires chaînées

Listes linéaires chaînées particulières

Liste circulaire ou anneau

C'est une Llc telle que le dernier élément pointe sur le premier. Elle est définie par l'adresse d'un élément quelconque.

$M_{Llcc} = M_{Llc}$

Listes linéaires chaînées

Listes linéaires chaînées particulières

Liste bidirectionnelle circulaire

C'est une Llc à double sens et dont le dernier(premier) pointe sur le premier(dernier).

$M_{Llcbc} = M_{Llcb}$

Listes linéaires chaînées

Implémentation : Dynamique (C)

```
/** Implémentation **\: LISTE DE ENTIERS**/
typedef int Typeelem_Li   ;
typedef struct Maillon_Li * Pointeur_Li ;

struct Maillon_Li
 {
   Typeelem_Li  Val ;
   Pointeur_Li Suiv ;
 } ;
```

Listes linéaires chaînées

Implémentation : Dynamique (C)

```
Pointeur_Li Allouer_Li (Pointeur_Li *P)
   {
     *P = (struct Maillon_Li *) malloc( sizeof( struct Maillon_Li)) ;
     (*P)->Suiv = NULL;
   }

void Liberer_Li ( Pointeur_Li P)
   { free (P);}
```

Listes linéaires chaînées

Implémentation : Dynamique (C)

```
Pointeur_Li Suivant_Li(  Pointeur_Li P)
   { return( P->Suiv ) ;  }

Typeelem_Li Valeur_Li( Pointeur_Li P)
   { return( P->Val) ; }
```

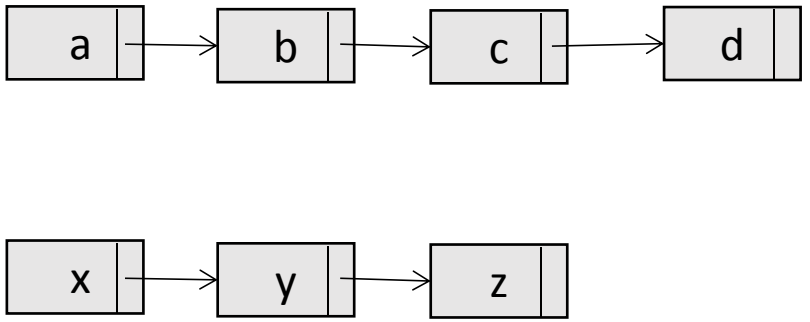
```
void Aff_val_Li(Pointeur_Li P, Typeelem_Li Val)
   {
     P->Val = Val ;
   }

void Aff_adr_Li( Pointeur_Li P,  Pointeur_Li Q)
   {
     P->Suiv = Q ;
   }
```

Listes linéaires chaînées

Implémentation : Statique (C)

Le tableau est un ensemble de triplets
(Element, Suivant, Occupe)



0	y	6	V
1	d	-1	V
2	b	5	V
→ 3	a	2	V
→ 4	x	0	V
5	c	1	V
6	z	-1	V
			F
			F
Max -1			

Listes linéaires chaînées

Implémentation : Statique (C)

Plusieurs listes dans un même tableau.

Le champ "Occupe" est nécessaire pour les opérations Allouer et Libérer.

Une phase d'initialisation est obligatoire avant l'utilisation de ce tableau.

Donc le tableau est global.

Une liste est définie par l'indice de son premier élément.

```
#define Max 100
#define True 1
#define False 0
#define Nil -1

typedef int Bool;
typedef int Typeqq;
struct Typeliste
 {
  Typeqq Element ;
   int Suivant;
   Bool Occupe;
} ;
```

```
struct Typeliste
Liste[ Max ];
```

Listes linéaires chaînées

Implémentation : Statique (C)

```
void Allouer ( int *I )
  {
    Bool Trouv;
    *I = 0;
    Trouv = False;
    while ( *I < Max && !Trouv )
     if ( Liste[*I].Occupe )
       *I++  ;
     else
       Trouv = True;
     if  ( !Trouv )  *I = -1;

  }
```

```
void Liberer ( int I )
   {
     Liste[I].Occupe = False ;
   }
```

Listes linéaires chaînées

Implémentation : Statique (C)

```
Typeqq Valeur ( int I )
  {
    return( Liste[I].Element );
  }

 int Suivant ( int I )
  {
    return ( Liste[I].Suivant ) ;
  }
```

```
void Aff_val ( int I, Typeqq Val)
  {
    Liste[I].Element = Val;
  }

void Aff_adr ( int I, int J)
  {
    Liste[I].Suivant = J;
  }
```

Listes linéaires chaînées

Implémentation : Dynamique (Pascal)

```
{ Implémentation : LISTE DE ENTIERS}

 { Listes lin?aires cha?n?es }
 TYPE
   Typeelem_LI = INTEGER;
   Pointeur_LI = ^Maillon_LI;
   Maillon_LI = RECORD
      Val : Typeelem_LI;
      Suiv : Pointeur_LI
   END;
```

```
PROCEDURE Allouer_LI ( VAR P : Pointeur_LI ) ;
  BEGIN  NEW(P)  END;

 PROCEDURE Liberer_LI ( P : Pointeur_LI ) ;
  BEGIN  DISPOSE(P)  END;
```

Listes linéaires chaînées

Implémentation : Dynamique (Pascal)

```
FUNCTION Suivant_LI( P : Pointeur_LI) : Pointeur_LI;
  BEGIN  Suivant_LI := P^.Suiv END;

FUNCTION Valeur_LI (P : Pointeur_LI) : Typeelem_LI;
  BEGIN   Valeur_LI := P^.Val  END;
```

```
PROCEDURE Aff_val_LI(P : Pointeur_LI; Val : Typeelem_LI );
  BEGIN P^.Val := Val  END;

PROCEDURE Aff_adr_LI( P, Q : Pointeur_LI ) ;
  BEGIN P^.Suiv := Q    END;
```

Listes linéaires chaînées

Implémentation : Statique (pascal)

Plusieurs listes dans un même tableau.

Le tableau est un ensemble de triplets (Element, Suivant, Occupe)

Le champ "Occupe" est nécessaire pour les opérations Allouer et Libérer.

Une phase d'initialisation est obligatoire avant l'utilisation de ce tableau.

Donc le tableau est global.

Une liste est définie par l'indice de son premier élément.

```
CONST Max = 100;
TYPE Typeqq = INTEGER;
TYPE Typeliste = RECORD
  Element: Typeqq ;
  Suivant : INTEGER;
  Occupe : BOOLEAN
END;

{ Le tableau }
VAR
  Liste : ARRAY[1..Max ] OF Typeliste;
```

Listes linéaires chaînées

Implémentation : Statique (pascal)

```
{ initialisation }
 PROCEDURE Init;
  VAR
   I : INTEGER;
  BEGIN
   FOR I:= 1 TO Max DO
     Liste[I].Occupe := FALSE;
  END;
```

```
PROCEDURE Allouer ( VAR I: INTEGER  );
  VAR
   Trouv :BOOLEAN;
  BEGIN
   I:= 1;
   Trouv := FALSE;
   WHILE ( (I <= Max) AND  NOT Trouv ) DO
    IF Liste[I].Occupe
    THEN  I := I + 1
    ELSE  Trouv := TRUE;
   IF  NOT Trouv THEN I := -1;
  END;
```

```
PROCEDURE Liberer ( I:INTEGER );
  BEGIN
   Liste[I].Occupe := FALSE ;
  END;
```

Listes linéaires chaînées

Implémentation : Statique (pascal)

```
FUNCTION Valeur ( I:INTEGER ) : Typeqq;
  BEGIN
    Valeur := Liste[I].Element;
  END;

 FUNCTION Suivant ( I:INTEGER ) : INTEGER;
  BEGIN
    Suivant :=  Liste[I].Suivant  ;
  END;
```

```
PROCEDURE Aff_val ( I:INTEGER; Val :Typeqq );
  BEGIN
    Liste[I].Element := Val;
  END;

 PROCEDURE Aff_adr (I:INTEGER; J: INTEGER);
  BEGIN
    Liste[I].Suivant := J;
  END ;
```