

Algorithmique et structures de données

CM 2

Pointeurs, structures récursives et listes chaînées

Plan du CM 2

Les structures

Les pointeurs

Les structures récursives

Les listes chaînées

Plan du cours

Les structures

Les pointeurs

Les structures récursives

Les listes chaînées

Types – rappels

Variables et types

Les données sont représentées par des **variables**

- toute variable est un **nom/identifiant** qui désigne un emplacement mémoire
- elle a un **type** qui détermine la place (le nombre d'octets) que cette variable occupe en mémoire

Types de base et types composés

- **types de base ou simples** (disponibles dans le langage)
 - booléen
 - entier
 - réel
 - caractère et chaîne de caractères
 - ...
- **types composés** (créés par l'utilisateur)
 - structure ou enregistrement
 - tableau
 - ...

Pour notre langage algorithmique, nous prendrons un typage statique

Enregistrements ou structures

Des types variés

Permet de regrouper des données avec des types différents

```
structure personne {  
    nom : chaîne de caractères  
    prénom : chaîne de caractères  
    âge : entier  
}
```

Les données sont appelées **champs** ou **attributs**.

Il n'y a pas d'ordre entre les attributs.

Déclaration et affectation

- **Déclaration d'une variable de type personne**

```
untel : personne
```

- **Affectation des attributs**

```
untel.prénom = "Pierre" ; untel.nom = "Quiroule" ; untel.âge = 20
```

Accès aux attributs

```
affichagepersonne(P : personne)  
    afficher P.nom ; afficher P.prénom ; afficher P.âge
```

Plan du cours

Les structures

Les pointeurs

Les structures récursives

Les listes chaînées

Les pointeurs – définition

A quoi sert un pointeur

- Un pointeur sert à pointer l'**emplacement de la mémoire** où se trouve un objet
- cet objet est appelé l'**objet pointé**
- le pointeur contient l'**adresse** de cet objet dans la mémoire

Constitution d'un pointeur

- Un pointeur a un **type** qui précise le **type de l'objet pointé**
- la **valeur** d'une variable pointeur est l'**adresse de l'objet pointé**

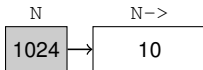
Les pointeurs – Notation de l'objet pointé

Notation de l'objet pointé

- si N est un pointeur, l'objet pointé est noté dans notre langage algorithmique $N- >$
- il est souvent noté $*n$ dans les langages de programmation

Exemple

N : pointeur sur entier



- l'adresse de N vaut 1024
- la valeur de $N- >$ vaut 10.

Les pointeurs – allocation (1)

Allocation

L'allocation de la mémoire pour la variable pointée peut se faire explicitement avec la commande **Nouveau(<Type>)**.

```
R : pointeur sur réel
```

```
R = Nouveau(réel) // R-> est créé et son adresse est donnée à R
```

```
R-> = 12.31
```

```
afficher R->
```

- la commande Nouveau(<Type>) entraîne une allocation de mémoire
- l'emplacement dépend du type

L'allocation est obligatoire

```
R : pointeur sur réel
```

```
R-> = 3.21
```

La dernière instruction entraîne une erreur car l'objet $R- >$ n'existe pas.

Les pointeurs – allocation (2)

Copie d'adresse

On peut également copier l'adresse d'un autre pointeur.

```
S : pointeur sur réel
```

```
S = R // le pointeur S contient la même adresse que R
```

```
afficher S->
```

```
S-> = 4.2
```

```
afficher R-> , S-> // on affiche 4.2 pour les deux pointeurs
```

- cela permet d'avoir un deuxième pointeur qui pointe sur un même objet
- il n'y a pas besoin d'effectuer une allocation de mémoire


Les pointeurs – allocation (3)

L'adresse None

- None signifie que le pointeur **ne pointe sur aucun objet**
- indispensable pour les **structures récursives**

N : pointeur sur entier

N = None

On le représente avec le schéma suivant 

Différence entre copie d'adresse et de valeur

Qu'affiche-t-on à la fin ?

A, B, C : pointeur sur entier

A = Nouveau(entier) ; A-> = 12

B = A

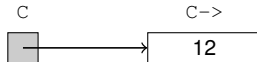
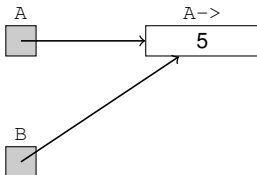
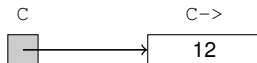
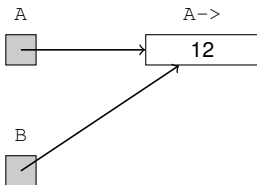
C = Nouveau(entier) ; C-> = A->

A-> = 5

afficher A->, B->, C->

Les pointeurs – allocation (4)

- A et B partagent la même adresse
- C possède sa propre adresse



Plan du cours

Les structures

Les pointeurs

Les structures récursives

Les listes chaînées

Les structures récursives (1)

Autoréférence

Une **structure** ou **enregistrement** est dite **récursive** si elle contient une ou plusieurs références (pointeurs) sur cette même structure.

Structure nœud

```
structure noeud {  
    valeur : entier // le type est fixé  
    suivant : pointeur sur noeud  
}
```

- Cela permet de construire le nombre de nœuds que l'on souhaite
- On utilise **None** pour définir le dernier nœud.

Accès à la valeur

```
p : pointeur sur noeud  
p = Nouveau(noeud)  
p->valeur = 10  
p->suivant = None
```

Simplification

On devrait écrire **p->.valeur** et **p->.suivant** au lieu de p->valeur et p->suivant

Les structures récursives (2)

Structures imbriquées

Il est possible d'imbriquer plusieurs structures entre-elles (cf composition en POO)

Définition d'une personne

```
structure personne {  
    nom : chaîne de caractères  
    prénom : chaîne de caractères  
    âge : entier  
}
```

Noeudpersonne

```
structure noeudPersonne {  
    valeur : personne // le type est fixé  
    suivant : pointeur sur noeud  
}
```

Les structures récursives (3)

Exemple avec une personne

```
p : pointeur sur noeudPersonne
p = nouveau(noeudPersonne)
p->valeur.nom = "Dupond"
p->valeur.prenom = "Jean"
p->valeur.âge = 37
unNoeud = p-> // p-> est un noeudPersonne
until = unNoeud.valeur // until est une personne
afficher until.age, p->valeur.age // on affiche 37 dans les deux cas
until.nom = "Dupont"
afficher p->valeur.nom // on affiche "Dupont"
```


Plan du cours

Les structures

Les pointeurs

Les structures récursives

Les listes chaînées

Listes chaînées – définition

Intérêt des structures récursives

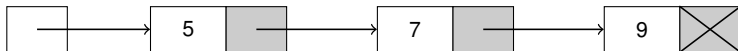
Les structures récursives ne sont intéressantes que si l'on manipule plusieurs objets.

- nombre d'objets non fixé au départ \neq tableaux
- gestion dynamique : insertion et suppression

Liste chaînée

Les éléments de la liste sont **chaînés** en utilisant la structure de nœud.

Exemple : une liste chaînée contenant les entiers 5, 7, 9.

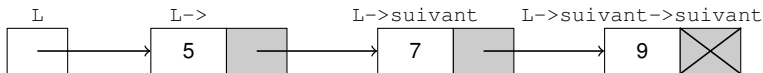


Type liste

```
type liste = pointeur sur nœud  
L1, L2 : liste
```

- il s'agit d'un **alias**
- nous pouvons indifféremment utiliser le terme *pointeur sur nœud* ou *liste*
- nous pouvons définir autant d'alias que souhaité

Listes chaînées – construction



Construction à la main d'une liste chaînée

```

L : pointeur sur noeud
L = Nouveau(noeud)
L->valeur = 5
L->suivant = Nouveau(noeud)
L->suivant->valeur = 7
L->suivant->suivant = Nouveau(noeud)
L->suivant->suivant->valeur = 9
L->suivant->suivant->suivant = None
  
```

- La méthode n'est pas pratique
- Nous souhaitons définir des procédures pour pouvoir construire une liste de taille quelconque

Listes chaînées – affichage

Affichage d'une liste

```
affichageListe(L : liste)
tant que L <> None faire
    afficher L->valeur
    L = L->suivant
```

Rappel : le passage de paramètre s'effectue par valeur.

```
L : liste
L = Nouveau(noeud) ; L->valeur = 5 ; L->suivant = None
affichageListe(L)
```

Après l'appel de la procédure *affichageListe*, la liste L reste inchangée.

Rappel : Alias entre pointeur sur nœud et liste

```
p = pointeur sur noeud
p = L
affichageListe(p)
```

Listes chaînées – insertion en début de liste

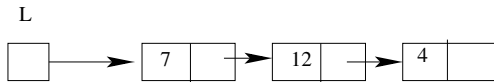
Insertion en début de liste

Variable tmp pour tampon, variable intermédiaire

```
insertionDebut(L : liste, n : entier)
    tmp : pointeur sur noeud    (1)
    tmp = Nouveau(noeud)        (1)
    tmp->valeur = n              (1)
    tmp->suivant = L             (2)
    retourner tmp                (3)
```

Le coût de l'opération est le même que soit la longueur de la liste.

Exemple, insertion de la valeur 8



Listes chaînées – insertion en début de liste

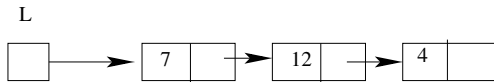
Insertion en début de liste

Variable tmp pour tampon, variable intermédiaire

```
insertionDebut(L : liste, n : entier)
  tmp : pointeur sur noeud  (1)
  tmp = Nouveau(noeud)      (1)
  tmp->valeur = n           (1)
  tmp->suivant = L          (2)
  retourner tmp             (3)
```

Le coût de l'opération est le même que soit la longueur de la liste.

Exemple, insertion de la valeur 8



Listes chaînées – insertion en début de liste

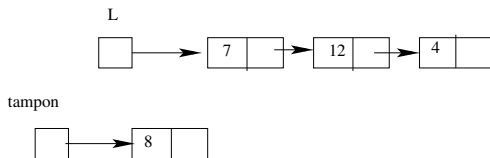
Insertion en début de liste

Variable tmp pour tampon, variable intermédiaire

```
insertionDebut(L : liste, n : entier)
  tmp : pointeur sur noeud  (1)
  tmp = Nouveau(noeud)      (1)
  tmp->valeur = n           (1)
  tmp->suivant = L           (2)
  retourner tmp              (3)
```

Le coût de l'opération est le même que soit la longueur de la liste.

Exemple, insertion de la valeur 8, étape (1)



Listes chaînées – insertion en début de liste

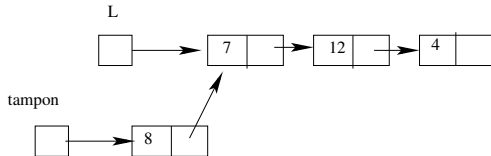
Insertion en début de liste

Variable tmp pour tampon, variable intermédiaire

```
insertionDebut(L : liste, n : entier)
  tmp : pointeur sur noeud  (1)
  tmp = Nouveau(noeud)      (1)
  tmp->valeur = n           (1)
  tmp->suivant = L          (2)
  retourner tmp             (3)
```

Le coût de l'opération est le même quelle que soit la longueur de la liste.

Exemple, insertion de la valeur 8, étape (2)



Listes chaînées – insertion en début de liste

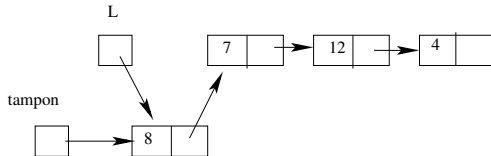
Insertion en début de liste

Variable tmp pour tampon, variable intermédiaire

```
insertionDebut(L : liste, n : entier)
  tampon : pointeur sur noeud  (1)
  tampon = Nouveau(noeud)      (1)
  tampon->valeur = n           (1)
  tampon->suivant = L           (2)
  retourner tmp                 (3)
```

Le coût de l'opération est le même quelle que soit la longueur de la liste.

Exemple, insertion de la valeur 8, étape (3)



Listes chaînées – insertion en début de liste

Construction de la liste précédente

```
maliste : liste  
maListe = insertionDebut (maListe, 9)  
maListe = insertionDebut (maListe, 7)  
maListe = insertionDebut (maListe, 5)  
affichageListe (maListe)
```

L'ordre est inversé.

On insère donc d'abord les derniers nœuds.

Listes chaînées – copie d'une liste (1)

Procédure récursive

```
copieListeRec(L : liste) : liste
  si L = None alors
    retourner None
  sinon res : liste
    res = Nouveau(noeud)
    res->valeur = L->valeur
    res->suivant = copieListeRec(L->suivant)
    retourner res
```

L'appel récursif se fait sur le nœud suivant

Listes chaînées – copie d'une liste (2)

Procédure itérative

```
copieListeIt(L : liste) : liste
  res, tmp : liste ; res = None
  tmp = res
  si L = None alors retourner res
  sinon tmp=Nouveau(noeud);tmp->valeur=L->valeur;tmp->suivant=None
  L = L->suivant
  tant que L <> None faire
    tmp->suivant=Nouveau(noeud);tmp->suivant->valeur=L->valeur
    tmp = tmp->suivant ; L = L->suivant
  tmp->suivant = None
  retourner res
```

Remarque

On utilise la boucle **tant que L <> None faire** lorsque l'on parcourt L **sans modifier** cette liste (copie, affichage,...).

Listes chaînées – copie d'une liste (3)

Méthode

Pour copier une liste en inversant l'ordre, on insère au début un à un les éléments.

Procédure itérative – méthode la plus simple

```
inversionListeIt (L : liste) : liste
  res = None
  tant que L <> None faire
    res = insertionDebut (res, L->valeur)
    L = L->suivant
  retourner res
```

Procédure récursive * – méthode la plus difficile

L1 : liste de départ, L2 : liste en sortie.

```
inversionListeRec (L1, L2 : liste)
  si L1 = None alors retourner L2
  sinon retourner inversionListeRec (L1->suivant,
                                     insertionDebut (L2, L1->valeur))
```

Pour l'inversion récursive, nous sommes obligés d'avoir un second argument pour la liste de sortie.

Modification d'une liste – suppression du dernier nœud

Première tentative

```
suppressionFin(L : liste) : liste
  tmp : liste
  si tmp = None retourner None
  tant que tmp <> None faire
    tmp = tmp->suivant
  tmp = None
  retourner L
```

En fait la procédure ne fait rien !

Il faut s'arrêter sur le dernier nœud.

Modification d'une liste – suppression du dernier nœud

Procédure correcte

```
suppressionFin(L : liste) : liste  
  tmp : liste  
  si tmp = None retourner None  
  tant que tmp->suivant <> None faire  
    tmp = tmp->suivant  
  désallouer(tmp)  
  tmp = None  
  retourner L
```

désallouer tmp permet de libérer la mémoire occupée par ce nœud.

