Cours et Excercice en Algorithme

INTRODUCTION GENERALE

I : ANALYSE, ALGORITHME, PROGRAMMATION :

But : acquérir une méthode, des outils : démarche a suivre d'un problème a résoudre à un programme informatique.

Domaines d'applications :

- ⇨ Gestion (facturation, paye,…)
- ⇨ Informatique scientifique (météorologie, astronomie,…)
- ⇨ Systèmes industriels (commandes numériques, robotique,…)
- ⇨ Informatique ludique (informatique personnelle, jeux,…)
- ⇨ Etc

Quelque soit le domaine, la démarche de conception du programme reste identique.
Démarche
Problème a résoudre

Etude Préalable	Compréhension du problème, modélisation du problème
Spécification des données et des résultats	Recenser les informations et préciser leur nature
Spécification de fonctionnalités	Recenser et préciser
Solution en langage naturel	Savoir résoudre le problème avant d'automatiser la solution
Données structurées / Algorithme	Mise en forme informatique des informations et des traitements à réaliser
Programmation	Choix du langage, traduction de la solution (algorithme) sous forme de programme
Programme exécutable	Compilation du programme en programme exécutable
Test et évaluation du travail réalisé	Test de la cohérence par rapport aux spécifications
Documentation	Manuel d'utilisateur, aide en ligne, manuel de maintenance du logiciel

II LA NOTION D'ALGORITHME :

Du mathématicien persan Al-Khwa-Rizm (Bagdad, 780 – 850)
Pour les notions de Al-Jabr (Algèbre) théorie du calcul

Plus ancien : Euclide (3eme siècle avant JC)
Babyloniens (1800 avant JC)

Selon le LAROUSSE, la définition d'algorithme est « un ensemble de règles opératoires dont l'enchaînement permet de résoudre un problème au moyen d'un nombre fini d'opérations. »

Quelques points importants :

- ⇨ Un algorithme décrit un traitement sur un ensemble fini de données de nature simple (nombres ou caractères), ou plus complexes (données structurées)
- ⇨ Un algorithme est constitué d'un ensemble fini d'actions composées d'opérations ou actions élémentaires. Ces actions élémentaires doivent être effectives (réalisable par la machine), non ambiguës.
- ⇨ Un algorithme doit toujours se terminer après un nombre fini d'opérations.
- ⇨ L'expression d'un algorithme nécessite un langage clair (compréhension) structuré (enchaînements d'opérations) non ambiguë, universel (indépendants du langage de programmation choisi)

Problème : un tel langage n'existe pas, on définit son propre langage.

III MÉTHODOLOGIE DE CONCEPTION D'UN ALGORITHME :

Analyse descendante : (ou programmation structurées) : on décompose un problème complexe en sous problèmes et ces sous problèmes en d'autres sous problèmes jusqu'à obtenir des problèmes faciles a résoudre c'est-à-dire connus.
On résout les sous problèmes simples sous forme d'algorithme puis on recompose les algorithmes pour obtenir l'algorithme global du problème de départ.

Garder à l'esprit :

- ⇨ La modularité : un module résout un petit problème donné. Un module doit être réutilisable.
- ⇨ Lisibilité de l'algorithme (mise en page, commentaires, spécification : dire quoi mais pas comment)
- ⇨ Attention à la complexité de l'algorithme :
  - *Complexité en temps* : mesure du temps d'exécution en fonction de la taille des données
  - *Complexité en espace* : espace mémoire nécessaire pour effectuer les traitements.
- ⇨ Ne pas réinventer la roue (c'est-à-dire ne pas refaire les programmes standard dont les solutions sont connues) ce qui implique avoir une certaine culture et un outil technique standard

I-Introduction

Un type de donnée détermine :
- un domaine : ensemble des valeurs possibles pour les objets de ce type.
- primitives : ensembles des opérations permettant de « manipuler » les objets de ce type.

On distingue :
- les types élémentaires : simples (entiers, caractères,…) prédéfinis dans la plupart des langages de programmation
- les types structurés : construits à l'aide de constructeurs de types (exemple : tableaux)
- les types abstraits : réservés à des structures plus évoluées (listes, files, arbres,…)

Remarque : certains types peuvent ne pas exister (ou pas totalement). Dans ce cas, il est possible de les simuler :
- coder les objets à l'aide des constructeurs existants (coder une liste par un tableau)
- écrire les primitives sous forme d'actions ou de fonctions

types élémentaires

1-LES ENTIERS

domaine [-N ; =N[ou N dépend du nombre d'octets
primitives :
- arithmétique : + - * div mod
- comparaison = ≥ ≠ <…
- fonctions particulières (ValAbs,…)

ATTENTION : problèmes de débordements (sortir du domaine)

2-LES RÉELS (NOMBRES A VIRGULES FLOTTANTES)

Domaine : un sous ensemble fini, d'un intervalle de la forme : [-N, -ε]U[0]U[ε , N]
Primitives : idem que les entiers, plus les fonctions mathématiques.
ATTENTION : les calculs sont toujours faux : ils sont approchés. Exemple : A≠ B*(A/B)

En pratique : on utilise des bibliothèques implémentant, par exemple, les décimaux à 1000000 de décimales en assurant, par exemple, une précision exacte jusqu'a la $n^{ème}$. décimale

3-LE TYPE BOOLÉEN

Domaine : Vrai, Faux
Primitives : opérateur booléenne, comparaisons
Remarque : se familiariser avec les écritures :
Tant que non fini ⇔ tant que fini = faux
Si correct alors ⇔ si correct = vrai alors
Retourner (trié) ⇔ Si trié = vrai alors retourner (vrai) sinon retourner (faux)

4-TYPE CARACTÈRES

Domaine : codés sur un octet (ASCII) voire deux octets.
Quelque soit le codage :

'a', 'b',…,'z' : sont consécutifs
'A', 'B',…,'Z' : sont consécutifs
'0',…,'9' : sont consécutifs
l'espace '_' précède les caractères alphabétiques.

Primitives :

ORD : 1 entier => le caractère ayant ce code
CHR : 1 caractère => son code

A savoir :

- Si (car≥ '0') et (car≤ '9') : tester un caractère chiffre
- Car<=chr(ord(car)+ord(A)-ord(a)) : convertir une minuscule en majuscule sans avoir a connaître le codage des lettres car les lettres sont codées consécutivement.
- Val<=ord(car)-ord('0') : récupère la valeur d'un chiffre.

Les chaînes de caractères

En ASD :
Var : ch1 : chaîne [30] : longueur ≤ 30
Ch2 : chaîne : longueur quelconque

Codage en C/C++

B	o	n	j	o	u	r

1 octet = 1 caractère
Primitives :

- accès au $i^{ème}$ car : ch(i-1)
- concaténation : +
- fonction longueur
- fonction spécifiques (cf doc)
- comparaison =, <, ≥

II-Eléments de base de l'algorithme

Notion d'objet

Un algorithme ou une action manipule des données pour obtenir un résultat.
Pour cela on manipule des objets simples ou structurés.

Exemple : on calcule le quotient q et le reste r de la division entière de a par b par soustraction successives. 25-6=19 donc q=1 puis 19-6=13 donc q=2 puis 13-6=7 donc q=3 puis 7-6=1 donc q=4 puis 1-6 impossible donc q=4 et r=1.

Un objet va être caractérisé par :

- ⇨ un identificateur (son nom) : pour le désigner cet identificateur doit être parlant : q=quotient…
- ⇨ Un type (nature de l'objet : entier, caractère…) simple ou structuré. Un type détermine en particulier les valeurs possibles de l'objet et les opérations primitives applicable à l'objet.
- ⇨ Une valeur (contenu de l'objet) unique. Cette valeur peut varier au cours de l'algorithme ou d'une exécution à l'autre : ces objets sont des variables.

Dans les cas contraires (valeur fixe) ce sont des constantes. Tous les objets manipulé par un algorithme doivent être clairement définis :

Mots clefs => const : PI=3.14
=> var a, b : entiers
x, y : caractères

a, b, x, y sont des identificateurs

Actions élémentaires :

Actions élémentaires : opérations simple, directement utilisable.

1-Affectation

Permet de donner une valeur à une variable :
A<=28 « reçoit » Si A avait une valeur auparavant, cette valeur disparaît : elle est écrasé par 28

Format général :
<id_variable> <= <expression> A<=28+13
A « l'exécution » : l'expression est évaluée (calculée) et sa valeur est rangée dans la variable.
Donc les types <id_variable> et <expression> doivent être compatibles.

Attention : A<=B+3
B doit avoir une valeur. Or au début d'une action, les variables ont une valeur indéterminée/ B doit avoir été initialisé.

2-Les opérations en entrée sortie

Elles permettent de récupérer une valeur venant de l'extérieur (lecture) ou de transmettre une valeur à l'extérieur (écriture) :

```
var A, B, C : entiers
Lire (A, B)
C<=A+B
Ecrire C
```

plus rapide :

```
lire (A, B)
écrire (A+B)
```

Remarques :

La notion de littéral : A<=28
28 est un objet caractérisé par son type (entier [numérique]), son nom (sa valeur), et sa valeur (28)
Le littéral est l'objet « constante », le nom est sa valeur
On note les littéraux de type caractère entre quotte 'A'.
On note les littéraux de type chaîne de caractères entre double quotte : ''bonjour''

Autre remarque : Lors de l'écriture d'un algorithme, éviter les détails de programmation au niveau des entrées sorties :
par exemple on écrira : écrire (A, B, C)
Et non écrire (''Le produit de'', A, ''par'', B, ''vaut'', C)

Les structures de contrôle conditionnelles :

Une action décrit un enchaînement d'actions élémentaires. L'enchaînement est décrit par les structures de contrôle.
Une structure de contrôle déjà vue : l'enchaînement séquentiel

```
lire (A, B)
C <= 2*A + B
```

La plupart des autres structures de contrôle utilise la notion de condition (expression booléenne) :
Une condition a une valeur qui est, soit vrai, soit fausse.
Pour déterminer la réalité de cette valeur on utilise :

- les opérateurs de comparaisons =, ≤ , ≥ , ≠
- les opérateurs booléens (logique) : ET, OU, NON

L'alternative SI-ALORS-SINON

Elle permet d'effectuer tel ou tel traitement en fonction de la valeur d'une condition.
Syntaxe :

```
Si <condition>
Alors < action _alors>
Sinon < action _sinon>
```

Exemple :

```
Lire (note)
Si note ≥ 10
Alors écrire ''Bravo''
Sinon écrire ''Désolé''
```

Remarque, la ligne *Sinon* <*action_sinon*> est facultative.

Principe de fonctionnement :
1 : la condition est évaluée
2 : Si la condition a la valeur vrai on exécute <action_alors>
Si la condition a la valeur fausse on exécute <action_sinon>

Remarque :
Les <action_alors> ou <action_sinon> peuvent être soit :
- des actions élémentaires
- des composées (bloc)

Dans ce cas on utilise les structures imbriquées.
Exemple de structure imbriquée:

```
Si A≥ 10
Alors   début
        Si A≤ 10
        Alors écrire (''oui'')
        Fin
Sinon écrire (''non'')
```

STRUCTURE À CHOIX MULTIPLES SELON-QUE :

Exemple :

```
Selon que
Note ≥  16 : écrire (''TB'')
Note ≥  14 : écrire (''B'')
Note ≥  12 : écrire (''AB'')
Note ≥  16 : écrire (''Passable'')
Sinon : écrire (''ajourné'')
Fin selon
```

Fonctionnement :
1 : la condition 1 est évaluée :
Si la condition 1 est vraie, alors on exécute l'action correspondante et on quitte la structure selon-que
Si la condition 1 est fausse , on évalue la condition 2…et ainsi de suite.
Si aucune n'est vraie on effectue l'action sinon.

Syntaxe :

```
Selon que
<condition 1> : <action 1>
<condition 2> : <action 2>
…
<condition n> : <action n>
sinon : <action_sinon>
fin selon
```

Remarque : en programmation, cette structure peut exister mais avec une forme ou un fonctionnement éventuellement différent. Si elle n'existe pas, il faut se souvenir que, en fait, SELON QUE est un raccourci d'écriture pour des SI imbriqués.

Actions et fonctions :

Un algorithme est composé d'un ensemble fini d'actions. En fait, on distingue :
- les actions qui réalisent un traitement (lecture d'un complexe, tri du fichier étudiant)
- les fonctions qui effectuent un calcul et retournent un résultat

En programmation, on trouve parfois cette distinction de façon explicite. Comme en Pascal, ou l'on a procedure et functions. En revanche, en C ou C++ cette distinction n'existe pas, il n'y a que des fonctions.

En algorithme, on s'avisera de faire cette distinction entre fonction et action, pour la simple raison que chacune ne fait pas intervenir le même type de variable ou de paramètres :

SYNTAXE D'UNE FONCTION :

Fonction <nom_fonction> (<liste des paramètres>) : <type de résultat>
< déclaration des objets locaux à la fonction>
début
{ corps de la fonction}
retourner résultat (en général vers une fonction ou action principale)
fin

SYNTAXE D'UNE ACTION :

Action <nom_action>
< déclaration des objets locaux à l'action>
début
{corps de l'action}
fin

EXEMPLE DE FONCTION :

Fonction périmètre rectangle (largeur, longueur : entiers) : entier
Début
Retourner [2*(largeur+longueur)]
Fin

UN EXEMPLE DE FONCTION ET D'ACTION QUI FAIT APPEL À UNE FONCTION :

Soit la fonction max3 qui donne le maximum de trois entiers :

```
Fontion max3 (x, y, z : entiers) : entier
{cette fonction détermine le max de trois entier}      <= Remarque : on peut insérer des commentaires entre {}
var : max :entier
début
si x < y
alors     début
          si z<y alors max <= y
          sinon max <=z
          fin
sinon     début
          si x < z alors max <= z
          sinon max <= x
          fin
retourner (max)
fin
```

Maintenant, on peut créer l'action qui détermine le maximum et le minimum de trois entiers, en faisant appel à la fonction max3 :

```
Action max&min
Var : a, b, c, min, max : entiers
Début
Lire (a, b, c)
Max <= max3 (a, b, c)
Min <= -max3 (-a, -b, -c)
Ecrire (min, max)
Fin
```

Remarques :

X, y, z sont les paramètres formels de la fonction max3.
Ce sont des paramètres d'entrés : lors de l'appel de la fonction max3, les valeurs des arguments d'appel (ici : a, b, c) ou (-a, -b, -c)) sont transmises aux paramètres x, y, z en fonction de leur ordre.

Les arguments sont des expressions (par exemple : max <= max3 (2*a+b, c-b, a*c)) qui sont évaluées à l'appel.
Leur valeur est transmise aux paramètres.
Naturellement, le type des expressions doit être compatible avec le type des paramètres.

Structures répétitives :

Idée : répéter un ensemble d'opérations, arrêter la répétition en fonction d'une condition

LA STRUCTURE TANT QUE :

Syntaxe :
Tant que <condition>
Faire <action>
Cette action peut être simple ou composée

Exemple :

```
I<=5
Tant que i≤ 5
Faire     début
          Ecrire (i*i)
          I<=i+1
Fin
```

Fonctionnement :
1 : la condition est évalué
2 : si la condition est fausse : c'est fini, on quitte le tant que
3 : si la condition est vraie, on exécute le contenu du tant que puis on remonte à l'étape 1 tester de nouveau la condition.

Remarque :
Le contenu de la structure tant que peut ne jamais être exécuté. Donc cette structure permet en réalité de répéter un traitement, 0, 1 ou plusieurs fois.
La condition étant évaluée au début, les variables étant utilisée dans la condition doivent avoir été initialisées.
On doit s'assurer de la terminaison (sinon le programme ne se termine jamais)
Pour cela, il faut nécessairement que dans le corps de la structure, la condition soit modifié quelque part.

Faisons l'analyse d'un programme écrit avec une structure tant que :

```
Action Division entière
{détermine et affiche le quotient et le reste de la division de 2 entiers}
Var : a, b, q, r : entiers
Début :  Lire (a, b)
         r<=a
         q<=0
tant que r≥ b
faire    début
         q<=q+1
         r<=r-b
         fin
écrire (q, r)
fin
```

Faire tourner à la main pour vérifier si le programme fonctionne. Par exemple vérifions qu'il marche pour la division de 15 par 6 :

Instructions	a	b	q	r
Lire (a,b)	15	6		
r<=a			0	15
r≥ b vrai				
q<=q+1			1	
r<=r-b				9

r≥ b vrai				
q<=q+1			2	
r<=r-b				3
r≥ b faux				
écrire			2	3

On peut se poser la question de la terminaison :
En effet si b<0 alors problème : il faut rajouter des conditions.
De plus, si b=0 la division n'est pas définie.
Donc cet algorithme ne marche que pour 2 entiers a et b lus au clavier tels que a ≥ 0 et b >0

STRUCTURE RÉPÉTER :

Syntaxe :
Répéter
<actions simples>
jusqu'à <condition>

Fonctionnement :
1 : on exécute le corps
2 : on évalue la condition
3 : si Vraie : on quitte le répéter
4 : si Fausse on recommence

Remarques :
Il y a toujours au moins une exécution du corps. La structure répéter permet de répéter un traitement 1 ou plusieurs fois.
Pour choisir entre répéter et tant que il faut se poser la question : faut-il éventuellement ne jamais faire le traitement ? Si oui : il faut utiliser tant que, sinon utiliser la structure répéter qui exécute au moins une fois l'action.

Attention, en C++ :
La structure est do…while : c'est à dire Faire…tant que.
Alors que la structure algorithmique est répéter…jusqu'à.
C'est à dire qu'en C++ on exécute l'action tant que une condition est vrai alors qu'en algorithme on exécute une action tant que le condition est fausse, c'est à dire jusqu'à ce que la condition inverse soit vraie.

STRUCTURE POUR :

Elle permet de parcourir un intervalle en répétant un traitement pour chacune des valeurs de cet intervalle.
Exemples :

1) Pour I de 1 à 5 faire Ecrire (I*I)
2) Pour I de 1 à 11 par pas de 3 faire Ecrire (I*I)

Syntaxe :
Pour <id_variable> DE <val_inférieure> A <val_supérieure>
[par pas de <val_pas>] <= facultatif
Faire <actions>

Les actions peuvent être simples ou composées.

Fonctionnement :
1 : Automatiquement, on a id_variable <= val_inférieure
Donc, on a pas besoin d'initialiser, la structure se charge de la faire
2 : id_variable > val_supérieure ?
Si oui, alors STOP, on quitte la structure
Sinon :

- on exécute le programme
- automatiquement, l'incrémentation se fait (+1 ou +pas si l'on a définit un pas particulier, par défaut, le pas est 1)
- on remonte au début du 2 tester la condition id_variable > val_supérieure ?

Remarques :
Il est possible que l'action ne soit jamais exécutée.
Il est cependant possible d'avoir un intervalle inversé à condition d'avoir un pas négatif.

IMPORTANT : Il est absolument interdit de modifier <id_variable>, <val_inférieure>, <val_supérieure>, <val_pas> dans le corps de boucle.
Parfois cette structure n'est pas présente dans un langage de programmation, il faut donc retenir que ce n'est qu'un raccourci pour écrire des tant que.

Utilisation du POUR :
On s'en sert dès que l'on connaît au début de la boucle le nombre de répétitions à effectuer.
Dans les cas contraire, on utilisera des TANT QUE ou des REPETER

Structures de données : les tableaux :

Exemple de tableau de 5 entiers :

0	1	2	3	4
7	32	-8	19	-3

T

T signifie que c'est un objet de type tableau.
Les numéros en indices 0, 1, 2, 3, 4 correspondent aux valeurs colonnes.
Le contenu de T : les 5 entiers (dans un certain ordre)
La première valeur est T[0] ou 0 correspond donc à l'indice de la première colonne.

Déclaration d'un tableau dans un algorithme :
Ctaille est la constante qui indique le nombre de case du tableau.

```
Const Ctaille=25
Var Tab : tableau [Ctaille] d'entiers
Pour I de 0 à Ctaille – 1
Faire Lire tab [I]
{Cet algorithme permettra notamment de rentrer les variables dans le tableau.}
```

Pour créer une fonction tableau :

```
Fonction Mintableau (T :tableau d'entiers, N entier) : entier
{pour créer la fonction Mintableau qui retournera par exemple le minimum d'un tableau, on désigne le tableau par T et N par le nombre de colonnes, ainsi, même si l'on a déclaré auparavant un tableau à 50 colonnes et que l'on n'utilise en fait que 30 colonnes, N=30 permet de définir à l'ordinateur le nombre de colonnes réellement utilisées et limiter la durée du traitement. N est donc indépendant de Ctaille}
```

Exemple : déterminer si un tableau est trié :

5	8	8	12	15	3	2	1

Le nombre n'est pas toujours fixé donc pas de structure POUR.

PREMIÈRE PROPOSITION

```
Fonction : est trié (T : Tableau [ ] d'entiers, N : entier) : booléen
Var : I : entier
Début :
I <= 0
Tant que (I < N-1) et (T[I] ≤ T[I+1]) faire I<=I+1
Retourner (I=N-1)
Fin
```

Remarque :
Il y a un problème qaudn I atteint le niveau N-1 : on évalue la condition I<N-1 et (T[I] ≤ T[I+1]) or la case T[I+1] n'existe pas. Or la première partie (I<N-1) renvoie faux : mais certains langages évalue quand même la deuxième

condition (ce n'est pas le cas du C++) : ce langage n'est pas universel, donc on ne peut pas l'appliquer en algorithmique.

DEUXIÈME PROPOSITION :

On utilise une variable booléenne :

```
Fonction : Est trié
Var : I entier
Trié : booléen
Début
I<=0   trié<=vrai
Tant que (I<N-1) et trié
Faire
        Si T[I]≤ T[I+1] alors I<=I+1
        Sinon trié<=faux
Retourner trié
Fin
```

PARTIE II
EXCERCICES CORRIGES

Exercices Simples

Exercice 1 :
Ecrire un algorithme d'une action qui échange deux variables A et B

أكتب تنظيم منهجي يقوم بتبديل المتغيريت A و B

```
Action : Echange
Var : A, B, C : réels
Début :  Ecrire (« Saisissez deux variables »)
         Lire (A, B)
         C <= A
         A <= B
         B <= C
Ecrire (« les valeurs de », A, « et de » , B, « ont été changées »)
Fin
```

Exercice 2 :
Ecrire une fonction qui donne les carré d'un réel

أكتب دالة تقوم بإعطاء مربع عدد حقيقي

```
Fonction : carré (x :réel) :réel
Var : x_au_carré
Début
x_au_carré <= x*x
retourner x_au_carré
fin
```

Remarques :
Dans une fonction, la seule variable qui est définie est celle du résultat, les autres sont définies dans la fonction mère, et apparaissent ici en temps qu'entrées.
Dans une fonction, ne pas oublier de retourner le résultat.

exercice en utilisant les structures SI…ALORS…SINON et SELON…QUE

Exercice 3 :
Ecrire une action qui fournit les félicitations ou l'ajournement d'un étudiant suivant sa note en utilisant Si-alors-sinon.

أكتب الحركة التي تطبع نجاح الطالب حسب النقطة المتحصل عليها

```
Action : Jury
Var : note : réel
Début :  lire (note)
         Si note <10 alors écrire (« ajourné »)
         Sinon écrire (« reçu »)
Fin
```

Exercice 4 :

Ecrire un programme qui donne la valeur absolue de 2 réels :

أكتب تنظيم منهجي يطبع القيمة المطلقة لعددين حقيقيين

```
Action : Valeur_absolue
Var : a, b : réels
Début :  Ecrire (« saisissez 2 réels »)
         Lire (A, B)
         Ecrire « les valeurs absolues de A et de B sont : »)
         Si A<0 alors écrire (-A)
         Sinon écrire (A)
         Ecrire (« et »)
         Si B<0 alors écrire (-A)
         Sinon écrire (A)
Fin
```

Remarque : on peut aller plus vite en créant une fonction valeur absolue et en faisant appel à cette fonction dans une action :

```
Fonction : valAbs (x :réel) :réel
Var : absx : réel
Début :  si x <0 alors absx <= -x
         Sinon absx <= x
Retourner absx
Fin
```

Et

```
Action : Valeur_absolue2
Var : A, B réels
Début :  Ecrire (« saisissez 2 réels »)
         Lire (A, B)
         Ecrire (« les valeurs de A et B sont : », valAbs(A), « et », valAbs(B))
```

Ecrire 5 :

Faire un programme qui donne le volume d'un cylindre en faisant appel à une fonction 'aire d'un cercle'.

أكتب تنظيم منهجي يحسب و يطبع حجم ، أستناد بي مساحة الدائرة

```
Fonction : aire_cercle (rayon :réel) :réel
Var : Aire : réel
Const : PI=3.14
Début :  Aire <= PI*rayon*rayon
         Retourner (Aire)
Fin

Fonction : volume_cercle (hauteur, rayon :réels) :réel
Var : volume : réel
Début :  Volume <=aire_cercle (rayon)*hauteur
         Retourner volume
Fin
```

Exercice 6 :

Ecrire un algorithme permettant de résoudre une équation du premier degré

أكتب تنظيم منهجي يقوم بحل المعادلة من درجة الأولى

```
Action : premierdegre
Var : a, b, x réels
Début :  Ecrire (« saisissez les valeurs a et b de l'équation ax+b=0 : »)
         Lire (a, b)
         Si a = 0 alors écrire (« pas de solution »)
```

```
        Sinon écrire (« la solution est x= », -b/a)
Fin
```

Exercice 7 :

Ecrire un algorithme permettant de résoudre une équation du second degré en utilisant des si alors..

أكتب تنظيم منهجي يقوم بحل المعادلة من درجة الثانية

```
Action : seconddegré
Var : a, b, c, delta
Début :  Ecrire (« saisissez les valeurs a, b et c de l'équation ax²+bx+c=0 : »)
         Lire (a, b, c)
         Si a=0 alors écrire (« équation du premier degré »)
         Sinon delta<=b²-4*a*c
                  Début
                  Si delta>0 alors écrire (« les solutions de l'équation sont », (-b-sqrt(delta))/(2*a), « et », (-
                  b+sqrt(delta))/(2*a))
                  Sinon
                           Début
                           Si d=0 alors écrire ( -b/(2a))
                           Sinon écrire (« pas de solutions réelles »)
                           Fin
                  Fin
Fin
```

Ecrire le même algorithme avec des selon-que :

الحل بحركة أخرة

```
Action : seconddegré
Var : a, b, c, delta
Début :  Ecrire ("saisissez les valeurs de a, b et c de l'équation ax²+bx+c)
         Lire (a, b, c)
Si a=0 alors écrire (« résoudre permier degré »)
Sinon    début
         Delta <= b²-4*a*c
         Selon que
         Delta > 0 : écrire ((-b-sqrt(delta))/(2*a), (-b+sqrt(delta))/(2*a))
         Delta = 0 : écrire (( -b/(2a))
         Sinon écrire (« pas de solution réelle »)
         Fin selon
Fin
```

Exercice 8

Ecrire un algorithme qui donne la durée de vol en heure minute connaissant l'heure de départ et l'heure d'arrivée.

1) on considère que le départ et l'arrivée ont lieu même jour
2) idem mais sans faire les conversions en minutes
3) on suppose que la durée de vol est inférieure à 24 heures mais que l'arrivée peut avoir lieu le lendemain.

أكتب تنظيم منهجي يحسب المدة الزمنية للطيران بالساعة و الدقيقة ، إذا كنا نعلم كل من ساعة الإنطلاق و الوصول.

1-نأخذ بعين الإعتبار الإنطلاق و الوصول في نفس اليوم

2- بدون تحويل إلى الدقيقة

3- نفترض أن المدة الطيران أقل من 24 ساعة اكن الوصول يكون غدا.

1)

```
Action : DuréeVol1
Var : h1, h2, m1, m2, hr, mr : entiers
Début :  Ecrire (« entrer horaire de départ et d'arrivée »)
         Lire (h1, m1, h2, m2)
         mr <= [h2*60+m2] – [h1*60+m1]
         hr <= mr/60
         mr <= mr%60
Ecrire (« durée de vol : » , hr, mr)
Fin
```

Remarque : l'opération % (modulo) permet de calculer le reste de la division entière.

2)
```
Action : DuréeVol2
Var : h1, h2, hr, m1, m2, mr : entiers
Début :  Ecrire (« entrer horaire de départ et d'arrivée »)
         Lire (h1, m1, h2, m2)
         Si m2>m1 alors
                  hr <= h2-h1 et mr <= m2-m1
                  Ecrire (hr, mr)
         Sinon
                  hr <= h2-h1-1 et mr <= m2+60-m1
                  Ecrire (hr, mr)
Fin
```

3)
```
Action : DuréeVol3
Var : h1, h2, m1, m2, hr, mr : entiers
Début :  Ecrire (« entrer horaire de départ et d'arrivée »)
         Lire (h1, m1, h2, m2)
         Si h2>h1 alors
                  Si m2>m1 alors
                           hr <= h2-h1 et mr <= m2-m1
                           Ecrire (hr, mr)
                  Sinon
                           hr <= h2-h1-1 et mr <= m2+60-m1
                           Ecrire (hr, mr)
         Sinon
                  Si m2>m1 alors
                           hr <= h2-h1+24 et mr <= m2-m1
                           Ecrire (hr, mr)
                  Sinon
                           hr <= h2-h1+24-1 et mr <= m2+60-m1
                           Ecrire (hr, mr)
Fin
```

Exercice 9

1) Ecrire une fonction max3 qui retourne le maximum de trois entiers
2) Ecrire une fonction min3 qui retourne le minimum de trois entiers
3) Ecrire une fonction max2 qui retourne le maximum de deux entiers
4) Ecrire une fonction max3 qui retourne le maximum de trois entiers en faisant appel à max2

1)
```
Fonction : max3(a, b, c : entier) : entier :
Var : max3 : entier
Début :  Si a>b alors
                  Si a>c alors max3 <= a
                  Sinon max3 <= c
         Sinon
                  Si c>b alors max3 <= c
                  Sinon max3 <= b
Retourner (max3)
Fin
```

2)
```
Fonction : min3(a, b, c : entier ) : entier :
Var min3 : entier
Début
Retourner (–max3(-a, -b, -c))
Fin
```

3)
```
Fonction : max2 (a, b : entier) : entier
```

```
Var : max2 : entier
Début :  Si a<b alors max2 <= b
         Sinon max2 <= a
Retourner (max2)
Fin
```

4)

```
Fonction : max3 (a, b, c : entier) : entier :
Var : max3 : entier
Début :  max3 <= max2 [max2 (a, b), c)
Retourner (max3)
Fin
```

Exercice 10

Ecrire avec des Si Alors Sinon une action permettant la saisie d'une note n (0≤ n≤ 20) et qui affiche la mention (n≥ 16 : Très Bien, n ≥ 14 : Bien, n ≥ 12 : Assez Bien, n ≥ 10 : Passable, n ≥ 10 : Ajourné)

أكتب تنظيم منهجي يستعمل الحركة التناوبية حسب النقاط في مايلي.

```
Action : Mention
Var Note : réel
Début :  Ecrire (« saisissez une note »)
         Lire (Note)
         Si Note≥ 16 alors écrire (« TB »)
         Sinon
                  Si Note≥ 14 alors écrire (« B »)
                  Sinon
                           Si Note≥ 12 alors écrire (« AB »)
                           Sinon
                                    Si Note≥ 10 alors écrire (« Passable »)
                                    Sinon écrire (« ajourné »)
Fin
```

Alternative : écrire le même Algorithme avec des Selon Que :

```
Action : Note
Var : Note : réel
Selon que
Note ≥  16 écrire (« TB »)
Note ≥  14 écrire (« B »)
Note ≥  12 écrire (« AB »)
Note ≥  10 écrire (« Passable »)
Sinon écrire (« ajourné »)
```

Exercice 11

Soit l'algorithme suivant :

```
Action : Permis_voiture
Var : permis, voiture : booléen
Début :  Ecrire (« avez-vous le permis ? (0/1) »)
         Lire (permis)
         Ecrire (« avez vous une voiture ? (0/1) »)
         Lire (voiture)
Si non permis ou voiture alors
         Si voiture alors écrire (« conduisez moi à la gare »)
         Sinon écrire (« j'ai une voiture pas chère »)
Sinon
         Si voiture alors écrire (« vous êtes hors la loi »)
         Sinon écrire (« vive le vélo »)
fin
```

1) Ecrire l'arbre des conditionnelles
2) Corriger les tests pour que tous les cas soient couvert de manière cohérente

3) Faites correspondre les actions et les tests correctement
4) Si possible, écrire cet algorithme avec des selon que.

Permis et voiture	Permis	voiture	Ni permis ni voiture
Gare	Vive le vélo	Conduisez moi à la gare	Voiture pas chère
Conduisez moi à la gare	Voiture pas chère	Hors la loi	Vive le vélo

En clair, selon l'algorithme proposé : si l'on a le permis et la voiture on peut amener quelqu'un à la gare ; si l'on a que le permis on dit vive le vélo, si l'on n'a que la voiture on conduit aussi à la gare, enfin si l'on a ni permis ni voiture alors on achète une voiture pas chère. Le cas hors la loi n'est pas évoqué et les correspondance sont inexactes. Il faut évidemment avoir :

- permis et voiture : conduire à la gare
- permis : j'ai une voiture pas chère
- voiture : vous êtes hors la loi
- ni voiture, ni permis : vive le vélo

Correction de l'algorithme proposé :

```
Action : Permis_voiture
Var : permis, voiture : booléen
Début :  Ecrire (« avez-vous le permis ? (0/1) »)
         Lire (permis)
         Ecrire (« avez vous une voiture ? (0/1) »)
         Lire (voiture)
Si permis ou voiture alors
         Si voiture alors écrire (« conduisez moi à la gare »)
         Sinon écrire (« j'ai une voiture pas chère »)
Sinon
         Si voiture alors écrire (« vous êtes hors la loi »)
         Sinon écrire (« vive le vélo »)
```

On peut effectivement écrire cet algorithme avec des selon-que :

```
Action : permis_voiture
Var : permis voiture : réel
Début :  Ecrire (« avez-vous le permis ? (0/1) »)
         Lire (permis)
         Ecrire (« avez vous une voiture ? (0/1) »)
         Lire (voiture)
Selon que :
Permis et voiture : écrire (« conduisez moi à la gare »)
Permis et non voiture : écrire (« j'ai une voiture pas chère »)
Non permis et voiture : (« vous êtes hors la loi »)
Non permis et non voiture : (« vive le vélo »)
Fin
```

Exercice 12

Ecrire un Algorithme calculatrice permettant la saisie de deux entiers et une opération –booléen- (+, - , / , x) et affichant le résultat. Donner avant cela les spécifications, la solution en langage naturel, les structures de données.

أكتب تنظيم منهجي يقوم يستعمل العمليات الحسابية و يطبع النتيجة للعددين صحيحين.

Spécifications :
Données : 2 opérandes et un opérateur
Résultat : résultat de l'opération choisie

Solution en langage naturel : Saisie des données, envisager tous les cas : +, - , x, /. Attention à la division par zéro qui est impossible

Structure de données : 2 opérandes : des entiers
Un opérateur booléen : +, -, * , /

Algorithme :

Action : calcul

```
Var : a, b : réel  op : booléen
Début   Ecrire (« saisissez le premier entier »)
        Lire (a)
        Ecrire (« saisissez l'opérateur »)
        Lire (op)
        Ecrire (« saisissez la deuxième variable »)
        Lire (b)
Selon que :
Op = '+' : Ecrire (a+b)
Op = '*' : Ecrire (a*b)
Op = '/' :          Si b= 0  alors écrire (« division impossible »)
                    Sinon écrire (a/b)
Op = '-' : Ecrire (a-b)
Fin selon
Fin
```

Exercices en utilisant les structures répétitives TANT QUE et REPETER… JUSQU'A et POUR

Exercice 13

Ecrire le Algorithme qui affiche la somme d'une suite d'entiers saisie par l'utilisateur se terminant par zéro.

أكتب تنظيم منهجي يقوم بحساب و طبع متتالية صحيحة و ينتهي بي صفر

Exemple : l'utilisateur entre 1, puis 5, puis 2, puis 0 : affiche : 8

1) donner les spécifications
2) donner la solution en langage naturel
3) indiquer les structures de données
4) faites l'algorithme

Spécifications :

- données : suite de nombre entiers se terminant par zéro
- résultat : la somme de ces entiers

Solution en langage naturel : tant que l'entier saisi n'est pas zéro, l'ajouter à la somme partielle et saisir l'entier suivant.

Structure de données :

- entier : entier courant (saisi)
- entier : somme partielle

Algorithme :

```
Action : Somme Suite
Var : a, s : entiers
Début   s<=0                    Attention : dans une structure tant que ne pas oublier d'initialiser!!!
        Lire (a)
        Tant que a≠ 0 faire
        Début
                s<=s+a
                Lire (a)
        Fin
        Ecrire (s)
Fin
```

Exercice 14

Ecrire un algorithme qui affiche la moyenne d'une suite d'entiers se terminant par zéro (le zéro n'entrant pas en compte dans la moyenne : il est juste la pour indiquer la fin de saisie)

1) donner les spécifications
2) donner la solution en langage naturel
3) indiquer les structures de données
4) faites l'algorithme

Spécification :

- données : suite d'entier se terminant par zéro

- résultat : la moyenne de ces entiers (zéro exclu)

Solution en langage naturel :
Tant que l'entier saisi différent de 0 alors ajouter l'entier aux entiers précédents et faire la moyenne (c'est à dire diviser par le nombre d'entiers

Structures de données :
- entier : entier saisi
- entier : résultat moyenne

Algorithme :
```
Action : Moyenne
Var : n, moy, s : entiers
Début :  moy<=0
         s<=0
         Lire (n)
         Tant que n≠ 0 faire
         Début
                  Moy <= moy*s+n)/(s+1)
                  s<=s+1
                  lire (n)
         fin
         Ecrire (moy)
Fin
```

Exercice 15

Ecrire un algorithme permettant la saisie d'une suite d'entiers se terminant par zéro et vérifier si cette suite contient deux entiers consécutifs égaux en utilisant les structures tant que.

1) donner les spécifications
2) donner la solution en langage naturel
3) indiquer les structures de données
4) faites l'algorithme

Spécifications :
- données : suite d'entier se terminant par zéro
- résultat : vrai si deux entiers consécutifs, faux sinon.

Solution en langage naturel : comparer l'entier courant et le précédent. Et tant que ils sont différents, on continu la lecture et tant que l'entier courant est différent de zéro.

Structures de données :
- entier : nombre courant
- entier : nombre précédent

Algorithme :
```
Action : Entiers consécutifs
Var : nc, np : entier
{on désignera par nc le nombre courant et np le nombre précédent}
Début    Lire (nc)
         np<=nc-1
{pour être sur que le nombre courant ne sera pas le même que le nombre précédent dès le départ on affecte la valeur nc-1 au nombre précédent. On aurait tout aussi bien pu lui donner la valeur zéro)
         Tant que nc≠ 0 et np ≠ nc faire
         Début
                  np<=nc
                  lire (nc)
         fin
         Si nc≠ 0 alors écrire (« oui »)
         Sinon écrire (« non »)
Fin
```

Refaire le même algorithme en utilisant une structure répéter jusqu'à

```
Action : Entiers consécutifs
Var : nc, np : entiers
Début
Lire (nc)
Si nc ≠ 0 alors    Répéter
                   Début
                   np <= nc
                   lire (nc)
                   jusqu'à (nc=np ou nc=0)
Si nc=0 alors écrire (« oui »)
Sinon écrire (« non »)
Fin
```

Exercice 16

Ecrire un algorithme qui affiche le maximum d'une suite se terminant par zéro

1) donner les spécifications
2) donner la solution en langage naturel
3) indiquer les structures de données
4) faites l'algorithme

Spécifications :

- données : une suite d'entiers se terminant par zéro
- résultat : un entier : le maximum de cette suite

Solution en langage naturel : comparer l'entier courant avec le maximum et tant que nc<max on continue, sinon on affiche la résultat et on continue, et tant que nc ≠ 0

Structures de données

- n : entier courant (saisi)
- max : entier max de la suite

Algorithme :

```
Action : max suite
Var : n, max : entiers
Début   Lire (n)
        Max<=n
        Tant que n≠ 0 faire
        Début
                Lire (n)
                Si max<n alors max<=n
        Fin
        Ecrire (max)
Fin
```

Exercice 17

Ecrire un programme mettant en œuvre le jeu suivant :

Le premier utilisateur saisi un entier que le second doit deviner. Pour cela, il a le droit à autant de tentatives qu'il souhaite. A chaque échec, le programme lui indique si l'entier cherché est plus grand ou plus petit que sa proposition. Un score indiquant le nombre de coups joués est mis à jour et affiché lorsque l'entier est trouvé.

1) donner les spécifications
2) donner la solution en langage naturel
3) indiquer les structures de données
4) faites l'algorithme

Spécifications :

- données : nombre entier
- résultat : nombre de tentatives

Solution en langage naturel : saisir un nombre entier par le premier joueur. Tant que le joueur 2 n≠ saisie, dire si n est > ou < à nombre cherché, incrémenter de 1 et continuer. Quand le résultat est trouvé, afficher le nombre de tentatives.

Structures de données :

- a : nombre saisi par l'utilisateur 1
- n : nombre saisi par l'utilisateur 2
- t : tentatives

```
Algorithme :
Action : devinette
Var : a, n, t : entiers
Début :  Lire (a)
         Lire (n)
         t=0
         Tant que a≠ n faire
         Début
                  Si n>a alors écrire (« nombre cherché plus petit « )
                  Sinon écrire (« nombre cherché plus grand »)
                  t<=t+1
                  lire (n)
         fin
         écrire (t+1)
fin
```

Exercice 18

Ecrire un algorithme permettant de calculer le PGCD de deux nombres en utilisant l'astuce suivante : soustraite le plus petit des deux entiers du plus grand jusqu'à ce qu'ils soient égaux
Ecrire le même programme en utilisant l'algorithme d'Euclide : d'une part en utilisant uniquement les structures TANT QUE, d'autre part en utilisant uniquement les structures REPETER JUSQU'A.

```
Action : PGCD
Var : a, b entiers
Lire (a, b)
Début
         a = ValAbs (a)
         b = ValAbs (b)
         Répéter
                  Selon que
                  a>b a<=a-b
                  a<b b<=b-a
         jusqu'a a=b
         écrire (a)
Fin
```

Même programme avec Euclide et des TANT QUE :

```
Action : PGCD
Var : a, b, r entiers
Lire (a, b)
r<=a%b
tant que r≠ 0 faire
         début
         a<=b
         b<=r
         r<=a%b
         fin
écrire (b)
fin
```

Même programme avec Euclide et des REPETER JUSQU'A :

```
Action : PGCD
Var : a, b, r entiers
```

```
Lire (a, b)
Répéter  r<=a%b
         a<=b
         b<=r
jusqu'à r=0
écrire (b)
fin
```

Exercice 19

Ecrire avec la commande POUR un algorithme qui permet de faire la somme d'une suite de nombre entrée par l'utilisateur. Faire la même chose en comptant par pas de –1.

```
Action :somme_nombre
Var : k, nb, n, somme : entier
Début :
        Somme <= 0
        Ecrire (« combien voulez-vous entrer de nombres ? »)
        Lire (nb)
        Pour k de 1 à nb faire
                Début
                Lire (n)
                Somme<=somme + n
                Fin
        Ecrire (somme)
Fin
```

```
Même programme par pas de –1 :
Action : somme_entier
Var : k, nb, n, somme : entiers
Début :
        Somme<=0
        Ecrire (« combien voulez-vous entrer de nombres ? »
        Lire (nb)
        Pour k de nb à 1 par pas de –1 faire
                Début
                Lire (n)
                Somme<=somme + n
                Fin
        Ecrire (somme)
Fin
```

Exercice 20

Traduire le POUR de l'algorithme suivant en REPETER JUSQU'A :

```
Action : bidon
Var : k, nb : entiers
Début
        Lire (nb)
        Pour k de 1 à nb faire
                Ecrire (k)
Fin
```

```
Action : Bidon
Var : k, nb : entier
Début
        Lire (nb)
        K<=1
        Si nb>0 alors
                Répéter écrire (k)
                K<=k+1
                Jusqu'à k>nb
Fin
```

Exercice 21
Ecrire une fonction qui fait la somme des entiers compris dans un intervalle.

```
Fonction : intervalle (a, b ; entiers) : entier
Var : k, somme : entier
Début
        Somme <= 0
        Pour k de a à b faire
        Somme<=somme + k
        Retourner (somme)
Fin
```

Exercice 22
Ecrire une fonction multiplication de a et b par addition successives.

```
Fonction : multiplication (a, b : entiers) : entier
Var : k, produit : entiers
Début
        produit<=0
        Pour k de 1 à a faire
        Produit<=produit + b
        Retourner (produit)
Fin
```

exercices sur les Tableaux

Exercice 23
Ecrire une action qui permette la saisie d'un tableau croissant : si T[k]<T[k+1] on enregistre, si T[k]>T[k+1] on redemande la saisie d'un nombre plus grand

```
Const : MAX=100
Ttype : Ttab=tableau [max]d'entier

Action : saisie_tableau_croissant
Var : tab : Ttab,  i : entier
Début
Lire (Tab[0])
        Pour i de 1 à MAX-1 faire
                Répéter lire (tab[i])
                jusqu'à tab[i] ≥  tab[i-1]
Fin
```

Exercice 24
Ecrire une fonction retournant le maximum d'un tableau de taille n.
Faire le même algorithme mais qui ne retourne que l'indice de la case du tableau contenant le maximum du tableau.

```
Fonction : maximum (tab : Tableau d'entier n :entier) : entier
Var : max, i : entiers
Début
        Max <= tab[0]
        Pour i de 1 à n-1 faire
                Si tab[i]>max alors max<=tab[i]
        Retourner (max)
Fin
```

```
Fonction : maximum (tab : Tableau d'entier n :entier) : entier
Var : indice, i, max : entiers
Début
```

```
        Max<=tab[0]
        Indice <=0
        Pour i de 1 à n-1 faire
                Si tab[i]>max alors
                        max<=tab[i]
                        indice<=i
        Retourner (indice)
Fin
```

Exercices généraux sur les actions paramétrées

Exercice 25

Ecrire une fonction Afficher qui affiche a l'écran le contenu d'un tableau. Ecrire aussi l'action principale qui permettra de comprendre comment fonctionne cette fonction afficher.

```
{Ne pas oublier d'indiquer les paramètres du tableau !}
Const : MAX : entier=100
Type : Ttab : Tableau [MAX] d'entier

Fonction Afficher (tab : tableau d'entiers, n entiers)
Var : i entier
Début :
        Pour i de 0 à n-1
        Ecrire (tab[i], «  »)
Fin

Action principale
Var t1 t2 : Ttab
Début
        T1[0]<=1
        T1[1]<=3
        T2[0]<=4
        T2[1]<=5
        T2[2]<=7
        Afficher (T1, 2)
        Afficher (T2, 3)
Fin
```

Résultat à l'écran :
1 3
4 5 7

Exercice 26

Ecrire une fonction qui permet la saisie d'un tableau. Faite aussi l'action principale qui permettra d'accéder a cette fonction saisie mais aussi d'afficher dans un second temps le résultat

```
Fonction : saisie (Stab : tableau d'entiers, N :entier)
Var : i entier
Début :
        Pour i de 0 à n-1 faire
                Lire (tab[i])
Fin

Action principale
Var : tabl : Ttab
Début
        Saisie (toto, 10)
        Afficher (toto, 10)
Fin
```

Ou afficher est la fonction de l'exercice 1.

Exercice 27

Ecrire une fonction qui calcule le nombre d'inversion d'un tableau de taille n (c'est à dire i<j et tab[i]>tab[j] pour tout i et j.)

```
Fonction inversion (tab : tableau d'entiers, N entier)
Var : j, C, i entiers
Début
        C<=0
        Pour i de 0 à n-2 faire
        Début
                Pour j de i+1 à n-1 faire
                Si tab[i]>tab[j] alors C<=C+1
        Fin
        Retourner ( C )
Fin
```

Exercice 28

Ecrire une action qui affiche les n premiers éléments de la suite définie par $u_0=1$ et u_{n+1}=somme de k=0 jusqu'à n de (u_k*u_{n-k})

Aide : stocker les éléments dans un tableau toto avec toto[0]=1. Puis on utilise une boucle imbriquée pour calculer toto[n+1]=somme k=0 à k=n de toto[k]*toto[n-k].

```
Action Suite (E : d :entier)
Var : toto : Ttab, i, k : entiers
Début :
        Toto[0]<=1
        Pour I de 1 à d-1 faire
        Toto[i]<=0
        Pour k de 0 à n-1 faire
        Toto[i]<=toto[i]+toto[k]+toto[i-1-k]
        Afficher (toto, d)
Fin
```

Exercice 29

Voyons maintenant quelques exercices rudimentaires de changements dans un tableau

Ecrire une action permettant de remplacer toutes les occurrences de x par y dans un tableau de taille n.

Ecrire un algorithme qui échange les valeurs des cases i et j dans un tableau.

Ecrire un programme qui inverse un tableau. (exemple : 1 5 6 7 3 devient 3 7 6 5 1)

```
Action : Remplacer (E : x : entier, E : y : entier, ES tab : tableau d'entiers, E : n : entier)
Var : i :entier
Début
Pour i de 0 à n-1 faire
        Si tab[i]=x alors tab[i]<=y
Fin
```

```
Action : Echanger (E : i : entier, E : j : entier, ES : tab : tableau d'entier, E : n :entier)
Var : temp
Début
        Si i<n ET j<n alors
                Temp<=tab[i]
                Tab[I]<=tab[j]
                Tab[j]<=temp
Fin
```

```
Action : inverser (ES : tab : tableau d'entiers, E : n : entier)
```

```
Var : i :entier
Début
        Pour i de 0 à n/2 – 1 faire
        Echanger (i, n-1-in tab, n)
        {ou Echanger est la deuxième action de cet exercice}

Fin
```

Tableaux triés et découpages fonctionnels

Exercice 30

Le but de l'exercice est de créer une action de saisie de tableau, qui trie, au fur et à mesure des entrées, les valeurs par ordre croissant dans le tableau.

Exemple :

Soit le tableau suivant :

0	1	2	3
2	5	7	9

Comment insérer 6 dans le tableau trié (en supposant qu'il n'y a pas de doublon dans le tableau) ?

- je cherche la bonne position (ici : la case d'indice 2)
- décalage à droite si nécessaire :

0	1	2	3	4
2	5	7	7	9

- Insertion de l'élément

0	1	2	3	4
2	5	6	7	9

On a donc ici le découpage fonctionnel :

On va donc créer une fonction IndiceEltSup qui cherche la bonne position, une action Insérer qui inclue le nombre entré dans la bonne case du tableau, et une action DécalageDroite qui décale comme dans l'exemple toutes les cases d'un rang vers la droite si nécessaire.

```
Const MAX=100
Type TtabVar = entité (tab : tableau[MAX] d'entiers, taille : entier)

Fonction IndiceEltSup (tvt : TtabVar, entier, n : entier) : entier
Var : i : entier
Début
        Tant que (i≤ tvt.taille ET tvt.tab[i]<n)
        i<=i+1
        retourner (i)
Fin

Action DécalageDroite (ES : tvt : TtabVar, E : i : entier)
Var : j : entier
Début
        Pour j de tvt.taille – 1 à i par pas de –1 faire
        Tvt.tab[j+1]<=tvt.tab[j]
        Tvt.taille++
Fin

Action Insérer (ES : tvt : TtabVar, E : i : entier, E : i : entier)
Début
        DécalageDroite (tvt, i)
        Tvt.tab[i]<=i
Fin

Action SaisieTrié (S : tvt : TtabVar)
Var : rep : chaîne, nb : entier, i : entier
Début
        Tvt.taille<=0
        Répéter
                Ecrire (Rentrer encore un entier ?)
```

```
                Lire (rep)
                Si rep ≠ « non » alors
                        Lire (nb)
                        I<=IndiceEltSup(tvt, nb)
                        Si non(i<tvt.taille ET tvt.tab[i]=nb)
                        Insérer (tvt, i, nb)
        Jusqu'à rep= « non »  ou tvt.taille=MAX
Fin
```

Exercice 31

Faire un algorithme qui fait une recherche dichotomique dans un tableau trié. On pourra utiliser les fonctions de l'exercice précédent.

Nous allons créer une action qui définie la zone de recherche, puis l'action RechercheDicho qui opérera la recherche dichotomique dans l'intervalle définie par la zone de recherche.

```
Action ZoneRecherche (E : tvt : TtabVar, E : n : entier, ES : Binf : entier, ES : Bsup : entier)
Var : milieu : entier
Début
        Milieu <= (Binf + Bsup)/2
        Si tvt.tab[milieu]=n alors
                Début
                Binf<=milieu
                Bsup<=milieu
                Fin
        Sinon
                Si tvt.tab[milieu]>n alors Bsup<=milieu –1
                Sinon Binf<=milieu+1
Fin

Fonction RechercheDicho (E : tvt : TtabVar, E : n : entier)
Var : Binf, Bsup : entiers
Début
        Binf<=0
        Bsup<=tvt.taille –1
        Tant que Bsup>Binf faire
                ZoneRecherche (tvt, n, Binf, Bsup)
                Si Bsup=Binf alors
                        Retourner (Binf)
                Sinon retourner ( -1)
```

Exercice 32

Faire un algorithme qui supprime une valeur dans un tableau trié. On pourra utiliser des fonctions des deux exercices précédents.

Le but est d'utiliser la recherche dichotomique de l'exercice précédent pour trouver dans le tableau l'indice de la valeur que l'on veut supprimer puis faire un décalage à gauche pour remettre en place les valeurs (sans qu'il y ait de vide dans une case du tableau)

```
Action Supprimer (ES : tvt : TtabVar, E : n : entier)
Var : i : entier
Début
        i<=RechercheDicho(tvt, n)
        Si i≠ -1 alors
        DecalageGauche (tvt, i)
Fin

Action DecalageGauche (Es : tvt : TtabVar, E : i : entier)
Var : j: entier
Début
```

```
        Pour j de i+1 à tvt.taille –1 faire
        Tvt.tab[j –1] <= tvt.tab[j]
        Tvt.taille <= tvt.taille –1
    Fin
```