Travaux Dirigés :
Algorithmique et Structure de Données

Enseignant : Maher Helaoui

Maher Helaoui est assistant contractuel à l'Institut Supérieur d'Informatique de Mahdia, Université de Monastir, TUNISIE. Depuis 2007.

Il est aussi chercheur en Intelligence Artificielle, au Pôle de Recherche en Informatique du Centre (PRINCE), à la Faculté des Sciences de Monastir **FSM**.

Contact : **maher.helaoui@gmail.com**

Site : **www.sites.google.com/site/maherhelaoui/Home/**

(ISIMa 2011)

TABLE DES MATIERES

INTRODUCTION

L'objectif de ce document est de présenter aux étudiants un résumé du cours algorithmique et structures de données.

Par la suite, une sélection d'exercices sera proposée. Les exercices sont choisis de sorte à aider les étudiants à développer une solution algorithmique qui résout un problème proposé.

Le document propose la solution de quelques exercices. La solution est destinée aux étudiants qui n'ont pas pu assister à une séance de Travaux Dirigés (TD), elle permet de les aider pour se rattraper…

CHAPITRE 1

LES VARIABLES

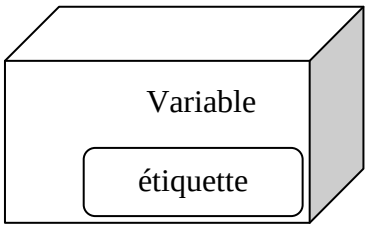
1. Objectif

Maitriser les notions : variable, type et valeur.

2. Déclaration

Un programme a besoin de stocker provisoirement des valeurs (information). Ces valeurs peuvent être de plusieurs types : elles peuvent être des nombres, du texte, etc. Toujours est-il que dès que l'on a besoin de stocker une information dans un programme, on utilise une **variable**.

Pour la schématiser, une variable est une boîte, repérée par une étiquette. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.



NB: Dans la mémoire vive de l'ordinateur, cette boîte et cette étiquette collée dessus n'existent pas. Il y a plutôt un emplacement de mémoire, désigné par une adresse binaire. La **compilation** d'un **langage informatique** se charge de nous épargner la gestion fastidieuse de ces emplacements mémoire et de leurs adresses. En affectant une étiquette choisie par le programmeur pour chaque adresse binaire.

La première chose à faire avant de pouvoir utiliser une variable est de préparer son emplacement mémoire : nous allons alors créer la boîte et lui coller une étiquette. Ceci se fait (pour les **variables globales**) au début de l'algorithme, avant même les **instructions** proprement dites. C'est ce qu'on appelle la **déclaration des variables**.

3. Type des variables

Pour créer une boîte (réserver un emplacement mémoire) nous devons préciser sa taille. Elle doit correspondre à ce que l'on voudra mettre dedans. Ça sera un gaspillage de mémoire de créer une boite plus grande par rapport à notre besoin. Et notre boite ne pourra pas stocker l'information dont nous avons besoin si elle n'a pas une taille suffisante. C'est

comme l’exemple de construire un château pour stocker un oiseau ou construire une petite cage pour stocker un éléphant.

Types numériques classiques

Type Numérique	Plage
Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	-3,40E38 à -1,40E-45 pour les valeurs négatives 1,40E-45 à 3,40E38 pour les valeurs positives
Réel double	1,79E308 à -4,94E-324 pour les valeurs négatives 4,94E-324 à 1,79E308 pour les valeurs positives

Question : Pourquoi ne pas déclarer toutes les variables numériques en réel double? (Réponse : pour éviter un gaspillage de mémoire réservée.)

Une déclaration algorithmique de variables aura ainsi cette forme :

Variable g **en Entier Long**

Variables PrixHT, TauxTVA, PrixTTC **en Réel Simple**

Types non numériques

Nos boîtes peuvent contenir une information autre que des nombres. Sans cela, on serait un peu embêté dès que l'on devrait stocker un nom de famille, par exemple.

On dispose donc du **type alphanumérique** (également appelé **type caractère**) : dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou de chiffres.

Une série de caractères forme une **chaîne de caractères**. Et une telle chaîne de caractères est toujours notée entre guillemets. Comme 2010 peut représenter le nombre 2010, ou la suite de caractères 2, 0, 1 et 0.

Un autre type est le **type booléen** : on y stocke uniquement les valeurs logiques **VRAI** et **FAUX**.

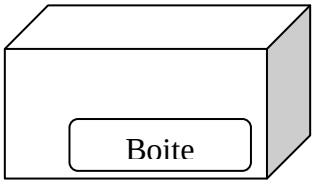
4. L'instruction d'affectation

Comme nous l'avons déjà présenté une variable permet de stocker une information, cette opération de stockage se fait à travers **l'affectation, c'est-à-dire lui attribuer une valeur**. En algorithmique, cette instruction se note avec le signe ←.

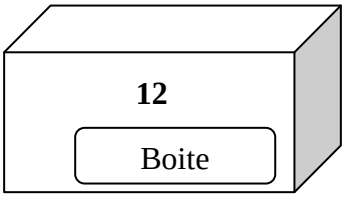
Exemple :

Variable Boite **en Entier**

//Cette ligne permet de réserver un espace mémoire suffisant pour contenir un entier au //niveau de la déclaration notre boite est vide.

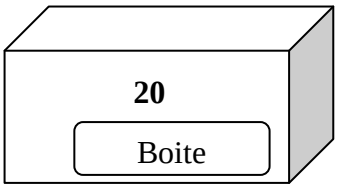


Boite ← 12



//Cette instruction affecte la valeur 12 dans notre boite

Boite ← 20



//Cette nouvelle instruction écrase l'ancienne valeur 12 et affecte la nouvelle valeur 20 //dans notre boite.

TRAVAUX DIRIGES 1

LES VARIABLES

Exercice 1

Soit l'algorithme ***Test1*** suivant :

```
Algorithme Test1
    A ← 4          (1)
    B ← 11         (2)
    A ← B - A      (3)
    B ← B - A      (4)
    A ← A + B      (5)
Fin
```

1. Que fait l'algorithme ci-dessus ?

2. Ce résultat est-t-il toujours vrai ? Etablir la trace de cet algorithme (sous forme de tableau) avec a et b pour valeurs initiales de A et B.

3. Ecrire un nouvel algorithme équivalant en n'utilisant que des variables ? (Utiliser une variable intermédiaire).

4. Que font les deux algorithmes ci-dessous ? (Trouver pour chacun ce qu'il fait grâce à un exemple puis le prouver)

```
Algorithme Test2
Début
    A ← A+B+C      (1)
    B ← B+C        (2)
    C ← A-C        (3)
    A ← A-C        (4)
    B ← C-B+A      (5)
    C ← C-B        (6)
Fin
```

```
Algorithme Test3
Début
    B ← B+C        (1)
    C ← B-C        (2)
    B ← B-C        (3)
    A ← A+B        (4)
    B ← A-B        (5)
    A ← A-B        (6)
Fin
```

5. Ecrire un algorithme permettant de réaliser la permutation circulaire de trois variables d'un même type quelconque A, B et C en effectuant d'abord la transposition des variables A et B, ensuite celle des variable B et C. (NB : On utilisera une seule variable intermédiaire.)

6. Ecrire un algorithme qui réalise directement la permutation circulaire des trois variables A, B et C en utilisant une variable intermédiaire et en ayant la décomposition de la permutation en transpositions.

7. Dénombrer les affectations et les opérations (additions et soustractions) effectuées lors de l'exécution des 4 algorithmes ci-dessus (questions 4 à 6). Sachant qu'un ordinateur prend une petite fraction de seconde pour effectuer chaque opération ou affectation,

8. Ecrire un algorithme réalisant la permutation circulaire des variables A, B, C et D. Qui sera optimisé au regard du temps d'exécution. Combien comporte-t-il d'affectations? Généraliser au cas de n variables.

Exercice 2

1. Compléter par le symbole de comparaison adéquat (< ou >):

"AUTO"	…..	"AVION"
1997	….	43
"1997"	….	"43"
"BAL"	….	"BALLON"

Exercice 3

2. Ecrire un algorithme complet pour calculer le périmètre du rectangle, en utilisant au mieux les instructions Saisir et Afficher, puis y ajouter le calcul de la surface.

3. Ecrire un algorithme qui demande le Prix Hors Taxes et la quantité d'un article, puis afficher une facture bien présentée (Taux de TVA est de 20,6%)

TRAVAUX DIRIGES 1

CORRECTION TD 1

Exercice 1

Soit l'algorithme ***Test1*** suivant :

```
Algorithme Test1
   A ← 4          (1)
   B ← 11         (2)
   A ← B - A      (3)
   B ← B - A      (4)
   A ← A + B      (5)
Fin
```

1. Que fait l'algorithme ci-dessus ?

Cet algorithme assure la permutation des valeurs des Variables A et B.

Instructions	A	B
1	4	--
2	4	11
3	7	11
4	7	4
5	11	4

2. Ce résultat est-t-il toujours vrai ? Etablir la trace de cet algorithme (sous forme de tableau) avec a et b pour valeurs initiales de A et B.

Oui, cet algorithme assure la permutation des valeurs des variable A et B quelque soit les valeurs initiaux a de la variable A et b de la variable B.

Instruction	A	B
1	a	--
2	a	b
3	b-a	b
4	b-a	a
5	b	a

3. Ecrire un nouvel algorithme équivalant en n'utilisant que des variables ? (Utiliser une variable intermédiaire).

```
Algorithme : Permuter2
// Permutation de deux variables
Variables A, B, Loulou : Entier
Début
    Afficher («Veillez donner la valeur de la variable A »)
    Saisir(A)
    Afficher («Veillez donner la valeur de la variable B»)
    Saisir(B)

    Loulou ← A
    A ← B
    B ← Loulou
    Afficher («Après exécution de l'algorithme»)
    Afficher («La valeur de la variable A est de », A)
    Afficher («La valeur de la variable B est de », B)
Fin
```

4. Que font les deux algorithmes ci-dessous ? (Trouver pour chacun ce qu'il fait grâce à un exemple puis le prouver)

```
Algorithme Test2
Début
    A ← A+B+C      (1)
    B ← B+C        (2)
    C ← A-C        (3)
    A ← A-C        (4)
    B ← C-B+A      (5)
    C ← C-B        (6)
Fin
```

```
Algorithme Test3
Début
    B ← B+C        (1)
    C ← B-C        (2)
    B ← B-C        (3)
    A ← A+B        (4)
    B ← A-B        (5)
    A ← A-B        (6)
Fin
```

Algorithme Test2 :

Instruction	A	B	C
0	a	b	c
1	a+b+c	b	c
2	a+b+c	b+c	c
3	a+b+c	b+c	a+b
4	c	b+c	a+b
5	c	a	a+b
6	c	a	b

Algorithme Test3 :

Instruction	A	B	C
0	a	b	c
1	a	b+c	c
2	a	b+c	b
3	a	c	b
4	a+c	c	b
5	a+c	a	b
6	c	a	b

Les algorithmes 2 et 3 font la permutation circulaire des variables A, B et C.

5. Ecrire un algorithme permettant de réaliser la permutation circulaire de trois variables d'un même type quelconque A, B et C en effectuant d'abord la transposition des variables A et B, ensuite celle des variable B et C. (NB : On utilisera une seule variable intermédiaire.)

```
Algorithme 4 Permuter3_1
// Permutation circulaire de trois variables en effectuant d'abord la transposition des variables A et B, ensuite celle des variable B et C.
Variables A, B, C, Loulou : Entier
Début
    Afficher («Veillez donner la valeur de la variable A »)
    Saisir(A)
    Afficher («Veillez donner la valeur de la variable B»)
    Saisir(B)
    Afficher («Veillez donner la valeur de la variable C»)
    Saisir(C)
    Afficher («Au début»)
    Afficher («La valeur de la variable A est de », A)
    Afficher («La valeur de la variable B est de », B)
    Afficher («La valeur de la variable C est de », C)
    Loulou ← A
    A ← B
    B ← Loulou
    Loulou ← B
    B ← C
    C ← Loulou
    Afficher («Après exécution de notre algorithme»)
    Afficher («La valeur de la variable A est de », A)
    Afficher («La valeur de la variable B est de », B)
    Afficher («La valeur de la variable C est de », C)
Fin
```

Ecrire un algorithme qui réalise directement la permutation circulaire des trois variables A, B et C en utilisant une variable intermédiaire et en ayant la décomposition de la permutation en transpositions.

```
Algorithme 5 Permuter3_2
//Permutation circulaire de trois variables en utilisant une variable intermédiaire et
en ayant la décomposition de la permutation en transpositions..
Variables A, B, C, Loulou : Entier
Début
   Afficher («Veillez donner la valeur de la variable A »)
   Saisir(A)
   Afficher («Veillez donner la valeur de la variable B»)
   Saisir(B)
   Afficher («Veillez donner la valeur de la variable C»)
   Saisir(C)
   Afficher («Au début»)
   Afficher («La valeur de la variable A est de », A)
   Afficher («La valeur de la variable B est de », B)
   Afficher («La valeur de la variable C est de », C)
   Loulou ← A
   A ← C
   C ← B
   B ← Loulou
   Afficher («Après exécution de notre algorithme»)
   Afficher («La valeur de la variable A est de », A)
   Afficher («La valeur de la variable B est de », B)
   Afficher («La valeur de la variable C est de », C)
Fin
```

6. Dénombrer les affectations et les opérations (additions et soustractions) effectuées lors de l'exécution des 4 algorithmes ci-dessus (questions 4 à 6). Sachant qu'un ordinateur prend une petite fraction de seconde pour effectuer chaque opération ou affectation,

Algorithm 2 : 14 fractions de seconde

Algorithm 3 : 12 fractions de seconde

Algorithm 4 : 6 fractions de seconde

Algorithm 5 : 4 fractions de seconde

L'algorithme 5 coûte le moins en temps.

7. Ecrire un algorithme réalisant la permutation circulaire des variables A, B, C et D. Qui sera optimisé au regard du temps d'exécution. Combien comporte-t-il d'affectations? Généraliser au cas de n variables.

Algorithme 6
// Permutation circulaire de trois variables en utilisant une variable intermédiaire et en ayant la décomposition de la permutation en transpositions..
Variables A, B, C, D, Loulou en Entier
Début
Afficher («Veillez donner la valeur de la variable A »)
Saisir(A)
Afficher («Veillez donner la valeur de la variable B»)
Saisir(B)
Afficher («Veillez donner la valeur de la variable C»)
Saisir(C)
Afficher («Veillez donner la valeur de la variable D»)
Saisir(D)
Afficher («Au début»)
Afficher («La valeur de la variable A est de », A)
Afficher («La valeur de la variable B est de », B)
Afficher («La valeur de la variable C est de », C)
Afficher («La valeur de la variable D est de », D)
Loulou ← A
A ← D
D ← C
C ← B
B ← Loulou
Afficher («Après exécution de notre algorithme»)
Afficher («La valeur de la variable A est de », A)
Afficher («La valeur de la variable B est de », B)
Afficher («La valeur de la variable C est de », C)
Afficher («La valeur de la variable D est de », D)
Fin

Exercice 2

1. Compléter par le symbole de comparaison adéquat (< ou >):

"AUTO"	…..	"AVION"
1997	>	43
"1997"	….	"43"
"BAL"	….	"BALLON"

Exercice 3

1. Ecrire un algorithme complet pour calculer le périmètre du rectangle, en utilisant au mieux les instructions Saisir et Afficher, puis y ajouter le calcul de la surface.

2. Ecrire un algorithme qui demande le Prix Hors Taxes et la quantité d'un article, puis afficher une facture bien présentée (Taux de TVA est de 20,6%)

```
Algorithme Calcul du périmètre du rectangle
Variables L, l, P en Entier
Début
Afficher («Veillez donner la longueur du rectangle »)
Saisir(L)
Afficher («Veillez donner la largeur du rectangle»)
Saisir(l)
P ← (L+l)*2
Afficher («Le périmètre de votre rectangle est de : », P)
Fin
```

CHAPITRE 2

LES STRUCTURES DE CONTROLE

1. Objectif

Définir, maitriser et manipuler les structures de contrôle.

2. Les structures de contrôle:

Il n'y a que deux formes possibles pour un test ;

L'instruction conditionnelle Si :

```
Si expression logique Alors
    Instructions
[Sinon
    Instructions]
FinSi
```

Si l'expression logique (condition) prend la valeur **vrai**, le premier bloc d'instructions est exécuté; si elle prend la valeur **faux**, le second bloc est exécuté (s'il est présent, sinon rien).

Les conditionnelles emboîtées:

```
Si expression1(test1) Alors
    Instructions 1
Sinon
    Si expression2(test2) Alors
            Instructions 2
     Sinon
            Instructions 3
    FinSi
FinSi
```

On peut remplacer la suite de **si** par l'instruction **selon (permet une facilité d'écriture)**

Syntaxe:

```
selon <identificateur>
        (liste de) valeur(s): instructions
        (liste de) valeur(s): instructions
```

…
[autres: instructions]

3. Les structures itératives

La boucle pour:

C'est l'instruction **pour** qui permet de faire des boucles déterministes. Il s'agit de répéter une suite d'instructions un certain nombre de fois.

```
Syntaxe:
   pour <var> de valinit à valfin [par <pas>]  faire
         Instructions à exécuter à chaque boucle
   Finpour
```

La boucle tant que:

Syntaxe:

initialisation des variables de condition

```
Tantque expression logique est vraie(test) faire
   Instructions à exécuter à chaque boucle
   réaffectation des variables de condition
FinTantque
```

Fonction: signifie répéter une suite d'instructions tant que la condition est remplie

La boucle répéter…tantque…

Syntaxe:

```
   répéter
         (ré)affectation des variables de condition
         Instructions à exécuter à chaque boucle
    Tantque  <expression logique est vraie>
```

Fonction: les instructions sont exécutées au moins une fois et répétées tant que la condition est remplie.

TRAVAUX DIRIGES 2

LES STRUCTURES DE CONTROLE

Exercice 1

Ecrire un algorithme qui permet de saisir un nombre puis détermine s'il appartient à un intervalle donné, sachant que les extrémités de l'intervalle sont fixées par l'utilisateur.

Exercice 2

Ecrire un algorithme permettant de lire la valeur de la température de l'eau et d'afficher son état :

GLACE	si la température est inférieure à 0,
LIQUIDE	si la température est strictement supérieure à 0 et inférieure à 100,
VAPEUR	si la température est supérieure à 100.

Exercice 3

Ecrire un algorithme qui demande une somme d'argent comprise entre 1 et 100 et qui affiche ensuite le nombre minimal de billets de 10, 5 et 1 qui la compose.

Exercice 4

Ecrire un algorithme qui permet de calculer an pour un nombre réel a et un entier positif n.

Exercice 5

Ecrire un algorithme qui permet de calculer le factoriel d'un nombre N positif, saisi au clavier.

Exercice 6

Ecrire un algorithme qui détermine si un entier N positif est parfait ou non. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs. Exemple : 6 = 1+2+3.

Exercice 7

Ecrire un algorithme qui permet de saisir un entier strictement positif N et d'afficher s'il est premier ou non. Un nombre est dit premier s'il est divisible uniquement par 1 et par lui-même.

TRAVAUX DIRIGES 2

CORRECTION TD 2

Exercice 1

Ecrire un algorithme qui permet de saisir un nombre puis détermine s'il appartient à un intervalle donné, sachant que les extrémités de l'intervalle sont fixées par l'utilisateur.

```
Algorithme Test d'appartenance à un intervalle
Variable    V,  Bi, Bs: réels

Début
   Répéter
            afficher("Veuillez choisir une borne inférieure d'un intervalle : ")
            saisir(Bi)
            afficher("Veuillez choisir une borne supérieure d'un intervalle : ")
            saisir(Bs)
   Tant que(Bi>Bs)
   afficher("Veuillez choisir une valeur : ")
   saisir(V)
   Si(V>=Bi et V<=Bs) Alors
            afficher("La valeur ", V, "appartient à l'intervalle[", Bi, ",",Bs, "]" )
   Sinon
afficher("La valeur ", V, "n'appartient pas à l'intervalle[", Bi, ",",Bs, "]" )
Finsi

Fin
```

Exercice 2

Ecrire un algorithme permettant de lire la valeur de la température de l'eau et d'afficher son état :

GLACE	si la température est inférieure à 0,
LIQUIDE	si la température est strictement supérieure à 0 et inférieure à 100,
VAPEUR	si la température est supérieure à 100.

```
Algorithme état de l'eau
Variable    T: réel

Début

    afficher("Veuillez donner la température de l'eau  : ")
    saisir(T)
    Si(T<=0) Alors
            afficher("L'état de l'eau est GLACE " )
            SinonSi(T>0 et T<100)Alors
afficher("L'état de l'eau est LIQUIDE " )
Sinon
afficher("L'état de l'eau est VAPEUR " )
Finsi
    Finsi
Fin
```

Exercice 3

Ecrire un algorithme qui demande une somme d'argent comprise entre 1 et 100 et qui affiche ensuite le nombre minimal de billets de 10, 5 et 1 qui la compose.

```
Algorithme Composition de billets d'une somme d'argent
Variable    Sa, B10,B5,B1: entiers

Début

    Répéter
afficher("Veuillez donner la valeur d'une somme d'argent comprise entre 1 et 100
 : ")
            saisir(Sa)
    Tant que(Sa<1 ou Sa>100)
    B10     ←       Sa Div 10
    B5      ←       (Sa Mod 10) Div 5
    B1      ←       Sa Mod 5
    afficher("Le nombre de billets de 10 est : ", B10)
    afficher("Le nombre de billets de 5   est : ", B5)
    afficher("Le nombre de billets de 1   est : ", B1)
    afficher("Le nombre minimal de billets est : ", B10+B5+B1)
Fin
```

Exercice 4

On veut calculer an pour un nombre réel a et un entier positif n.

```
Algorithme a à la puissance n
Variable    n,i: entiers
            a,ai: réels

Début

afficher("Veuillez donner un entier  : ")
saisir(n)
afficher("Veuillez donner un réel  : ")
saisir(a)
ai ← a
   pour (i de 2 à n)  faire
            a ← a * a
   Finpour
   Si( n=0)  faire
            a ← 1
   FinSi
afficher(ai,"à la puissance", n,  "est égale à  : ", a)

Fin
```

Exercice 5

Ecrire un algorithme qui permet de calculer le factoriel d'un nombre N positif, saisi au clavier.

```
Algorithme factoriel
Variable    N,f: entiers
Début
   Répéter
afficher("Veuillez donner un entier positif  : ")
saisir(N)
   Tant que(N<0)
f ← 1
   pour (i de 2 à N)  faire
            f ← f * i
   Finpour
afficher("Le  factoriel du nombre ", N,  "est égale à  : ", f)
Fin
```

Exercice 6

Ecrire un algorithme qui détermine si un entier N positif est parfait ou non. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs. Exemple : 6 = 1+2+3.

```
Algorithme Nombre parfait ?
Variable    N,S: entiers
Début
    Répéter
afficher("Veuillez donner un entier positif  : ")
saisir(N)
    Tant que(N<0)
S  ←        0
    pour (i de 1 à N)  faire
            Si (N Mod i = 0)  faire
                    S        ←        S+ i
            FinSi
    Finpour
afficher("Le  factoriel du nombre ", N,  "est égale à  : ", f)
Fin
```

Exercice 7

Ecrire un algorithme qui permet de saisir un entier strictement positif N et d'afficher s'il est premier ou non. Un nombre est dit premier s'il est divisible uniquement par 1 et par lui-même.

```
Algorithme Nombre premier
Variable   N: entiers
           Premier : boolien

Début
   Répéter
           Afficher ("Veuillez donner un entier positif : ")
           Saisir (N)
   Tant que(N<=0)
      Premier ← vrai
      i  ← 2
   Tant que (i < N et Premier)  faire
           Si (N Mod i )  faire
                   Premier  ← faux
           FinSi
           i  ← i+1
   Fin Tant que
   Si (Premier)  faire
           Afficher ("Le  nombre ", N, "est  premier")
   Sinon
           Afficher ("Le  nombre ", N,  "n'est pas   premier")
   FinSi
Fin
```

CHAPITRE 3

LES TABLEAUX A UNE DIMENSION

1. Objectif

Maitriser la définition et la manipulation des tableaux à une dimension.

Maitriser les algorithmes de tri.

2. Définition

Un tableau est une variable de type complexe, elle peut être définie comme un ensemble de variables (de type simple) de même types. Chaque cellule dans un tableau est considérée comme une variable (de type simple). On peut accéder à cette cellule à travers son indice.

3. Exemple

Soit un tableau Table de type entier et qui est composé de 10 éléments.

Déclaration du tableau :

```
Types :
Tab : Tableau[1..50] d'Entiers
Variables :
Table : Tab

//Pour affecter la valeur x à la cellule de Table d'indice k il suffit de faire
Table[k]←x
//k est l'indice de la kième cellule de Table.
```

4. Algorithmes de tri

Trier un tableau c'est le fait d'organiser l'ensemble de ses éléments selon un ordre déterminé.

Les algorithmes de tri ne sont pas tous identiques. En effet, ils peuvent être différenciés par la complexité algorithmique (fixer une borne supérieure du nombre d'opérations qui seront nécessaires pour trier un ensemble de n éléments), les ressources nécessaires (notamment en termes d'espace mémoire utilisé) et le caractère stable (garder l'ordre relatif des quantités égales).

TRAVAUX DIRIGES 3

LES TABLEAUX A UNE DIMENSION

Exercice 1 : Exemples d'algorithmes de tri

1. Ecrire un programme qui lit la dimension N d'un tableau T du type entier (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

2. Ecrire un algorithme Tri_à_bulles qui permet de parcourir le tableau, et comparer les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, les permuter. Après chaque parcours complet du tableau, recommencer l'opération jusqu'aucune permutation n'a lieu pendant un parcours.

3. Ecrire un algorithme Tri_par_sélection dont le principe est de:

   - rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
   - rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 2 ;
   - continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

4. Ecrire un algorithme Tri_par_insertion dont le principe est de:

   - parcourir le tableau à trier du début à la fin.
   - au moment de considérer le i-ème élément, les éléments qui le précèdent doivent être déjà triés
   - insérer le i-ème élément à sa place parmi ceux qui précèdent. Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion.

5. Dans le chapitre récursivité nous allons traiter d'autres exemples.

6. Calculer et afficher ensuite la somme des éléments du tableau.

7. Ranger les éléments du tableau T dans l'ordre inverse sans utiliser de tableau d'aide. Afficher le tableau résultant.

   (Idée: Echanger les éléments du tableau à l'aide de deux indices qui parcourent le tableau en commençant respectivement au début et à la fin du tableau et qui se rencontrent en son milieu.)

Exercice 2

Ecrire un programme qui lit la dimension N d'un tableau T du type entier (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

Copiez ensuite toutes les composantes strictement positives dans un deuxième tableau TPOS et toutes les valeurs strictement négatives dans un troisième tableau TNEG. Afficher les tableaux TPOS et TNEG.

Exercice 3 : Produit scalaire de deux vecteurs

Ecrire un programme qui calcule le produit scalaire de deux vecteurs d'entiers U et V (de même dimension).

Exemple:

```
 \           /  \           /
 | 3  2  -4 | * | 2  -3  5 | = 3*2+2*(-3)+(-4)*5 = -20
/            \  /           \
```

Exercice 4 : Maximum et minimum des valeurs d'un tableau

Ecrire un programme qui détermine la plus grande et la plus petite valeur dans un tableau d'entiers A. Afficher ensuite la valeur et la position du maximum et du minimum. Si le tableau contient plusieurs maxima ou minima, le programme retiendra la position du premier maximum ou minimum rencontré.

Exercice 5 : Insérer une valeur dans un tableau trié

Un tableau A de dimension N+1 contient N valeurs entières triées par ordre croissant; la

(N+1) ième valeur est indéfinie. Insérer une valeur VAL donnée au clavier dans le tableau A de manière à obtenir un tableau de N+1 valeurs triées.

Exercice 6 : Recherche d'une valeur dans un tableau

Problème: Rechercher dans un tableau d'entiers A une valeur VAL entrée au clavier. Afficher la position de VAL si elle se trouve dans le tableau, sinon afficher un message correspondant.

La valeur POS qui est utilisée pour mémoriser la position de la valeur dans le tableau, aura la valeur -1 aussi longtemps que VAL n'a pas été trouvée.

Implémenter deux versions:

a) La recherche séquentielle

Comparer successivement les valeurs du tableau avec la valeur donnée.

b) La recherche dichotomique ('recherche binaire', 'binary search')

Condition: Le tableau A doit être trié

Comparer le nombre recherché à la valeur au milieu du tableau,

- s'il y a égalité ou si le tableau est épuisé, arrêter le traitement avec un message correspondant.

- si la valeur recherchée précède la valeur actuelle du tableau, continuer la recherche dans le demi-tableau à gauche de la position actuelle.

- si la valeur recherchée suit la valeur actuelle du tableau, continuer la recherche dans le demi-tableau à droite de la position actuelle.

Ecrire le programme pour le cas où le tableau A est trié par ordre croissant.

Question: Quel est l'avantage de la recherche dichotomique? Expliquer brièvement.

Exercice 7 : Fusion de deux tableaux triés

Problème: On dispose de deux tableaux A et B (de dimensions respectives N et M), triés par ordre croissant. Fusionner les éléments de A et B dans un troisième tableau FUS trié par ordre croissant.

Méthode: Utiliser trois indices IA, IB et IFUS. Comparer A(IA) et B(IB); remplacer

FUS(IFUS) par le plus petit des deux éléments; avancer dans le tableau FUS et dans le tableau qui a contribué son élément. Lorsque l'un des deux tableaux A ou B est épuisé, il suffit de recopier les éléments restants de l'autre tableau dans le tableau FUS.

Exercice 8 : Tri par sélection du maximum

Problème: Classer les éléments d'un tableau A par ordre décroissant.

Méthode: Parcourir le tableau de gauche à droite à l'aide de l'indice I. Pour chaque élément A(I) du tableau, déterminer la position PMAX du (premier) maximum à droite de A(I) et échanger A(I) et A(PMAX).

Exercice 9 : Tri par propagation (bubble sort)

Problème: Classer les éléments d'un tableau A par ordre croissant.

Méthode: En recommençant chaque fois au début du tableau, on effectue à plusieurs reprises le traitement suivant: On propage, par permutations successives, le plus grand élément du tableau vers la fin du tableau (comme une bulle qui remonte à la surface d'un liquide).

CHAPITRE 4

LES TABLEAUX A DEUX DIMENSIONS

1. Objectif

Maitriser la définition et la manipulation des tableaux à deux dimensions.

2. Définition

Une matrice est un tableau à deux dimensions. Chaque ligne d'une matrice est un tableau à une dimension.

3. Exemple

Soit une matrice MaMatrice de type entier et qui est composé de 10 lignes et 10 colonnes.

Déclaration du tableau :

Types :

Tab : MaMatrice[1..50 ; 1..50] d'Entiers

Variables :

Table : Tab

//Pour affecter la valeur x à la cellule de la ligne l et la colonne c de la matrice

// il suffit de faire

Matrice[l][c] ←*x*

//l est l'indice de la lième ligne de MaMatrice.

////c est l'indice de la cième colonne de MaMatrice.

TRAVAUX DIRIGES 4

LES TABLEAUX A DEUX DIMENSIONS

Exercice 1. (Mise à zéro de la diagonale principale d'une matrice)

Ecrire un programme qui met à zéro les éléments de la diagonale principale d'une matrice carrée A donnée par l'utilisateur.

Exercice 2. (Matrice unitaire)

Ecrire un programme qui construit et affiche une matrice carrée unitaire U de dimension N.

Une matrice unitaire est une matrice, telle que:

/ 1 si i=j

uij =

\ 0 si ij

Exercice 3. Transposition d'une matrice

Ecrire un Algorithme qui effectue la transposition tA d'une matrice A de dimensions N et M en une matrice de dimensions M et N.

a) La matrice transposée sera mémorisée dans une deuxième matrice B qui sera ensuite affichée.

b) La matrice A sera transposée par permutation des éléments.

Rappel:

```
tA = t |a b c d  | =  | a  e  i |
       |e f  g  h |    | b  f  j |
       |i  j  k  l |   | c  g k |
                       | d  h l |
```

Exercice 4. Multiplication d'une matrice par un réel

Ecrire un algorithme qui réalise la multiplication d'une matrice A par un réel X.

Rappel:

```
        | a   b   c   d |      | X*a   X*b   X*c   X*d |
  X *   | e   f   g   h | =    | X*e   X*f   X*g   X*h |
        | i   j   k   l |      | X*i   X*j   X*k   X*l |
```

a) Le résultat de la multiplication sera mémorisé dans une deuxième matrice A qui sera ensuite affichée.

b) Les éléments de la matrice A seront multipliés par X.

Exercice 5. Addition de deux matrices

Ecrire un algorithme qui réalise l'addition de deux matrices A et B de mêmes dimensions N et M.

Rappel:

```
| a b c  d  |     | a' b' c' d' |      | a+a'   b+b'   c+c'   d+d' |
| e f  g  h | +   | e' f' g' h' |  =   | e+e'   f+f'    g+g'   h+h' |
| i j  k  l |     | i' j' k' l' |      | i+i'    j+j'    k+k'    l+l' |
```

a) Le résultat de l'addition sera mémorisé dans une troisième matrice C qui sera ensuite affichée.

b) La matrice B est ajoutée à A.

Exercice 6. Multiplication de deux matrices

En multipliant une matrice A de dimensions N et M avec une matrice B de dimensions M et P on obtient une matrice C de dimensions N et P:

```
A(N,M) * B(M,P) = C(N,P)
```

La multiplication de deux matrices se fait en multipliant les composantes des deux matrices lignes par colonnes:

$$c_{ij} = \sum_{k=1}^{k=M} (a_{ik} * b_{kj})$$

Rappel:

$$\begin{pmatrix} a & b & c \\ e & f & g \\ h & i & j \\ k & l & m \end{pmatrix} * \begin{pmatrix} p & q \\ r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a*p + b*r + c*t & a*q + b*s + c*u \\ e*p + f*r + g*t & e*q + f*s + g*u \\ h*p + i*r + j*t & h*q + i*s + j*u \\ k*p + l*r + m*t & k*q + l*s + m*u \end{pmatrix}$$

Ecrire un algorithme qui effectue la multiplication de deux matrices A et B. Le résultat de la multiplication sera mémorisé dans une troisième matrice C qui sera ensuite affichée.

Exercice 7. Triangle de Pascal

Ecrire un algorithme qui construit le triangle de PASCAL de degré N et le mémorise dans une matrice carrée P de dimension N+1.

Exemple: *Triangle de Pascal de degré 6:*

```
n=1 1
n=2 1 1
n=3 1 2 1
n=4 1 3 3 1
n=5 1 4 6 4 1
n=6 1 5 10 10 5 1
n=7 1 6 15 20 15 5 1
```

Méthode:

Calculer et afficher seulement les valeurs jusqu'à la diagonale principale (incluse). Limiter le degré à entrer par l'utilisateur à 13.

Construire le triangle ligne par ligne:

- Initialiser le premier élément et l'élément de la diagonale à 1.

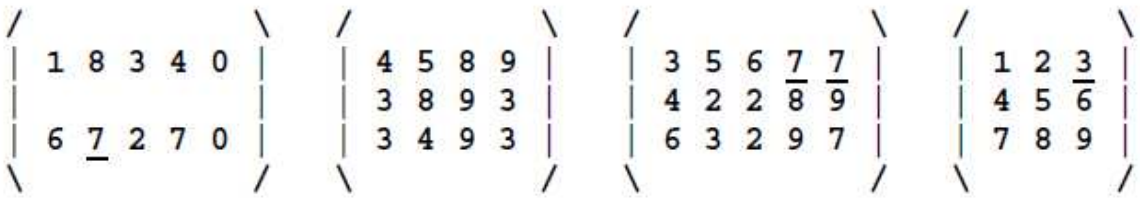
- Calculer les valeurs entre les éléments initialisés de gauche à droite en utilisant la relation:

$P_{i,j} = P_{i-1,j} + P_{i-1,j-1}$

Exercice 8. Recherche de 'points-cols'

Rechercher dans une matrice donnée A les éléments qui sont à la fois un maximum sur leur ligne et un minimum sur leur colonne. Ces éléments sont appelés des *points-cols*. Afficher les positions et les valeurs de *tous* les points-cols trouvés.

Exemples: Les éléments soulignés sont des points-cols:



Méthode: Etablir deux matrices d'aide MAX et MIN de même dimensions que A, telles que:

```
          / 1 si A[i,j] est un maximum
MAX[i,j] =| sur la ligne i
          \ 0 sinon

          / 1 si A[i,j] est un minimum
MIN[i,j] =| sur la colonne j
          \ 0 sinon
```

TRAVAUX DIRIGES 5

LES FONCTIONS

Objectif

Maitriser la définition et la manipulation des tableaux à deux dimensions.

Exercice 1

1. Ecrire un programme se servant d'une fonction MOYENNE pour afficher la moyenne arithmétique de deux nombres réels entrés au clavier.

2. Ecrire une fonction MIN et une fonction MAX qui déterminent le minimum et le maximum de deux nombres réels.

Ecrire un programme se servant des fonctions MIN et MAX pour déterminer le minimum et le maximum de quatre nombres réels entrés au clavier.

3. Ecrire la fonction NCHIFFRES qui obtient une valeur entière N (positive ou négative) comme paramètre et qui fournit le nombre de chiffres de N comme résultat.

Ecrire un petit programme qui teste la fonction NCHIFFRES:

Exemple:

Introduire un nombre entier : 6457392

Le nombre 6457392 a 7 chiffres.

4. En mathématiques, on définit la fonction factorielle de la manière suivante:

0! = 1

n! = n*(n-1)*(n-2)* ... * 1 (pour n>0)

Ecrire une fonction FACT qui reçoit la valeur de l'entier N comme paramètre et qui fournit la factorielle de N comme résultat. Ecrire un petit programme qui teste la fonction FACT.

Exercice 2

N est un nombre entier positif donné par l'utilisateur du programme.

Soit les trois fonctions suivantes:

SOMME_1 = 1+2+…+N ;

SOMME_2=12+22+…+N2 ;

PRODUIT_1 = 1*2*3*….*N

Ecrire un programme qui calcule:

SOMME= SOMME_1 + SOMME_2 + PRODUIT_1

MOYENNE = (SOMME_1 + SOMME_2 + PRODUIT_1)/3

Et

PRODUIT = SOMME_1 * SOMME_2 * PRODUIT_1

Exercice 3

Ecrire un programme de Test de code de la route.

Voici le test à programmer :

Répondez par O pour Oui et par N pour Non (l'utilisateur doit obligatoirement répondre par O ou par N)

1. Peut-t-on passer au feu rouge ?
2. Doit-t-on marquer un arrêt au stop ?
3. Doit-t-on ralentir au carrefour ?
4. Peut-t-on rouler à 80 Km/H au centre ville ?
5. Le feu est vert et un piéton est au milieu de la route, on doit s'arrêter et l'attendre ?

TRAVAUX DIRIGES 6

LES ENREGISTREMENTS

Exercice1 :

Quel est l'apport des tableaux par rapport aux variables?

Un tableau est une extension d'une variable. En effet, il est composé de plusieurs variables de même type appelées les éléments du tableau.

Quel est l'apport des enregistrements par rapport aux variables et aux tableaux que nous avons utilisé jusqu'à présent ?

Un enregistrement est une extension d'un tableau. En effet, il est composé de plusieurs variables de types différents appelées les éléments de l'enregistrement. Un enregistrement est donc un type composé.

Exemple :

//Pour définir le type nombre complexe nous pouvons faire recours aux enregistrements :

```
Types
   Complexe =    Struct
                       reel     : Réel
                       imag     : Réel
                 FinStruct
```

*(*Pour manipuler des variables de type enregistrement il est nécessaire de préciser le nom de l'enregistrement, suivi d'un point puis de l'indicateur du champ concerné.*

Pour créer un nombre complexe C_0= 2.5+3i)*

```
Variables
   C0 : Complexe
Début
   C0.reel ←      2.5
   C0.imag      ←      3
Fin
```

Exercice 2 :

Ecrire un algorithme qui lit deux nombres complexes C_1 et C_2 et qui affiche ensuite leur somme et leur produit.

Sachant que :

(a+bi)+(c+di)=(a+c)+(b+d)i

(a+bi)*(c+di)=(ac-bd)+(ad+bc)i

```
Algorithme CalculComplexe
Types
   Complexe =    Struct
                     reel    : Réel
                     imag    : Réel
                 FinStruct

Variables
   S, P, C1, C2 : Complexe
Début
   Ecrire(«Veillez donner la partie réelle du 1er nombre complexe : » )
   Lire(C1.reel)
   Ecrire(«Veillez donner la partie imaginaire du 1er nombre complexe : » )
   Lire(C1.imag)
   Ecrire(«Veillez donner la partie réelle du 2eme  nombre complexe : » )
   Lire(C12.reel)
   Ecrire(«Veillez donner la partie imaginaire du 2eme nombre complexe : » )
   Lire(C2.imag)
   S.reel ← C1.reel + C2.reel
   S.imag ← C1.imag + C2.imag
   Ecrire(«Somme = », S.reel, « + », S.imag, «i »)
   P.reel ← (C1.reel * C2.reel)-( C1.imag * C2.imag)
   P.reel ← (C1.reel * C2.imag)+( C1.imag * C2.reel)
   Ecrire(«Produit = », P.reel, « + », P.imag, «i »)
Fin
```

CHAPITRE 7

LES FICHIERS SEQUENTIELS

1. Objectif

Comprendre les concepts de base relatifs aux fichiers.

Manipuler des fichiers à organisation séquentielle.

2. Questions

Quel est l'apport des fichiers pour un programmeur ?

Les algorithmes sur les fichiers permettent la sauvegarde permanente des données, comme ils permettent aussi la lecture des données stockées dans des fichiers de données. Ceci est très utile pour le programmeur. Imaginez qu'un programme nécessite quelques jours pour offrir un résultat final et qu'il a besoin de lire des nouvelles données chaque 2H.

Les fichiers permettent de prévoir la lecture automatique de ces données en absence du programmeur. D'où un gain de temps et d'effort.

Si on prend aussi l'exemple d'un jeu, la sauvegarde du jeu se fait automatiquement et en temps réel. Cette sauvegarde se fait par les fichiers.

3. Exemples

*(*Déclaration d'une variable de type fichier*)*

```
Types
   Etudiant =      Struct
                        CIN          : Entier
                        Nom          : Chaine[30]
                        Prénom       : Chaine[30]
                        Classe       : Chaine[5]
                   FinStruct
   Fetud = Fichier
```

*(*Procédure permettant de créer et remplir un fichier fe d'étudiants*)*

```
Procédure Création(Var fe :Fetud)
Variables
   et :Etudiant (*variable tampon*)
Début
   Ouvrir(fe,E)
   Ecrire(« CIN de l'étudiant : »)
   Lire(et.CIN)
   TantQue(et.CIN≠0)Faire
         Ecrire(« Donner le nom de l'étudiant : »)
         Lire(et.Nom)
         Ecrire(« Donner le prénom de l'étudiant : »)
         Lire(et.Prénom)
         Ecrire(« Donner la classe de l'étudiant : »)
         Lire(et.Classe)
         Ecrire(fe,et)
         Ecrire(« CIN de l'étudiant : »)
         Lire(et.CIN)
   FinTQ
   Fermer(fe)
Fin
```

*(*Procédure qui assure le parcours d'un fichier à organisation séquentielle*)*

```
Procédure Consultation(Var fe :Fetud)
Variables
   et :Etudiant (*variable tampon*)
Début
   Ouvrir(fe,L)
   Lire(fe,et)
   TantQue(NON(FDF(fe)))Faire
         Ecrire (et.CIN,et.Nom,et.Prénom,et.Classe)
         Lire(fe,et)
   FinTQ
   Fermer(fe)
Fin
```

*(*Procédure qui assure le parcours d'un fichier de type texte à organisation séquentielle*)*

```
Procédure ParcoursFichText(Var ftext :Fetud)
Variables
   ligne :Chaine (*variable tampon*)
Début
   Ouvrir(ftext,L)
   Lire_Lig(ftext,ligne)
   TantQue(NON(FDF(ftext)))Faire
         Ecrire (ligne)
         Lire(ftext,ligne)
   FinTQ
   Fermer(ftext)
Fin
```

TRAVAUX DIRIGES 7

LES FICHIERS SEQUENTIELS

Exercice 1

1. Ecrire une procédure permettant de créer et remplir le fichier (de type Fi_employés) qui contient des informations sur les employés d'une entreprise (matricule, nom, prénom, grade, salaire).

*(*Déclaration d'une variable de type fichier*)*

```
Types
   Employé =      Struct
                        matricule          : Entier
                        nom          : Chaine[30]
                        prénom       : Chaine[30]
                        grade        : Chaine[10]
                        salaire      : Entier
                  FinStruct
   Fi_employés = Fichier
```

*(*Procédure permettant de créer et remplir un fichier de type* Fi_employés**)*

```
Procédure Création(Var fe : Fi_employés)
Variables
   em : Employé (*variable tampon*)
Début
   Ouvrir(fe,E)
   Ecrire(« matricule de l'étudiant : »)
   Lire(em.matricule)
   TantQue(em.matricule≠0)Faire
         Ecrire(« Donner le nom de l'employé : »)
         Lire(em.nom)
         Ecrire(« Donner le prénom de l'employé : »)
         Lire(em.prénom)
         Ecrire(« Donner le grade de l'employé : »)
         Lire(em.grade)
         Ecrire(« Donner le salaire de l'employé : »)
         Lire(em.salaire)
         Ecrire(fe,em)
         Ecrire(« matricule de l'employé : »)
         Lire(em.matricule)
   FinTQ
   Fermer(fe)
Fin
```

2. Ecrire une procédure permettant d'afficher la liste des employés à partir du fichier de type Fi_employés.

*(*Procédure qui assure le parcours d'un fichier à organisation séquentielle*)*

```
Procédure Consultation(Var fe : Fi_employés)
Variables
   em : Employé
Début
   Ouvrir(fe,L)
   Lire(fe,em)
   TantQue(NON(FDF(fe)))Faire
            Ecrire (em.matricule,em.nom,em.prénom,em.grade,em.salaire)
            Lire(fe,em)
   FinTQ
   Fermer(fe)
Fin
```

3. Ecrire une fonction permettant d'afficher la liste des employés dont le salaire est compris entre 500 et 700 DT.

*(*Procédure qui assure le parcours d'un fichier à organisation séquentielle*)*

```
Procédure Consultationbis(Var fe : Fi_employés)
Variables
   em : Employé
Début
   Ouvrir(fe,L)
   Lire(fe,em)
   TantQue(NON(FDF(fe)))Faire
            Si(em.salaire>=500 ET em.salaire<=700)alors
                     Ecrire (em.matricule,em.nom,em.prénom,em.grade,em.salaire)
            FinSi
            Lire(fe,em)
   FinTQ
   Fermer(fe)
Fin
```

4. Ecrire une procédure permettant de rechercher un employé dans le fichier Fi_employés à partir de son matricule. Si l'employé est trouvé, l'algorithme doit afficher son nom, son prénom et son grade, sinon il doit afficher ce message « Ce matricule ne figure pas dans le fichier »

```
Procédure Recherchemat(Var fe : Fi_employés, MER : Entier)
Variables
   em : Employé
Début
   Ouvrir(fe,L)
   Lire(fe,em)
   Si(em.matricule=MER)alors
            Ecrire (em.nom,em.prénom,em.grade)
   FinSi
   TantQue(NON(FDF(fe)) ET em.matricule ≠ MER  )Faire
```

```
            Lire(fe,em)
            Si(em.matricule=MER)alors
                  Ecrire (em.nom,em.prénom,em.grade)
            FinSi
      FinTQ
      Fermer(fe)
Fin
```

5. Copier les salaires des employés dans un tableau T_Salaire

```
Type
Tab :Tableau[1..50] :Entier

Procédure CopieSalaires(Var fe : Fi_employés, T_Salaire :Tab)
Variables
   em : Employé
   i   : Entier

Début
   i ← 1
   Ouvrir(fe,L)
   Lire(fe,em)
   TantQue(NON(FDF(fe)))Faire
            T_Salaire[i] ← em.salaire
            i ← i+1
            Lire(fe,em)
   FinTQ
   Fermer(fe)
Fin
```

6. Trier le tableau T_Salaire dans l'ordre décroissant.

```
Procédure Trie(Var T_Salaire :Tab)
Variables
i,x : Entier
échange : Booléen
Début
Répéter
Echange ← Faux
Pour i de 1 à 50 Faire
Si(T_Salaire[i]< T_Salaire[i+1])alors
                              x ← T_Salaire[i]
                              T_Salaire[i] ← T_Salaire[i+1]
                              T_Salaire[i+1] ← x
                              échange ← Vrai
FinSi
            FinPour
Jusqu'à(échange = Faux)
Fin
```

7. Copier les Salaires triés dans un nouveau fichier Fi_Salaires

```
Types
   Fi_Salaire = Fichier d'entiers

Procédure CréationFiSalaire(Var fs : Fi_Salaire, T_Salaire :Tab)
Variables
   i                : Entier
Début
   Ouvrir(fs,E)
   Pour i de 1 à 50Faire
         Ecrire(fs,T_Salaire[i])
   FinPour
   Fermer(fs)
Fin
```

8. Ecrire une fonction permettant d'afficher les Salaires triés des employés à partir de Fi_Salaires.

```
Procédure Consultationbis(Var fs : Fi_Salaire)
Variables
   Salaire : Entier
Début
   Ouvrir(fs,L)
   Lire(fs,Salaire)
   TantQue(NON(FDF(fs)))Faire
         Ecrire (Salaire)
         Lire(fs,Salaire)
   FinTQ
   Fermer(fs)
Fin
```

Exercice 2

Ecrire un algorithme permettant de :

1. Créer et remplir un fichier « Fnote » qui contient les notes de 30 étudiants (CIN, nom, prénom, note).
2. Copier les notes dans un tableau Tnote.
3. Trier le tableau Tnote dans l'ordre croissant.
4. Copier les notes triées du tableau vers le fichier fnotes.

Exercice 3

1. Quelles sont les spécificités d'un fichier de type Texte.

2. Ecrire une fonction permettant de lire et d'afficher le contenu d'un fichier de type texte.

Exercice 4

Créer sur disquette puis afficher à l'écran le fichier INFORM.TXT dont les informations sont structurées de la manière suivante:

Numéro de matricule (entier)

Nom (chaîne de caractères)

Prénom (chaîne de caractères)

Le nombre d'enregistrements à créer est à entrer au clavier par l'utilisateur.

Exercice 5

Ecrire un programme qui crée sur disquette un fichier INFBIS.TXT qui est la copie exacte (enregistrement par enregistrement) du fichier INFORM.TXT.

Exercice 6

Ajouter un nouvel enregistrement (entré au clavier) à la fin de INFORM.TXT et sauver le nouveau fichier sous le nom INFBIS.TXT.

Exercice 7

Insérer un nouvel enregistrement dans INFORM.TXT en supposant que le fichier est trié relativement à la rubrique NOM et sauver le nouveau fichier sous le nom INFBIS.TXT.

Exercice 8

Supprimer dans INFORM.TXT tous les enregistrements:

a) dont le numéro de matricule se termine par 8

b) dont le prénom est "Paul"

c) dont le nom est un palindrome. Définir une fonction d'aide PALI qui fournit le résultat 1 si la chaîne transmise comme paramètre est un palindrome, sinon la valeur zéro.

Sauver le nouveau fichier à chaque fois sous le nom INFBIS.TXT.

Exercice 9

Créer sur disquette puis afficher à l'écran le fichier FAMILLE.TXT dont les informations sont structurées de la manière suivante:

Nom de famille

Prénom du père

Prénom de la mère

Nombre d'enfants

Prénoms des enfants

Le nombre d'enregistrements à créer est entré au clavier.

Attention: Le nombre de rubriques des enregistrements varie avec le nombre d'enfants!

TRAVAUX DIRIGES 8

LA RECURSIVITE

Objectifs

- Résoudre les problèmes récursifs
- Comparer les approches de programmation itérative et récursive.

Définition : Une procédure (ou une fonction) récursive est une procédure (ou une fonction) dont le corps contient un ou plusieurs appels à elle-même.

Exemple :

1. Ecrire un algorithme **Tri_par_fusion** dont le principe est décrit récursivement comme suit:
   - Découper en deux parties à peu près égales les données à trier
   - Trier les données de chaque partie
   - Fusionner les deux parties

2. Ecrire un algorithme **Tri_rapide** dont le principe est récursivement décrit comme suit:

   Placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, définir un nouveau pivot et répèter l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Exercice 1

Transformer la procédure suivante en une fonction (ou une procédure) récursive:

```
Procédure Factoriel (N : Entier, P : Réel)
Variables :
  i : Entier
Début
```

P ← 1

```
   Pour i de 1 à N faire
     P ← P * i
   Fin Pour
 Fin
```

```
Fonction Factoriel (n:Entier)
Début
Si (n<=1) alors
      Factoriel ← 1
   Sinon
      Factoriel ← n * Factoriel (n-1)
   Finsi
Fin
```

```
Procédure Factoriel (i:Entier,n:Entier,  p: Entier)
Début
   Si (i<=n) alors
     p ← p * i;
     Factoriel (i+1,n,p)
   FinSi
Fin
```

Exercice 2

Ecrire une procédure itérative et une procédure récursif qui affiche tout le contenu d'un tableau T de taille n.

```
Pour i de 1 à n faire
  Ecrire(T[i])
FinPour
```

```
Procédure Afficher (i: Entier; T:Tab; n:Entier)
Début
   Si (i <= n) alors
      ECRIRE(T[i])
      Afficher(i+1, T, n)
   FinSi
Fin
```

```
Procédure Afficher (T:Tab; n:Entier)
Début
   Si (n>=1) alors
      Ecrire (T[n])
      Afficher(T, n-1)
   FinSi
Fin
```

Exercice 3

Transformer la procédure ci-dessous en sous-programme récursif :

```
Procédure F( T: Tab, N: Entier)
Variables :
  i, m : Entier
Début
  m ← 1
  POUR i de 2 à n FAIRE
    SI (T[m] < T[i]) ALORS
      m ← i
    FINSI
  FINPOUR
  F ← m
Fin
```

```
Procédure Recherche (i: Entier; T:Tab; n:Entier;  m: Entier)
Début
   Si (i <= n) alors
      Si (T[m]<T[i]) alors
         m <-- i
      FinSi
      Recherche (i+1, T, n, m)
   FinSi
Fin
```

Exercice 4

Calcul du PGCD par la méthode d'Euclide :

Ecrire une fonction récursive PGCD_Euc qui retourne le PGCD de 2 entiers a et b en utilisant l'algorithme d'Euclide qui s'appuie sur les propriétés suivantes :

PGCD(a,b)=b si b est un diviseur de a

PGCD(a,b)=PGCD(b, a mod b) sinon

Exemple PGCD(36,20) = PGCD(20,16)=PGCD(16,4)=4.

```
Fonction PGCD_Eu (a:Entier, b:Entier) :Entier
Début
   Si (a mod b = 0) alors
      PGCD_Eu ← b
   Sinon
      PGCD_Eu ← PGCD_Eu (b, a mod b)
   finsi
Fin
```

Exercice 5

Calcul du nombre de combinaisons de p éléments parmi n.

Ecrire une fonction récursive CNP qui calcule le nombre de combinaison de p éléments parmi n éléments différents par la formule suivante

$C_n^{} = {}^{}C_n^{} = 1$

$C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$

```
Fonction CNP (n:Entier, p:Entier) :Entier
Début
   Si (p = 1) alors
      CNP ← n
   sinonsi(p=n)alors
      CNP ← 1
   Sinon
     CNP ← CNP(n-1,p)+ CNP(n-1,p-1)
   finsi
Fin
```

Exercice 6

Ecrire une fonction récursive Palind qui vérifie si une chaine de caractères est un palindrome ou non.

Formellement une chaine S de n caractères est un palindrome si et seulement si :

Pour tout i, $1 \leq i \leq n \text{ div } 2$, S[i]= S[n-i+1]

Une condition nécessaire est que les caractères extrêmes soient identiques et que la sous chaine privée des caractères extrêmes soit également un palindrome.

```
Fonction Palindrome (s:Chaine) :Booléen
Début
   Si (Long(s)<2) alors
      Palindrome ← Vrai
   sinon
      Palindrome ← (s[1]=s[Long(s)] ET Palindrome(copie(s,2,Long(s)-2)))
   finsi
Fin
```

Exercice 7

Fonction 91 de MacCarty

Ecrire une fonction récursive f qui calcule f(n) selon la formule suivante :

- f(n) = n-10 Si n > 100
- f(n) = f(f(n+11)) Sinon

```
Fonction F91 (n:Entier) :Entier
Début
   Si (n>100) alors
      F91 ← n-10
   Sinon
      F91 ← F91(F91(n+11))
   finsi
Fin
```

Exercice 8

Transformer en sous-programme récursif :

Module de calcul de l'occurrence (nombre d'apparitions) d'un entier X dans un tableau T de taille n.

Version itérative

```
Fonction Occurrence(T:Tab, n:Entier, X:Entier): Entier
  Variables :
   i : Entier
Début
   p ← 0
   Pour i de 1 à n Faire
      Si (T[i] = X) Alors
         p ← p + 1
      FINSI
   FinPour
   Occurrence ← p
Fin
```

```
Procédure  Occurrence (i: ENTIER, T:TAB, n:ENTIER, X:ENTIER, VAR p: ENTIER)
Début
   Si ( i <= n) alors
      Si (T[i] = X) alors
         p <-- p + 1
      FinSi
      Occurrence(i+1, T, n, X, p)
  FinSi
Fin
```

```
Fonction Occurrence (i: ENTIER, T:TAB, n:ENTIER, X:ENTIER):Entier
```

```
Début
   Si (i<=n) alors
      Si (T[i]=X) alors
         Occurrence <-- 1 + Occurrence(i+1,T,n,X)
      Sinon
         Occurrence <-- Occurrence(i+1,T,n,X)
      FinSi
   FinSi
Fin
```

CHAPITRE 9

LES POINTEURS

1. Objectifs

Accéder aux données dans la mémoire de l'ordinateur à l'aide de *pointeurs*, c.-à-d. à l'aide de variables auxquelles on peut attribuer les *adresses d'autres variables*.

Les pointeurs nous permettent d'écrire des programmes plus compacts et plus efficients et fournissent souvent la seule solution raisonnable à un problème.

2. Définition: Pointeur

Un **pointeur** est une variable spéciale qui peut contenir l'**adresse** d'une autre variable.

Si un pointeur P contient l'adresse d'une variable A, on dit que **'P pointe sur A'**.

Remarques

Les pointeurs et les noms de variables ont le même rôle: Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence:

* Un ***pointeur*** est une variable qui peut 'pointer' sur différentes adresses.

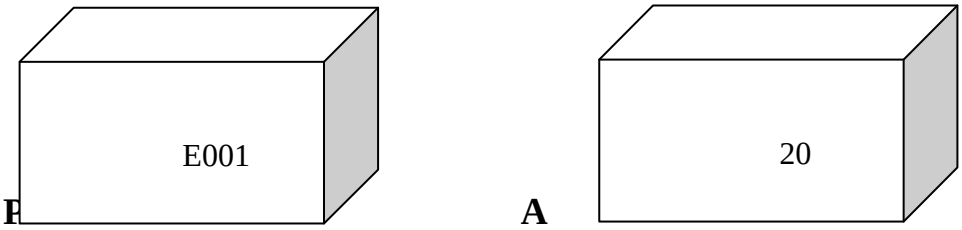
* Le ***nom d'une variable*** reste toujours lié à la même adresse.

3. Déclaration d'un pointeur :

```
Types :
  Pointeur : ^Entier
Variables
  A : Entier
  P : Pointeur
```

*/*le pointeur **P** prend l'adresse d'une autre variable A (**P pointe sur A**)*/*

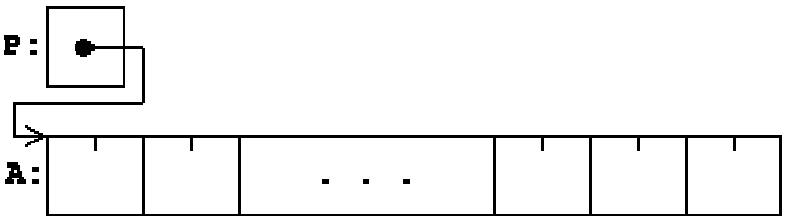
P ← Adresse(A)



Adresse : E001

Le nom d'un tableau représente l'adresse de son premier élément.

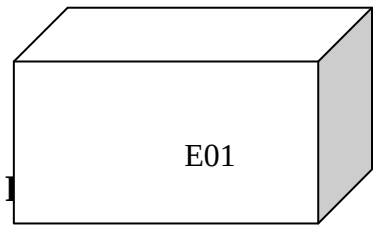
```
Types :
  Tab : Tableau[1..50] de Entiers
  Pointeur : ^Entier
Variables
  A : Tab
  P : Pointeur
```



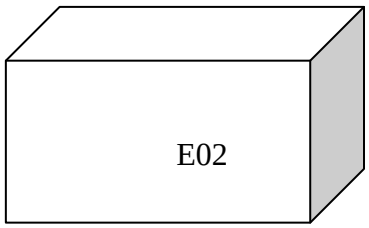
Adresses : E01 E02 … En-2 En-1 En

*/*le ponteur **P** prend l'adresse de la première cellule du tableau A (**P pointe sur le premier élément de A**)*/*

P ← A



A



P←P+1

P^← 12

/*Le contenu de l'élément de A dont l'adresse est E02 (Le contenu de P) prend la valeur 12.*/

A

TRAVAUX DIRIGES 9

LES POINTEURS

Exercice 1

Ecrire un programme qui lit deux tableaux d'entiers A et B et leurs dimensions N et M au clavier et qui ajoute les éléments de B à la fin de A. Utiliser le formalisme pointeur.

```
Algorithme Ajoutdetab
 Variables
  A : Tableau[1..100] de Entiers
  B : Tableau[1..50] de Entiers
  N, M : Entiers  /* dimensions des tableaux */
  I : Entier     /* indice courant */
 Début
/* Saisie des données */
Ecrire("Dimension du tableau A (max.50) : ")
Lire(N)
Pour I de 1 à N Faire
    Ecrire("Elément", I, " : ")
    Lire(A[I])
Fin pour
Ecrire("Dimension du tableau B (max.50) : ");
Lire(M)
Pour I de 1 à M Faire
    Ecrire("Elément", I, " : ")
    Lire(B[I])
Fin pour
/* Affichage des tableaux */
Ecrire("Tableau donné A :")
Pour I de 1 à N Faire
    Ecrire((A+I)^)
Fin pour
Ecrire("Tableau donné B :")
Pour I de 1 à M Faire
    Ecrire((B+I)^)
Fin pour
/* Copie de B à la fin de A */
Pour I de 1 à M Faire
    (A+N+I)^ ← (B+I)^
Fin pour
/* Nouvelle dimension de A */
N ← N+M;
 /* Edition du résultat */
Ecrire("Tableau résultat A :")
```

```
Pour I de 1 à N Faire
    Ecrire((A+I)^)
Fin pour
Fin
```

Exercice 2

Ecrire un programme qui lit un entier X et un tableau A du type **Entier** au clavier et élimine toutes les occurrences de X dans A en tassant les éléments restants. Le programme utilisera les pointeurs P1 et P2 pour parcourir le tableau.

```
Algorithme EliminerXdeA
Types :
  Tab : Tableau[1..100] de Entiers
  Pointeur : ^Entier
Variables
  A : tab
  N : Entier  /* dimension du tableau */
  X : Entier
  P1,P2 : Pointeurs
Début
 /* Saisie des données */
 Ecrire("Dimension du tableau A (max.100) : ")
 Lire(N)
 Pour P1 de A à A+N Faire
     Ecrire("Elément", P1-A, " : ")
     Lire(P1^)
 Fin pour
 Ecrire("Valeur à supprimer du tableau ")
 Lire(X)
 /* Affichage des tableaux */
 Ecrire("Tableau donné A :")
 Pour P1 de A à A+N Faire
     Ecrire((P1)^)
 Fin pour
 /* Effacer toutes les occurrences de X et comprimer :  */
 /* Copier tous les éléments de P1 vers P2 et augmenter */
 /* P2 pour tous les éléments différents de X.          */
P2 ← A
Pour P1 de A à A+N Faire
     P2^ = P1^;
     si (P2^ ≠ X)alors
         P2←P2+1
     Fin si
 Fin pour
/* Nouvelle dimension de A */
 N ← P2-A;
  /* Edition du résultat */
 Pour P1 de A à A+N Faire
    Ecrire((P1)^);
 Fin pour
```

```
Fin
```

Exercice 3

Ecrire un programme qui range les éléments d'un tableau A du type **Entier** dans l'ordre inverse. Le programme utilisera des pointeurs P1 et P2 et une variable numérique AIDE pour la permutation des éléments.

```
Algorithme RangeInverse
Types :
  Tab : Tableau[1..100] de Entiers
  Pointeur : ^Entier
Variables
  A : tab
  N : Entier  /* dimension du tableau */
  AIDE : Entier
  P1,P2 : Pointeurs
Début
 /* Saisie des données */
 Ecrire("Dimension du tableau A (max.100) : ")
 Lire(N)
 Pour P1 de A à A+N Faire
     Ecrire("Elément", P1-A, " : ")
     Lire(P1^)
 Fin pour
 /* Affichage du tableau */
 Ecrire("Tableau donné A :")
 Pour P1 de A à A+N Faire
     Ecrire((P1)^)
 Fin pour
 /* Inverser la tableau */
 P1←A
 P2← A+(N-1)
 Tant que P1<P2 Faire
     AIDE   ← (P1)^
     (P1)^  ← (P2)^
     (P2)^  ← AIDE
    P1    ← P1+1
    P2    ← P2-1
 Fin Tq
  /* Edition du résultat */
 Pour P1 de A à A+N Faire
    Ecrire((P1)^);
 Fin pour
Fin
```

Exercice 4

Ecrire un programme qui lit deux tableaux d'entiers A et B et leurs dimensions N et M au clavier et qui ajoute les éléments de B à la fin de A. Utiliser deux pointeurs PA et PB pour le transfert et afficher le tableau résultant A.

Exercice 5

Ecrire un programme qui lit une chaîne de caractères CH et détermine la longueur de la chaîne à l'aide d'un pointeur P. Le programme n'utilisera pas de variables numériques.

```
/* Placer P à la fin de la chaîne */
 P←CH
 Tant que P^ Faire
    P←P+1
 Fin Tq
  /* Affichage du résultat */
 Ecrire("La chaîne \", CH, "\ est formée de", P-CH, "caractères.")
```

Exercice 6

Ecrire un programme qui lit une chaîne de caractères CH au clavier et qui compte les occurrences des lettres de l'alphabet en ne distinguant pas les majuscules et les minuscules.

Utiliser un tableau ABC de dimension 26 pour mémoriser le résultat et un pointeur PCH pour parcourir la chaîne CH et un pointeur PABC pour parcourir ABC. Afficher seulement le nombre des lettres qui apparaissent au mois une fois dans le texte.

Exemple:

Entrez une ligne de texte (max. 100 caractères) :

Jeanne

La chaîne "Jeanne" contient :

1 fois la lettre 'A'

2 fois la lettre 'E'

1 fois la lettre 'J'

3 fois la lettre 'N'

```
/* Initialiser le tableau ABC */
 Pour PABC de ABC à ABC+26 Faire
    PABC^ ← 0
 Fin pour
 /* Compter les lettres */
 PCH ← CH
 Tant que PCH^ Faire
     si PCH^>='A' ET PCH^<='Z' alors
       ((ABC+(PCH^-'A'))^)← ((ABC+(PCH^-'A'))^)+1
     Finsi
     si (PCH^>='a' && PCH^<='z')alors
       ((ABC+(PCH^-'a'))^) ← ((ABC+(PCH^-'a'))^)+1
     Finsi
     PCH←PCH+1
 Fin Tq
 /* Affichage des résultats */
 /* (PABC-ABC) est le numéro de la lettre de l'alphabet. */
 Ecrire("La chaîne contient :")
 Pour PABC de ABC à ABC+26 Faire
    Si (PABC^) alors
      Ecrire(PABC^, "  fois la lettre  ", 'A'+(PABC-ABC))
    Finsi
 Fin pour
```

Exercice 7

Ecrire un programme qui lit une matrice A de dimensions N et M au clavier et qui affiche les données suivantes en utilisant le formalisme pointeur à chaque fois que cela est possible:

a) la matrice A

b) la transposée de A

c) la matrice A interprétée comme tableau unidimensionnel

Exercice 8

Ecrire un programme qui lit deux matrices A et B de dimensions N et M respectivement M et P au clavier et qui effectue la multiplication des deux matrices. Le résultat de la multiplication sera affecté à la matrice C, qui sera ensuite affichée. Utiliser le formalisme pointeur à chaque fois que cela est possible.

Exercice 9

Ecrire un programme qui lit 5 mots d'une longueur maximale de 50 caractères et les mémorise dans un tableau de chaînes de caractères TABCH. Inverser l'ordre des caractères à l'intérieur des 5 mots à l'aide de deux pointeurs P1 et P2. Afficher les mots.

TRAVAUX DIRIGES 10

LES LISTES CHAINEES

1. Objectifs

- Maitriser la définition et la manipulation des listes chainées simples.
- Représenter un ensemble d'éléments contenus chacun dans une **cellule**. Celle-ci contient en plus de l'élément, l'adresse de l'élément suivant appelé pointeur.

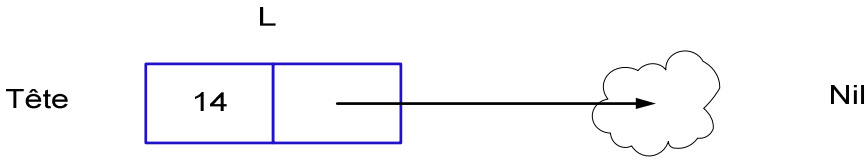
Problème A

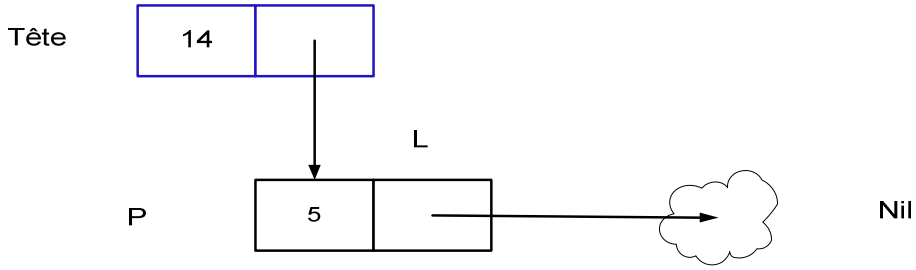
1. Ecrire une procédure «CréatListe » permettant de créer une liste chainée simple de n éléments de type entier.

```
Types
   Liste = ^Cellule
   Cellule =      Struct
                        Elem        : Entier
                        Suiv        : Liste
                  FinStruct
Variables
   L              : Liste
```

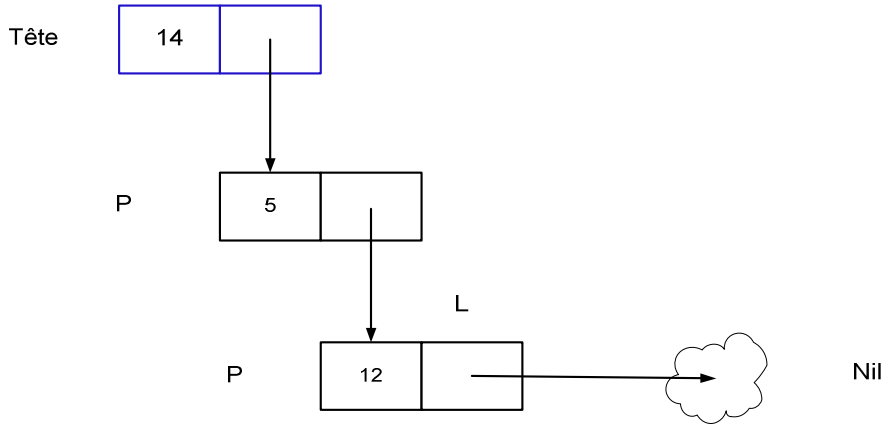
```
Procédure CréatListe(n :entier, L :Liste)
Variables
   Tête,P         : Liste
   i              : Entier
Début
   Allouer(Tête)
   Ecrire(« Entrer l'élément de tête : »)
   Lire(Tête^.Elem)
   Tête^.Suiv ← Nil
   L ← Tête
   Pour i de 2 à n Faire
         Allouer(P)
         Ecrire(« Entrer un élément : »)
         Lire(P^.Elem)
         P^.Suiv ← Nil
         L^.Suiv ← P
         L ← P
```

```
    FinPour
    L ← Tête
Fin
```

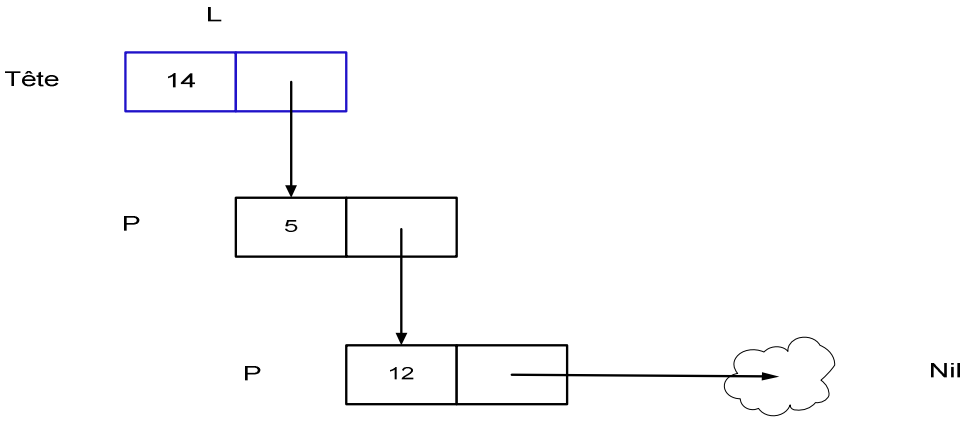




(*Pour n =2*)



(*Pour n =3*)

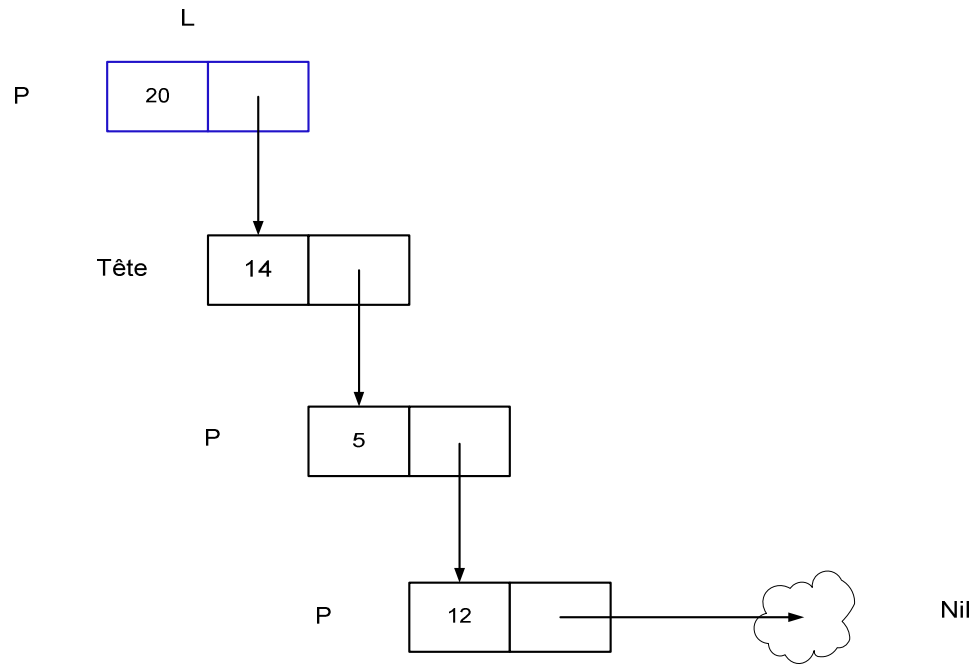


2. Ecrire une procédure « AffichListe » permettant de parcourir et afficher les éléments d'une liste chainée simple L.

```
Procédure AffichListe(L :Liste)
Variables
    P              : Liste
Début
P←L
TantQue(P≠Nil)Faire
    Ecrire(P^.Elem)
    P←P^.Suiv
FinTQ
Fin
```

3. Ecrire une procédure « InsertTete » permettant d'insérer un élément en tête d'une liste chainée simple L.

```
Procédure InsertTête(L :Liste)
Variables
    P              : Liste
Début
Allouer(P)
Ecrire(« Entrer un élément : »)
Lire(P^.Elem)
P^.Suiv ←L
L←P
Fin
```



4. Ecrire une fonction « Recherche » qui vérifie si un entier x figure dans une liste chainée simple L.

```
Fonction Recherche(x : Entier, L :Liste) :Booléen
Variables
   P              : Liste
   Trouve         :Booléen
Début
Trouve ← Faux
P←L
TantQue(P≠Nil)ET(Trouve=Faux)Faire
   Trouve ← (P^.Elem=x)
   P←P^.Suiv
FinTQ
Recherche←Trouve
Fin
```

5. En supposant qu'une valeur x existe une seule fois dans une liste chainée simple L. Ecrire une procédure « supprimer » permettant de supprimer cet élément x de la liste L. (Notez que : supprimer x revient à mettre à jour les liens de façon que le successeur du prédécesseur de x devient le successeur de x. En plus, un traitement particulier doit être fait si l'élément à supprimer est le premier élément de la liste.)

```
Procédure Supprimer(x : Entier, L :Liste)
Variables
   P ,Q           : Liste

Début
   Si(L≠Nil)Alors
   P←L
   Si(L^.Elem=x)Alors
            L←L^.Suiv
            Libérer(P)
   Sinon
            Q←L^.Suiv
TantQue(Q≠Nil)ET(Q^.Elem≠x)Faire
                  P ← Q
                  Q←Q^.Suiv
FinTQ
Si(Q≠Nil)Alors
   P^.Suiv ← Q^.Suiv
   Libérer(Q)
FinSi
   FinSi
FinSi
Fin
```

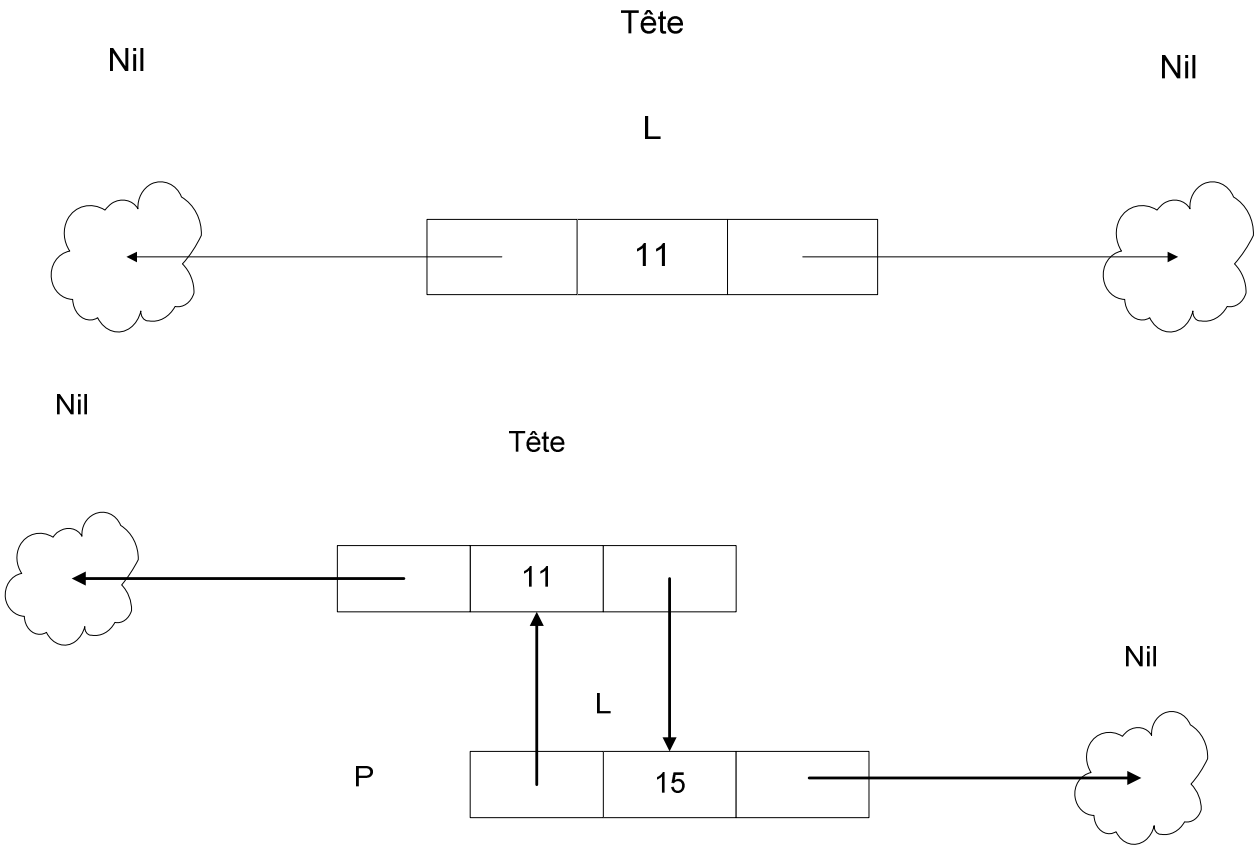
6. Ecrire une procédure qui permet d'inverser une liste chainée simple L en une liste chainée simple L_inv.
7. Ecrire une procédure qui permet de trier les éléments d'une liste chainée simple L dans l'ordre croissant.

Problème B

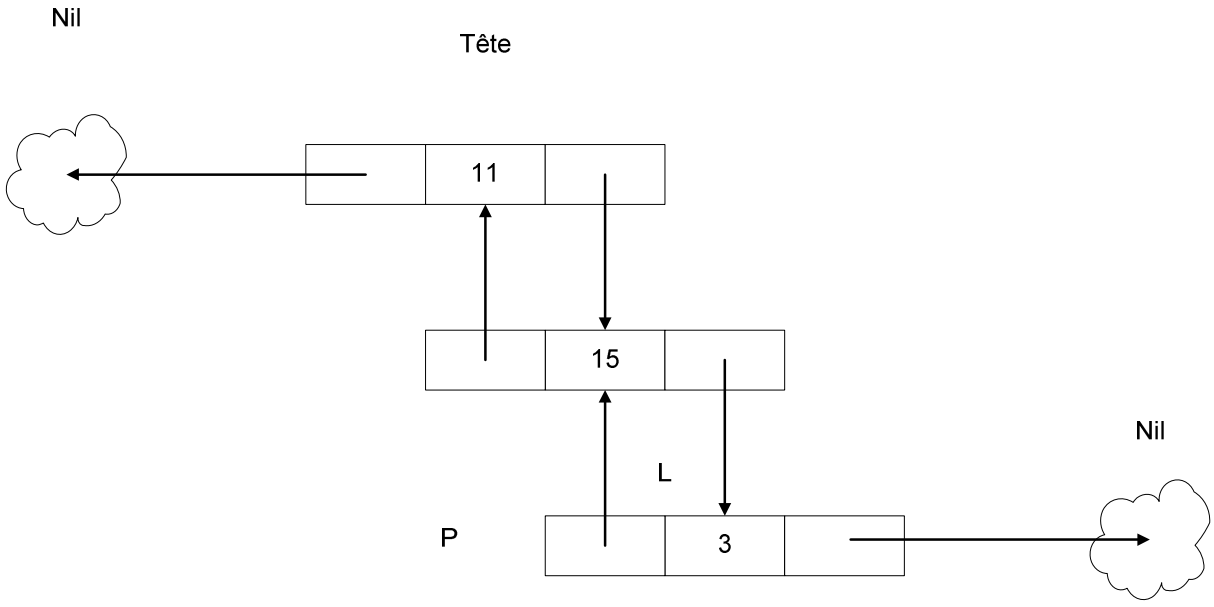
1. Ecrire une procédure «CréatListe » permettant de créer une liste à chainage double de n éléments de type entier.

```
Types
   Liste = ^Cellule
   Cellule =      Struct
                        Prec          : Liste
                        Elem          : Entier
                        Suiv          : Liste
                  FinStruct
Variables
   L              : Liste
```

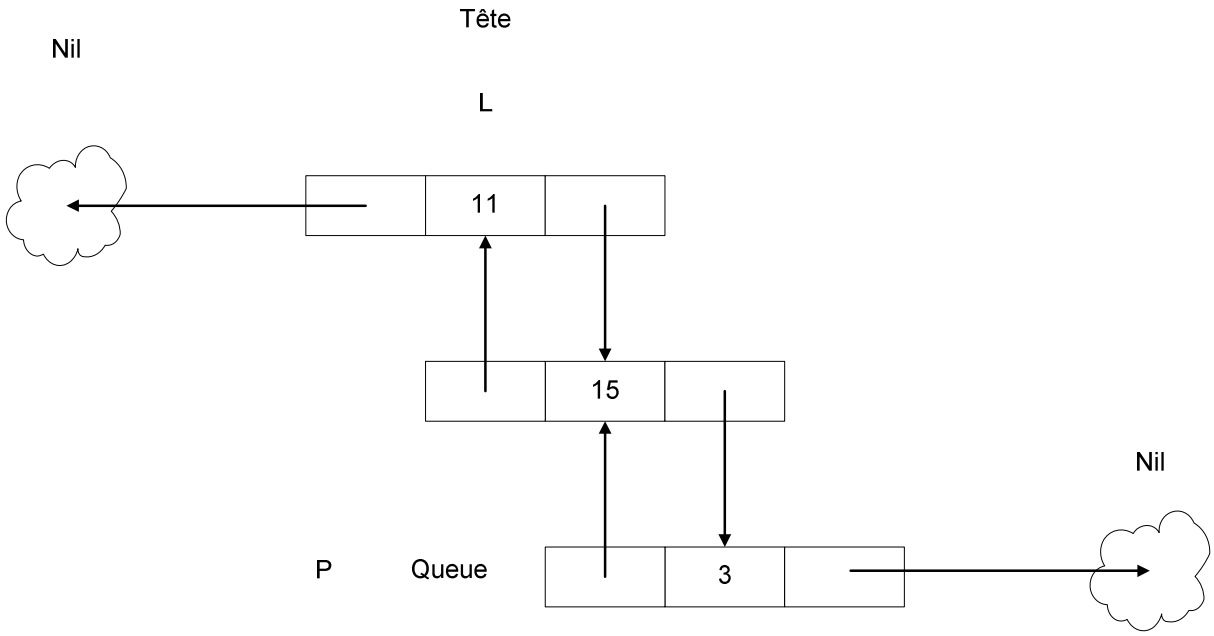
```
Procédure CréatListe(n :entier, L :Liste, Queue :Liste)
Variables
   Tête,P         : Liste
   i              : Entier
Début
   Allouer(Tête)
   Ecrire(« Entrer l'élément de tête : »)
   Lire(Tête^.Elem)
   Tête^.Suiv ← Nil
   Tête^.Prec ← Nil
   L ← Tête
   Pour i de 2 à n Faire
         Allouer(P)
         Ecrire(« Entrer un élément : »)
         Lire(P^.Elem)
         P^.Suiv ← Nil
         P^.Prec ← L
         L^.Suiv ← P
         L ← P
   FinPour
   Queue ← P
L ← Tête
Fin
```



(*Pour n =2*)



(*Pour n =3*)



2. Ecrire une procédure « AffichListe » permettant de parcourir et afficher les éléments d'une liste à chainage double L.

```
Procédure AffichListe(L :Liste)
Variables
    P              : Liste
Début
P←L
TantQue(P≠Nil)Faire
    Ecrire(P^.Elem)
    P←P^.Suiv
FinTQ
Fin
```

ou

```
Procédure AffichListe(Queue :Liste)
Variables
    P              : Liste
Début
P←Queue
TantQue(P≠Nil)Faire
    Ecrire(P^.Elem)
    P←P^.Prec
FinTQ
Fin
```

3. Ecrire une procédure « InsertTete » permettant d'insérer un élément en tête d'une liste à chainage double L et une procédure « InsertQueue » permettant d'insérer un élément en queue d'une liste à chainage double L

```
Procédure InsertTête(L :Liste)
Variables
   P              : Liste
Début
Allouer(P)
Ecrire(« Entrer un élément : »)
Lire(P^.Elem)
P^.Suiv ←L
P^.Prec ←Nil
L^.Prec ←P
L←P
Fin
```

```
Procédure InsertQueue(Queue :Liste)
Variables
   P              : Liste
Début
Allouer(P)
Ecrire(« Entrer un élément : »)
Lire(P^.Elem)
P^.Suiv ←Nil
P^.Prec ←Queue
Queue^.Suiv ←P
Queue←P
Fin
```

4. Ecrire une fonction « Recherche » qui vérifie si un entier x figure dans une liste à chainage double L.

5. En supposant qu'une valeur x existe une seule fois dans une liste à chainage double L. Ecrire une procédure « supprimer » permettant de supprimer cet élément x de la liste L. (Notez que : supprimer x revient à mettre à jour les liens de façon que le successeur du prédécesseur de x devient le successeur de x. En plus, un traitement particulier doit être fait si l'élément à supprimer est le premier élément de la liste.)

6. Ecrire une procédure qui permet d'inverser une liste à chainage double L en une liste à chainage double L_inv.

7. Ecrire une procédure qui permet de trier les éléments d'une liste à chainage double L dans l'ordre croissant.

TRAVAUX DIRIGES 11

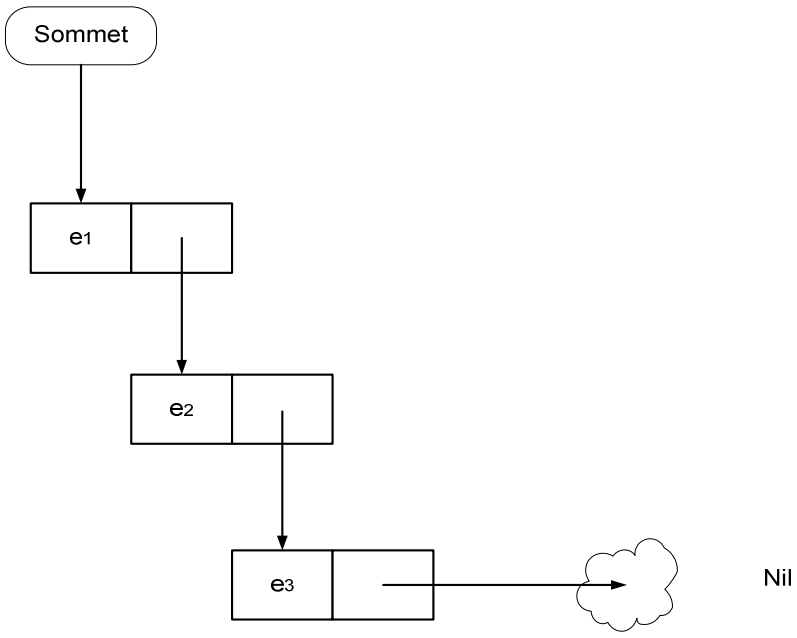
LES PILES ET LES FILES

1. Objectif

Définir et déterminer les primitives d'une pile et d'une file qui sont un cas particulier d'une liste chainée.

2. Définition des Piles

Une pile est une liste chainée où l'insertion et la suppression d'un élément se font toujours en tête de liste.



L'image intuitive d'une pile peut être donnée par une pile d'assiette ou une pile de dossiers à condition de supposer qu'on prend un seul élément à la fois celui du sommet.

On peut résumer les contraintes d'accès d'une pile par le principe « Dernier entré, premier sorti »

Quand on fait recours à une pile ?

Lorsqu'on mémorise des informations qui devront être traitées dans l'ordre inverse de leur entrée.

Déclaration d'une pile :

```
Types
   Pile = ^Cellule
   Cellule =      Struct
                        Elem          : Entier
                        Suiv          : Pile
                  FinStruct
Variables
   P              : Pile
```

Procédure Initialiser(P : Pile) (*créer une Pile P vide.*)

Fonction Pile_Vide(P :Pile) : Boolèen (*renvoyer la valeur vrai si la Pile est vide.*)

Procédure Empiler(x :Entier, P :Pile) (*ajouter l'élément x au sommet de la Pile*)

Procédure Dépiler(x :Entier, P :Pile) (* supprimer le sommet de la Pile après de l'avoir sauvegarder dans une variable x.

```
Procédure Initialiser(P :Pile)
Début
   P←Nil
Fin
```

```
Fonction Pile_Vide(P :Pile) :Booléen
Début
   Pile_Vide←(P=Nil)
Fin
```

```
Procédure Empiler(x :entier, P :Pile)
Variables
   Q              : Pile
Début
   Allouer(Q)
   Q^.Elem ← x
   Q^.Suiv ← P
   P ← Q
Fin
```

```
Procédure Dépiler(x :entier, P :Pile)
Variables
   Q              : Pile
Début
   Si NON(Pile_Vide(P)) alors
         x ← P^.Elem
         Q ← P
         P ← P^.Suiv
         Libérer(Q)
```

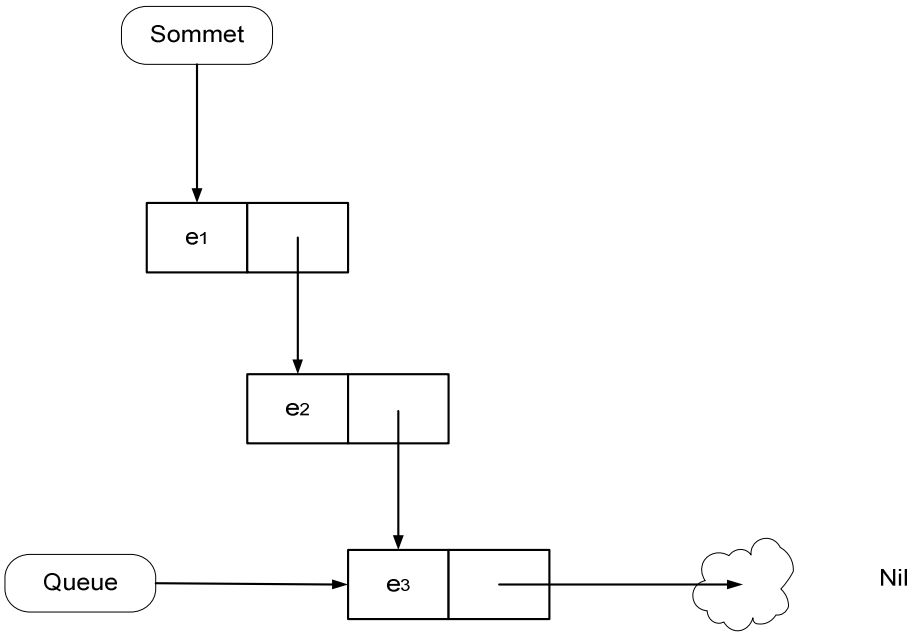
```
    Sinon
            Ecrire(« Impossible la pile est vide »)
Fin
```

3. Définition des Files

Une File est une liste chainée dont les contraintes d'accès sont définies comme suit :

- On ne peut ajouter un élément qu'en dernier rang de la liste.
- On ne peut supprimer que le premier élément.

On peut résumer les contraintes d'accès par le principe « premier entré premier sorti ».



Quand on fait recours à une File ?

Lorsqu'on mémorise des informations qui devront être traitées dans le même ordre de leur entrée.

Déclaration d'une file :

```
Types
   Liste = ^Cellule
   Cellule =        Struct
                        Elem        : Entier
                        Suiv        : Pile
                    FinStruct

File =              Struct
                        Tête        : Liste
                        Queue       : Liste
                    FinStruct
Variables
   F                : File
```

Procédure Initialiser(F : File) (*créer une File F vide.*)

Procédure Ajouter(x :Entier, F :File) (*ajouter l'élément x à la fin de la File*)

Procédure Extraire(x :Entier, F :File) (* extraire le sommet de la File après de l'avoir sauvegarder dans une variable x.

```
Procédure Initialiser(F :File)
Début
   F.Tête   ←Nil
   F.Queue ←Nil
Fin
```

```
Procédure Ajouter(x :entier, F :File)
Variables
   P              : Liste
Début
   Allouer(P)
   P^.Elem ← x
   P^.Suiv ← Nil
   Si(F.Queue ≠ Nil)alors
            F.Queue^.Suiv← P
   Sinon
            F.Tête←P
   FinSi
   F.Queue ← P
Fin
```

```
Procédure Extraire(x :entier, F :File)
Variables
   P              : Liste
Début
   Si (F.Tête=Nil) alors
            Ecrire(« Impossible la pile est vide »)
   Sinon
            P ← F.Tête
            x ← F.Tête^.Elem
            F.Tête ← F.Tête^.Suiv
            Libérer(P)
   FinSi
Fin
```

TRAVAUX DIRIGES 12

LES ARBRES

1. Objectifs

Connaitre les concepts de base relatifs aux arbres binaires.

2. Problème

1. Ecrire un algorithme et une procédure «Construire » permettant de construire un arbre d'entiers à partir d'une suite de nombres entrés par l'utilisateur. Il faut que le sous arbre gauche d'un nœud de valeur n contient les éléments < à n, tandis que le sous arbre droit contient les éléments supérieurs à n.

```
Algorithme ArbreBinaire
Types
   Arbre = ^Nœud
   Nœud =      Struct
                    Valeur      : Entier
                    Gauche: Arbre
                    Droite      : Arbre
               FinStruct
Variables
   Racine      : Arbre
   x           : Entier
```

```
Procédure Construire(Elt :Entier, B :Arbre)
Début
   Si(B=Nil)alors
         Allouer(B)
         B^.Valeur ← Elt
         B^.Gauche←Nil
         B^.Droite←Nil
   Sinon Si(Elt<B^.Valeur)alors
         Construire(Elt, B^.Gauche)
   Sinon
         Construire(Elt, B^.Droite)
   FinSi
Fin
```

```
Début Programme principal
   Racine ← Nil
   Ecrire(«Entrer un entier :»)
   Lire(x)
   Tant que(x)faire
            Construire(x,racine)
            Ecrire(«Entrer un entier :»)
            Lire(x)
   FinTQ
Fin
```

2. On cherche à parcourir un arbre binaire selon une stratégie dite « en profondeur d'abord » ou dans l'ordre préfixe. Ecrire la procédure ParcoursPréfixe.

```
Procédure ParcoursPréfixe(B :Arbre)
Début
   Si(B≠Nil)alors
            Ecrire(B^.Valeur)
            ParcoursPréfixe(B^.Gauche)
            ParcoursPréfixe(B^.Droite)
   FinSi
Fin
```

3. On cherche à parcourir un arbre binaire dans l'ordre infixe. Ecrire la procédure ParcoursInfixe.

```
Procédure ParcoursInfixe(B :Arbre)
Début
   Si(B≠Nil)alors
            ParcoursInfixe(B^.Gauche)
            Ecrire(B^.Valeur)
            ParcoursInfixe(B^.Droite)
   FinSi
Fin
```

4. On cherche à parcourir un arbre binaire dans l'ordre postfixe. Ecrire la procédure ParcoursPostfixe.

```
Procédure ParcoursPostfixe(B :Arbre)
Début
   Si(B≠Nil)alors
            ParcoursPostfixe(B^.Gauche)
            ParcoursPostfixe(B^.Droite)
            Ecrire(B^.Valeur)
   FinSi
Fin
```

5. Ecrire une fonction « Recherche » permettant de chercher un élément x dans un arbre binaire ordonné B.

```
Fonction Recherche(x :Entier, B :Arbre) :Booléen
Début
   Si(B=Nil)alors
         Recherche←Faux
   Sinon Si(x= B^.Valeur)alors
         Recherche←Vrai
   Sinon Si(x<B^.Valeur)alors
         Recherche(x, B^.Gauche)
   Sinon
         Recherche(x, B^.Droite)
   FinSi
Fin
```

6. Ecrire une procédure qui permet d'ajouter un élément à un arbre binaire (on suppose que cet élément n'existe pas dans l'arbre).

7. Ecrire une procédure qui permet de supprimer un élément d'un arbre binaire.

TRAVAUX DIRIGES 13

EXERCICES DE REVISIONS

Exercice : Pointeur

Ecrire un programme qui lit une chaîne de caractères CH au clavier et qui compte les occurrences des lettres de l'alphabet en ne distinguant pas les majuscules et les minuscules.

Utiliser un tableau ABC de dimension 26 pour mémoriser le résultat et un pointeur PCH pour parcourir la chaîne CH et un pointeur PABC pour parcourir ABC. Afficher seulement le nombre des lettres qui apparaissent au mois une fois dans le texte.

Exemple:

Entrez une ligne de texte (max. 100 caractères) :

Jeanne

La chaîne "Jeanne" contient :

1 fois la lettre 'A'

2 fois la lettre 'E'

1 fois la lettre 'J'

3 fois la lettre 'N'

Exercice : Récursivité

Calcul du PGCD par la méthode d'Euclide :

Ecrire une fonction récursive PGCD_Euc qui retourne le PGCD de 2 entiers a et b en utilisant l'algorithme d'Euclide qui s'appuie sur les propriétés suivantes :

PGCD(a,b)=b si b est un diviseur de a

PGCD(a,b)=PGCD(b, a mod b) sinon

Exemple PGCD(36,20) = PGCD(20,16)=PGCD(16,4)=4.

Exercice : Fichier

1. Ecrire une procédure permettant de créer et remplir le fichier (de type Fi_employés) qui contient des informations sur les employés d'une entreprise (matricule, nom, prénom, grade, salaire).

2. Ecrire une procédure permettant d'afficher la liste des employés à partir du fichier de type Fi_employés.

3. Ecrire une fonction permettant d'afficher la liste des employés dont le salaire est compris entre 500 et 700 DT.

Exercice : Enregistrement

Ecrire un algorithme qui lit deux nombres complexes C_1 et C_2 et qui affiche ensuite leur somme et leur produit.

Sachant que :

(a+bi)+(c+di)=(a+c)+(b+d)i

(a+bi)*(c+di)=(ac-bd)+(ad+bc)i

Exercice : liste chainées simples

1. Ecrire une procédure «CréatListe » permettant de créer une liste chainée simple de n éléments de type entier.
2. Ecrire une procédure « AffichListe » permettant de parcourir et afficher les éléments d'une liste chainée simple L.
3. Ecrire une procédure « InsertTete » permettant d'insérer un élément en tête d'une liste chainée simple L.
4. Ecrire une fonction « Recherche » qui vérifie si un entier x figure dans une liste chainée simple L.
5. En supposant qu'une valeur x existe une seule fois dans une liste chainée simple L. Ecrire une procédure « supprimer » permettant de supprimer cet élément x de la liste L. (Notez que : supprimer x revient à mettre à jour les liens de façon que le successeur du prédécesseur de x devient le successeur de x. En plus, un traitement particulier doit être fait si l'élément à supprimer est le premier élément de la liste.)
6. Ecrire une procédure qui permet d'inverser une liste chainée simple L en une liste chainée simple L_inv.
7. Ecrire une procédure qui permet de trier les éléments d'une liste chainée simple L dans l'ordre croissant.

REFERENCES DE BASE

1. Baghdadi Zitouni. ***Algorithme & Structure de Données***, Centre de Publication Universitaire, 2003.

2. Christophe Darmangeat. ***Cours d'algorithmique et de programmation***, enseigné à l'Université Paris 7, dans la spécialité PISE du Master SSAMECI (ancien DESS A.I.G.E.S.), **http://www.pise.info/algo/introduction.htm** , 2011.

3. Faber Frédéric. ***INTRODUCTION A LA PROGRAMMATION EN ANSI-C***, un manuel pour études d'autodidacte comprenant exercices et solutions, **http://www.ltam.lu/cours-c/** , 1993-1997.