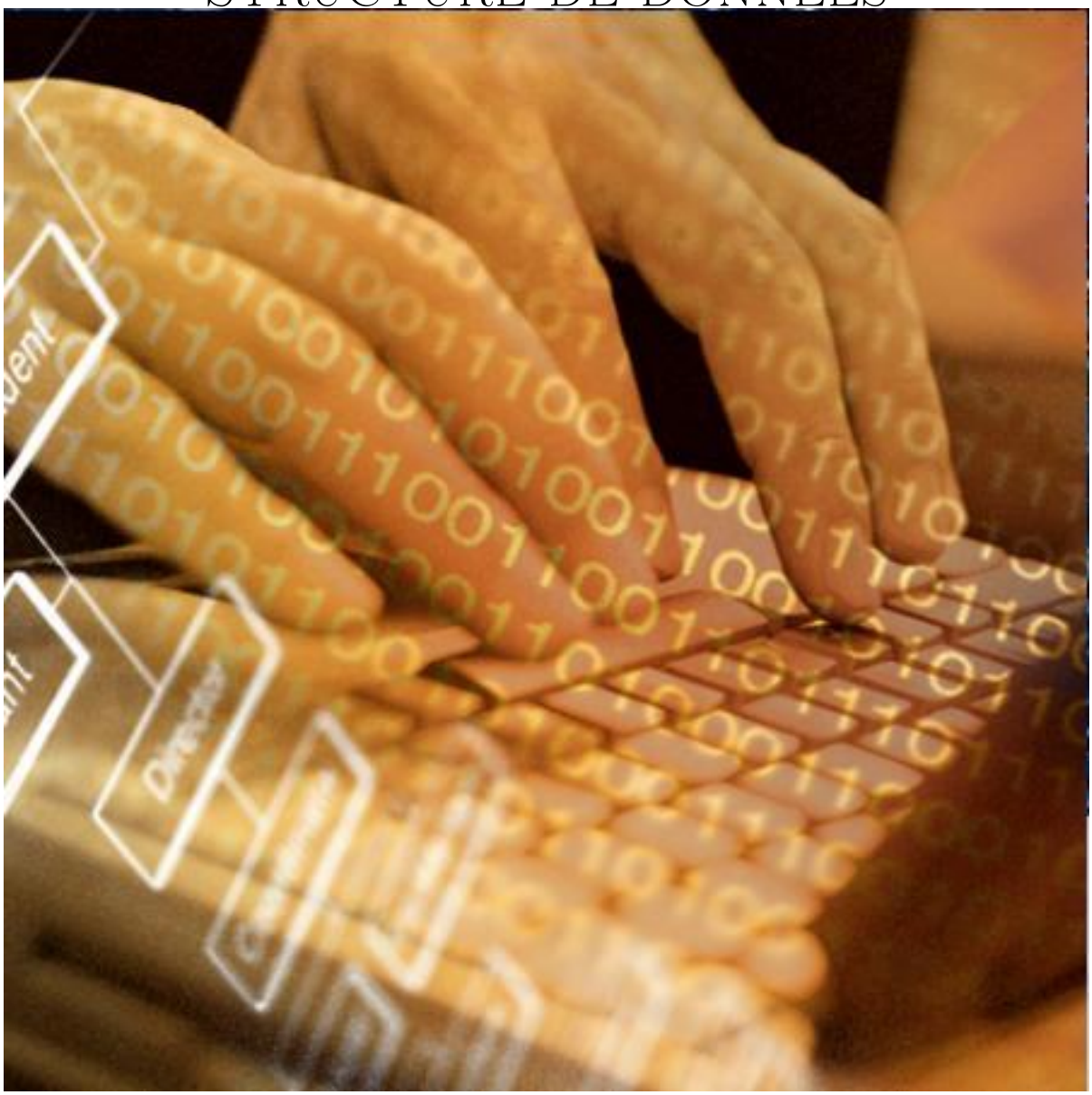
Structure de données



Madani BEZOUI
mad.bezoui@gmail.com
http://mbezoui.webs.com

Année Universitaire 2012-2013

Ceci est un manuscrit établit pour permettre aux étudiants de deuxième année Recherche Opérationnelle, de l'Université de Boumerdes, une bonne assimilation et une bonne révision des cours de ce module. Veuillez me signaler les éventuelles erreurs de frappe de saisie,... à l'adresse mail suivante : madani.bezoui@gmail.com, Merci.

©. M. BEZOUI. Janvier 2013.

Table des matières

Chapitre 1

RAPPELS SUR L'ALGORITHMIQUE

"Les premiers 90% du code prennent les premiers 10% du temps de développement. Les 10% restants prennent les autres 90% du temps de développement."
Tom Cargill

1.1 Rôle de l'algorithme dans l'informatique

Qu'es qu'un algorithme ?

✍ Un algorithme est une séquence d'étapes de calcul qui transforment l'entrée en sortie.

✍ Un algorithme est un outil permettant de résoudre un problème de calcul bien spécifié.

1.2 La structure de données

Définition 1. *Une structure de données est un moyen de stocker et organiser des données pour faciliter l'accès à ces données et leur modification.*

1.3 Opérateurs de bases principaux

1.3.1 Symboles arithmétiques

Voici quelques symboles arithmétiques

symbole	exemple	valeur	rôle
$+$	$2+3$	5	addition des entiers et decimaux
$-$	$2-3$	-1	soustraction des entiers et decimaux
$*$	$2*3$	6	multiplication des entiers et decimaux
$/$	$2/3$	0,33	division des entiers et decimaux
div	$7 div 2$	3	résultat d'une division entière
mod	$3\ mod\ 2$	1	reste de la division entière

Symboles logiques

symbole	exemple	valeur	rôle
=	2 = 3	0	teste d'égalité
<>	2 <> 3	1	test de différence
<	3 < 2	0	test d'infériorité stricte
>	3 > 2	1	test de superiorité stricte
<=	3 <= 3	1	test d'inferiorité au sens large
>=	3 >= 2	1	test de superiorité au sens large
ou	0 *ou* 1	1	le "ou" logique (inclusif)
et	0 *et* 1	0	le "et" logique

1.4 Structures conditionnelles ou Alternatives

1.4.1 Instruction Si

Algorithme 1 Instruction conditionnelle : si

```
1: si Condition(s) alors
2:     Instruction(s)
3: finsi
```

deuxième forme :

Algorithme 2 Instruction conditionnelle : si

```
1: si Condition(s) alors
2:     Instruction(s)
3: sinon
4:     Instruction(s)
5: finsi
```

Exemple 1. *Afficher le plus grand de deux nombres.*

1.4.2 Le choix switch

Algorithme 4 Instruction conditionnelle : selon (switch)

```
1: selon variable ou expression
2: cas valeur1 : action 1
3: ⋮ ⋮ ⋮
4: cas valeur n : action n
5: Autrement action par défaut
6: fin selon
```

Algorithme 3 Correction_exemple_1

```
1: var a,b : entier ;
2: début
3: écrire("Entrez deux nombres :")
4: lire(a,b)
5: si a > b alors
6:    écrire("Le plus grand des deux est : ",a)
7: sinon
8:    si a = b alors
9:       écrire("Les nombres",a,"et",b,"sont égaux")
10:   sinon
11:      écrire(" Le plus grand des deux est : ",b)
12:   finsi
13: finsi
14: fin.
```

1.5 Structure itératives

1.5.1 La boucle pour

Algorithme 5 Boucle_pour

```
pour i allant de val_initiale à val_finale faire
   Instruction(s)
fin pour
```

où :

val_initiale : Valeur initiale de notre compteur.

val_finale : Valeur finale, pour laquelle notre algorithme sort de la boucle.

1.5.2 La boucle tant que

Algorithme 6 Boucle_tant_que

```
1: tantque Condition(s) faire
2:    Instruction(s)
3: fin tantque
```

La condition(s) est vérifie avant chaque itération. Si elle est évaluée à faux, on sort de la boucle.

1.5.3 La boucle répéter ... jusqu'à

L'algorithme répète les instructions "Instruction", jusqu'à ce que la condition "condition" n'est pas respectée.

Algorithme 7 Boucle_repeter

1: **répéter**
   Instruction(s)
2: **jusqu'à** Condition(s)

1.6 Les tableaux

1.6.1 Tableaux à une dimension

- Les tableaux peuvent être vus comme des types secondaires : ils sont construits sur les types de bases, et ont une structure voisine de celle des matrices mathématiques à une dimension. Leur taille est fixe, et leurs éléments sont tous de même type.
- Un tableau se déclare de la façon suivante :

  variable T[1,..,taille] : tableau de TYPE

  où :
  - TYPE : désigne un type de base (entier, réel, booléen,..)
  - T : un nom de variable
  - taille : un entier qui donne le nombre d'éléments du tableau.
- Pour accéder au i^{me} élément d'un tableau nommé T, il faut utiliser la syntaxe suivante : $T[i]$.

1.6.2 Tableaux à plusieurs dimensions

On les appel aussi matrices, Les tableaux à plusieurs dimensions se déclarent de la façon suivante.

points[1,..,2 ;1,..,7] :tableau d'entiers

où :

– entiers : n'est que le type, que j'ai choisi pour illustrer cet exemple, vous pouvez toute fois définir comme réels, chaîne de caractères.

L'accès à un élément (i,j) de la matrice M se fait d'une manière analogue à celle vue dans le cas de tableaux : $M[i, j]$

1.7 Les sous-programmes

1.7.1 Les fonctions

Une fonction est un sous-algorithme qui retourne une valeur.

Algorithme 8 exemple_fonction

```
1: var a,b :entier ;moy :réel ;{Les variables globales}
2: fonction moyenne(arg1 :entier,arg2 :entier) :réel ; {arg1,arg2 sont des variables locales de type entier, réel, est le type de la variable sortante.}
3: var M :réel ;{Ceci est aussi une variable locale}
4: début{Début de la fonction}
5: M ← arg1/arg2 ;
6: renvoyer(M) ;{Instruction obligatoire pour renvoyer la valeur de la fonction}
7: fin ;{Fin de la fonction}
8: début{Début de l'algorithme principal}
9: écrire("Introduire deux nombres !") ;
10: lire(a,b) ;
11: moy ← moyenne(a, b) ;
12: écrire("La moyenne entre les nombres",a,"et",b,"est :",moy)
13: fin.{Fin de l'algorithme principal}
```

1.7.2 Les procédures

Les procédures sont des sous programmes ayant aucun ou plusieurs arguments de sortie.

Algorithme 9 exemple_procedure

```
1: var a,b :entier ;moy :réel ;
2: procédure signe(arg1 :entier)
3: début{Début de la procédure}
4: si arg1>0 alors
5:    écrire("Ce nombre est positif") ;
6: sinon
7:    si arg1=0 alors
8:       écrire("Ce nombre est nul") ;
9:    sinon
10:      écrire("Ce nombre est négatif")
11:   finsi
12: finsi
13: fin ;{Fin de la procédure}
14: début{Début de l'algorithme principal}
15: écrire("Introduire un nombre") ;
16: lire(a) ;
17: signe(a) ;{Appel de la procédure d'affichage du signe de a}
18: fin.{Fin de l'algorithme principal}
```

1.8 Les enregistrements

Un ensemble d'enregistrements constitue un fichiers. Il s'agit d'un nouveau type de structure. leurs déclarations se fait de la manière suivante :

Type *Identifiant=Enregistrement*

Champ1 : Type1
Champ2 : Type2
⋮ ⋮
Champ N : Type N
{ Les types : Type 1, Type 2, ... sont les types usuels pré-définis}
Fin Enregistrement
Variable
Var1, Var2 : Identifiant
debut
{ L'accès se fait comme suit :}
Identifiant.Champ1← Valeur ;

Exemple 2. Type Voiture=Enregistrement
Marque : chaîne de 20 caractères
Couleur : chaîne de 20 caractères
Matricule : Entier
Fin Enregistrement
Variable
V : Voiture
Debut
V.Marque← 'BMW'
V.Couleur← 'Rouge'
V.Matricule← 235001306

Chapitre 2

RÉCURSIVITÉ

"Une personne qui n'a jamais commis d'erreurs n'a jamais tenté d'innover."
Albert Einstein

2.1 Définition

Définition 2. *La récursivité est une méthode de description d'algorithmes qui permet à une procédure de s'appeler elle-même (directement ou indirectement). Une notion est récursive si elle se contient elle-même en partie, ou si elle est partiellement définie à partir d'elle-même.*

L'expression d'algorithmes sous forme récursive permet des descriptions concises et naturelles. Le principe est d'utiliser, pour décrire l'algorithme sur une donnée D, l'algorithme lui-même appliqué à un ou plusieurs sous-ensembles de D, jusqu'à ce que le traitement puisse s'effectuer sans nouvelle décomposition. Dans une procédure récursive, il y a deux notions à retenir :

- ✈ *la procédure s'appelle elle-même : on recommence avec de nouvelles données.*
- ✈ *il y a un test de fin : dans ce cas, il n'y a pas d'appel récursif. Il est souvent préférable d'indiquer le test de fin des appels récursifs en début de procédure.*

2.2 Exemples

2.2.1 La Factorielle

La factorielle d'un nombre n donné est le produit des nombres entiers inférieurs ou égaux à ce nombre n. Cette définition peut se noter de différentes façons. Une première façon consiste à donner des exemples et à essayer de généraliser.
$0! = 1$
$1! = 1$
$2! = 1x2$
$3! = 1x2x3$
$n! = 1 \; si \; n = 0$

$n! = 1 * 2 * ... * (n-1) * n \ si \ n > 0$

2.2.2 Algorithme récursif de la factorielle d'un entier

- fonction fact1(n) :entier
  {Fonction qui fait le calcul récursif de la factorielle d'un entier $n >= 0$}
- var n,f : entier
- debut
- si (n=0)ou(n=1) alors
  - f $\longleftarrow$ 1 ;
- sinon
  - f$\longleftarrow$ $n * fact1(n-1)$;
- finsi
- renvoyer(f) ;
- fin.

2.2.3 Nombre de Fibonacci

La suite des nombres de Fibonacci se définit comme suit :

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \quad si \ n > 1 \end{aligned}$$

2.2.4 Algorithme Récursif qui calcul les termes la suite de Fibonacci

- fonction fibonacci(n)
  var n,fib : entier
- debut
- si ($n <= 1$) alors
  - fib$\longleftarrow$ n
- sinon
  - fib$\longleftarrow$fibonacci(n-1)+fibonacci(n-2) ;
- finsi
- renvoyer(fib) ;

fin.

2.2.5 Exemple d'exécution de la suite de Fibonacci

Remarque 1. *Une fonction récursive peut s'appeler elle-même plusieurs fois avec des paramètres différents. La fonction fibonacci (n) s'appelle récursivement 2 fois en fibonacci (n-1) et fibonacci (n-2).*

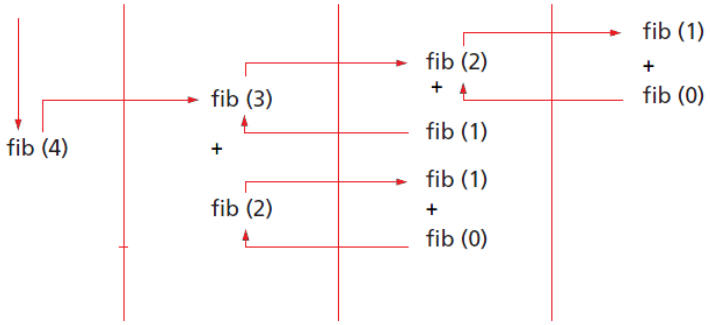


FIGURE 2.1 – Représentation graphique du calcul récursif de fib(4)

Chapitre 3

LA PROGRAMMATION DYNAMIQUE

"L'ordinateur a de la mémoire mais aucun souvenir."
Anonyme

3.1 Mémoire statique et mémoire dynamique

Les données d'un programme en cours d'exécution sont rangées dans deux zones de mémoires distinctes :

- la mémoire statique utilisée pour les données de taille constante connue du compilateur
- la mémoire dynamique utilisée pour les données de taille susceptible de varier

3.1.1 Mémoire Statique

Les langages de programmation sont tels que :

- Les variables globales et les variables locales des sous programmes ne peuvent contenir que des données de taille constante
- Ces variables sont rangées dans la mémoire statique
- La mémoire statique est gérée comme une pile (stack)
- à chaque appel d'un sous-programme, on réserve en sommet de pile la zone de mémoire nécessaire aux variables locales de ce sous-programme,
- à chaque sortie de sous-programme, on libère cette mémoire

3.1.2 Mémoire Dynamique

- La mémoire dynamique est occupée par des données
  - dont la taille peut varier d'une exécution du programme à l'autre et même au cours d'une même exécution
- Ces données sont toujours des valeurs manipulées à l'aide de pointeurs
- La mémoire dynamique est gérée comme un tas de mémoire (heap) dans lequel :

- ☞ on puise de la mémoire (allocation) selon les besoins
- ☞ on la remet à disposition (libération) lorsque l'on n'en a plus besoin

✍ On connaît les vecteurs

- ☞ Une structure de données statique
  - ✈ Chaque vecteur occupe en mémoire la taille maximum envisagée

✍ Que peut-on faire si on ne souhaite pas perdre de place ?

- ☞ Il faudrait une structure de donnée dynamique
  - ✈ À chaque instant, la place occupée par les données dépend uniquement de la taille de celles-ci

✍ On propose **les listes linéaires chaînées**.

3.2 Les pointeurs

Lorsqu'on déclare une variable et ce, quel que soit le langage de programmation, le compilateur réserve en mémoire l'espace nécessaire au contenu de cette variable à une adresse donnée. Toute variable possède donc une adresse en mémoire. La plupart du temps on ne s'intéresse pas à ces adresses. Mais quelque fois, ce type de renseignement peut s'avérer très utile.

3.2.1 Notion de pointeur

Un pointeur est une variable qui au lieu de contenir l'information proprement dite, contient son adresse en mémoire.

FIGURE 3.1 – Représentation graphique d'un pointeur et une variable

P est une variable de type pointeur contenant une adresse, et à cette adresse se trouve la donnée.

A la différence de la variable X classique qui contient directement l'information "71", la variable Y déclarée comme pointeur contient son adresse. On dit que Y pointe vers l'information "71" grace à son contenu, qui est l'adresse de cette information.

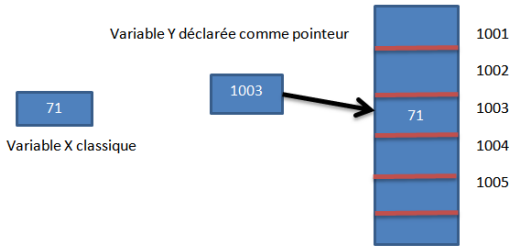
– *Déclaration d'un type "pointeur sur entier" :*

- ☞ *type tpoint_ent :↑ entier ;*

– *Déclaration d'un type "pointeur sur un caractère"*

- ☞ *type tpoint_caractère :↑ caractere ;*

La première déclaration se traduit comme suit : La variable "tpoint_ent", est un pointeur qui pointe vers une information de type Entier.



Exemple 3.

FIGURE 3.2 – Représentation graphique d'un pointeur et une variable

3.3 Opération sur les pointeurs

3.3.1 Initialisation des pointeurs

- nous vous engageons, dans un premier temps, à initialiser systématiquement vos variables pointeur avec la constante symbolique NIL.
- une variable pointeur non initialisée pointe sur n'importe quoi
- une variable pointeur initialisée à NIL ne pointe sur rien ! ! !, mais une tentative dŠaccès produira un arrêt immédiat de lŠexécution et non un comportement erratique.

3.3.2 Accès aux données

Les pointeurs contiennent des adresses, et on accèdes au contenu de ces adresses en utilisant l'opérateur ↑ placé derrière le nom de la variable pointeur.

exp

Déclaration :

P :↑entier
X :entier
début

$$X \leftarrow P\uparrow +1$$

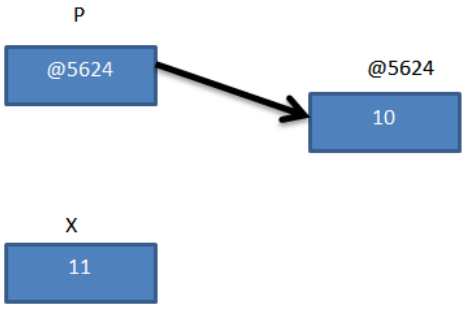


FIGURE 3.3 – Représentation graphique d'un pointeur et une variable

3.3.3 Affectation

Soient deux variables pointeurs P1 et P2 de type pointeur permettant l'accès respectivement aux données a et b.

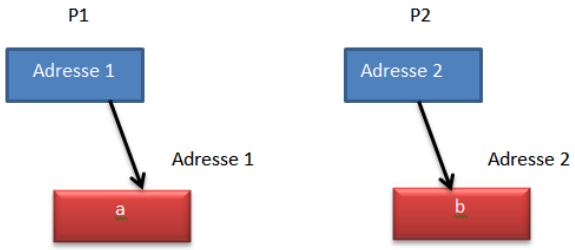


FIGURE 3.4 – Observez bien cet exemple !

L'exécution de l'instruction $P1 \leftarrow P2$

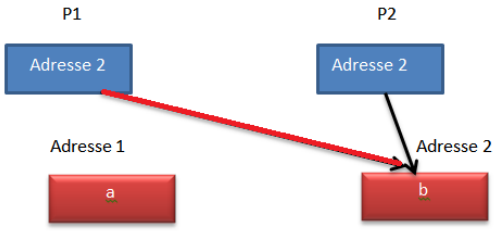


FIGURE 3.5 – $P1 \leftarrow P2$

On observe deux choses :

1. La donnée "a" n'est plus accessible, car on a perdu son adresse qui se trouve dans P1,
2. On peut atteindre b par l'intermédiaire de P1 et P2.

L'exécution de l'instruction $P1 \uparrow \leftarrow P2 \uparrow$

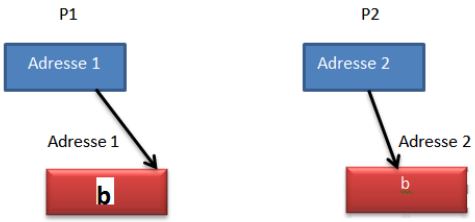


FIGURE 3.6 – $P1 \uparrow \leftarrow P2 \uparrow$

On observe deux choses :

1. La donnée "a" n'existe plus (avant elle existait mais elle été inaccessible)
2. La donnée "b" existent en double et dans deux adresses différentes.

Chapitre 4

LES LISTES CHAÎNÉES

"L'imagination est plus importante que le savoir."
Albert Einstein

4.1 Introduction

Lorsqu'on souhaite utiliser des informations en grand nombre, on peut utiliser un tableau. Toutefois, la déclaration préférable de la taille de ce tableau demeure une contrainte, notamment lorsqu'il est impossible, à priori, de connaître le nombre d'élément qui seront nécessaires.

Le concepts de listes chaînées, que nous allons développer dans ce chapitre, permet de lever cette contrainte : chaque fois que l'on a besoin d'enregistrer en mémoire une information, on demande au compilateur de nous allouer l'espace nécessaire en mémoire. Ces information éparpillées en mémoire sont chaînées entre elles au moyen de variables de type pointeur.

4.2 Gestion dynamique de la mémoire

L'allocation de l'espace mémoire au fur et à mesure des besoins, et sa restitution à la fin du besoin, s'appelle : gestion dynamique de la mémoire.

Deux procédures permettent de gérer dynamiquement la mémoire :

La procédure nouveau(P) qui permet à chaque appel d'obtenir un espace mémoire dont l'adresse sera retournée dans la variable pointeur P.

La procédure laisser(P) qui permet de libérer un espace mémoire d'adresse P dont on plus besoin.

Ces deux procédures permettent d'obtenir et de rendre un espace mémoire au fur et à mesure des besoin de l'algorithme. On parle de gestion dynamique de la mémoire contrairement à la gestion statique des tableaux (dimension fixe).

4.3 Définition d'une liste chaînée

Une liste chaîne est un ensemble d'éléments d'informations appelées noeuds. En plus de l'information dont il est porteur, un noeud possède un pointeur. Ce pointeur contient l'adresse du noeud suivant dans la liste. C'est l'adresse du premier noeud qui détermine la liste. Cette adresse doit se trouver dans la variable que nous appellerons très souvent début.

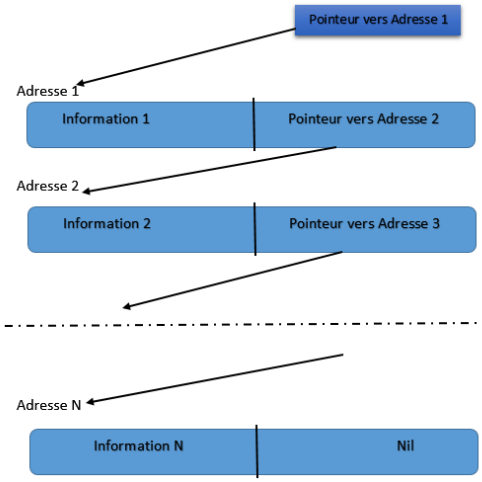


FIGURE 4.1 – Représentation graphique d'une liste

Une liste chaînée n'est pas limitée à un nombre fixe d'éléments. On peut insérer et supprimer des éléments de la liste autant que nécessaire, et ce, sans avoir besoin de connaître, à priori, le nombre d'éléments de la liste : c'est la gestion dynamique de la mémoire. Elle apporte donc une réponse optimale à bon nombre de problèmes d'implantation de l'information de l'information dans l'algorithme.

4.4 Le noeud

L'adresse du noeud est rangée dans la variable P. On dira noeud d'adresse P ou noeud pointé par P. Le noeud contient deux parties : La partie information contenant les variables **membre 1**, **membre 2**, ..., et le pointeur **suivant** contenant l'adresse.

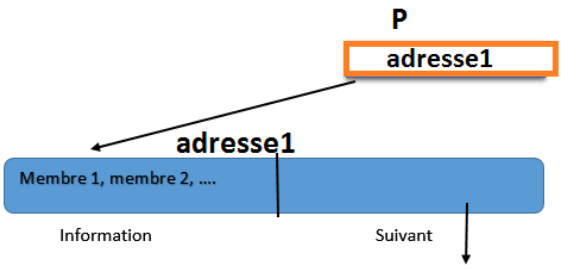


FIGURE 4.2 – Représentation graphique d'un noeud

4.5 Opérations sur les Noeuds

4.5.1 Déclaration d'un noeud

La syntaxe de déclaration d'un noeud est :

TYPE pointeur=↑ noeud
noeud=enregistrement
membre 1 :type 1
membre 2 :type 2
⋮
suivant :pointeur
Fin_ enregistrement
VAR P : pointeur ;

On définit un type pointeur sur des variables de type noeud, puis on définit le type noeud. Une variable P de type pointeur contient l'adresse d'une variable de type noeud. La variable P peut être représentée schématiquement par :

L'adresse du noeud est rangée dans la variable P. Par abus de langage, on dira noeud d'adresse P ou noeud pointé par P. Le noeud contient deux parties : La partie information contenant les variables **membre 1**, **membre 2**, ..., et le pointeur **suivant** contenant l'adresse.

On accède aux variables du noeud d'adresse P comme On accède au variables membres du noeud d'adresse P comme suit :

$$P\uparrow.membre1,\ P\uparrow.membre2, ...$$

4.5.2 Accès à un élément (noeud) d'une liste

Soit le noeud suivant, voire figure 4.2. On accède aux variables du noeud d'adresse P comme on accède au variables membres du noeud d'adresse P comme suit :

$$P\uparrow.membre1,\ P\uparrow.membre2, ...$$

On accède de la même façon à l'adresse contenue dans le noeud d'adresse P :

$$P\uparrow.suivant$$

4.6 Création d'une liste

Algorithme de création d'une liste
algorithme création_liste
type pointeur=↑etudiant
etudiant=enregistrement
nom :chaine de 20 caractères
prénom : chaîne de 20 caractères
age : entier
suivant : pointeur
fin enregistrement
variables début : pointeur ; R :caractère ;
debut

nouveau(debut) ;courant←debut ;
répéter
écrire("Entrer le nom, le prénom et l'age. SVP !")
lire(courant↑.nom,courant↑.prénom,courant↑.age)
écrire("Voulez vous continuer ? Y/N")
lire(R)
si (R='Y')**alors**
nouveau(courant↑.suivant)
courant←courant↑.suivant
sinon
courant↑.suivant←nil
fin si
jusqu'à (R='N')
fin

4.7 Manipulation des listes

4.7.1 Consultation d'une liste chaînée

{Lecture du N^{eme} élément de la liste} *courant←début ;*
pour *i allant de 1 à (n-1) faire*
courant←courant↑ .suivant
fin pour*crire(courant↑.nom,courant↑.prénom,courant↑.age)*

4.7.2 Insertion d'un élément à la tête d'une liste chaînée

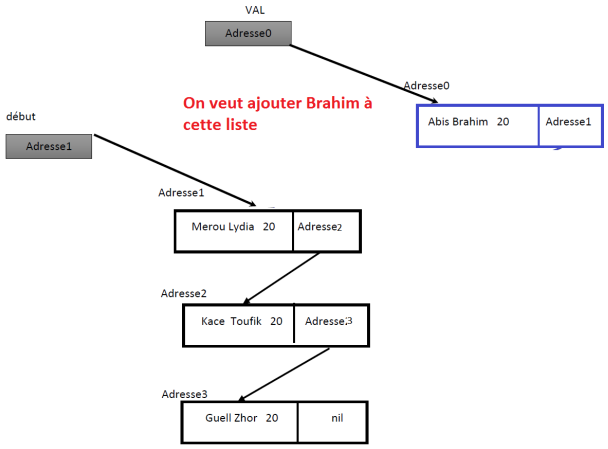


FIGURE 4.3 – Ce qu'on veut faire !

On veut ajouter un élément à la tête d'une liste : comme s'est indiqué dans la figure suivante 4.3.

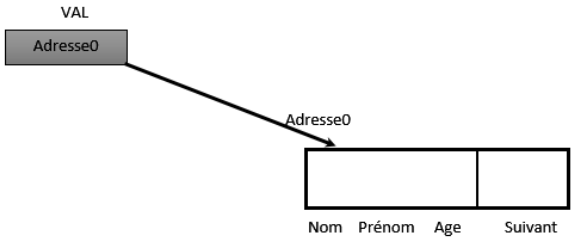


FIGURE 4.4 – Première étape : Nouveau(Val)

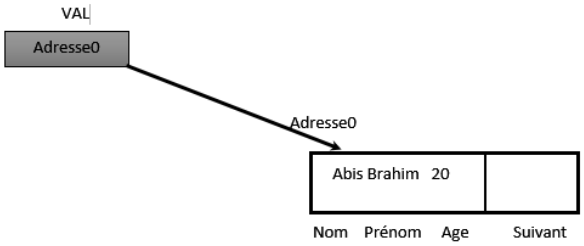


FIGURE 4.5 – Deuxième étape : lire("Val↑.nom,Val↑.prénom,Val↑.age")

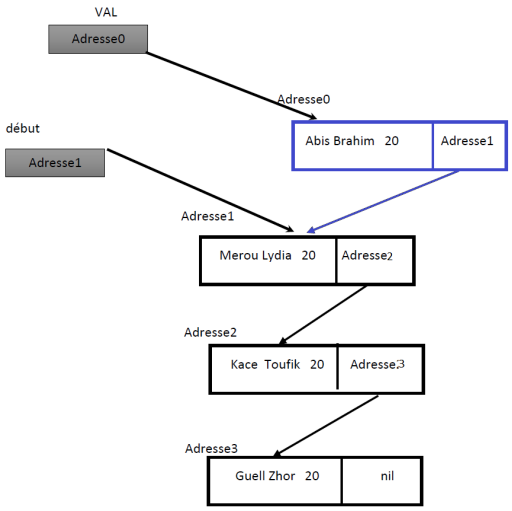


FIGURE 4.6 – Troisième étape : Val↑.suivant←début ;

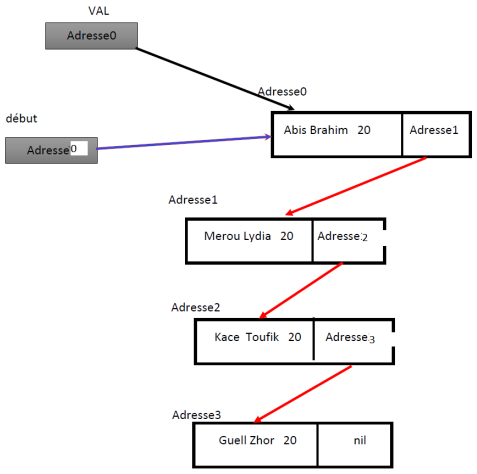


FIGURE 4.7 – Quatrième étape : début←Val ;

Algorithme générale

{Étape 1 : Création d'un noeud "Val" de type Pointeur}
Nouveau(Val) ;
{Étape 2 : Saisi des valeur du noeud "Val"}
écrire("Entrez un nom, un prénom et un age") ;
lire("Val↑.nom, Val↑.prénom, Val↑.age") ;
{Étape 3 : Chaînage du nouvel élément "Val" }
Val↑.suivant←début ;
{Étape 4 : Le nouvel élément "Val" devient le premier élément de la liste }
début←Val ;

4.7.3 Insertion d'un élément en queue de liste

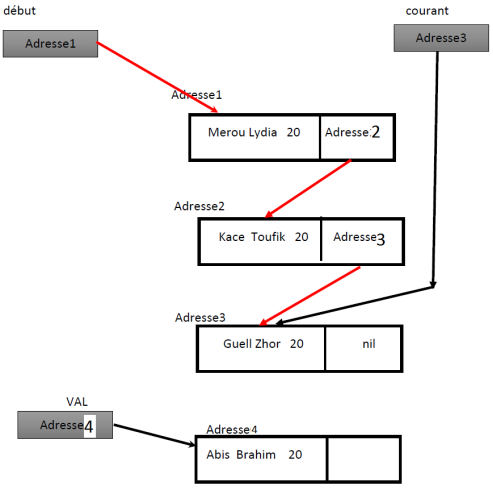


FIGURE 4.8 – Troisième étape : courant←début ;
tant que courant↑.suivant<>nil faire
courant← courant↑.suivant ;
fin tant que

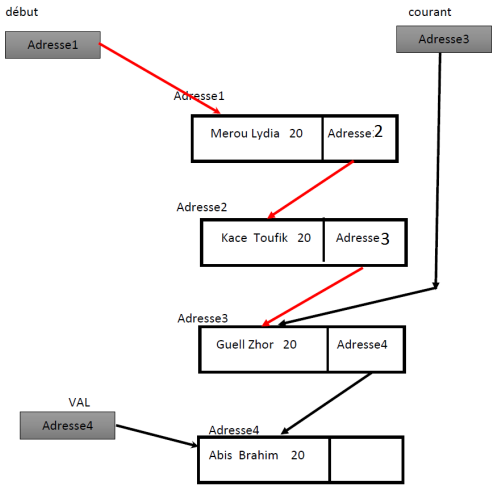


FIGURE 4.9 – Quatrième étape : courant↑.suivant←Val ;

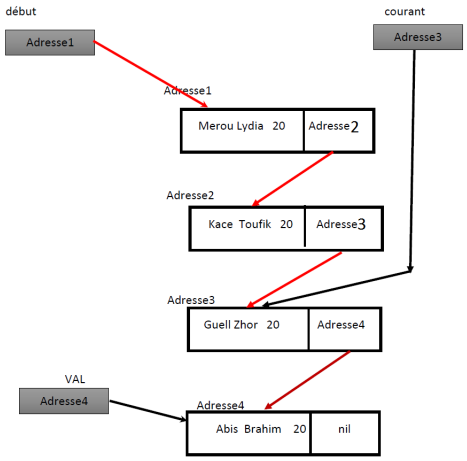


FIGURE 4.10 – Cinquième étape : Val.suivant←nil ;

Algorithme générale

{Étape 1 : Création d'un noeud "Val" de type Pointeur}
Nouveau(Val) ;
{Étape 2 : Saisi des valeur du noeud "Val"}
écrire("Entrez un nom, un prénom et un age") ;
lire("Val↑.nom,Val↑.prénom,Val↑.age") ;
{Étape 3 : Recherche du dernier élément}
courant←début ;
tant que courant↑.suivant<>nil faire
courant← courant↑.suivant ;
fin tant que
{Étape 4 : Chaînage du nouvel élément "Val" }
courant↑.suivant←Val ;
{Étape 5 : Le nouvel élément "Val" devient le dernier élément de la liste}
Val.suivant←nil ;

4.7.4 Insertion d'un élément à la k^{eme} position

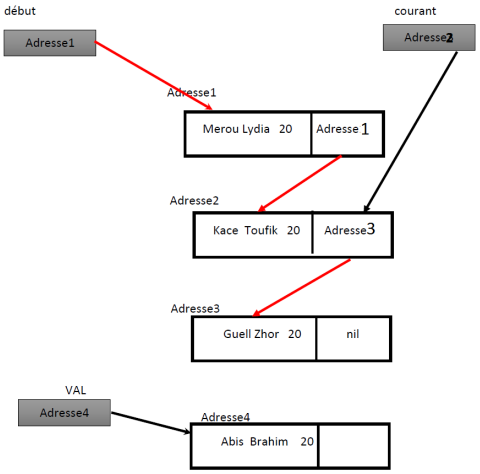


FIGURE 4.11 – Troisième étape : courant←début ;$i \leftarrow 1$;
tant que courant↑.suivant<>nil ou $(i < k - 1)$ faire
courant← courant↑.suivant ; i←i+1 ;
fin tant que

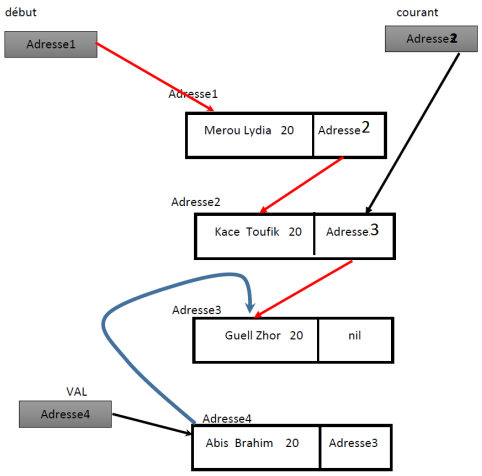


FIGURE 4.12 – Quatrième étape : Val↑.suivant←courant↑.suivant ;

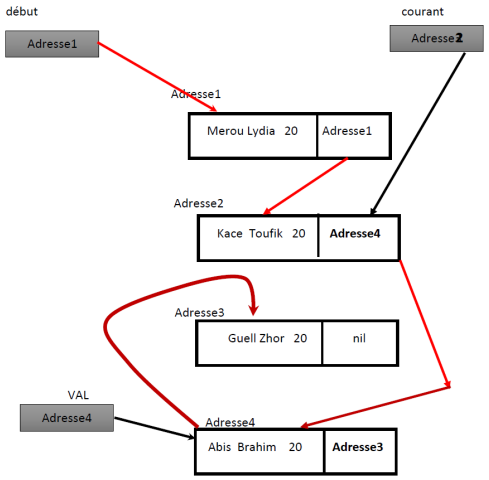


FIGURE 4.13 – Cinquième étape : courant↑.suivant ←Val ;

Algorithme générale

{Étape 1 : Création d'un noeud "Val" de type Pointeur}
Nouveau(Val) ;
{Étape 2 : Saisi des valeur du noeud "Val"}
écrire("Entrez un nom, un prénom et un age") ;
lire("Val↑.nom,Val↑.prénom,Val↑.age") ;
{Étape 3 : Recherche de la k^{me} position}
courant←début ;i=1 ;
tant que courant↑.suivant<>nil ou $(i < k-1)$ faire
courant← courant↑.suivant ; i←i+1 ;
fin tant que
{Étape 4 : Chaînage du nouvel élément "Val" }
Val↑.suivant←courant↑.suivant ;
{Étape 5 : Le nouvel élément "Val" devient le k^{me} élément de la liste}
courant↑.suivant ← Val ;

4.8 Listes particulières

4.8.1 Listes circulaires

C'est une liste où le dernier noeud (cellule) pointe le premier, formant ainsi un cercle. La tête de la liste est l'adresse de n'importe quel noeud de la liste circulaire. Le même modèle des LLC est utilisé pour écrire des algorithmes sur ce type de listes.

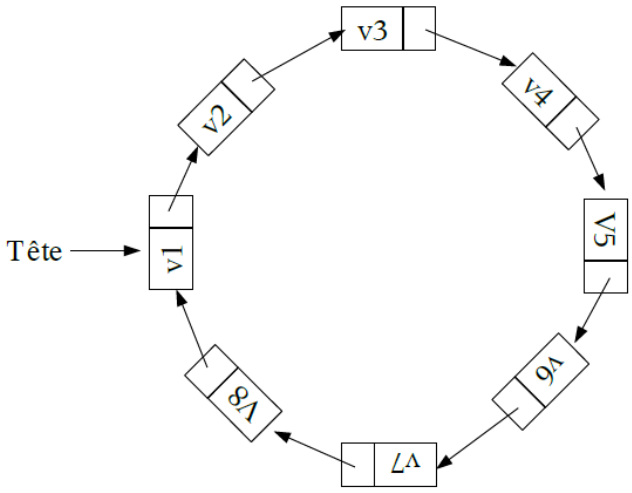


FIGURE 4.14 – Représentation graphique d'une liste circulaire

4.8.2 Liste Bi-directionnelle

Ce sont des listes que l'on peut parcourir dans les deux sens : de gauche à droite et de droite à gauche. Pour cela on rajoute dans chaque noeud un deuxième pointeur indiquant l'adresse du noeud précédent.

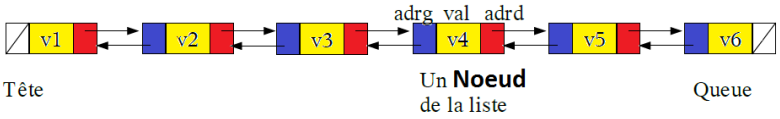


FIGURE 4.15 – Représentation graphique d'une liste bi-directionnelle

Chapitre 5

LES PILES

L'art de découvrir en mathématique est plus précieux que la plupart des choses qu'on découvre
Leibnitz.

5.1 Définition

Définition 3. *Une pile est une liste d'éléments telle qu'un élément ne peut lui être ajouté ou retiré que par une extrémité appelée* sommet *de la pile. En terme plus pratique, on dira que les éléments sont retirés par l'ordre inverse par lequel il ont été ajoutés (Dernier Entré, Premier Sorti, LIFO en anglais)*

- ☞ *Le dernier élément d'une pile est appelé* ***base*** *de la pile*
- ☞ *Le premier élément d'une pile est appelé* ***sommet*** *de la pile.*

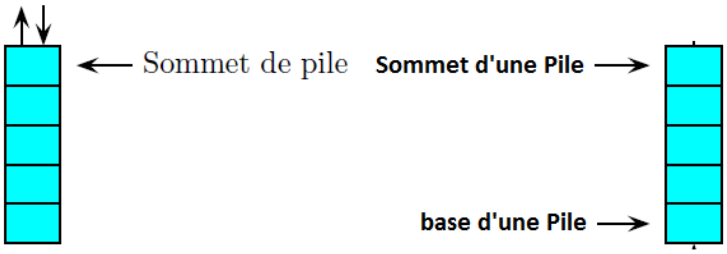


FIGURE 5.1 – Représentation Graphique d'une pile

5.1.1 Opérations sur les piles

PileVide Une fonction qui retourne vrai au faux selon que la pile est vide ou non.

PilePleine Une fonction qui retourne vrai ou faux selon que la pile est pleine ou non, utilisée seulement dans le cas statique.

Empiler(P,V) Insère l'élément V dans la pile P, si P n'est pas pleine. (sinon elle **déborde**).

Dépiler(P,V) Si la pile n'est pas vide, retirer l'élément qui se trouve au sommet de la pile et mettre dans la variable V. Si la pile est vide alors, on dit qu'**elle déborde négativement**.

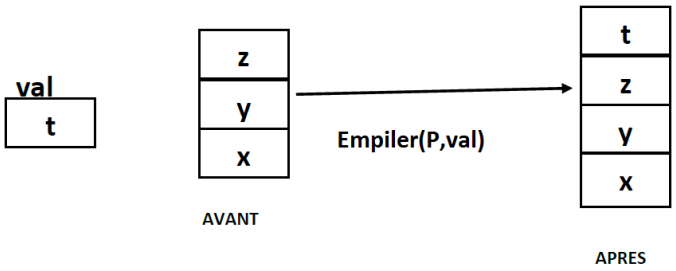


FIGURE 5.2 – La procédure Empiler

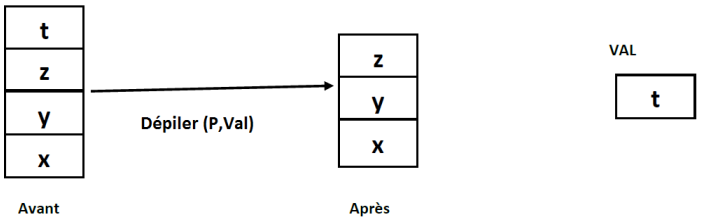


FIGURE 5.3 – La procédure Dépiler

5.2 Modélisation par tableau

La première manière de modéliser une pile consiste à utiliser un tableau. L'ajout d'un élément se fera à la suite du dernier élément du tableau. Le retrait d'un élément de la pile se fera en enlevant le dernier élément du tableau. La figure suivante représente une pile par cette modélisation. Elle reprend la même pile que pour l'exemple de modélisation par liste chaînée.

5.2.1 Déclaration d'une pile à l'aide de tableaux

Type Pile=enregistrement
Sommet :entier
info :tableau [1,..,Val_Max] de type quelconque
fin enregistrement ;
Var P :Pile ;

Avantage : Accès rapide à l'information

Inconvénient : Gaspillage ou insuffisance de l'espace mémoire à cause de la limitation de la taille du tableau

5.3 Modélisation par liste chaînée

La première façon de modéliser une pile consiste à utiliser une liste chaînée en n'utilisant que les opérations empiler et dépiler. Dans ce cas, on s'aperçoit que le dernier élément entré est toujours le premier élément sorti. La figure suivante représente une pile par cette modélisation. La pile contient les chaînes de caractères suivantes qui ont été ajoutées dans cet ordre : "Alger", "Boumerdès", "Bouira" et "Béjaia".

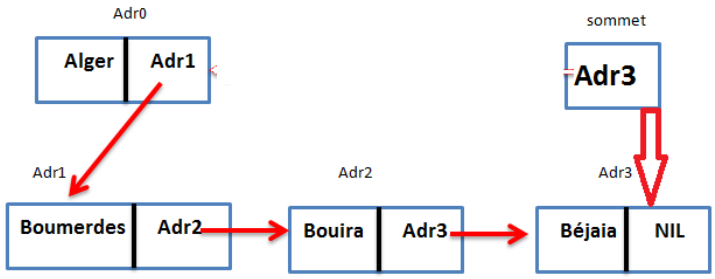


FIGURE 5.4 – Représentation graphique d'une Piles avec les listes chaînées

Pour cette modélisation, la structure d'une pile est celle d'une liste chaînée.

5.3.1 Déclaration d'une pile à l'aide de liste chaînée

Type $P = \uparrow pile$
pile=enregistrement
info : de type quelconque
suivant :P
fin enregistrement ;
Var P1 :P

Avantage : Gain en espace mémoire.

Inconvénient : Accès moins rapide.

5.3.2 Saisie d'une Pile

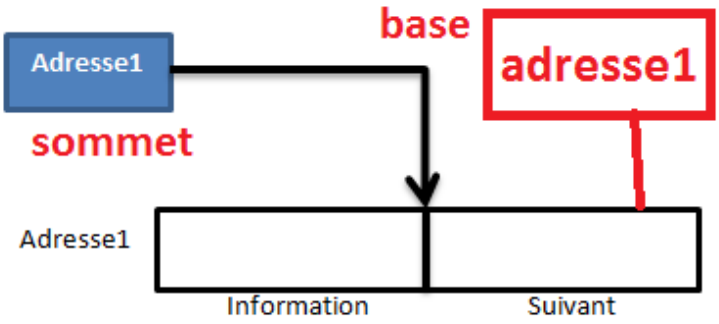


FIGURE 5.5 – Nouveau(sommet) ;$base \leftarrow sommet$

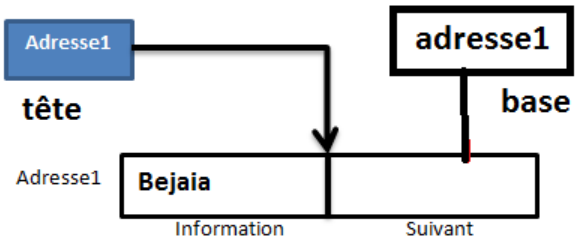


FIGURE 5.6 – écrire("Entrez le nom de la ville")Lire($sommet \uparrow .Information$)

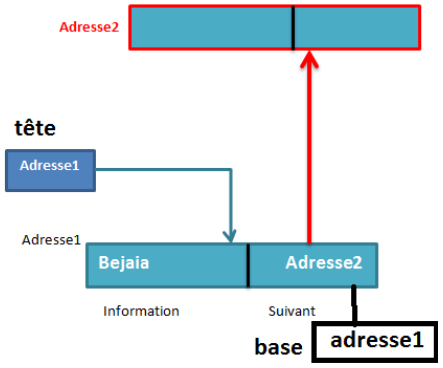


FIGURE 5.7 – Nouveau($sommet \uparrow .Suivant$)

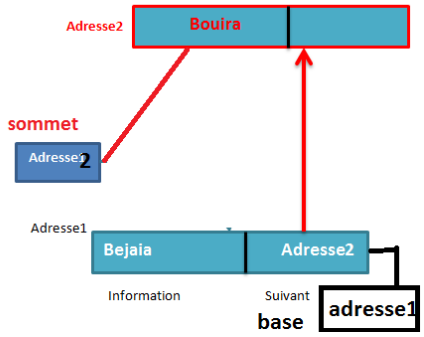


FIGURE 5.8 – $sommet \leftarrow sommet \uparrow .suivant$
écrire("Entrez le nom de la ville")
Lire($sommet \uparrow .Information$)

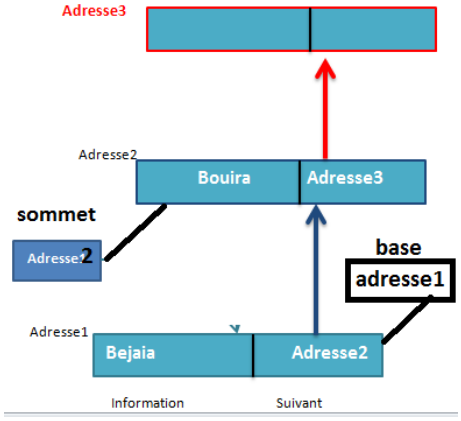


FIGURE 5.9 – Nouveau($sommet \uparrow .suivant$)

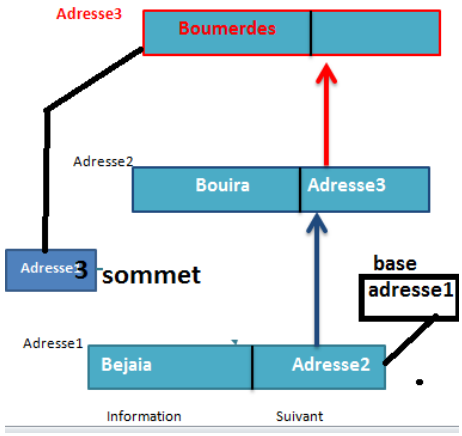


FIGURE 5.10 – *sommet* ← *sommet* ↑ *.suivant*
écrire("Entrez le nom de la ville") ; Lire(*sommet* ↑ *.Information*)

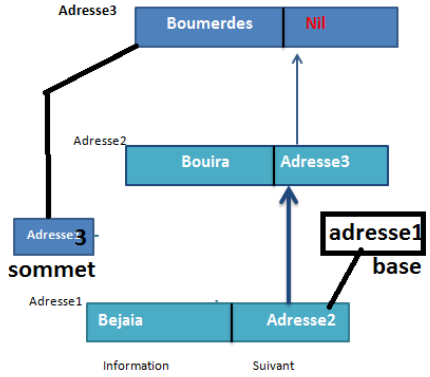


FIGURE 5.11 – *sommet* ↑ *.suivant* ← *Nil*

Chapitre 6

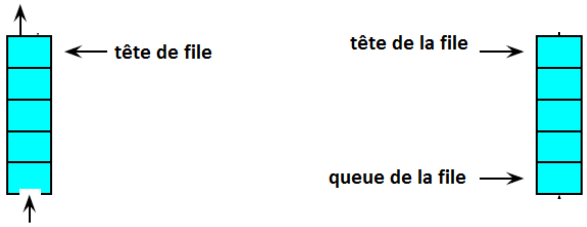
LES FILES

"Ne vous occuper pas d'une belle présentation des résultats avant que le programme soit correct"
Anonyme

6.1 Définitions

6.1.1 Qu'es qu'une file ?

Une file est une structure de données dans laquelle l'opération d'insertion se fait à la fin de la pile et l'opération de retrait se fait au début de la file.



6.2 Opérateurs sur les files

Quatres opérateurs principales peuvent être définies pour manipuler une file :

- file_vide(F) : est une fonction qui retourne vrai si la file est vide, elle retourne faux sinon.
- file_pleine(F) : est une fonction qui retourne vrai si la file pleine, elle retourne faux sinon.
- enfiler(F,V) : est une procédure qui consiste à insérer une valeur V à la fin de la file.
- défiler(F,V) : est une procédure qui consiste à retirer l'élément qui se trouve au début de la file. Ceci est possible si la file est non vide.

6.3 Implémentation statique

L'implémentation statique utilise un tableau pour stocker les éléments de la file.

6.3.1 Déclaration

type file=enregistrement
tete,queue :entier ;
information : tableau[1,..,val_max] de type qlq ;
fin enregistrement
var F :file ;

6.4 Implémentation dynamique

On utilise une liste linéaire chaînée

6.4.1 Déclaration

type file=enregistrement
information :type qlq ;
suivant : ↑file
fin enregistrement
var tete,F :file ;

6.5 Saisie d'une File

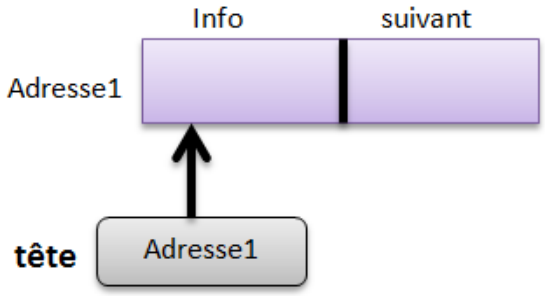


FIGURE 6.1 – nouveau(tete)

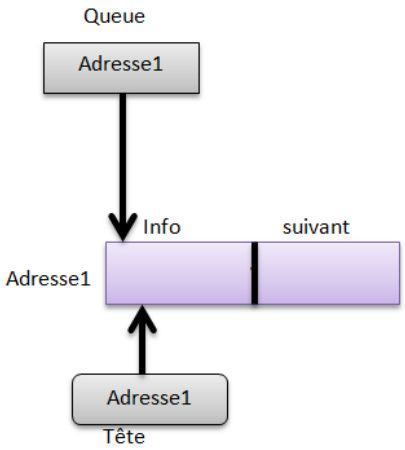


FIGURE 6.2 – queue←tete

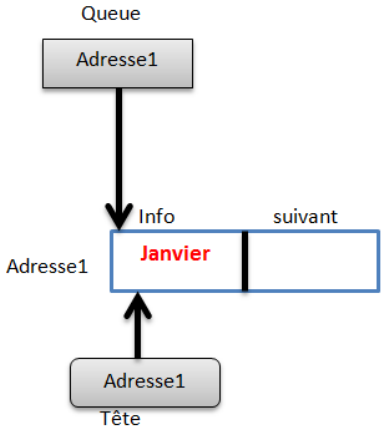


FIGURE 6.3 – Écrire("Entrez un mois")Lire(queue↑.information)

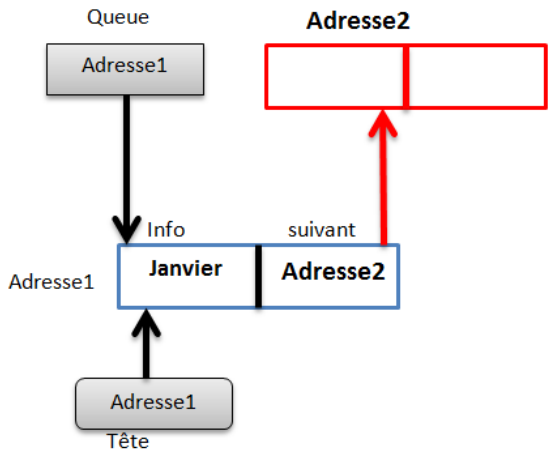


FIGURE 6.4 – Nouveau(queue↑.suivant)

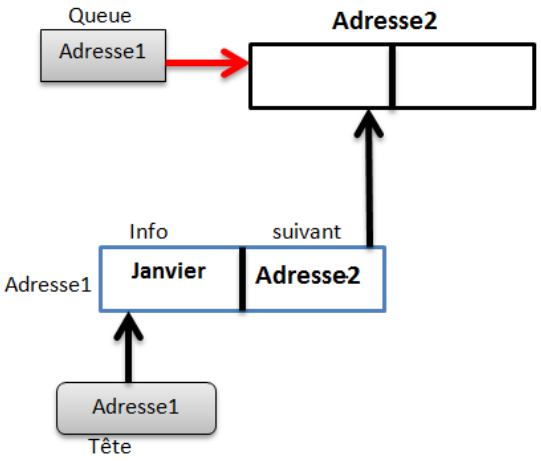


FIGURE 6.5 – queue←queue↑ .*suivant*

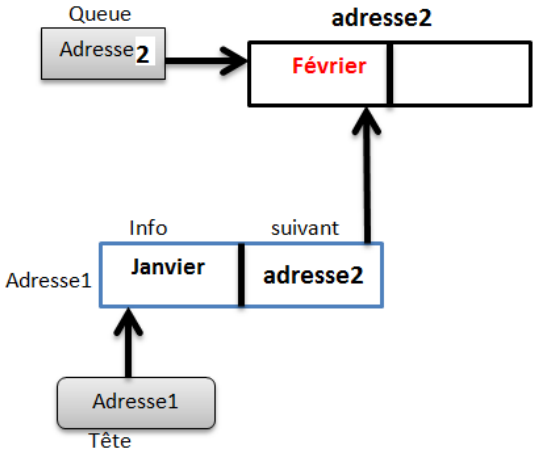


FIGURE 6.6 – Écrire("Introduire le mois") ; Lire(queue↑.information)

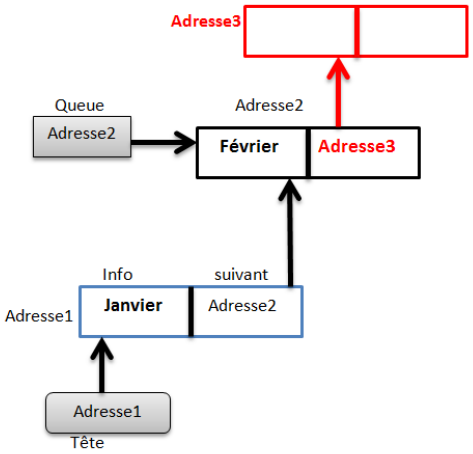


FIGURE 6.7 – Nouveau(queue↑.suivant)

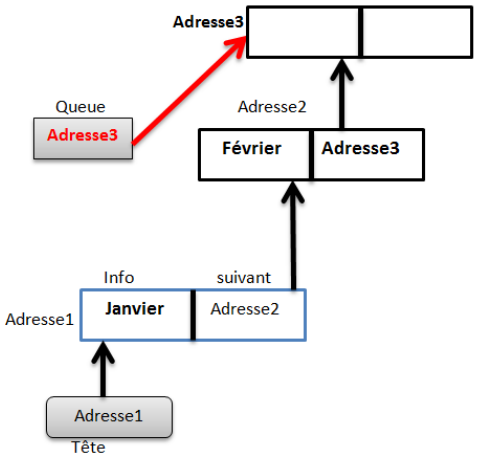


FIGURE 6.8 – *queue* ← *queue* ↑ *.suivant*

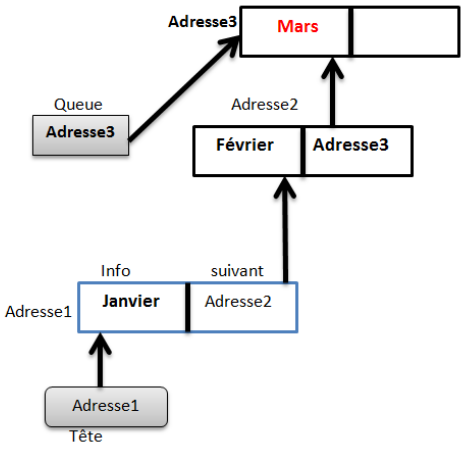


FIGURE 6.9 – Écrire("Introduire un mois") ;Lire(queue↑.information)

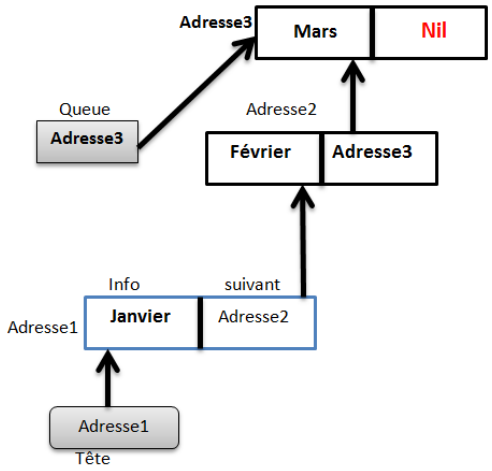


FIGURE 6.10 – *queue* ↑ *.suivant* ← *Nil*

Chapitre 7

LES ARBRES

7.1 Définitions

Définition 4. *Un arbre est un ensemble d'éléments appelés noeuds liés par une relation dite de parenté, induisant une structure hiérarchique parmi ces noeuds*

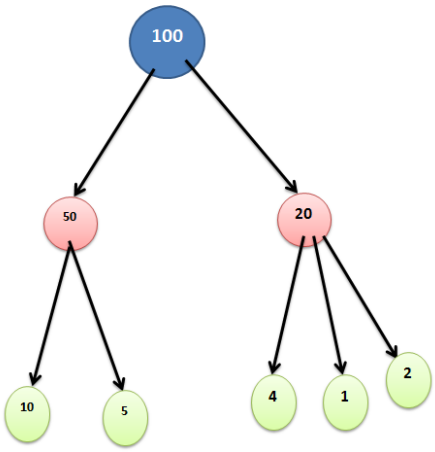


FIGURE 7.1 – Exemple d'arbre

7.2 Terminologie

Fils *chaque noeud pointe vers un ensemble éventuellement vide d'autre noeuds. Chaque élément de cet ensemble pointé est appelé fils. Dans l'exemple : 10 et 5 sont des fils.*

Père *Un père P est un père de F ssi F est le fils de P. Dans l'exemple 50 est à la fois père de 10 et 5, et aussi fils de 100.*

Racine *Est le seul noeud sans père. Dans l'exemple : 100 est une Racine.*

Feuille *Est un noeud sans fils.*

Noeud Interne *Est un noeud qui n'est pas terminale (pas une feuille). Dans l'exemple : 50 et 20 sont des noeuds internes.*

Degré d'un noeud *est le nombre de ses fils. Dans l'exemple : le degré de 50 est 2.*

Degré d'un arbre *est le plus grand degré d'un noeud de l'arbre. Sur l'exemple : le degré de l'arbre est : 3 correspondant au degré de 20 qui est de 3*

Sous arbre *Un sous arbre d'un noeud de A est composé de tous les descendants d'un noeud de A. Sur l'exemple :* $\{50, 10, 5\}$, $\{20, 4, 1, 2\}$ *sont deux sous arbre du noeud 100 (racine)*

Hauteur (profondeur ou niveau) d'un noeud *est la longueur du chemin qui relie la racine à ce noeud. sur l'exemple :*
la hauteur de la racine 100 c'est 0.
la hauteur du noeud 12 est 50. la hauteur du noeud 1 est 3.

Hauteur d'un arbre *Plus grande profondeur des noeuds d'un arbre. La hauteur de l'arbre est 3.*

7.3 Arbre Binaire

Définition 5. *est un arbre ou chaque à au plus deux fils.*

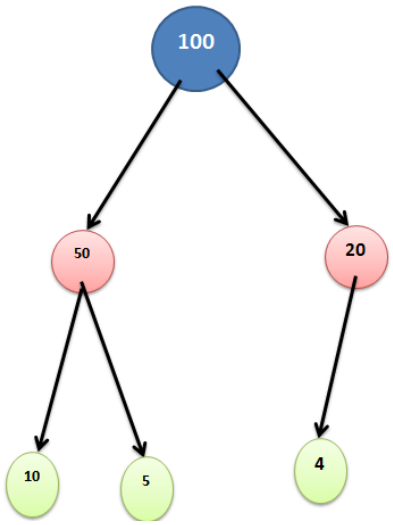


FIGURE 7.2 – Exemple d'arbre Binaire