ALGORITHMIQUE L3 MIASHS-IDS

Fadila Bentayeb

Organisation du cours

- Volume horaire : 21CM
  - Partie magistrale
  - Partie Travaux Dirigés

- Evaluation
  - Partiel sur table

Plan du cours

- Structures de données simples
- Structures de contrôle simples
- Procédures & Fonctions
- Types construits
  - Enregistrement
  - Tableau
  - Fichier
- Structures de données dynamiques : Pointeur
- Récursivité

Algorithme

- Définition
  - « Une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat, et cela indépendamment des données ».

Encyclopedia universalis

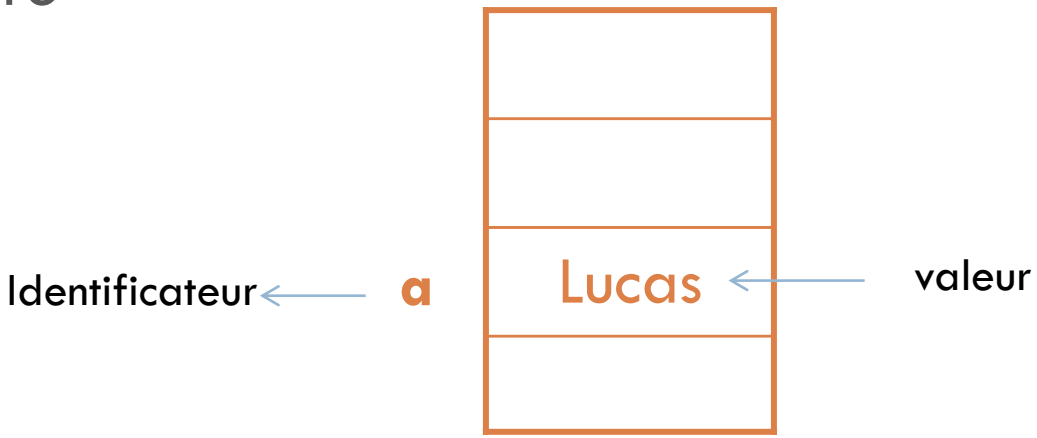
- Exemples
  - Résoudre une équation du second degré
  - Trouver un chemin dans un labyrinthe
  - Gérer un portefeuille de titres

Méthodologie de programmation

- Structures de données
  - Variables simples
  - Tableau
  - Fichier
  - Pointeurs
  - …
- Structures de contrôle
  - Saisie et affichage de données
  - Affectation
  - Conditionnelle
  - Boucle
  - Sous-programmes

Variable

- Définition
  - Emplacement mémoire
  - Identificateur
  - Type
  - Valeur



- Exemple
  - a : string → Type

Types de données

- Type de base
  - Entier, réel, caractère, string, booléen
- Types construits
  - Enregistrement, tableau, fichier, pointeur

- Exercice
  - Donner les valeurs et les types des variables suivantes :
    - x1 := 5
    - x2 := 2.5
    - y := vrai
    - z := non(y)
    - w := x1 < x2

Conditionnelle

- **Sélection simple**
  - Si Condition Alors Actions Fin Si
- **Sélection double**
  - Si Condition Alors Actions 1 Sinon Actions 2 Fin Si
- **Sélections imbriquées**
  - Si condition 1
    Alors Si Condition 2 Alors Actions 2 Sinon Actions 3 Fin Si
    Sinon Actions 1
    Fin Si
- **Sélection multiple**
  - Selon Nom_Variable
    Valeurs 1 : Actions 1
    … : ….
    Valeurs N : Actions N
    Autre : Actions Autres
    Faire Fin Selon

Boucle

- La boucle POUR

  Pour compteur := valeur_initiale à valeur_finale

  Faire Actions

  Fin Pour

- La boucle TANT QUE

  Tant que Condition

  Faire Actions

  Fin Tant que

- La boucle REPETER
  - Répéter Actions Jusqu'à Condition

Procédures

- **Définition :** Une procédure est un sous-programme qui prend 0 ou plusieurs paramètres en entrée et retourne 0 ou plusieurs résultats

- Déclaration d'une procédure

  Procédure NomProc (liste d'arguments : type)

  Déclaration de variables locales

  Début

  Corps de la procédure

  Fin

  Paramètres formels

- Appel d'une procédure dans le programme principal

  NomProc (liste d'arguments)

  Paramètres effectifs

Exemple de procédure

```
Algorithme test
Var
x, y, total : entier
« déclaration de la procédure somme »
Procédure somme (a, b, resultat : entier)
Début
resultat := a+b
Fin

Début  « programme principal »
Saisir(x, y)
somme(x, y, total)
Afficher (total)
Fin
```

Fonction

- **Définition** : Une fonction est un sous-programme qui prend zéro ou plusieurs paramètres en entrée et retourne un et un seul résultat
- **Déclaration d'une fonction**

  Fonction NomFonction (liste d'arguments : type) **: Type**

  Déclaration de variables locales

  Début

  Corps de la fonction

  Fin

- **Appel d'une fonction dans le programme principal**

  variable-globale := NomFonction (liste d'arguments)

Exemple de fonction

```
Algorithme test
 Var
 x, y, total : entier

 « déclaration de la fonction somme »
 Fonction somme (a, b : entier) : entier
 Var
 resultat : entier
 Début
 resultat := a+b
 somme := resultat
 Fin

 Début  « programme principal »
 Saisir(x, y)
 total := somme(x, y)
 Afficher (total)
Fin
```

Portée d'une variable

- Variable globale
  - Elle est utilisable dans tous les traitements dépendant du traitement dans lequel cette variable a été déclarée
  - Les variables déclarées dans le programme principal sont globales

- Variable locale
  - Elle est utilisable uniquement dans le traitement dans lequel cette variable a été déclarée
  - Les variables déclarées dans un sous-programme sont locales à ce sous-programme

Passage de paramètres

- L'échange de données entre sous-programmes peut se faire par le partage de variables globales ou par les variables locales passées comme paramètres.

- Il existe plusieurs types de passage de paramètres
  - Par adresse ou par variable
  - Par valeur
  - Par nom

Passage de paramètres par adresse

- Les procédures appelées peuvent modifier les valeurs des paramètres transmis
- Les paramètres formels contiennent les adresses des paramètres réels

- Notation
  - Les paramètres formels passés par adresse sont précédés par le mot clé : var

Exemple de passage de paramètre par adresse

Algorithme test

Var

x, y, total : entier

« déclaration de la procédure somme »

Procédure somme (a, b, var resultat : entier)

Début

resultat := a+b

Fin

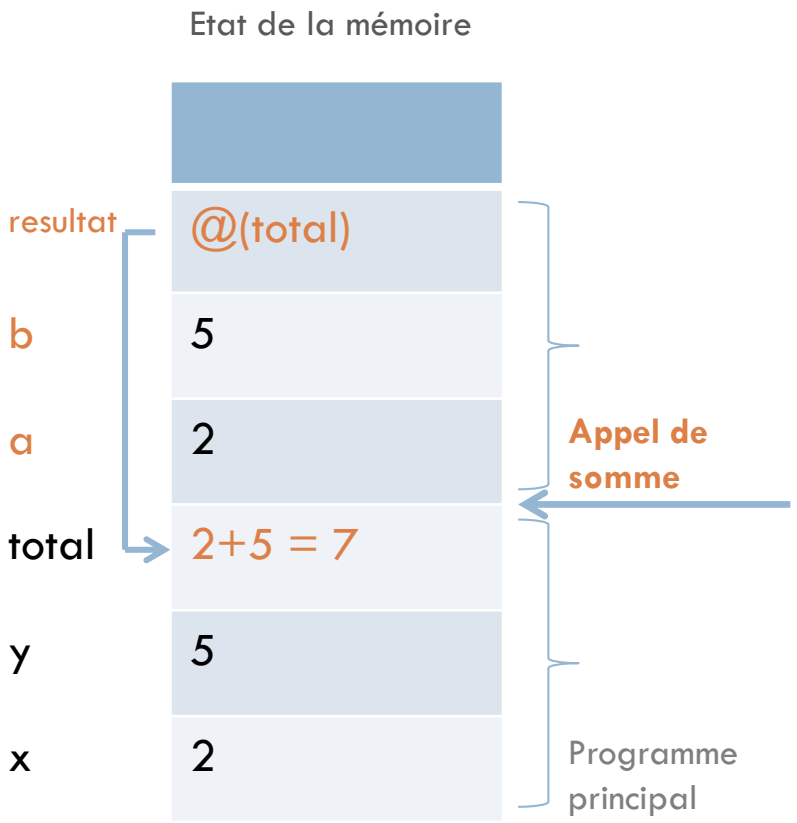
Début « programme principal »

Saisir(x, y)

somme(x, y, total)

Afficher (total)

Fin



Passage de paramètres par valeur

- Les valeurs des paramètres transmis peuvent être utilisées dans les procédures appelées sans qu'il soit possible de les modifier.
- Les paramètres formels contiennent une copie des paramètres réels.

- Notation
  - Les paramètres formels passés par valeur ne sont précédés par aucun mot clé particulier

Exemple de passage de paramètre par valeur

Algorithme test

Var

x, y, total : entier

« déclaration de la procédure somme »

Procédure somme (a, b, resultat : entier)

Début

resultat := a+b

Fin

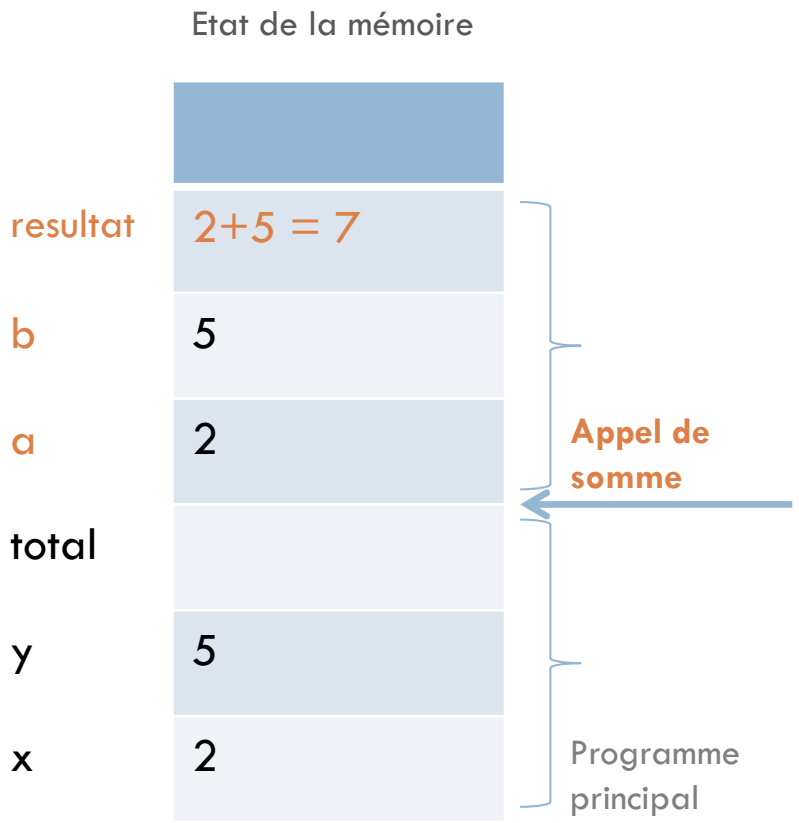
Début « programme principal »

Saisir(x, y)

somme(x, y, total)

Afficher (total)

Fin



Types de données construits

- Structures de données statiques
  - Type énuméré
  - Type Intervalle
  - Type Tableau
  - Type Enregistrement
  - Type Fichier
- Structures de données dynamiques
  - Type Pointeur

Type énuméré

- On définit en extension toutes les valeurs que peuvent prendre les variables de ce type. Les valeurs doivent être ordonnées. Elles sont en nombre fini.

```
Type
Jour = (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche)
Var     j1, j2 : jour ;
Début
        j1 := mardi ; j2 := dimanche
Fin
```

- Les opérateurs relationnels ($<$, $\leq$, $>$, $\geq$, $=$, $\neq$) s'appliquent sur les différentes valeurs de ce type.

  lundi $<$ mardi $<$ mercredi $<$ jeudi $<$ vendredi $<$ samedi $<$ dimanche

Type Intervalle

- On définit en compréhension toutes les valeurs que peuvent prendre les variables de ce type.
- Les valeurs doivent être ordonnées et on ne précisera que le minimum et le maximum.
- Remarque : Les valeurs appartenant à un intervalle sont nécessairement de type déjà défini. Bien que les types énuméré et intervalle sont peu utilisés, ils permettent néanmoins d'accroître la lisibilité des algorithmes et facilitent les contrôles.
- Exemple

```
Type
Chiffre   = 0 .. 9
Lettre = A .. Z
Jour_ouvrable = lundi .. Vendredi
```

Exemple d'utilisation des types énuméré et intervalle

```
Algorithme enu_inter

Type
Jour = (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche)
Emploi_du_temps = tableau [1..52, lundi..dimanche] de string

Var
An_01_02 : Emploi_du_temps
Semaine : Entier
J : Jour

Début
Pour semaine:=1 à 4 Faire *emploi du temps du mois de janvier*
    Pour j:= lundi à dimanche Faire
    Ecrire (An_01_02[semaine, j])
    Fin Pour
Fin Pour
Fin
```

Tableau

- Déclaration

  Type

  Nom_Type = tableau [Min .. Max] de Type de base

- Exemple

```
Algorithme test
Type
    Tab = tableau [1..10] de réel
Var
    T : Tab
Début
    Pour i := 1 à 10 Faire
    Saisir(T[i])
    Fin Pour
Fin
```

T : identificateur

15.2	**1 : indice**
65.8	2
256.8	3
14.3	4
89	5
1000	6
2154.6	7
845.7	8
789.1	9
1025.5	10

Enregistrement

- Déclaration

```
Type
Nom_enregistrement = Enregistrement
                        Champ 1 : type
                        Champ 2 : type
                        …          :  …
                        Champ n : type
                        Fin
```

- Exemple

```
Etudiant = Enregistrement
            Numéro, titre auteur, éditeur : String
            date_de_parution : [1900..2070]
           Fin
```

- Contrairement aux tableaux les enregistrements peuvent regrouper des données de types différents

Type Enregistrement : autre exemple

```
Algorithme essai
Type
Ouvrage= Enregistrement
            Code   : entier;
            Titre    : Chaîne[40] ;
            Auteur  : Chaîne[30] ;
            Editeur : Chaîne[25] ;
            Date     :  enregistrement
                          Mois   : 1..12;
                          Année: 1900..2050
                           fin
        Fin
Var
livre : Ouvrage
Début
Saisir(livre.Code,  livre.Titre,  livre.Auteur,  livre.Editeur,  livre.Date.Mois,  livre.Date.Année)
Fin
```

Fichier

- Fichier = ensemble de données organisées
- Modes d 'Organisation d 'un fichier
  - Séquentielle
  - Séquentielle indexée
- Modes d'Accès
  - Séquentiel
  - Direct

Fichier Texte

- Organisation
  - Séquentielle

- Accès
  - Séquentiel

- Déclaration

Type

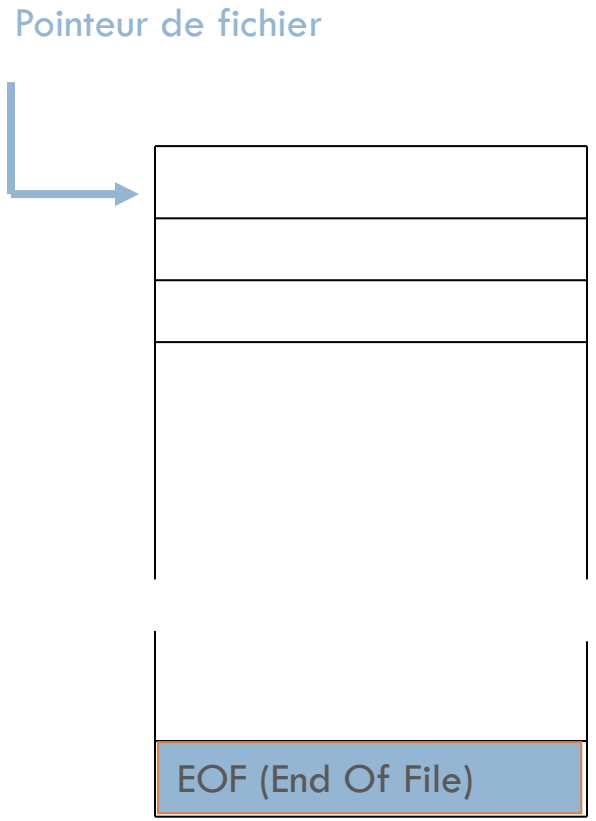
Livre = Fichier Texte

Fichier d'enregistrement

- Organisation
  - Séquentielle
  - Séquentielle Indexée
- Accès
  - Séquentiel
  - Direct
- Déclaration

Type

Nom_Fichier = Fichier de Nom_type_enregistrement



Fichier d'enregistrement : Exemple

```
Type personne = enregistrement
                    nom      : string
                    prénom : string
                    fin

    employé = fichier de personne
```

Manipulation des fichiers

- Exemple

```
Type
Etudiant= enregistrement
                nom : String
             prénom : string
                âge : [0..130]
             fin
Promotion = Fichier de Etudiant
Var
f : Promotion
p : Etudiant
…
```

Opérations sur les fichiers

- Ouverture du fichier : Ouvrir(f)
- Lecture d'un enregistrement p à partir d'un fichier f : Lire(f, p)
- Ecriture d'un enregistrement p dans un fichier f : Ecrire(f, p)
- Mises à jour d'un fichier (Ajout, Suppression, Modification)
- Fermeture du fichier : Fermer(f)
- La fonction EOF(f) teste si le pointeur du fichier f pointe vers la fin du fichier

Structures de données dynamiques

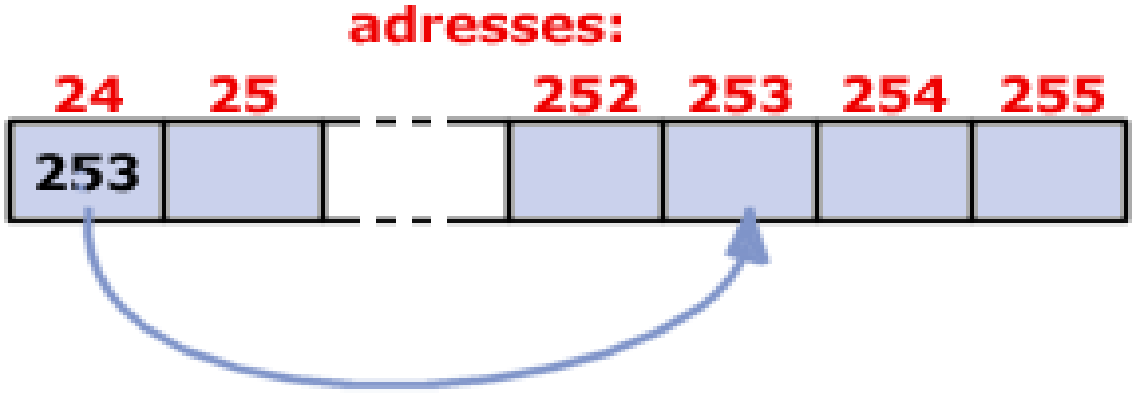
- Variables statiques
  - définies au début du programme
  - maintenues tout au long du programme.
- Variables dynamiques
  - variations en format et en taille durant l'exécution du programme
  - création et suppressions en cours d'exécution.

Notion d'adresse

- La mémoire est constituée d'un ensemble de cases de 8 bits appelés octets (byte)
- Une variable : selon son type (donc sa taille), va ainsi occuper une ou plusieurs de ces cases
- Exemple
  - une variable de type char occupera un seul octet
  - une variable de type entier ccuppera 8 octets ou 16 octets
- Chacun de ces emplacements mémoire (appelées blocs) est identifié par un numéro. Ce numéro s'appelle adresse
  - Numéro = adresse

Accès aux variables

- On peut donc accéder à une variable de 2 façons :
  - grâce à son nom
  - grâce à l'adresse du premier bloc alloué à la variable
- Il suffit donc de stocker l'adresse de la variable dans un pointeur afin de pouvoir accéder à celle-ci (on dit que l'on "pointe vers la variable").



Comment connaît-on l'adresse d'une variable?

- L'adresse d'une variable change à chaque lancement de programme étant donné que le système d'exploitation alloue les blocs de mémoire qui sont libres, et ceux-ci ne sont pas les mêmes à chaque exécution.

- Il existe donc une syntaxe permettant de connaître l'adresse d'une variable, connaissant son nom: il suffit de faire précéder le nom de la variable par le caractère ^ pour désigner l'adresse de cette variable: ^Nom_de_la_variable

Déclaration d'un pointeur

- Un pointeur est une variable qui doit être définie en précisant le type de variable pointée de la façon suivante:

Type P = ^ **Nom_du_pointeur**

- Le type de variable pointée peut-être aussi bien un type primaire (tel que entier, car, ...) qu'un type complexe (tel que record, ...).

Type P = ^ **Nom_du_pointeur**

Initialisation d'un pointeur

- Après avoir déclaré un pointeur il faut l'intialiser
- Cette démarche est très importante car lorsque vous déclarez un pointeur, celui-ci contient ce que la case où il est stocké contenait avant, c'est-à-dire n'importe quel nombre
- Autrement dit, si vous n'initialisez pas votre pointeur, celui-ci risque de pointer vers une zone hasardeuse de votre mémoire, ce qui peut être un morceau de votre programme ou ... de votre système d'exploitation !

Un pointeur non initialisé représente un danger!

Initialisation d'un pointeur

- Pour initialiser un pointeur, il faut utiliser l'opérateur d'affectation ':=' suivi de l'opérateur d'adresse '@' auquel est accollé un nom de variable (celle-ci doit bien sûr avoir été définie avant...) :

Nom_du_pointeur = @nom_de_la_variable_pointée

```
Var         a: entier, b: car
            p1:^entier, p2:^car
Début       a := 2;
            p1 := @a
            p2 := @b
```

Accéder à une variable pointée

- Après (et seulement aprè) avoir déclaré et initialisé un pointeur, il est possible d'accéder au contenu de l'adresse mémoire pointée par le pointeur

Exemple : a : entier, b : car

P1^ := 10

P2^ := ‘x';

Après ces deux instructions, le contenu des variables a et b sera respectivement 10 et x

Exercice

- L'opérateur ^ est utilisé agit comme un opérateur d'indirection qui, appliqué à une valeur de type pointeur, délivre la valeur pointée

Quels sont les type et valeur de j ?

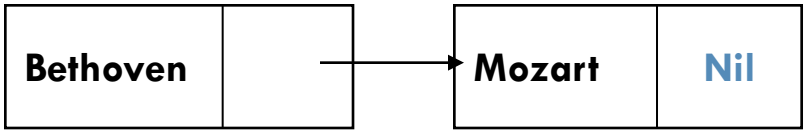
```
Algorithme Exo1
Var i : entier
pi : ^entier

    Début
    i : = 10
    pi := @i       /*   initialisation du pointeur pi  */
    pi ^ := 2      /*   initialisation de la valeur pointée par pi  */
    j := pi^ + 1   /*   une utilisation de la valeur pointée par pi */
    Fin
```

Listes chaînées

- Une liste chaînée est une structure de données dynamique autoréférentielle

- Une structure autoréferrentielle (parfois appelée structure récursive) correspond à une structure dont au moins un des champs contient un pointeur vers une structure de même type



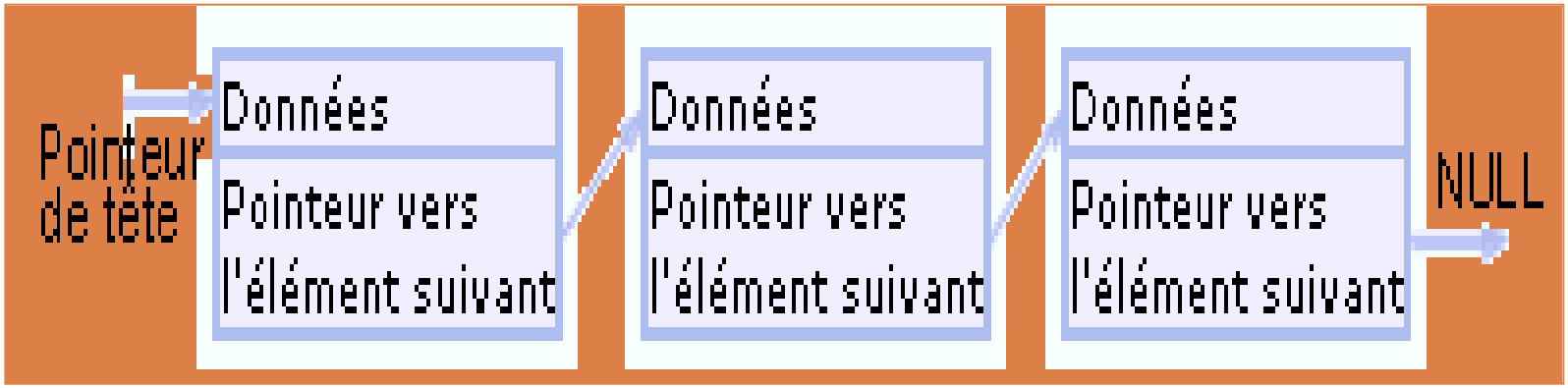
Pointeur de Tête et valeur Nil

- La déclaration de la structure récursive des pointeurs crée une chaîne d'enregistrements liés par des liens logiques, mais cela n'est pas suffisant

- Il faut conserver une "trace" du premier enregistrement afin de pouvoir accéder aux autres :
  - c'est pourquoi un pointeur vers le premier élément de la liste est indispensable
  - Ce pointeur est appelé pointeur de tête

- Le dernier enregistrement ne pointe vers rien
  - il est donc nécessaire de donner à son pointeur la valeur NULL (Nil)

Pointeur de Tête et valeur Nil

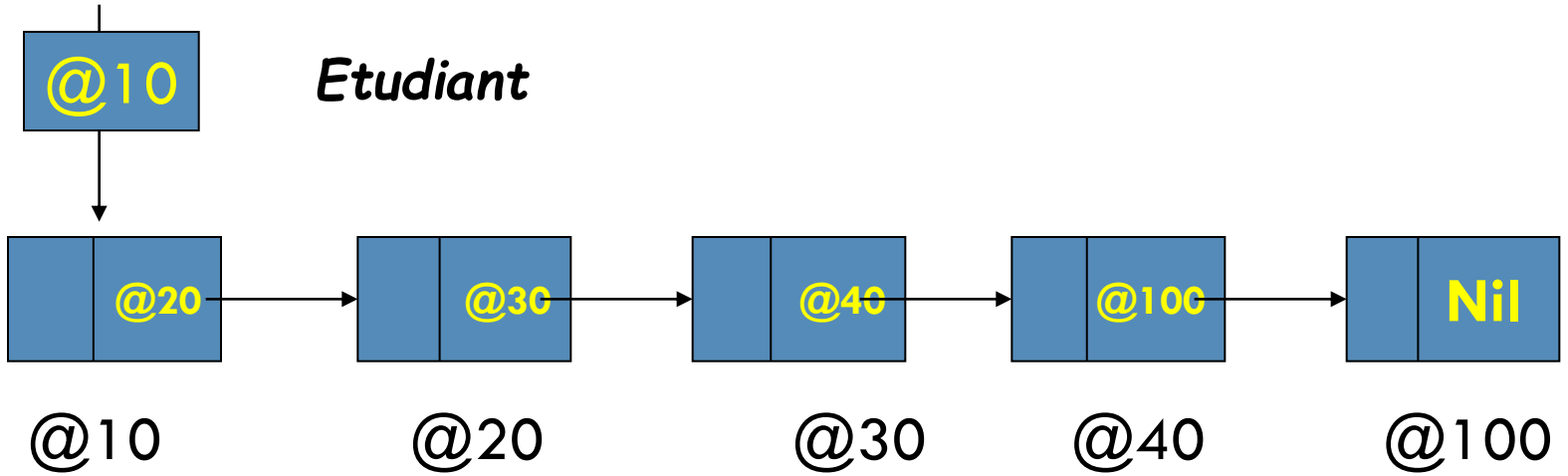


Différents types de listes

- Liste chaînée
  - Lorsque la structure contient des données et un pointeur vers la structure suivante
- Liste chaînée double
  - Lorsque la structure contient des données, un pointeur vers la structure suivante, et un pointeur vers la structure précédente
- Arbre binaire
  - Lorsque la structure contient des données, un pointeur vers une première structure suivante, et un pointeur vers une seconde structure

Listes chaînées

- Une liste est une structure qui permet de stocker de manière ordonnée des éléments
  - Elle est composée de maillons
  - un maillon est une structure qui contient un élément à stocker et un pointeur (au sens large) sur le prochain maillon de la liste
- Un pointeur qui pointe sur rien aura la valeur VIDE notée Nil

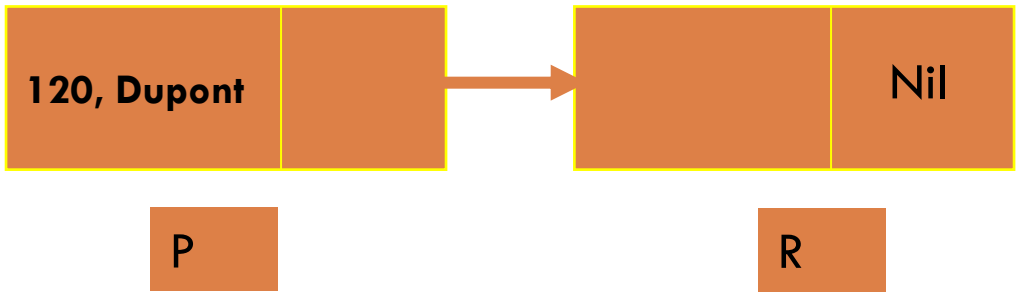


Listes chaînées : Exemple

```
Type
Pointeur = ^maillon

maillon = Enregistrement
              element : string
              suivant : Pointeur
          Fin
```

```
Var
P, R: Pointeur
Début
P^.element := '120, Dupont '
P^.suivant := R
Fin
```



Listes chaînées : Exemple

```
Type
Liste = ^Personne
Personne = Enregistrement
                Nom      : Chaîne[30]
               Suivant  : Liste
              fin
Var
Etudiant :  Liste
```

- Etudiant est un pointeur qui pointe vers une variable dynamique de type Personne »

Allocation/Désallocation des variables dynamiques

- On utilise 2 procédures pour réserver/libérer des variables dynamiques
  - Nouveau
  - Libérer
- Les variables créées par la procédure Nouveau sont stockées dans une structure de type pile appelée : le tas

```
Procedure Nouveau (var  P : Personne)
Début
…
Fin
```

```
Procedure Libérer (P : Personne)
Début
…
Fin
```

Création d'un pointeur

```
Type
                        Personne  =  ^Cellule
                        Cellule  =  Enregistrement
                                      Info  :  Chaîne[100]
                                      Suiv  :  Personne
                                    fin

Var                     mus   :  Personne;

Création                Nouveau(mus)
                        mus^.Info    :=  …
                        mus^.Suiv    :=  …
```

Ajout d'un premier élément

- Une fois la structure et les pointeurs définis, il est possible d'ajouter un premier maillon à la liste chaînée, puis de l'affecter au pointeur Tete. Pour cela il est nécessaire de :
  - d'allouer la mémoire nécessaire au nouveau maillon grâce à la fonction Nouveau, selon la syntaxe suivante: Nouveau (P)
  - d'assigner au champ "pointeur" du nouveau maillon, la valeur du pointeur vers le maillon de tête : P^.Suivant = Tete
  - définir le nouveau maillon comme maillon de tête : Tete = P

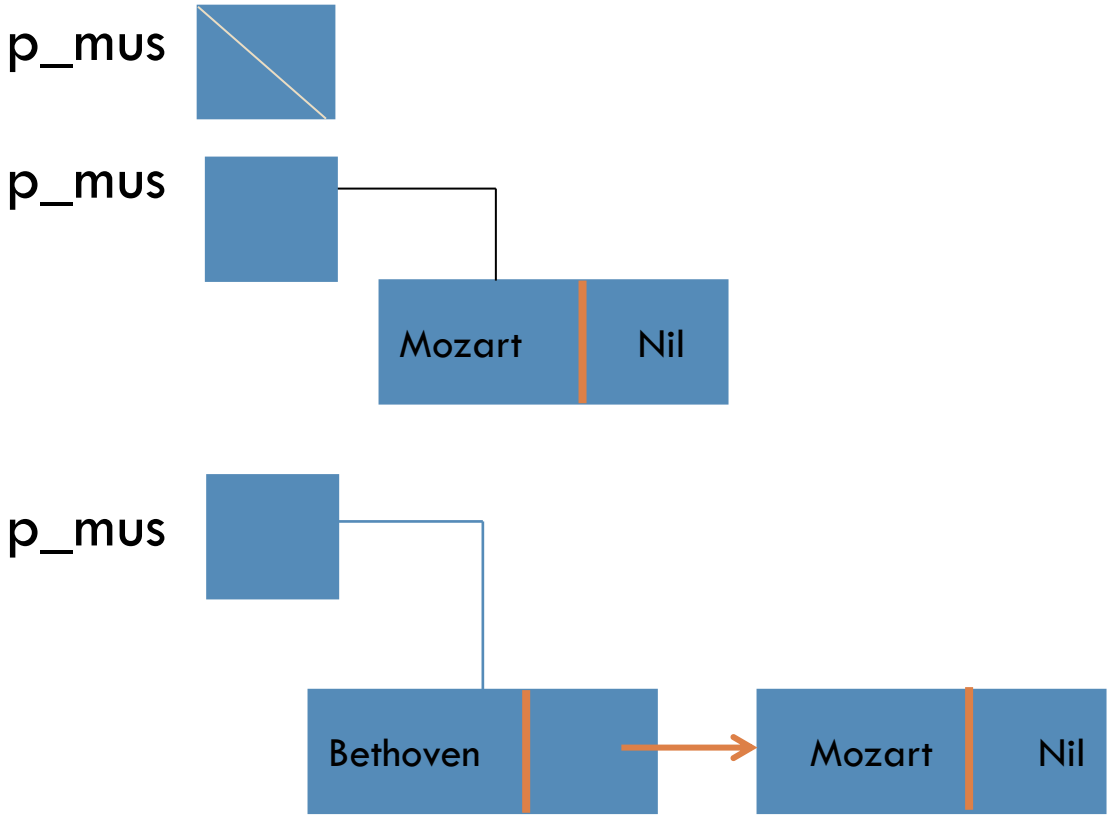
Création d'une pile

- p_mus, mus : Personne

```
p_mus : = Nil
```

```
Nouveau(mus)
mus^.Info : = 'Mozart'
mus^.Suiv := p_mus
p_mus := mus
```

```
Nouveau(mus)
mus^.Info : = 'Bethoven'
mus^.Suiv := p_mus
p_mus := mus
```



Création d'une liste par empilement

```
Algorithme pile
Var
p_mus, mus     :  Personne;
       M       :  Chaîne[20]
Début
    p_mus  :=  NIL;
    Répéter
          Nouveau(mus)
          Saisir(M)
          mus^.Info  := M
          mus^.Suiv  := p_mus
          p_mus  :=  mus
    Jusqu'à    M = "STOP"
Fin
```

Opérations de mise à jour sur les listes

- Ajout d'un maillon
  - au début de la liste
  - à la fin de la liste
  - à un endroit précis de la liste
- Suppression d'un maillon
- Modification d'un maillon

Ajout d'un élément en tête de liste

```
Algorithme Ajouter_tête

Var
p_mus, mus  :  Personne
M  :  Chaîne[20]

Début
    Nouveau(mus)
    Saisir(M)
    mus^.Info  := M
    mus^.Suiv := p_mus
    p_mus  :=  mus
Fin
```

Ajout d'un élément en fin de liste

```
Algorithme Ajouter_fin
Var         p_mus, mus, temp :  Personne
            M  :  Chaîne[20]
 Début
    temp := p_mus
    Tant que  temp^.suiv <> Nil Faire
            temp := temp^.suiv
    Fin Tant que
    Nouveau(mus)
    Saisir(M)
    mus^.Info  := M
    mus^.Suiv := Nil
    temp^.suiv  :=  mus
Fin
```

Récursivité

- Les définitions par récurrence sont assez courantes en mathématiques.
- Prenons le cas de la suite de Fibonacci, définie par :
  - Uo = u1 =1
  - Un = Un-1+Un-2 pour n>1

- On obtient donc la suite :
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, …

- Nous allons voir que ces définitions récursives s'implémentent très simplement en informatique en utilisant des procédures ou/et des fonctions récursives

Récursivité

- Déclaration d'une procédure récursive

```
Procédure Recursive (paramètres)
* déclaration des variables locales
Début
    Si TEST_D'ARRET   Alors
                        < Instructions >
                        < Instruction pour le point d'arrêt >
                    Sinon
                        * appel récursif
                        < Instructions >
                        < Appel récursif de la procédure (paramètres changés) >
                        < Instructions >
    Fin Si
Fin
```

Fonction récursive

- Pour calculer la suite de Fibonacci, une transcription littérale de la formule est la suivante :

```
Fonction Fib (n : integer) : integer
var r : integer

Début

Si n = 0 ou n = 1
                Alors       r := 1
                Sinon       r := Fib (n-1) + Fib (n-2)  * appel récursif de la fonction
Finsi
Fib := r

Fin Fonction
```

Fonction Récursive

- Déroulement
- Appel de la fonction Fib

Fib(4)

```
> Fib (3)                                          +  Fib (2)
> (Fib (2)                  +        Fib (1))      +  Fib (2)
> ((Fib (1) + Fib (1))      +        Fib (1))      +  Fib(2)
> ((1       + Fib(1))       +        Fib (1))      +  Fib(2)
> ((1       +     1)        +        Fib (1))      +  Fib(2)
> (         2               +        Fib(1))       +  Fib(2)
> (2                        +        1)            +  Fib(2)
> 3                                                 +  Fib(2)
> 3                                                 +  (Fib (1) + Fib (1))
> 3                                                 +  (1       + Fib(1))
> 3                                                 +  (1       + 1)
> 3                                                  +        2
> 5
```

Récursivité : Autre exemple

```
Fonction Fact (n: integer): integer
 Var r : integer
Début

    Si              n <= 1
                    Alors     r := 1
                    Sinon     r := n * Fact(n-1)     * appel récursif

    Fin si

Fact := r
Fin
Fin Fonction
```