Algorithmique et Structures de Données

Akkouche Samir :
E-mail : samir.akkouche@univ-lyon1.fr

Pronost Nicolas:
E-mail : nicolas.pronost@univ-lyon1.fr

LISTE D'ELEMENTS

Définition d'une liste

Liste = <> -- VIDE

Liste = < Premier(*Elément*) | Reste(*Liste*) >

Structure de données à accès séquentiel

Opérations sur les Listes

- InitialiserListe //La liste est vide
- Listevide // teste si la liste est vide (renvoie vrai après l'initialisation)
- PremierElement // renvoie la valeur du premier élément de la liste
- AjouterenTete // Le nouvel élément devient le premier élément de la liste.
- SupprimerElement //Supprime un element

Comment gérer en mémoire une liste?

On a besoin de repérer le premier élément de la liste

On a besoin de repérer la place libre pour stocker les nouveaux arrivants

Tableau?

Structure?

Autre?

Stockage dans un tableau simple

premier = 0 | reste = 1 | dernier = 5

0	1	2	3	4	5	6	7	8	9	10
12	34	6	14	78	-------	-------	--------	-------	-------	-------

Opérations sur les Listes:

- Initialiser : par exemple dernier = 0 ou premier = -1
- Listevide : renvoie vrai si dernier = 0
- PremierElement : renvoie la valeur en position 0
- AjouterenTete : Tous les éléments sont décalés à droite d'une position, dernier = dernier +1 , le nouvel élément est mis en 0.
- SupprimerElement : Repérer l'élément et décaler à gauche à partir de la position suivante et dernier = dernier -1

Stockage dans un tableau simple

Avantages : Le premier élément est toujours en 0, le reste en 1. Il faut simplement repérer la première place libre

Inconvénient : On fait beaucoup de décalages

Si on veut éviter de décaler les éléments :

- Il faut laisser des « trous » lorsqu'on supprime par exemple un élément
  - Comment gérer l'existence de ces trous???
- Il faut repérer la position du premier élément et du reste
- Il faut repérer la position de la première place libre pour y insérer un nouvel élément

⇒ Deux solutions :

⇒ Stocker dans un tableau et tout gérer à la main : Place occupée et place libre

⇒ Utiliser le tas pour ne pas avoir à gérer la place libre

Stockage dans un tableau : autre solution

```
Cellule: structure
        info : Element
        suivant : entier
    fin structure
```

```
Liste: structure
        premier : entier
        place_libre : entier
        t : tableau [1..10000] de Cellule
    fin structure
```

Exemple de liste l : Liste

l.premier = 0

l.place_libre = 1

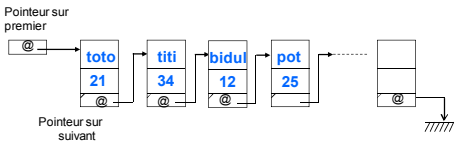
0	1	2	3	4	5	6	7	8	9	10
12	-------	-------	14	78	-------	-------	6	-------	-------	-------
3	2	5	7	-1	6	8	4	9	10	-1

Opérations sur les Listes (suite)

- AjouterenQueue// Le nouvel élément est en fin de liste
- RechercherunElement// renvoie « l'emplacement » éventuel de l'élément
- AfficherListe // Parcourt la liste et affiche les éléments
- …

LES LISTES CHAINEES

- **Ensemble d'objets rangés linéairement**
- **Contrairement aux tableaux, l'accès se fait à l'aide de pointeurs**
- **Chaque objet contient un ensemble d'informations dont un pointeur qui indique l'emplacement de l'objet suivant,**
- **Le pointeur qui désigne le premier objet est stocké à part.**
- **Le pointeur du dernier objet doit avoir un code spécial= NULL**

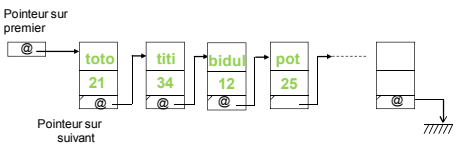


Représentation graphique d'une liste chaînée

- **Chaque objet contient un ensemble d'informations dont un pointeur qui indique l'objet suivant,**

Cellule = Structure
info : Element
suivant : pointeur
Fin structure

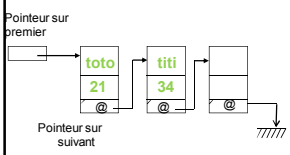
```
struct Cellule {
        Element info;
        Cellule * suivant;
};
```

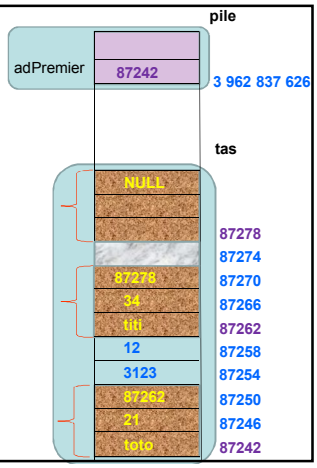


Représentation graphique d'une liste chaînée

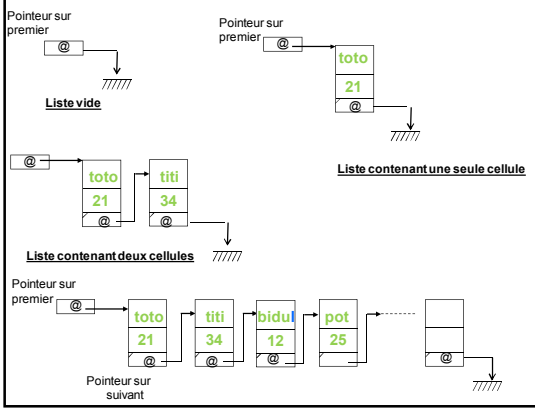
Particularité des Listes chainées

- Les Cellules sont créés dynamiquement dans le TAS
- Le pointeur qui désigne la première Cellule est stocké dans la PILE
- L'objectif est de créer et libérer selon les besoins



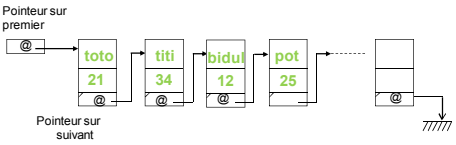


Evolution d'une liste



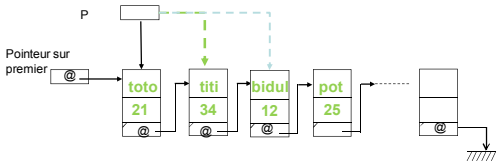
Traitements dans les listes chainées

- Création de Cellules
- Chaînage d'une cellule créée dans le TAS : En tête, en queue, ailleurs…
- Décrochage et libération d'une Cellule de la liste
- Parcours d'une liste : Affichage, recherche, réorganisation…



Manipulation d'une liste

- Opération classique : Parcours d'une liste,
  - affichage, recherche d'un élément….

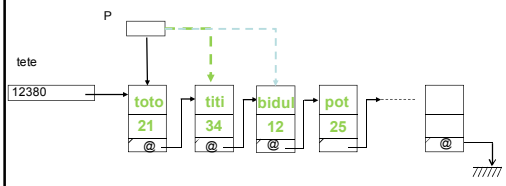


- Méthode :
  - Utiliser un pointeur de travail,
  - Le faire pointer sur la première cellule,
  - Le faire évoluer en passant de cellule en cellule.

Comment faire en pratique?

Manipulation d'une liste

- Supposons que le pointeur sur la première Cellule s'appelle tete et qu'il contient la valeur 12380



Cellule * p; // Déclaration du pointeur de travail

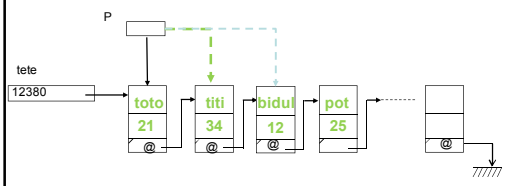
p = tete; // la variable p prend la valeur tete, maintenant p pointe sur la première //cellule

Grâce à ce pointeur de travail, on peut accéder aux champs contenant toto et 21

Manipulation d'une liste

- Comment passer à la cellule suivante ?

p = p->suivant; // On peut accéder maintenant aux champs contenant titi et 34



Pour parcourir toute la liste on procède de la façon suivante :

```
p=tete; // maintenant p contient la valeur 3 245 768 000

while (p != NULL)
  {
        //traitement …….
        p=p->suivant; //On passe à la cellule suivante.

  }
```

Implémentation des Listes chaînées

```
Type
classe Cellule
    info : Element
    suivant : pointeur sur Cellule
Fin classe

classe Liste
   adPremiere: pointeur sur Cellule
Fin classe
```

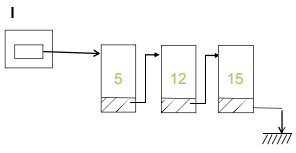
```
class Cellule
    {
        public:
          Element  info ;
          Cellule * suivant ;
    } ;

class Liste
   {
        public:
            Cellule * adPremiere;
   };
```

Implémentation des Listes chaînées (suite)

Chainage en tête

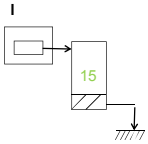
Exemple de création de liste chaînée

on veut créer une liste d'entiers (arrivés dans l'ordre suivant : 15,12,5)

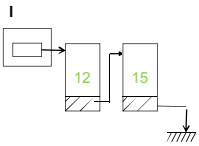


Etape 1: initialisation

Etape 2



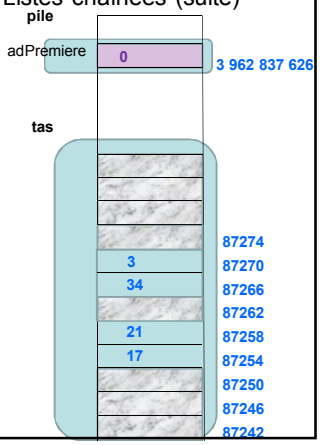
Etape 3



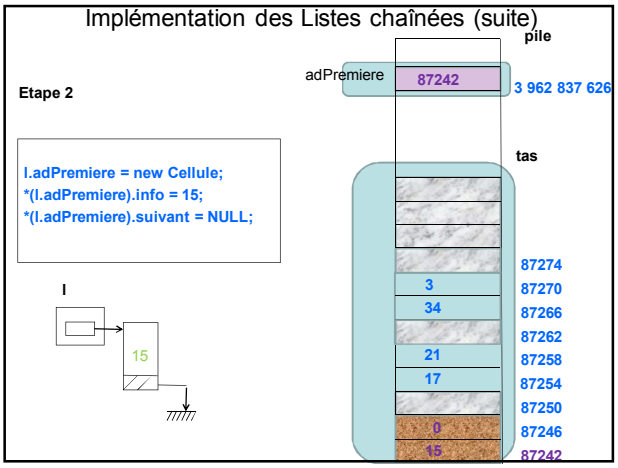
Implémentation des Listes chaînées (suite)

Etape 1: initialisation

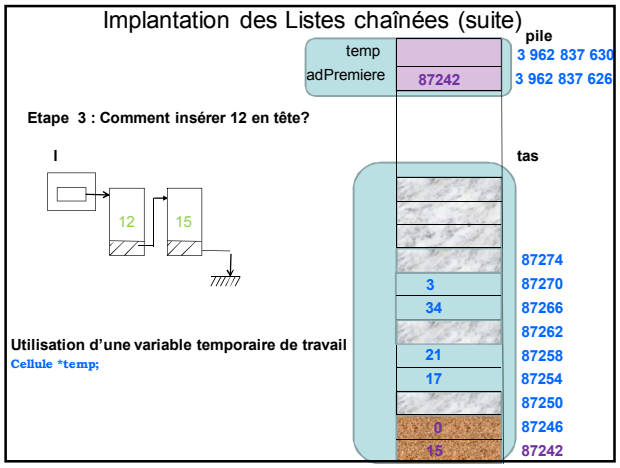
```
Liste :: Liste( )
{
   adPremiere = NULL;
}
----------------------------------------
Liste l;
```



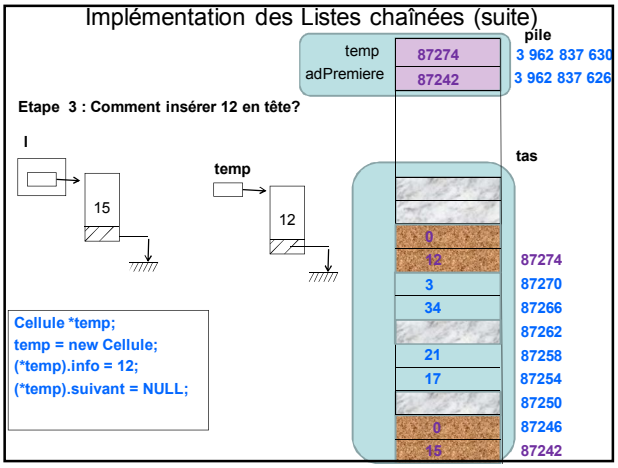
Implémentation des Listes chaînées (suite)



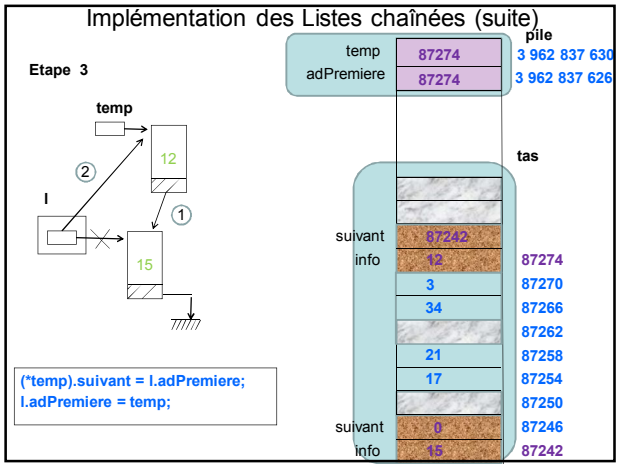
Implantation des Listes chaînées (suite)



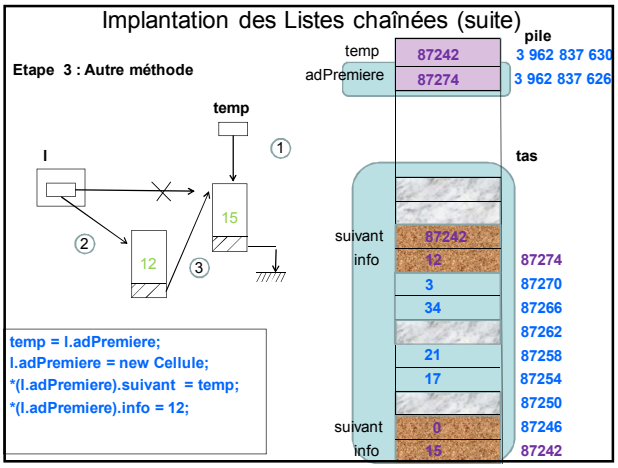
Implémentation des Listes chaînées (suite)



Implémentation des Listes chaînées (suite)



Implantation des Listes chaînées (suite)



TDA Liste

module Liste

importer **module** Element

exporter type **Cellule**, **Liste**

procedure initialiser(l : Liste)

Précondition : l n'est pas initialisée

Postcondition : l est une liste vide

Paramètre en mode donnée-résultat : l

procedure testament(l : Liste)

Précondition : l est bien initialisée

Postcondition : l est une liste vide

Paramètre en mode donnée résultat : l

----------A suivre----------

implantation

Finmodule

TDA Liste(suite)

module Liste(suite)

fonction estVide(l : Liste) : *Booleen*

Précondition : l est bien initialisée

Résultat : renvoie vrai si la liste est vide

Paramètre en mode donnée : l

fonction premierElement(l : Liste) : *Elemen*t

Précondition : l est bien initialisée

Résultat : renvoie l'élément situé en tête de liste

Paramètre en mode donnée : l

----------A suivre----------

implantation

Finmodule

TDA Liste (suite)

procedure ajouteEnTete(l : Liste, x : Element)

Précondition : l est bien initialisée

Postcondition : l'élément x est mis en tête de liste

Paramètre en mode donnée résultat : l

Paramètre en mode donnée : x

procedure ajouterEnQueue(l : Liste, x : Element)

Précondition : l est bien initialisée

Postcondition : l'élément x est mis en fin de liste

Paramètre en mode donnée résultat : l

Paramètre en mode donnée : x

----------A suivre----------

implantation

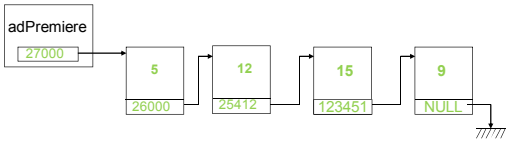
TDA Liste (suite)

procedure afficher(l : Liste)
Précondition : l est bien initialisée
Postcondition : rien
Paramètre en mode donnée : l

fonction recherche(l : Liste, x : Element) : *pointeur sur Cellule*
Précondition : l est bien initialisée
Résultat : renvoie l'adresse de la cellule contenant x et NULL si x n'existe pas
Paramètres en mode donnée : l et x

implantation

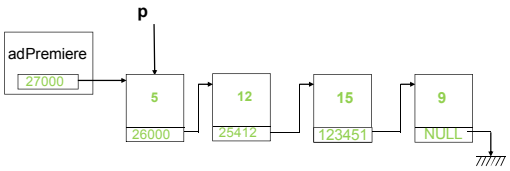
Libération d'une liste



Problème : Comment libérer proprement toutes ces cellules?

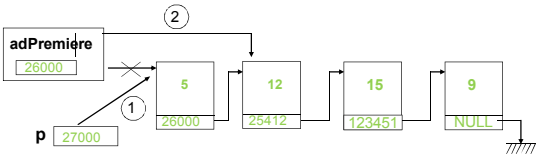
Libération d'une liste

Méthode :
1. Utiliser un pointeur de travail P
2. Décrocher, à l'aide de P, la Cellule de tête sans perdre la liste
3. Libérer la Cellule

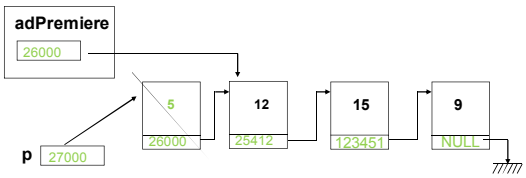


Libération d'une liste

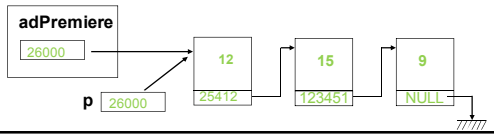
- Décrocher avec P la boite de tête sans perdre la liste
  - P ← l.adPremiere ①
  - l.adPremiere ← (l.adPremiere ↑.suivant) ②



Libération d'une liste



- Libérer la boite pointée par P :
  libere p → delete p;
- P ← l.adPremiere



Libération d'une liste

Algorithme :
Tant que l.adPremiere <> NULL faire
  p ← l.adPremiere
  l.adPremiere ← l.adPremiere ↑.suivant
  libere p
 fin tantque

```
while(l->adPremiere !=NULL)
   {
      p = l->adPremiere;
      l->adPremiere = (l-> adPremiere)→suivant;
    delete p;
   }
```