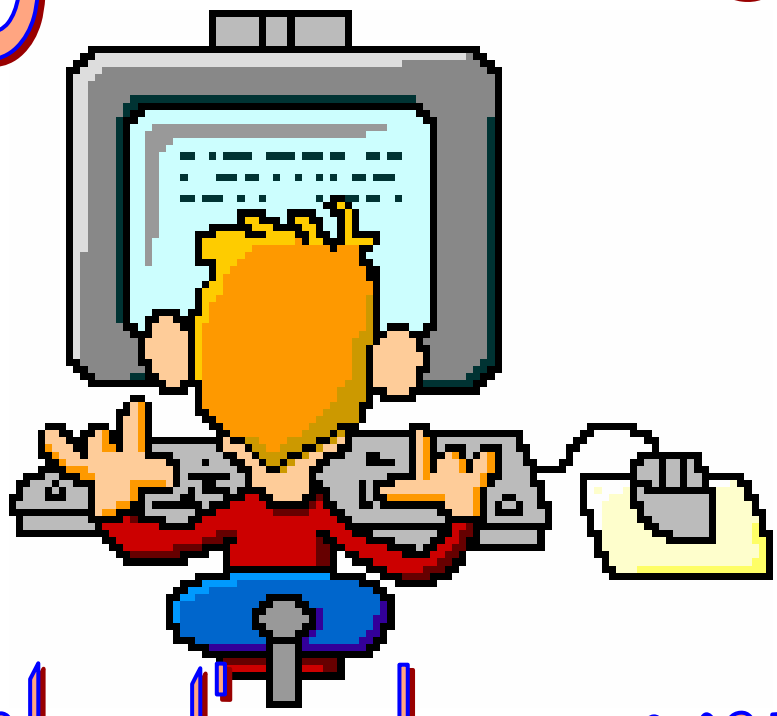
Algorithmes



Et structuration de programmes

Sommaire

1. Les structures de base d'un langage de programmation

Pour qu'un ordinateur fonctionne, il est nécessaire de lui dire **quoi faire**. Toute action réalisée par une machine a été programmée par un être humain (du moins pour l'instant); un ordinateur ne décide de rien, il fait "bêtement" ce qu'il lui a été programmé.

Mais qu'est-ce que programmer ?
C'est écrire une série d'actions élémentaires compréhensibles par le "cerveau" de la machine, cette succession permettant de réaliser une action plus compliquée.
Chacune de ces actions plus compliquées, étant connue de la machine, peut être utilisée comme les actions élémentaires du départ pour construire des actions encore plus complexes.

La machine a son propre langage appelé langage machine. Il serait trop compliqué d'écrire directement les programmes en langage dit de bas niveau. Nous utilisons donc des langages dits "évolués" compréhensibles pour un initié. Ce langage sera ensuite traduit en langage machine. Malgré que les langages soient de plus en plus proches du langage humain, ils ne sont pas directement lisibles. C'est pourquoi, dans ce qui suit, nous allons utiliser un pseudo-langage, comportant toutes les structures de base d'un langage de programmation. Il suffira ensuite de traduire notre "pseudo" en langage évolué en fonction des possibilités de ce langage. Par exemple, le langage Java permet plus de type d'actions qu'un langage tel que le Cobol

Un programme est donc une suite d'instructions exécutées par la machine.

Ces instructions peuvent :

- soit s'enchaîner les unes après les autres, on parle alors **de séquence d'instructions**;
- ou bien s'exécuter dans certains cas et pas dans d'autres, on parle alors de **structure alternative**;
- ou se répéter plusieurs fois, on parle alors de **structure répétitive**.

1.1. La séquence d'instructions

Une instruction est une action que l'ordinateur est capable d'exécuter.
Chaque langage de programmation fournit une liste des instructions qui sont implémentées et que l'on peut donc utiliser sans les réécrire en détail.
Structurer un programme s'effectue que le programme soit écrit en C, en Java, en Visual Basic. Dans un premier temps, nous écrirons nos programmes en utilisant un pseudo-langage que nous pourrons traduire ensuite dans l'un des langages de programmation.

Pour illustrer notre propos, prenons l'exemple du déroulement de la journée de Buggy.

Une séquence d'instruction serait :

```
Se lever
Prendre sa douche
Prendre le petit déjeuner
S'habiller
```

Vous voyez que l'ordre des instructions a de l'importance : "S'habiller" puis "prendre sa douche" conduit à un résultat pas génial que nous appellerons un "bug". Cependant certaines instructions peuvent se dérouler dans un ordre indifférent: "prendre sa douche" et "prendre son petit déjeuner" peuvent être inversés sans préjudice pour le résultat.

Une alternative s'exprime par si ..... sinon.....

```
Si fin de semaine ou congé
        Se lever
        Prendre son petit déjeuner
        Prendre sa douche
        Mettre sa tenue de sport
        Faire son jogging
Sinon
        Se lever
        Prendre son petit déjeuner
        Prendre sa douche
        Mettre sa tenue de travail
        Aller travailler
```

Que la condition soit réalisée (**condition vraie**) ou qu'elle ne le soit pas (**condition fausse**) les premières actions sont les mêmes et se passent dans le même ordre ce qui permet la simplification suivante :

```
Se lever
Prendre son petit déjeuner
Prendre sa douche
Si fin de semaine ou congé
        Mettre sa tenue de sport
        Faire son jogging
Sinon
        Mettre sa tenue de travail
        Aller travailler
```

Remarquez que les actions si vrai ou si faux sont décalées par rapport aux autres instructions afin de permettre une meilleure lisibilité; on parle d'**indentation**.

Pour illustrer l'itérative ou répétitive (les deux termes sont équivalents), continuons notre exemple en supposant que Buggy dont nous vivons la journée palpitante, soit un employé de banque. La routine journalière de cet employé est :

```
Ouvrir guichet
Appeler premier client
Tant que client dans file d'attente et pas fin de journée
        Traiter client
        Appeler client suivant
FinTantQue
```

Les deux actions "Traiter client" et "Appeler client suivant" vont se répéter tant que la condition située derrière l'instruction "Tant que" est vérifiée.

Considérons maintenant le programme complet de la journée de Buggy

```
1.  Se lever
2.  Prendre son petit déjeuner
3.  Prendre sa douche
4.  Si fin de semaine ou congé
5.      Mettre sa tenue de sport
6.      Faire son jogging
7.      Passer une journée de détente
8.  Sinon
9.      Mettre sa tenue de travail
10.     Aller travailler
11.     Faire travail
12. FinSi
13. Rentrer à la maison
14. Dîner
15. Aller se coucher
16.
17. Fonction travail
18.     Ouvrir guichet
19.     Appeler premier client
20.     Tant que pas heure de déjeuner
21.         Faire guichet
22.     FinTantQue
23.     Déjeuner
24.     Tant que client et pas heure de sortie
25.         Faire guichet
26.     FinTantQue
27. Fin Fonction
28.
29. Fonction Guichet
30.         Si client en file d'attente
31.             Traiter client
32.             Appeler client suivant
33.         Sinon
34.             Classer
35.         FinSi
36.     Fin Fonction
```

Remarques : Faire travail étant une instruction assez complexe, elle a été détaillée dans la **fonction** Travail pour une meilleure lisibilité du programme. De même, nous avons créé une fonction guichet afin de ne pas répéter la même séquence d'instructions deux fois dans le programme.

Notre programme a donc été scindé en deux parties :

- ☞ le corps du programme de la ligne 1 à la ligne 15
- ☞ les fonctions ou sous-programmes internes à partir de la ligne 17.

Comment cela se passe-t-il lorsque nous rencontrons un appel de fonction ?
A la ligne 11 nous avons "Faire travail" qui indique à la machine qu'elle doit aller en ligne 17 qui correspond au début de la fonction appelée. La procédure fait les actions des lignes 18, 19 et 20. Elle trouve à nouveau un appel de fonction, cette fois-ci "Faire guichet" donc elle se débranche vers la ligne 29 et exécute les instructions jusqu'à la ligne 36 où se trouve "Fin fonction" se qui ramène la machine à l'instruction se situant juste après " Faire guichet" c'est à dire ligne 22. puis les actions des lignes 20, 21, 22 vont se répéter n fois jusqu'à l'heure de déjeuner. Ensuite, la ligne 23 est exécutée et à nouveau il y aura répétition de faire guichet (avec débranchement à la ligne 29 et retour à la ligne 26. Lorsque nous arrivons sur la ligne 27 "Fin fonction", nous retournons à la ligne 12.

Dans notre pseudo-langage, nous n'aurons que la liste minimum d'instructions, nécessaire et suffisante pour les programmes que nous aurons à écrire.

1.2. L'affectation

```
Variable ⇐ Valeur
```

Ce qui se lit "variable reçoit valeur" et qui signifie que nous mémorisons la valeur à un endroit nommé variable. Nous pourrions aussi dire que nous rangeons une valeur dans des cases de la mémoire que nous nommons variable.
Par exemple :

```
i ⇐ 1
Termine ⇐ VRAI*
```

Il ne faut jamais confondre valeur et variable. Une variable est caractérisée par :

- une adresse c'est à dire un emplacement dans la mémoire de la machine,
- un type permettant d'indiquer la nature de l'information contenue,
- éventuellement une longueur si le type ne le définit pas.

Quand nous aurons affaire à une variable numérique, nous écrirons

```
nCompteur ⇐ 0
nSalaire ⇐ nSalaireBase
```

Dans le premier exemple, nous envoyons la constante 0 dans une variable nommée nCompteur.
Dans le deuxième exemple, nous envoyons le contenu de la variable nSalaireBase comme contenu de la variable nSalaire.

Quand nous sommes en présence d'une variable alphanumérique, nous écrirons

```
strMessage ⇐"Bonjour"
strMesage ⇐ Bonjour
```

Dans le premier cas, nous envoyons la constante Bonjour dans une variable nommée strMessage.
"**AFFICHER** strMessage" donnera comme résultat l'affichage du mot Bonjour.
Dans le deuxième exemple, nous envoyons le contenu de la variable nommée Bonjour qui peut contenir **BYE** comme contenu de la variable strMessage.
"**AFFICHER** strMessage" donnera comme résultat l'affichage du mot **BYE**.

1.2.1. les opérations

Les opérations arithmétiques courantes, addition, soustraction, multiplication et division s'écrivent ainsi :

```
variable  ⇐  val1 + val2
variable  ⇐  val1 - val2
variable  ⇐  val1 * val2
variable  ⇐  val1 / val2
```

On doit toujours affecter le résultat d'une opération dans une variable.

1.2.2. le dialogue avec l'utilisateur

Pour permettre au programme de dialoguer avec l'utilisateur, c'est à dire d'afficher un résultat à l'écran et de lire une entrée au clavier, il faut au moins deux instructions une pour lire, l'autre pour afficher.

Dans le pseudo-langage, elles s'écrivent ainsi :

```
LIRE Variable
AFFICHER Texte Variable Variable Texte ...
```

La première lit tous les caractères qui sont saisis au clavier, jusqu'à ce que l'utilisateur appuie sur la touche entrée, et range le résultat dans la variable.

La seconde affiche sur l'écran le ou les textes et la valeur des variables.

exemple :

```
AFFICHER "Quel est ton nom ? "
LIRE nom_utilisateur
AFFICHER "Ton nom est : " nom_utilisateur ". Le mien est TOTO"
```

1.3. La structure alternative

Il est souvent nécessaire lorsque nous écrivons un programme de distinguer plusieurs cas conditionnant l'exécution de telles ou telles instructions. Pour ce faire, on utilise une structure alternative : si on est dans tel cas, alors on fait cela sinon on fait ceci.

La syntaxe de cette instruction est la suivante :

```
SI condition ALORS
        actions  si vrai
[ SINON
        actions si faux]
FINSI
```

Les crochets signifient que la partie "**sinon actions**" est facultative.

```
SI condition ALORS
        actions  si vrai
FINSI
```

Exemples :
Pour avoir la valeur absolue (appelée nAbsolu dans notre exemple) d'un nombre (n1 dans notre exemple), s'il est négatif nous le multiplions par –1 sinon la valeur absolue égale le nombre :

```
SI (n 1 < 0)  ALORS
        nAbsolu ⇐ n1 * (-1)
SINON
        nAbsolu ⇐  n1
FINSI
```

Ne faire la division que si le diviseur n2 n'est pas égal à zéro :

```
SI (n2 <> 0) ALORS
        nResultat ⇐ n3 / n2
FINSI
```

1.3.1. Les conditions

Pour exprimer les conditions on utilise les opérateurs conditionnels suivants :

=	égal
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal
<>	différent

```
(n1 < 1)
(strToto <> strTiti)
```

On peut combiner des conditions à l'aide des opérateurs logiques suivants : ET OU NON XOR(ou exclusif)

```
((n1<2) ET ((n2 = 0) OU (n3 <>n1)) XOR (n4 = 1))
```

Lorsque l'on écrit de telles conditions, il est recommandé de mettre toutes les parenthèses afin d'éviter les erreurs car les opérateurs ont une hiérarchie qui ne collera pas nécessairement avec votre logique.

1.3.2. Les actions

Les actions qui suivent **sinon** ou **alors** peuvent être :

- ☞ - une simple instruction
- ☞ - une suite d'instructions séparées par des ;
- ☞ - une autre alternative
- ☞ - une répétitive

1.4. La structure répétitive

Un programme a presque toujours pour rôle de répéter la même action un certain nombre de fois. Pour ce faire, nous utiliserons une structure permettant de dire " Exécute telles actions jusqu'à ce que telle condition soit remplie".

Bien qu'une seule soit nécessaire, la plupart des langages de programmation proposent trois types de structure répétitive.
Voici celles de notre pseudo-langage.

1.4.1. TantQue

```
TantQue condition
        actions
FinTQ
```

Ce qui signifie : tant que la condition est vraie, on exécute les actions.

exemple :

```
bSuite ⇐ Vrai;
TantQue (bSuite)
        n1 ⇐ n1+1;
        SI (strFin = "oui")
        ALORS bFini ⇐ faux
        FinSi
FinTQ
```

1.4.2. Faire Jusqu'à

```
FAIRE
        actions
JUSQUA condition
```

Ce qui signifie que l'on exécute les actions jusqu'à ce que la condition soit vraie.
exemple :

```
bFini ⇐-FAUX;
Faire
        n1 ⇐ n1+1;
        SI (n2 / n1 < nEpsilon)
                ALORS bFini ⇐ VRAI
        SINON  n2  ⇐ n2 / n1
        FINSI
Jusqua (bFiini  =  VRAI);
```

1.4.3. Pour

Très souvent, nous utilisons une structure répétitive avec un compteur et nous arrêtons lorsque le compteur a atteint sa valeur finale.

```
n1 ⇐ 1
TantQue (n1  <  10)
        nFactoriel ⇐ nFactoriel  *  n1
        n1 ⇐ n1 + 1
FINTQ
```

C'est pourquoi la plupart des langages de programmation offrent une structure permettant d'écrire cette répétitive plus simplement. Dans le pseudo-langage c'est la structure **pour** :

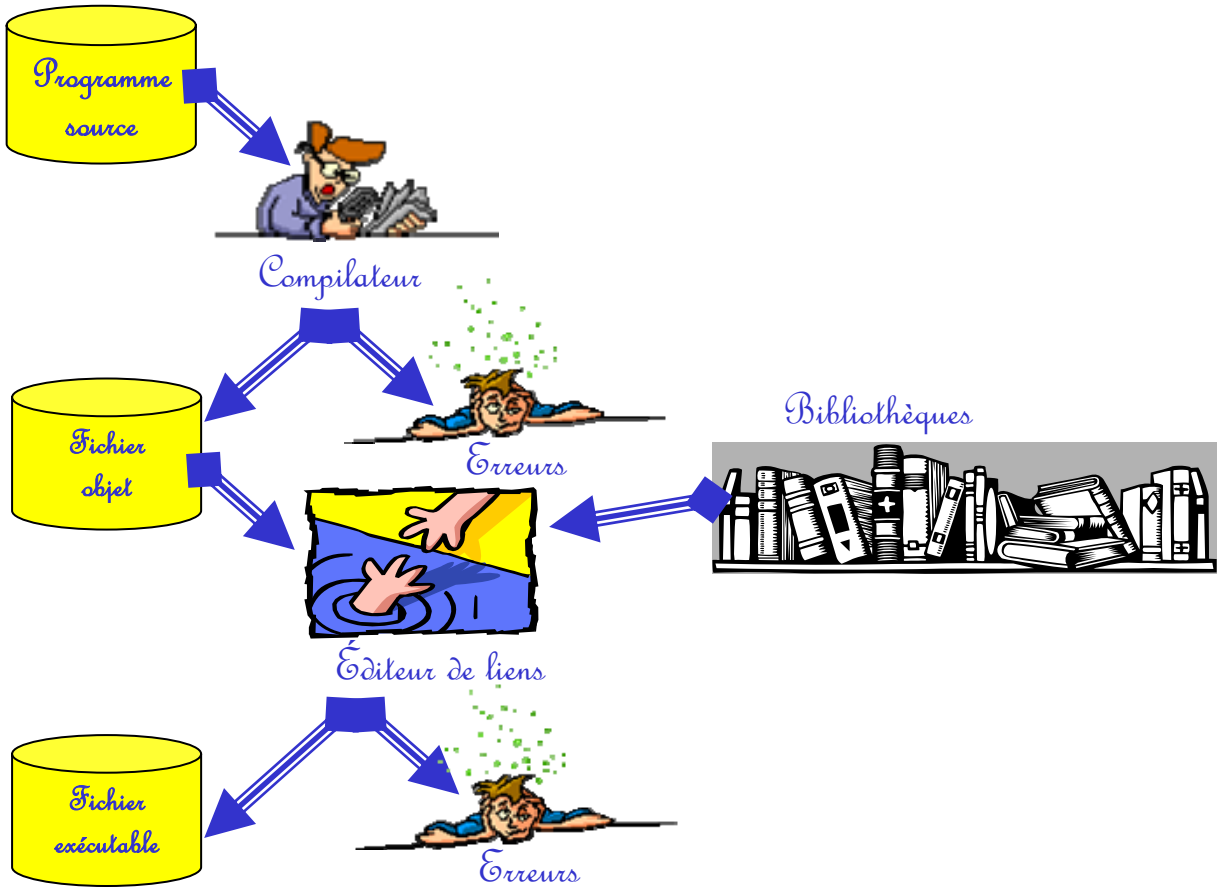
```
POUR variable ALLANT DE valeur initiale A valeur finale  [PAS valeur du pas]
        actions
FinPour
```

Lorsque le **PAS** est omis, il est supposé égal à +1.

```
POUR  n1  ALLANT DE  1  A  10
        nFactoriel ⇐ nFactoriel  *  n1
FinPour
```

1.5. La compilation

Un langage de programmation sert à écrire des programmes de manière à les exécuter. Des outils permettent de traduite le langage écrit par le programmeur en langage machine.
Ils fonctionnent de la manière suivante :

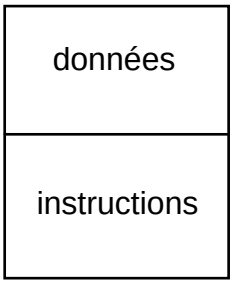


Le compilateur analyse le langage source afin de vérifier la syntaxe et de générer un fichier objet en langage intermédiaire assez proche du langage machine. Tant qu'il y a des erreurs de syntaxe, le compilateur est incapable de générer le fichier objet.
Souvent, on utilise dans un programme des fonctions qui soit ont été écrites par quelqu'un d'autre soit sont fournies dans une bibliothèque (graphique par exemple). Dans ce cas, le compilateur ne les connaît pas et ne peut donc pas générer le langage intermédiaire correspondant.

C'est le travail de l'éditeur de liens que d'aller résoudre les références non résolues. C'est à dire que lorsqu'il est fait appel dans le fichier objet à des fonctions ou des variables externes, l'éditeur de liens recherche les objets ou bibliothèques concernés et génère l'exécutable. Il se produit une erreur lorsque l'éditeur de liens ne trouve pas ces références.

1.6. La déclaration des variables

Un programme exécutable est composé de deux parties :



La partie instructions contient les instructions à exécuter.
La partie données contient toutes les variables utilisées par le programme.
Un programme exécutable est chargé dans la mémoire centrale de l'ordinateur, les valeurs que l'on a affectées aux variables doivent être conservées tout le temps du déroulement du programme. Par conséquent, il faut que le programme soit capable de réserver la place nécessaire aux variables.
Pour ce faire, les variables doivent être déclarées afin que le compilateur sache quelle place elles vont occuper.

1.6.1. Les types de variables

Les variables que l'on utilise dans les programmes ne sont pas toutes de même nature, il y a des nombres, des caractères, ... On dit que les variables sont typées.
Il est nécessaire de donner un type aux variables, car cela permet d'une part de contrôler leur utilisation (ex: on ne peut pas diviser un caractère par un entier) et d'autre part car cela indique quelle place il faut réserver pour la variable.
Généralement les langages de programmation offrent les types suivants :

entier : il s'agit des variables destinées à contenir un nombre entier positif ou négatif.
Dans notre pseudo-langage, nous écrirons la déclaration des variables de type entier :

```
ENTIER variable, variable, ... ;
```

Généralement un entier occupe 2 octets, ce qui limite les valeurs de -32768 à +32768.
Cependant cela dépend des machines, des compilateurs, et des langages.

Certains langages distinguent les entiers courts (1 octet), les entiers longs (4 octets) et les entiers simples (2 octets).

réel : il s'agit des variables numériques qui ne sont pas des entiers, c'est à dire qui comportent des décimales. Généralement un nombre réel est codé sur 4 octets (voir cours sur le codage des informations).

Dans notre pseudo-langage, la déclaration des variables de type réel est la suivante :

```
REEL variable, variable, ... ;
```

caractère : Les variables de type caractère contiennent des caractères alphabétiques ou numériques (de 0 à 9), mais dans ce cas ils ne sont pas considérés comme étant des nombres et on ne peut pas faire d'opérations dessus.
Un caractère occupe un octet.
Dans notre pseudo-langage, une variable de type caractère se déclare ainsi :

```
CAR variable , variable , ...;
```

Remarque : les chaînes de caractères, dans notre langage sont des tableaux de caractères (voir 1.6.2)

booléen : Il est souvent nécessaire lorsque l'on écrit un programme d'introduire des variables qui prennent les valeurs **vrai** ou **faux** ou les valeurs **oui** ou **non**.

Pour cela, il existe un type particulier dont les variables ne peuvent prendre que 2 valeurs : **vrai** ou **faux**.. Dans notre pseudo-langage, la déclaration s'écrit :

```
BOOLEEN variable, variable, ... ;
```

1.6.2. les tableaux

On peut regrouper plusieurs variables sous un même nom, chacune étant alors repérée par un numéro. C'est ce que l'on appelle un tableau. On peut faire un tableau avec des variables de n'importe quel type.
Dans tous les cas le ième élément d'un tableau appelé tab sera adressé par tab(i).
Généralement on fait des tableaux à une dimension, mais il existe également des tableaux à deux dimensions, dans ce cas tab(i,j) représente la jème colonne et la ième ligne.

```
TABLEAU type  variable [ longueur ] ;
```

exemple :

```
TABLEAU CAR mot[10];
TABLEAU ENTIER liste_nb[25];
TABLEAU CAR MOTS[10][20];
```

Le premier exemple montre la déclaration d'un tableau de 10 postes de type caractère dont les postes seront nommées mot (i) i allant de 1 à 10.

Dans certains langages i ira de 0 à 9

1.7. Les fonctions et procédures

Lorsque l'algorithme devient trop compliqué, nous avons envie de le découper, de manière à ce que chaque partie soit plus simple et donc plus lisible.
De même, lorsqu'une partie de code doit être exécutée plusieurs fois à des endroits différents ou réutilisée ultérieurement on pourra ne l'écrire qu'une fois et lui donner un nom en en faisant une **fonction** ou une **procédure**.

1.7.1. Procédure

Une **procédure** est une suite d'instructions servant à réaliser une tâche précise en fonction d'un certain nombre de paramètres.
Ce lot d'instructions est écrit une fois dans la procédure mais peut être exécutée autant de fois que voulu en appelant la procédure : on aura donc une ligne de code pour exécuter n instructions.
De plus une procédure peut fonctionner avec des paramètres qui sont indiqués au moment de l'appel et pris en compte dans l'exécution de la procédure.

Les paramètres sont de deux types.
Les paramètres de type VAL : ils contiennent une valeur qui sera utilisée dans la procédure.
Les paramètres de type VAR : ils représentent une variable du programme appelant qui pourra être lue et modifiée si nécessaire.

Dans notre pseudo-langage, une procédure se déclare de la manière suivante :

```
PROCEDURE nom procédure ( Param1, Param2,..... Paramn )
déclarations de variables locales
DEBUT
        actions
FIN
```

Les variables que nous déclarons à l'intérieur de procédure ne sont connues que dans cette procédure. Elles sont d'ailleurs nommées variables locales.

Pour utiliser cette procédure dans un programme appelant, on écrit :

```
nom procédure ( P1, P2, ...,Pn.) ;
```

```
PROCEDURE remplir_chaine (CAR caractere,, ENTIER longueur,
                          TABLEAU CAR chaine [ MAXCAR] )
ENTIER  n1,  nLong;
DEBUT
        SI longueur > MAXCAR
                ALORS  nLong ⇐ MAXCAR
                SINON  nLong ⇐ longueur
        FINSI
POUR  n1 ALLANT DE 1 A  nLongueur
        chaine( n1) ⇐ caractere;
FinPour
FIN
```

Cette procédure attend en entrée, trois paramètres pour fonctionner. Le premier sera de type caractère, le second de type numérique entier et le troisième de type tableau de caractères

Nous appellerons cette fonction comme ceci :

```
remplir_chaine("*",12, ligne);
remplir_chaine(longmot, rep, message);
```

Dans le premier cas, nous envoyons en paramètres deux constantes et une variable : la première constante("*")de type caractère qui sera reçue dans caractère, la seconde (12) qui sera reçue dans longueur et la variable (ligne) qui sera reçue dans chaîne.
Dans le deuxième appel tous les paramètres sont transmis sous forme de variables.

Appel par remplir_chaine("*",12, ligne);

PROCEDURE remplir_chaine (CAR caractere, ENTIER longueur,TABLEAU CAR chaine [MAXCAR])

Appel par remplir_chaine(rep, longmot, message);

Les paramètres doivent être transmis dans l'ordre et doivent respecter le type défini dans la procédure.

1.7.2. Fonction

Une **fonction** est une procédure dont le but est de déterminer une valeur et de la retourner au programme appelant.
Dans notre pseudo-langage, elle se déclare de la manière suivante :

```
type  FONCTION  nom de la fonction  ( VALtype nom VARtype nom, type nom, ... )
        déclarations de variables locales
        DEBUT
                actions
        RETOURNE valeur
        FIN
```

Les mêmes remarques que pour la procédure s'appliquent.
De plus, il faut noter que la fonction retourne une valeur (ou le contenu d'une variable) et que donc à la déclaration on doit indiquer son type, c'est à dire le type de cette valeur.

L'appel d'une fonction s'écrit :

```
variable  ⇐  nom de la fonction ( valeur1 ,  valeur2 , .. ) ;
```

Une fonction ne peut donc pas retourner un tableau.

exemples :

```
ENTIER FONCTION valeur_absolue (VAL ENTIER nombre)
DEBUT
SI (nombre < 0)
        ALORS
        RETOURNE (nombre * (-1))
SINON
        RETOURNE nombre
FINSI
FIN
```

Nous l'utiliserons ainsi :

```
        n1  ⇐- valeur_absolue( n1 );
        nLimite ⇐  valeur_absolue(-120);
```

2. Règles de programmation

Un programme doit être **le plus lisible possible**, de manière à ce que n'importe qui d'autre que l'auteur soit capable de comprendre ce qu'il fait rien qu'en le lisant. Pour cela il faut suivre les quelques règles suivantes :

- le nom des variables doit être significatif, c'est à dire indiquer clairement à quoi elles servent
- un algorithme ne doit pas être trop long (une page écran). S'il est trop long, il faut le découper en fonctions et procédures (voir 1.7)
- les structures de contrôle doivent être indentées, c'est à dire, par exemple, que les instructions qui suivent le **alors** doivent toutes être alignées et décalées d'une tabulation par rapport au **si**. Il en est de même pour les répétitives.
- A chaque imbrication d'une structure de contrôle, on décale d'une tabulation.

exemples :

mal écrit :	mieux écrit :
ENTIER a,b,c,i,j,k	ENTIER i,j;
TABLEAU CAR tab(10)(20)	ENTIER nbmots, nbavecx;
	BOOLEEN trouve;
i ⇐1; a ⇐-0; b⇐0;	TABLEAU CAR mots (10) (20)
FAIRE	
SI (tab(i,1) <> ' ')	i ⇐-1; nbmots ⇐0; nbavecx ⇐0;
ALORS	FAIRE
c ⇐ 0 ; a ⇐ a+1;	SI (mots(i,1) <> ' ')
j ⇐ 1	ALORS
TANTQUE ((j <20) ET (c <> 0)) FAIRE	trouve ⇐FAUX;
SI tab(i,j) = 'X' ALORS c ⇐ 0 FINSI	nbmots ⇐nbmots + 1;
j ⇐ j+1;	j ⇐ 1;
FINTQ FINSI	TANTQUE ((j < 20) ET (NON trouve))
SI (c = 0) ALORS b ⇐b+1 FINSI i ⇐ i+1; JUSQUA (i > 10);	SI (mots(i,j) = 'X')
	ALORS
	trouve ⇐VRAI
	FINSI
	FINTQ
	SI trouve ALORS nbavecx ⇐ nbavecx +1; FINSI
	FINSI
	i ⇐i+1;
	JUSQUA (i > 10)

En ce qui concerne les fonctions et procédures, il y a aussi des règles à respecter :

- On doit toujours être capable de donner un nom significatif à une procédure ou à une fonction.
- Le nombre de paramètres ne doit pas être trop grand (en général inférieur à 5) car cela nuit à la lisibilité du programme.
- Une procédure ou une fonction doit être la plus générale possible de manière à pouvoir être réutilisée dans d'autres circonstances.
- Si le but d'une procédure est de calculer une valeur simple, il est préférable d'en faire une fonction.
- Il est souvent plus clair d'écrire une fonction booléenne plutôt qu'une condition complexe.

3. La syntaxe du pseudo-langage

Un programme comportera

- Une partie déclaration
- Une partie encadrée par **début** **fin** où sont décrites les actions

```
programme :
déclarations;
DEBUT
        Actions
FIN
```

Dans la partie déclarations, nous trouvons :

- déclaration de variables
- déclaration de fonction
- déclaration de procédure

3.1.1. Déclaration de variables :

```
type   nom
ou
TABLEAU   type   nom ( entier ) [ ( entier ) ] ;
```

Où type peut être: **ENTIER OU REEL OU CAR OU BOOLEEN**

3.1.2. Déclaration de fonction :

```
type FONCTION ( VAL type nom, type nom, ...VAR type nom, type nom, ...)
        déclaration de variables
        DEBUT
        Actions
        RETOURNE valeur
        FIN
```

3.1.3. Déclaration de procédure:

```
PROCEDURE nom (     VAL type nom, type nom, ...VAR type nom, type nom, ...)
déclaration de variables
DEBUT
        Actions
FIN
```

Dans la partie actions, nous trouvons :

- suite d'instructions
- alternative
- répétitive

3.1.4. Suite d'instructions :

```
instruction;
instruction; ...
```

3.1.5. Alternative :

```
SI condition ALORS
        Actions si vrai
[ SINON
        actions si faux]
FINSI
```

3.1.6. Répétitive :

- Tantque
- Faire
- Pour

```
TantQue condition
        actions  si vrai
FINTQ
```

```
Faire
        Actions
Jusqua condition
```

```
POUR variable ALLANT DE entier A entier  [PAS entier]
        Actions
FinPour
```

3.1.7. Pour exprimer une condition :

- Opérateurs logiques **ET, OU, NON, XOR**
- Opérateurs de comparaison =, <, >, <=, >=, <>

3.1.8. Les instructions

- variable ← valeur
- **LIRE** variable
- **AFFICHER** [texte valeur]
- Appel de fonction : variable ← nomFonction (param1, param2, ...) ;
- Appel de procédure nomProcédure (param1, param2, ...) ;

Pour exprimer une valeur, nous pouvons utiliser :

- Un nom de variable
- Une constante

4. Comment analyser un problème ?

Vous vous rendrez vite compte que chaque personne a un mode de raisonnement qui lui est propre

Dans les quelques pages qui suivent, nous vous indiquerons quelques raisonnements tenus pour arriver à un pseudo code répondant à une question donnée mais il s'agit de notre façon de raisonner : à vous de vous construire la votre.

Première étape : bien comprendre la question.

Bien souvent c'est une analyse de texte qu'il faut faire sauf si la question est très simple.

Dans les entraînements qui vous sont proposés ensuite, nous avons expliqué en quelques phrases le besoin et nous l'avons fait suivre par un exemple :

- qu'avons-nous au départ ?
- à quel résultat devons-nous arriver?

Cela est un début de démarche

- Imaginez plusieurs points de départ possibles.
- Pour chacun d'eux, quel est le résultat que vous devez obtenir ?

Dans les points d'entrée possible, essayez de penser aux cas extrêmes.

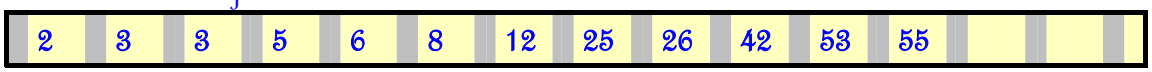
Pour illustrer notre discours, nous allons utiliser le problème suivant :

Cet algorithme permet de ranger un élément à sa place dans une liste de manière très rapide.
Nous possèdons une liste de **N** entiers triés, nous oulons placer un nombre **X** au bon endroit parmi ceux déjà triés.
Comment ? On compare **X** à l'élément du milieu de tableau, s'il est inférieur on réduit le tableau à sa partie gauche, s'il est supérieur on réduit le tableau à sa partie droite.
On réitère l'opération jusqu'à ce que le tableau ait moins de 3 éléments.

Quels sont les différents types de cas à traiter ?

Premier cas

Les éléments de départ sont :



Nous voulons insérer le nombre **7 au milieu**

Le résultat devrait être :

Deuxième cas

Les éléments de départ sont :

Nous voulons insérer le nombre **7 dans une liste ne comportant aucun élément**

Le résultat devrait être :

Troisième cas

Les éléments de départ sont :

Nous voulons insérer le nombre **1 en première case**

Le résultat devrait être :

Quatrième cas

Les éléments de départ sont :

Nous voulons insérer le nombre **60 qui sera en fin de liste**

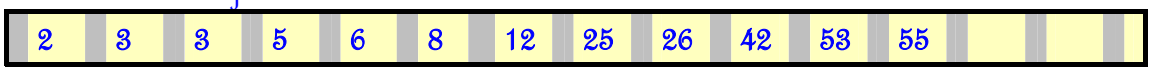
Le résultat devrait être :

Deuxième étape : comment passer des éléments de départ au résultat

Se consacrer d'abord au premier cas qui représente fréquemment le cas le plus courant.
Puis, voir si le deuxième cas fonctionne correctement avec notre démarche du premier cas.
Si ce n'est pas le cas, modifier la démarche pour que le deuxième cas fonctionne correctement.

Dès que c'est fait, reprendre la démarche avec le premier cas qui lui ne fonctionne peut-être plus.

Puis, voir si le troisième cas fonctionne correctement avec notre démarche.
Sinon, adapter la démarche et re vérifier le cas 1 et le cas 2 et ainsi de suite.....
Comment faire avec le premier cas ?
Les éléments de départ sont :



Nous voulons insérer le nombre **7 bien placé dans la liste**
Nous devons connaître le milieu du tableau : pour cela:

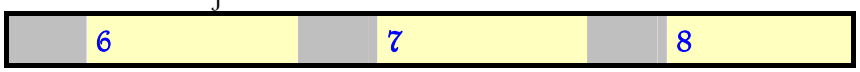
- Il faut déterminer le nombre d'éléments contenus au départ soit 12 éléments
- Puis le diviser par 2 pour avoir la position de l'élément du milieu soit 6

6ème élément est 8 ➔ 7 est plus petit que 8 donc nous insèrerons 7 dans

- Nous re divisons le nombre d'éléments 6 par 2 soit 3
- .le 3ème élément est 3 ==> on insère 7 dans

- Nous re divisons le nombre d'éléments soit 4 par 2
- le 2ème élément est 5 ==> on insère 7 dans

- Nous re divisons le nombre d'éléments soit 3 par 2 soit 1,5 que nous arrondissons à 2
- .le 2ème élément est 6 inférieur à 7 ==> on insère 7 entre 6 8



Résumons notre démarche :

Soit un nombre à insérer **nAInserer**

Soit 1 le départ de notre recherche **nDebut**

Soit le nombre d'élements à consuler **nFin**

☞ **nbElement** prend la valeur de **nFin**

Nous cherchons le nombre d'éléments à consulter **nbElement**

Nous divisons ce nombre par 2 **nPosition**

Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**

Nous comparons le nPositioniéme numéro de la liste avec notre nombre à Insérer

Si ce nombre est inférieur au nombre à insérer,

☞ nous prenons la partie droite du tableau depuis l'élément en nPosition jusqu'à la fin; cela revient à dire que nous changeons la position de début du tableau **nDebut** prend la valeur de **nPosition**.

☞ autrement nous prenons la partie gauche depuis le 1er élément jusqu'à l'élément en nPositio; cela revient à dire que nous changeons la position de fin du tableau **nFin** prend la valeur de **nPosition**

Nous cherchons le nombre d'éléments restant à consulter **nbElement**

☞ **nbElement** prend la valeur de **nFin – nDebut + 1**

Nous divisons ce nombre par 2 **nPosition**

Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**

Voyez que nous sommes en train de recommencer la démarche telle qu'elle a été décrite un peu au dessus. Nous sommes donc en train de répéter une même séquence de ligne ce qui veut dire que nous sommes en présence d'une répétitive.

Notre démarche devient donc
Soit un nombre à insérer **nAInserer**
Soit 1 le départ de notre recherche **nDebut**
Soit le nombre d'élements à consuler **nFin**
Nous cherchons le nombre d'éléments à consulter **nbElement**

- la première fois **nbElement** prend la valeur de **nFin**

Nous répétons

Nous divisons ce nombre par 2 **nPosition**
Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**
Nous comparons le nPositioniéme numéro de la liste avec notre nombre à Insérer
Si ce nombre est inférieur au nombre à insérer,

- nous prenons la partie droite du tableau depuis l'élément en nPosition jusqu'à la fin; cela revient à dire que nous changeons la position de début du tableau **nDebut** prend la valeur de **nPosition**.
- autrement nous prenons la partie gauche depuis le 1er élément jusqu'à l'élément en nPosition; cela revient à dire que nous changeons la position de fin du tableau **nFin** prend la valeur de **nPosition**

Nous cherchons le nombre d'éléments restant à consulter **nbElement**

- les autres fois **nbElement** prend la valeur de **nFin – nDebut + 1**

Fin de la séquence répétée

A ce point de raisonnement, il faut se poser la question " répéter combien de fois?" Qu'est ce qui fait que la répétition doit cesser? C'est une question qui demande un solide bons sens; si nous n'y répondons pas bien nous avons deux écueils possibles

- Nous ne rentrons jamais dans la séquence d'instructions
- Nous n'en sortons jamais : c'est ce que nous appelons "boucler"; c'est le mouvement perpétuel.

Dans notre exemple, nous avons terminé la répétition quand notre nombre d'éléments à consulter est arrivé à 2

Notre démarche devient donc

Soit un nombre à insérer **nAInserer**

Soit 1 le départ de notre recherche **nDebut**

Soit le nombre d'élements à consuler **nFin**

Nous cherchons le nombre d'éléments à consulter **nbElement**

- la première fois **nbElement** prend la valeur de **nFin**

Nous répétons **TANTQUE** nombre d'éléments > 2

Nous divisons ce nombre par 2 **nPosition**

Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**

Nous comparons le nPositioniéme numéro de la liste avec notre nombre à Insérer

Si ce nombre est inférieur au nombre à insérer,

- nous prenons la partie droite du tableau depuis l'élément en nPosition jusqu'à la fin; cela revient à dire que nous changeons la position de début du tableau **nDebut** prend la valeur de **nPosition**.
- autrement nous prenons la partie gauche depuis le 1er élément jusqu'à l'élément en nPosition; cela revient à dire que nous changeons la position de fin du tableau **nFin** prend la valeur de **nPosition**

FINSI

Nous cherchons le nombre d'éléments restant à consulter **nbElement**

- les autres fois **nbElement** prend la valeur de **nFin – nDebut + 1**

Fin de la séquence répétée **FINTQ**

Nous sommes maintenant avec deux éléments à consulter.

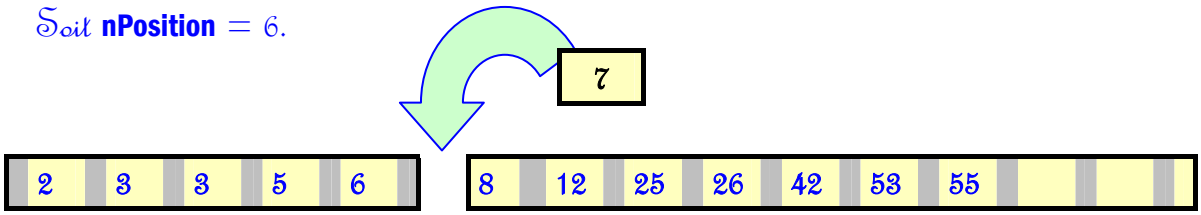
Toujours en se basant sur le bon sens, nous savons que le deuxième élément est nécessairement plus grand ou égal que le nombre à insérer car quand nous coupions notre tableau en deux c'était par un "inférieur". Il suffit donc de comparer notre nombre à insérer au premier élément.

SI l'élément en position nDebut est inférieur à notre nombre à insérer

- La position dans laquelle nous allons insérons notre nombre à insérer est **nDebut** + 1
- Autrement c'est **nDebut**

FINSI

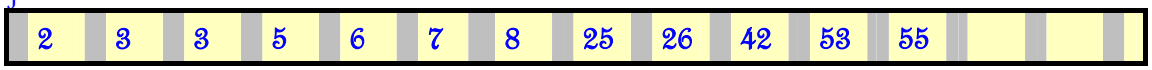
A ce point du raisonnement, nous savons qu'il faut insérer 7 en 6ème position
Soit **nPosition** = 6.



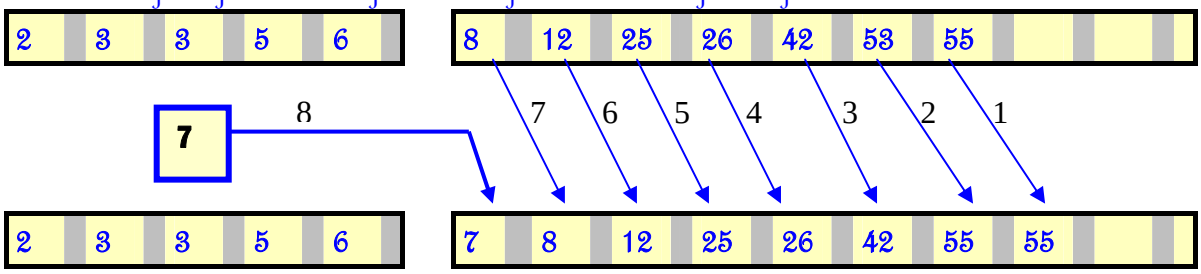
Si nous envoyons notre nombre à insérer en 6ème position, nous allons "écraser" la valeur qui y était soit 8 : Notre résultat serait alors

C'est différent du résultat prévu :

Ce qu'il faut faire c'est décaler 8 vers la droite c'est à dire à la place de 12 pour mettre 7 à la place



Si nous procédons ainsi nous perdons toujours un des éléments initiaux.
Comment faire pour ne rien perdre ? il faut commencer par la fin.



Traduite en français, notre démarche serait :
Soit le nombre d'éléments de la liste d'origine **nFin**
Décaler toutes les valeurs de la position **nFin** à la position **nPosition** vers la droite.

Qui se traduit aussi par
POUR n **ALLANT** de **nFIN** à **nPosition** en faisant –1 à chaque tour
Prendre la valeur en position n et la mettre dans la valeur en position n+1
FINPOUR
Mettre la nombre à insérer dans la valeur qui est en position **nPosition**

En résumé, notre démarche est devenue :

Soit un nombre à insérer **nAInserer**

Soit 1 le départ de notre recherche **nDebut**

Soit le nombre d'élements à consuler **nFin**

Nous cherchons le nombre d'éléments à consulter **nbElement**

- la première fois **nbElement** prend la valeur de **nFin**

Nous répétons **TANTQUE** nombre d'éléments > 2

Nous divisons ce nombre par 2 **nPosition**

Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**

Nous comparons le nPositioniéme numéro de la liste avec notre nombre à Insérer

SI ce nombre est inférieur au nombre à insérer,

- nous prenons la partie droite du tableau depuis l'élément en nPosition jusqu'à la fin; cela revient à dire que nous changeons la position de début du tableau **nDebut** prend la valeur de **nPosition**.
- autrement nous prenons la partie gauche depuis le 1er élément jusqu'à l'élément en nPosition; cela revient à dire que nous changeons la position de fin du tableau **nFin** prend la valeur de **nPosition**

FINSI

Nous cherchons le nombre d'éléments restant à consulter **nbElement**

- les autres fois **nbElement** prend la valeur de **nFin – nDebut + 1**

Fin de la séquence répétée **FINTQ**

SI l'élément en position nDebut est inférieur à notre nombre à insérer

- La position dans laquelle nous allons insérons notre nombre à insérer est **nDebut** + 1
- Autrement c'est **nDebut**

FINSI

POUR n **ALLANT** de **nFIN** à **nPosition** en faisant –1 à chaque tour

Prendre la valeur en position n et la mettre dans la valeur en position n+1

FINPOUR

Mettre la nombre à insérer dans la valeur qui est en position **nPosition**

Examinons maintenant le deuxième cas

Les éléments de départ sont :

Nous voulons insérer le nombre **7 dans une liste ne comportant aucun élément**

Le résultat devrait être :

Par rapport au cas précédant, nous nous apercevons que la recherche de la position d'insertion est inutile car il n'y a pas de liste d'éléments au départ.
De même, il ne sera pas nécessaire de faire des décalages puisqu'il n'y a rien à décaler.

Pour ce cas, la procédure sera :

Soit un nombre à insérer	**nAInserer**
Soit 1 le départ de notre recherche	**nDebut**
Soit le nombre d'élements à consuler	**nFin**
Nous cherchons le nombre d'éléments à consulter	**nbElement**
La position d'insertion est 1	**nPosition**

Mettre la nombre à insérer dans la valeur qui est en position **nPosition**

Nous adaptons notre démarche du premier cas
Soit un nombre à insérer **nAInserer**
Soit 1 le départ de notre recherche **nDebut**
Soit le nombre d'élements à consuler **nFin**
Nous cherchons le nombre d'éléments à consulter **nbElement**
☞ la première fois **nbElement** prend la valeur de **nFin**
La position d'insertion est 1 **nPosition**
Si nFin n'est pas zéro

Nous répétons **TANTQUE** nombre d'éléments > 2

Nous divisons ce nombre par 2 **nPosition**

Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**

Nous comparons le nPositionième numéro de la liste avec notre nombre à Insérer

Si ce nombre est inférieur au nombre à insérer,

- ✓ nous prenons la partie droite du tableau depuis l'élément en nPosition jusqu'à la fin; cela revient à dire que nous changeons la position de début du tableau nDebut prend la valeur de nPosition.
- ✓ autrement nous prenons la partie gauche depuis le 1er élément jusqu'à l'élément en nPosition; cela revient à dire que nous changeons la position de fin du tableau nFin prend la valeur de nPosition

FINSI

Nous cherchons le nombre d'éléments restant à consulter **nbElement**

les autres fois nbElement prend la valeur de nFin – nDebut + 1

Fin de la séquence répétée **FINTQ**

SI l'élément en position **nDebut** est inférieur à notre nombre à insérer

- ✓ La position d'insertion est **nDebut** + 1
- ✓ Autrement c'est **nDebut**

FINSI

Soit le nombre d'élements à consuler **nFin**

POUR n ALLANT de **nFIN** à **nPosition** en faisant –1 à chaque tour

Prendre la valeur en position n et la mettre dans la valeur en position n+1

FINPOUR

FINSI

Mettre la nombre à insérer dans la valeur qui est en position **nPosition**

Examinons le troisième cas

Les éléments de départ sont :

Nous voulons insérer le nombre **1 en première case**

Le résultat devrait être :

La démarche écrite pour les deux premiers cas fonctionne

Examinons le quatrième cas

Les éléments de départ sont :

Nous voulons insérer le nombre **60 qui sera en fin de liste**

Le résultat devrait être :

Lorsque nous déroulons la démarche précédemment écrite, nous arrivons à une anomalie.

Au premier **TANTQUE**, les valeurs des variables sont :

Variable	NDébut	NFin	NPosition	nbElement
Valeur	1	12	1	12

A la fin de ce premier **TANTQUE** , elles valent :

Variable	NDébut	NFin	NPosition	nbElement
Valeur	6	12	1	7

Démarrons le premier **TANTQUE**, **nPosition** devient 7 / 4 = 3,5 ramené à l'entier supérieur soit 4. Problème, car 4 n'est pas compris entre 6 et 12. il Nous faut changer le calcul de **nPosition** par Ce calcul devient :

Nous divisons ce nombre par 2 **nPosition**

Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**

Ajouter **nDebut** à **nPosition**

Si nous reprenons les cas précédant, cette nouvelle démarche fonctionne toujours.

La démarche finale est :
Soit un nombre à insérer **nAInserer**
Soit 1 le départ de notre recherche **nDebut**
Soit le nombre d'élements à consuler **nFin**
Nous cherchons le nombre d'éléments à consulter **nbElement**
La position d'insertion est 1 **nPosition**
Si nFin n'est pas zéro

Nous répétons **TANTQUE** nombre d'éléments > 2

Nous divisons ce nombre par 2 **nPosition**

Si le résultat n'est pas entier, nous arrondissons à l'entier supérieur **nPosition**

Ajouter **nDebut – 1** à **nPosition**

Nous comparons le nPositioniéme numéro de la liste avec notre nombre à Insérer

Si ce nombre est inférieur au nombre à insérer,

- ✓ nous prenons la partie droite du tableau depuis l'élément en nPosition jusqu'à la fin; cela revient à dire que nous changeons la position de début du tableau nDebut prend la valeur de nPosition.
- ✓ autrement nous prenons la partie gauche depuis le 1er élément jusqu'à l'élément en nPosition; cela revient à dire que nous changeons la position de fin du tableau nFin prend la valeur de nPosition

FINSI

Nous cherchons le nombre d'éléments restant à consulter **nbElement**

les autres fois nbElement prend la valeur de nFin – nDebut + 1

Fin de la séquence répétée **FINTQ**

SI l'élément en position **nDebut** est inférieur à notre nombre à insérer

- ✓ La position d'insertion est **nDebut** + 1
- ✓ Autrement c'est **nDebut**

FINSI

Soit le nombre d'élements à consuler **nFin**

POUR n ALLANT de **nFIN** à **nPosition** en faisant –1 à chaque tour

Prendre la valeur en position n et la mettre dans la valeur en position n+1

FINPOUR

FINSI

Mettre la nombre à insérer dans la valeur qui est en position **nPosition**

5. Un peu d'entraînement

Pour chacun des entraînements proposés, vous avez :

- ☞ Le sujet sur la première page
- ☞ Un corrigé possible sur la page suivante
- ☞ Eventuellement des explications complémentaires

Faites-les en ne regardant pas le corrigé avant; cela peut paraître une évidence mais la tentation sera grande Résistez !!!!

5.1. L'alternative

Un patron décide de calculer le montant de sa participation au prix du repas de ses employés de la façon suivante :

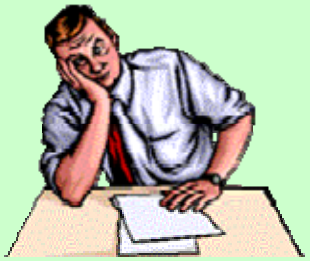
S'il est célibataire	participation de 20%
S'il est marié	participation de 25%
S'il a des enfants	participation de 10% supplémentaires par enfant

La participation est plafonnée à 50%
Si le salaire mensuel est inférieur à 6000f la participation est majorée de 10%

Écrire le programme qui lit les informations au clavier et affiche pour un salarié, la participation à laquelle il a droit.

Deuxième partie de l'exercice :
Corriger l'exercice précédent pour ne pas être obligé de relancer le programme pour chaque employé.

Corrigé



```
Car strRepMarie;
Entier nEnfants,
REEL nSalaire
REEL nParticipation,

AFFICHER "L'employé est-il marié  (O/N) ?"
LIRE strRepMarie
AFFICHER "Combien a-t-il d'enfants ?"
LIRE nEnfants
AFFICHER "Quel est son nSalaire ?"
LIRE nSalaire

SI (strRepMarie = 'N') ALORS
        nParticipation  ⇐ 0.2
SINON
        nParticipation ⇐ 0.25
FINSI
nParticipation ⇐ nParticipation + nEnfants*0.1
SI (nSalaire <= 6000) ALORS
        nParticipation ⇐  nParticipation + 0.1
FINSI
SI (nParticipation > 0.5) ALORS
        nParticipation ⇐  0.5
FINSI
```

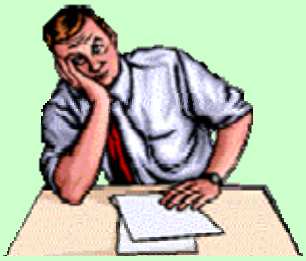
Pour la deuxième partie de l'exercice, il faut rajouter une boucle :

```
Car strReponse
Booleen bSuite
bSuite = vrai
TANTQUE  bSuite
         voir ci-dessus
         AFFICHER "Y a-t-il un autre employé (O/N) ?"
         LIRE strReponse
        Si strReponse = "N"
                BSuite = faux
FINTQ
```

5.2. La boucle

Autre sujet : Écrire un programme qui saisit des entiers et en affiche la somme et la moyenne (on arrête la saisie avec la valeur 0)

Corrigé



```
REEL nSomme
REEL nMoyenne
REEL nNombreFrappes
REEL nSaisie

nSomme ⇐ 0
nNombreFrappes ⇐ 0
nSaisie = 99999                // tout sauf zéro
TANTQUE nSaisie <> 0
        AFFICHER "entrer un entier"
        LIRE nSaisie
        SI (nSaisie <> 0)
        ALORS
                nNombreFrappes  ⇐ nNombreFrappes + 1;
                nSomme  ⇐ nSomme + nSaisie;
        SINON
                nMoyenne  ⇐ (nSomme / nNombreFrappes)
                AFFICHER "La somme = " somme " La moyenne = " moyenne
        FINSI
FINTQ
```

Les statistiques

On saisit des entiers (comme exercice précédent) et on les range dans un tableau (maximum 50)

Écrire un programme qui affiche le maximum, le minimum et la valeur moyenne de ces nombres.

A vos neurones

Corrigé

```
ENTIER nMini, nMaxi,,nSomme
REEL nMoyenne
Entier nIndice, nbNombre, nSaisie
TABLEAU ENTIER nTab[50]

AFFICHER "entrer un entier"
LIRE nSaisie
nbNombres ⇐ 0
TANTQUE (nSaisie <> 0 ET nbNombre < 50
        SI (nombre <> 0)
                nb_nombres ⇐ nb_nombres + 1
                TAB(nb_nombres) ⇐ nombre
        FINSI
        AFFICHER "entrer un entier"
        LIRE nSaisie
FINTQ
nSomme  ⇐ 0
nMini ⇐ TAB(1)
nMaxi⇐ TAB(1)
POUR nIndice ALLANT  DE 1 A nbNombres
        somme ⇐ somme +TAB(nIndice)
        SI (TAB(i) < nMini ALORS min ⇐ TAB(nIndice)
        SI (TAB(i) > nMaxi) ALORS max ⇐-TAB(nIndice)
FinPour
nMoyenne ⇐ --(nSomme / nbNombres)
AFFICHER "La somme = " nSomme " La moyenne = " nMoyenne
AFFICHER "Le minimum = " nMini " Le maximum = " nMaxi
```

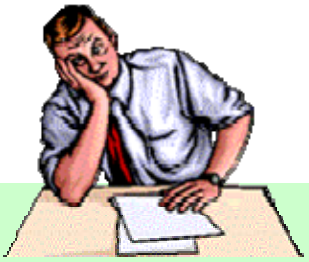
5.3. Contraction

Recopier une phrase dans une autre en ôtant toutes les occurrences d'un caractère
Soit une phrase terminée par un point.
Il s'agit de la restituer en supprimant les occurrences d'un caractère donné.

Exemple :

phrase :	abbcccdeeeffg
caractère :	c
résultat :	abbdeeeffg

Donnez le jeu d'essai qui permet de tester cette procédure.
Donnez l'algorithme de la procédure en pseudo code.



Corrigé

```
1.  TABLEAU CAR strPhrase1[100], strResultat[100]
2.  ENTIER nIndice 1, nIndice2
3.  CAR strOctet
4.  String strSaisie
5.  AFFICHER "entrer la phrase terminée par un point"
6.  LIRE strSaisie
7.  strPhrase1 ⇐ strSaisie;
8.  AFFICHER "entrer le caractère à ôter""
9.  LIRE strSaisie
10. StrOctet⇐ strSaisie
11. //
12. nIndice1 ⇐1;
13. nIndice2 ⇐ 1;
14. TANTQUE strPhrase1(nIndice1) <> "."
15.         SI strPhrase1(nIndice1) <> strOctet  ALORS
16.             strResultat(nIndice2) = strPhrase1(nIndice1)
17.             nIndice2 ⇐ nIndice2 +1
18.         FINSI;
19.         nIndice1 ⇐ nIndice1 +1
20. FINTQ
21. AFFICHER "le résultat est : ", strResultat
```

5.4. Doublons

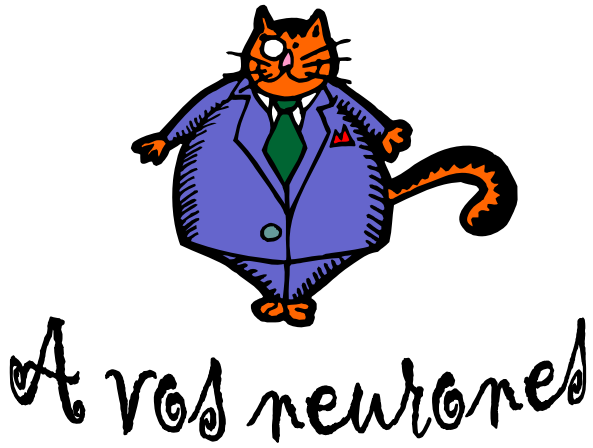
Recopier une phrase dans une autre en ôtant tous les doublons de caractères successifs
Soit une phrase terminée par un point.
Il s'agit de la restituer en supprimant tous les doublons de caractères successifs.

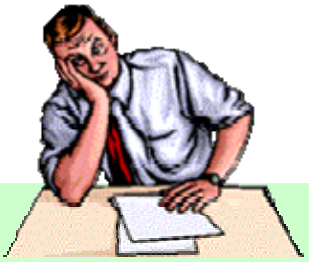
Exemple : abbcccdeeefffg. donne abcdefg.

Donnez le jeu d'essai qui permet de tester cette procédure.

Pour tester le programme, c'est à dire voir s'il répond bien à nos attentes, s'il n'a pas de "bug", avant de la faire "tourner" sur la machine nous imaginons un jeu d'essai avec tous les cas à tester et le résultat attendu pour chaque cas : **c'est le jeu d'essai.**

Donnez l'algorithme de la procédure.





Corrigé

```
TABLEAU CAR strPhrase1[100], strResultat[100]
ENTIER nIndice 1, nIndice2
CAR strOctet
STRING strSaisie

AFFICHER "entrer la phrase terminée par un point"
LIRE strSaisie
strPhrase1 ⇐ strSaisie;
//
nIndice1 ⇐1
nIndice2 ⇐ 1
strOctet  ⇐ " "
TANTQUE strPhrase1(nIndice1)  <>  "."
        SI strPhrase1(nIndice1) <> strOctet   ALORS
                strResultat(nIndice2) = strPhrase1(nIndice1)
                nIndice2 ⇐ nIndice2 +1
                strOctet  ⇐ strPhrase1(nIndice1)
        FINSI
        nIndice1 ⇐ nIndice1 +1
FINTQ
AFFICHER "le résultat est : ", strResultat
```

Jeu d'essai:

bbbbiidooooonn.	donne	bidon
aaaaaaaaaaaaa.	donne	a
meilleur	donne	meilleur

Les deux derniers cas sont des cas extrêmes mais dans un jeu d'essai sérieux, ce sont des cas à ne pas oublier

Si nous voulions être très rigoureux, nous devrions tester que la phrase saisie se termine bien par un point. Si ce n'est pas le cas, le **TANTQUE** va provoquer une boucle dont nous ne sortirons jamais. Nous pourrions appeler çà le mouvement perpétuel.

5.5. Equivalence

Déterminer si deux phrases sont équivalentes.
Soit deux phrases terminées par un même terminateur.
Elles sont dites équivalentes si elles ont les mêmes lettres dans le même ordre mais avec un nombre d'occurrences de ces lettres qui peut différer entre les deux phrases.

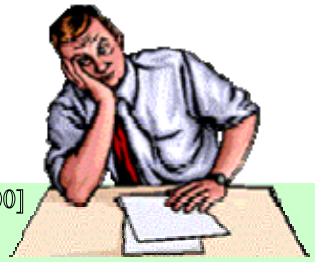
On supposera qu'il existe une fonction longueur **lg** de chaîne qui renvoie un entier

Exemple :

abbcccdeeefffg
aabcdefffffg sont équivalentes

Donnez le jeu d'essai qui permet de tester cette procédure.
Donnez l'algorithme de la procédure toujours en pseudo code.

A vos neurones



Corrigé

```
1.    TABLEAU CAR strPhrase1[100], strPhrase2[100]
2.    ENTIER nIndice 1, nIndice2
3.    CAR strOctet
4.    BOOLEEN bOK
5.    STRING strSaisie
6.    AFFICHER "entrer la première phrase"
7.    LIRE strSaisie
8.    strPhrase1 ⇐ strSaisie;
9.    AFFICHER "entrer la deuxième phrase"
10.   LIRE strSaisie
11.   StrPhrase2 ⇐ strSaisie;
12.   //
13.   bOk = vrai
14.   nIndice1 ⇐1;
15.   nIndice2 ⇐ 1;
16.   TANTQUE bOk = vrai ET  nIndice1 <strPhrase1.lg  ET nIndice2 <strPhrase2.lg
17.       strOctet ⇐ strPhrase1(nIndice1)
18.       SI strPhrase2(nIndice2) = strOctet   ALORS
19.           TANTQUE strPhrase2(nIndice2) = strOctet   ET  nIndice2 strPhrase2.lg
20.                       nIndice2 ⇐ nIndice2 +1;
21.           FINTQ
22.       SINON
23.               bOk = faux
24.       FINSI;
25.       TANTQUE strPhrase1(nIndice1) = strOctet   ET  nIndice1 <strPhrase1.lg
26.               nIndice1 ⇐ nIndice1 +1;
27.       FINTQ;
28.   FINTQ
29.   SI bOK = vrai
30.       AFFICHER "Les phrases sont équivalentes";
31.   SINON
32.       AFFICHER "Les phrases sont différentes";
```

Faisons tourner le programme à la main, ligne par ligne avec les phrases :
Phrase1 → aabbbccddde Phrase2 : abccdeee

Dans le tableau qui suit, nous notons tous les changements de valeurs subies au fur et à mesure de l'exécution du programme. La première colonne indique la ligne du programme où les valeurs sont notées (les valeurs sont changées une fois la ligne d'affectation passée).

N° ligne	nIndice1	nIndice2	StrPhrase1()	StrPhrase2()	StrOctet	bOk
14						vrai
15	1					
16		1				
18					a	
21		2				
27	2					
27	3					
18					b	
21		3				
26	4					
26	5					
26	6					
21		4			c	
21		5				
27	7					
27	8					
18					d	
21		6				
27	9					
27	10					
27	11					
18					e	
21		7				
21		8				
21		9				
27	12					

Il est assez fastidieux de "dérouler" le programme de cette manière; Des outils appelés "déboggeur" vous permettent de laisser le travail à la machine. Ils permettent de :

- ☞ Faire avancer le programme "pas à pas" c'est à dire ligne par ligne
- ☞ Lister le contenu des variables à un point d'arrêt c'est à dire à une ligne précise (nous vous rappelons que les valeurs ne sont modifiées qu'une fois l'affectation finie; il faut donc se positionner sur la ligne après l'affectation pour voir la valeur prise par la variable.
- ☞ Faire avancer le programme de point d'arrêt en point d'arrêt
- ☞ Eventuellement de modifier les valeurs des variables : attention si vous travaillez dans un contexte aussi utilisé par les utilisateurs, vous risquez de faire des bêtises en modifiant la base de données réelle.

Cas n° 1

Phrase n° 1 ccccccddddaaaagggg
Phrase n° 2 cccdddggggaa Résultat → phrases différentes

Cas n° 2

Phrase n° 1 aaazzzeeerrty
Phrase n° 2 azerty Résultat → phrases équivalentes

Cas n° 3

Phrase n° 1 Immobile
Phrase n° 2 imobile Résultat → phrases différentes ou équivalentes

En effet, une lettre majuscule est différente d'une lettre minuscule sur un système "casse sensitive". Elles sont équivalence sur les autres systèmes.

5.6. Eparpillement

Chercher les lettres d'un mot éparpillées dans une phrase, dans le même ordre.

Soient un caractère terminateur et une phrase terminée par ce caractère terminateur.
Soient un mot donné
Il s'agit de vérifier si les lettres du mot sont bien présentes dans la phrase, ce dans le même ordre que celui du mot.

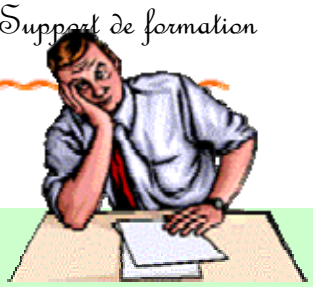
Exemple :

terminateur : .
phrase : le chat est gris et boit.
mot : lattis
longueur : 6

donne vrai

Donnez l'algorithme de la procédure en pseudo code.

A vos neurones



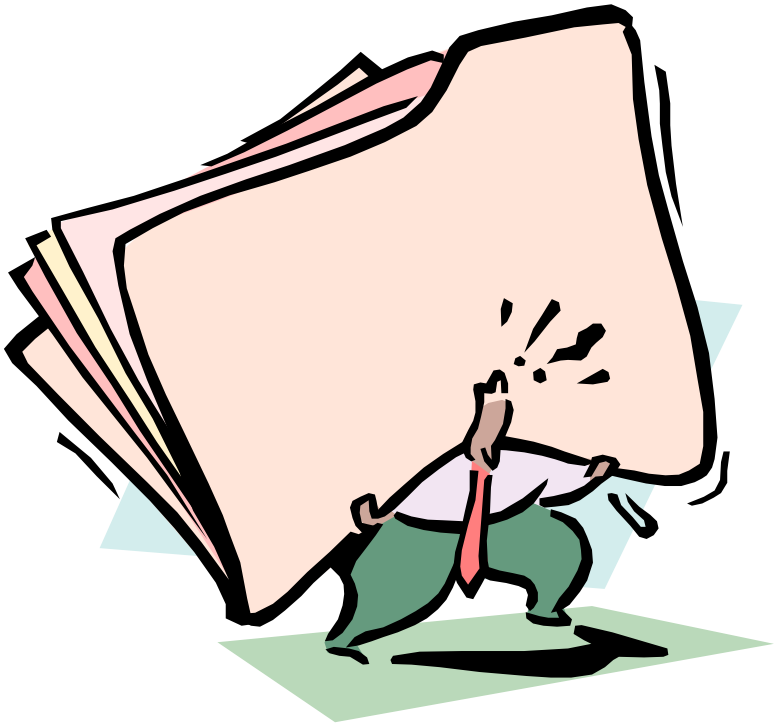
Corrigé

```
CAR strSaisie
ENTIER nIndice 1,nIndice2,nLong1,nLong2
TABLEAU CAR strPhrase1[100], strPhrase2[100]

AFFICHER "entrer la phrase terminée par un point"
LIRE strSaisie
nLong1⇐ longueur(strSaisie) - 1
strPhrase1 ⇐ strSaisie
AFFICHER "entrer le mot recherché"
LIRE strSaisie
NLong2 ⇐ longueur(strSaisie)
strPhrase2 ⇐ strSaisie
AFFICHER "entrer le longueur du mot recherché"
LIRE strSaisie

nIndice1 ⇐1
nIndice2 ⇐ 1
TANTQUE  nIndice2  <  nLong2 ET nIndice1 < nLong1
        SI strPhrase1(nIndice1) = strPhrase2[nIndice2]
                nIndice2 ⇐ nIndice2 +1
        FINSI
        nIndice1 ⇐ nIndice1 +1
FINTQ
SI nIndice2 < nLong2
        AFFICHER "le résultat est : faux"
SINON
        AFFICHER "le résultat est :vrai"

ENTIER FONCTION longueur(CAR str1)
DEBUT
TABLEAU CAR strCarac[100]
StrCarac ⇐ str1
TANTQUE  strCarac[nIndice] <> ' '
        nIndice = nIndice + 1
FINTQ
RETOURNE nIndice
FIN
```

5.7. Inversion

Effectuer la saisie d'une chaîne de caractères qui contiendra un nom et un prénom.
Les prénoms composés seront obligatoirement séparés par des tirets.
Afficher une chaîne de caractères sous forme prénom nom séparés par un espace, en ayant fait disparaître les tirets saisis dans le prénom.
Ecrire la procédure en pseudo-code (éventuellement ensuite avec un langage).

Ne pas utiliser les instructions de type concaténation et recherche d'un caractère dans une chaîne.



Corrigé

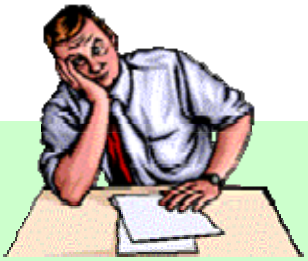
```
CAR strNomPrenom
TABLEAU CAR strOctet[30], strNom[30]
ENTIER nCurseur, nCurseur2, nIndice, nFinPrenom, nDebutPrenom;

AFFICHER "entrer un nom et un prénom"
LIRE strNomPrenom
StrOctet ⇐ strNomPrenom;
nCurseur⇐ longueur(strSaisie) - 1        //  utilisation d'une procédure écrite en page 56
TANTQUE nCurseur  >= 1   ET strOctet(nCurseur) = " "
        nCurseur ⇐ nCursur –1
FINTQ
SI nCurseur = 0 ALORS
        AFFICHER "Vous n'avez rien saisi"
SINON
        nFinPrenom⇐ nCurseur
        TANTQUE nCurseur  >= 1   ET strOctet(nCurseur) <> " "
                nCurseur ⇐ nCurseur –1
        FINTQ
        SI nCurseur = 0 ALORS
                AFFICHER "Vous n'avez saisi que le prénom";
        SINON
                NCurseur2 ⇐ 1;
                nDébutPrenom⇐ nCurseur + 1;
                POUR nIndice ALLANT DE nDebutPrenom A nFinPrenom PAS 1
                        SI strOctet (nIndice) = "-" ALORS
                                StrResultat(nCurseur2) ⇐ " ":
                        SINON
                                StrResultat(nCurseur2) ⇐ StrOctet(nIndice);
                        FINSI;
                nCurseur2 ⇐ nCurseur2 + 1;
                FINPOUR;
                nCurseur ⇐ nDebutPrenom  - 2;          //  envoi du nom après le prénom
                StrNom(nCurseur2) ⇐ " ":
                POUR nIndice ALLANT de  1 A nCurseur
                        nCurseur2 ⇐ nCurseur2 + 1;
                        StrNom(nCurseur2) ⇐ StrOctet(nIndice);
                FINPOUR;
        FINSI
FINSI
AFFICHER "Le résultat est : " strNom
```

5.8. Palindrome

Déterminer si une chaîne de caractères est un palindrome.

Un palindrome est une phrase qui peut se lire dans les deux sens.
Les espaces sont ignorés.
Exemple : esope reste ici et se repose.
Le terminateur est ici un point.

Donnez l'algorithme du programme.



Corrigé

```
CAR strSaisie, strOctet
TABLEAU CAR strPhrase1[100];
ENTIER nIndice 1, nIndice2;
BOOLEEN bOK;
AFFICHER "entrer la phrase terminée par un point"
LIRE strSaisie
strPhrase1 ⇐ strSaisie
bOK = vrai
nIndice1 ⇐1;
// recherche longueur de la phrase
nIndice2 ⇐ 1;
TANTQUE strPhrase1(nIndice2)  <>  "."
        nIndice2 ⇐ nIndice2 +1
FINTQ
nIndice2 ⇐ nIndice2 -1
TANTQUE nIndice1  <= nIndice2  ET bOk = vrai
        SI strPhrase1(nIndice1) = " " ALORS
                TANTQUE strPhrase1(nIndice1) = " " ET nIndice1 <= nIndice2
                        nIndice1 ⇐ nIndice1 +1
                FINTQ
        FINSI
        SI strPhrase1(nIndice2) = " " ALORS
                TANTQUE strPhrase1(nIndice2) = " " ET nIndice1 <= nIndice2
                        nIndice2 ⇐ nIndice2 - 1
                FINTQ
        FINSI
        SI strPhrase1(nIndice1) = strPhrase1(nIndice2)    ALORS
                nIndice2 ⇐ nIndice2 - 1
                nIndice1 ⇐ nIndice1 +1
        SINON
                bOK = faux
        FINSI
 FINTQ
SI bOk = vrai ALORS
        AFFICHER "La phrase est un palindrome";
SINON
        AFFICHER "La phrase n'est pas un palindrome";
FINSI
```

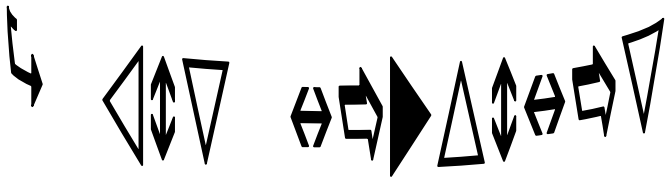
5.9. Cryptage

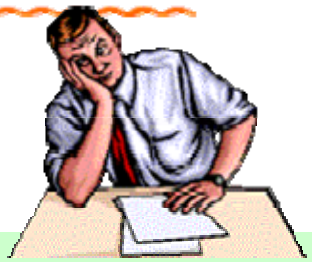
Crypter une phrase en codant les lettres en fonction du mot où elles se trouvent.

Soit une phrase terminée par un point.
Les espaces sont des séparateurs de mot et sont transcrits sans modification.
Il s'agit de la crypter en codant chaque mot suivant son rang dans la phrase.
Au mot de rang 1, on crypte ses lettres avec les lettres qui suivent dans l'alphabet.
Au mot de rang 2, on crypte ses lettres avec les lettres qui suivent de 2 caractères dans l'alphabet. etc ...
Par convention, la lettre suivant le caractère Z est le caractère A.
Les espaces sont des séparateurs de mot et sont transcrits sans modification.

Exemple :

Phrase :	**LE CHAT EST GRIS .**
Rang :	1 2 3 4
Résultat :	**MF EJCV HVW KVMW.**





Corrigé

```
CAR strSaisie,
ENTIER nIndice1,nIndice2,nRang,
TABLEAU CAR strPhrase1[100], strAlphabet[26] strResultat[100]

AFFICHER "entrer la phrase terminée par un point"
LIRE strSaisie
strPhrase1 ⇐ strSaisie
strAlphabet  ⇐ [A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z]
nIndice1 ⇐1
nRang ⇐ 1

TANTQUE strPhrase1(nIndice1)  <>  "."
        SI strPhrase1(nIndice1) <> " "   ALORS
        // recherche indice lettre
                nIndice2 ⇐ 0
                TANTQUE strPhrase1(nIndice1) <> strAlphabet (nIndice2)
                        nIndice2 ⇐ nIndice2 + 1
                FINTQ
                // recherche rang indice de la lettre résultante
                nIndice2 ⇐ nIndice2 +nRang
                SI nIndice2 > 26 ALORS
                        nIndice2 ⇐ nIndice2 – 26
                FINSI;
                StrResultat (nIndice1) ⇐  strAlphabet (nIndice2)
        SINON
                StrResultat (nIndice1) ⇐  " "
                NRang ⇐ nRang + 1
        FINSI
        nIndice1 ⇐ nIndice1 +1
FINTQ
AFFICHER "le résultat est : " , strResultat;
```

5.10. Comptage

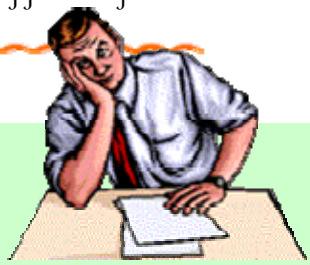
Compter le nombre de mots d'une phrase ayant une terminaison donnée.
Soit une phrase terminée par un point.
Les espaces sont des séparateurs de mot.
Il s'agit de donner le nombre de mots de la phrase ayant pour terminaison la chaîne intitulée terminaison.

Exemple :	Caractère final : .
Phrase :	rien ne sert de courir il faut partir à point il ne faut pas rire.
Terminaison :	rir
Résultat :	1

Note : les terminaisons de longueur nulle indiquent à la procédure qu'il faut renvoyer le nombre de mots de la phrase.

Ecrire la procédure en pseudo code

Corrigé



```
CAR strSaisie,
TABLEAU CAR strPhrase1[100], strTermine[10], strMot[26];
BOOLEEN bOk;
ENTIER nIndice1,nIndice2, nIndice3,nIndice4,nLong,, nLongMot, nCpt;
AFFICHER "entrer la phrase terminée par un point"
LIRE strPhrase1
nLong Phrase ⇐ 0;                                   // longueur de la phrase saisie
TANTQUE strPhrase1(nLongPhrase )<> "." ET nLongPhrase < 100
        NLongPhrase ⇐ nLongPhrase + 1
FINTQ

AFFICHER "entrer la terminaison "
LIRE strTermine
nCpt ⇐ 0
nLong ⇐ 0                                           // longueur de strTermine
TANTQUE strTermine(nLong) >= 'A' et strTermine(nLong) <= "Z"
        NLong ⇐ nLong + 1
FINTQ

nIndice1 ⇐1;
TANTQUE strPhrase1(nIndice1)  <>  "." ET nIndice1< nLongPhrase
        nIndice2 ⇐1                                 // recherche le mot
        strMot ⇐ "    "
        TANTQUE strPhrase1(nIndice1) <> " " ET strPhrase1(nIndice1)  <>  "."
                StrMot(nIndice2) ⇐ strPhrase1(nIndice1)
                nIndice2 ⇐ nIndice2 + 1
                nIndice1 ⇐ nIndice1 + 1
        FINTQ;
        nLongMot ⇐ nIndice2
        SI nLong <> 0 ALORS
                SI nLong <= nLongMot ALORS
                        bOk ⇐ vrai
                        nIndice3 ⇐ 1
                        nDepart ⇐ nLongMot  - nLong
                        POUR nIndice4 ALLANT DE nDepart  A nLongMot  PAS 1
                                SI strTermine(nIndice3) <> strMot(nIndice4) ALORS
                                        bOk ⇐ faux
                                                FINSI
                                nIndice3 ⇐ nIndice3 + 1;
                        FINPOUR;
```

```
                                        SI bOk = vrai ALORS      nCpt ⇐ nCpt + 1;          FINSI
                        FINSI;
                SINON;
                        NCpt ⇐ nCpt + 1;
                FINSI;
                nIndice1 ⇐ nIndice1 +1;
FINTQ
AFFICHER "le résultat est : " , nCpt;
```

5.11. Les tris

Dans tous les exercices qui suivent, nous étudierons différents algorithmes permettant de trier un tableau de 10 entiers.
Afin d'expliquer les algorithmes, on prendra en exemple le tableau suivant :

52 10 1 25 62 3 8 55 3 23

5.11.1. Le tri par sélection

Le premier algorithme auquel on pense pour effectuer ce tri est celui-ci :
on cherche le plus petit élément du tableau et on le place en 1er, puis on cherche le plus petit dans ce qui reste et on le met en second, etc...

52 10 1 25 62 3 8 55 3 23
1 52 10 25 62 3 8 55 3 23
1 3 52 10 25 62 8 55 3 23
1 3 3 52 10 25 62 8 55 23
1 3 3 8 52 10 25 62 55 23
1 3 3 8 10 52 25 62 55 23
1 3 3 8 10 23 52 25 62 55
1 3 3 8 10 23 25 52 62 55
1 3 3 8 10 23 25 52 62 55
1 3 3 8 10 23 25 52 55 62

Écrire l'algorithme qui permet de réaliser ce tri.

Corrigé

```
ENTIER nIndice1, nIndice2, nPetit, nPosition
TABLEAU ENTIER nTab[10]

ChargTableau                          // appelle d'une procédure de chargement du tableau

POUR nIndice1 ALLANT de 1 A 9
        nPetit ⇐ nTab( nIndice1 )
        POUR nIndice2 ALLANT DE nIndice1 A 10
                        SI (nTab( nIndice2 ) < nPetit) ALORS
                        nPetit ⇐ nTab( nIndice2 )
                         nPosition ⇐ nIndice2
                FINSI
        FinPour
        POUR nIndice2 ALLANT DE  nPosition  A  nIndice1+1  PAS -1
                nTab( nIndice2 ) ⇐ nTab(nIndice2-1)
        FinPour
        nTab( nIndice1 ) ⇐ nPetit
FinPour
```

5.11.2. Le tri bulle

Le tri bulle est un tri plus astucieux. Son principe est de faire remonter petit à petit un élément trop grand vers le haut du tableau en comparant les éléments deux à deux.
Si l'élément de gauche est supérieur à son voisin de droite on les inverse et on continue avec le suivant.
Lorsque l'on est en haut du tableau on repart au début et on s'arrête lorsque tous les éléments sont bien placés.

52	10	1	25	62	3	8	55	3	23
10	52	1	25	62	3	8	55	3	23
10	1	52	25	62	3	8	55	3	23
10	1	25	52	62	3	8	55	3	23
10	1	25	52	62	3	8	55	3	23
10	1	25	52	3	62	8	55	3	23
10	1	25	52	3	8	62	55	3	23
10	1	25	52	3	8	55	62	3	23
10	1	25	52	3	8	55	3	62	23
10	1	25	52	3	8	55	3	23	62

On a parcouru tout le tableau, on recommence, jusqu'à ce que tout soit bien placé.

Écrire l'algorithme qui réalise ce tri.

Corrigé

```
ENTIER nIndice1, nIndice2, nTampon
BOOLEEN bInversion;
TABLEAU ENTIER nTab[10]

chargTableau                                    // appelle d'une procédure de chargement du tableau

bInversion ⇐ vrai
TantQue bInversion = vrai
        inversion ⇐ FAUX
        POUR nIndice1 ALLANT DE 1 A  9
                SI (nTab(nIndice1) > nTab(nIndice1+1))
                        nTampon ⇐ nTab(nIndice1)
                        nTab(nIndice1) ⇐ nTab(nIndice1+1)
                        nTab(nIndice1+1) ⇐ nTampon
                        bInversion ⇐ VRAI
                FINSI
        FinPour
FINTQ
```

5.11.3. Le tri par permutation

Le tri par permutation est le tri du jeu de cartes.

On parcourt le tableau jusqu'à ce que l'on trouve un élément plus petit que le précédent, donc mal placé. On prend cet élément et on le range à sa place dans le tableau puis on continue la lecture. On s'arrête à la fin du tableau.

52	10	1	25	62	3	8	55	3	23
10	52	1	25	62	3	8	55	3	23
1	10	52	25	62	3	8	55	3	23
1	10	25	52	62	3	8	55	3	23
1	3	10	25	52	62	8	55	3	23
1	3	8	10	25	52	62	55	3	23
1	3	8	10	25	52	55	62	3	23
1	3	3	8	10	25	52	55	62	23
1	3	3	8	10	23	25	52	55	62

Écrire l'algorithme qui réalise ce tri.

Corrigé

```
ENTIER nIndice1, nIndice2, nIndice3, nABouger
TABLEAU ENTIER nTab[10]

chargTableau                                    // appelle d'une procédure de chargement du tableau

POUR nIndice1 ALLANT DE 1 A 9
        SI (nTab(nIndice1+1) < nTab(nIndice1))
                abouger ⇐  nTab(nIndice1+1);
                nIndice2 ⇐ 1;
                TantQue ((nIndice2 < nIndice1) ET (nTab(nIndice2) < nTab(nIndice1+1)))
                        nIndice2 ⇐ nIndice2+1
                FINTQ
                POUR nIndice3 ALLANT DE nIndice1+1 A  nIndice2+1   PAS -1
                        nTab(nIndice3) ⇐ nTab(nIndice3-1)
                FinPour
                nTab(nIndice2) ⇐ abouger;
        FINSI
FinPour
```

OU BIEN

```
POUR nIndice1 ALLANT DE 2 A 10
        SI (nTab(nIndice1) < nTab(nIndice1 - 1))
                nIndice2 ⇐ nIndice1
                FAIRE
                                x⇐ nTab(nIndice2-1)
                                nTab(nIndice2-1) ⇐ nTab(nIndice2)
                                nTab(nIndice2) ⇐ x
                                nIndice2 ⇐ nIndice2-1
                JUSQUA  ((nIndice2  = 1) OU (nTab(nIndice2 - 1) < nTab(nIndice2)) )
        FINSI
FINPOUR
```

5.11.4. le tri par comptage

Le tri par comptage consiste pour chaque élément du tableau à compter combien d'éléments sont plus petits que lui, grâce à ce chiffre on connaît sa position dans le tableau résultat.

	52	10	1	25	62	3	8	55	3	23
nbre de plus petit	7	4	0	6	9	1	3	8	1	5
position	8	5	1	7	10	2	4	9	3	6

1	3	3	8	10	23	25	52	55	62

Corrigé

```
ENTIER nIndice1, nIndice2
TABLEAU ENTIER nTab[10], nResultat[10], nPosition[10]

ChargTableau                                    // appelle d'une procédure de chargement du tableau

POUR nIndice1 ALLANT DE 1 A 10
        nResultat( nIndice1 ) ⇐0
        nPosition( nIndice1 ) ⇐ 0
/* calcul des compteurs */
        POUR nIndice2 ALLANT DE 1 A 10
                SI  nTab( nIndice2 ) < nTab( nIndice1 )
                        nPosition( nIndice1 ) ⇐ nPosition ( nIndice1 ) + 1
                FINSI
        FinPour
FinPour

POUR nIndice1 ALLANT DE 1 A 10
        nIndice2 ⇐ nPosition ( nIndice1 )
        TantQue nResultat(nIndice2) <> 0            /* pour le cas des doubles */
                nIndice2 ⇐ nIndice2+1
        FINTQ
        nResultat(nIndice2) ⇐ nTab( nIndice1 )
FinPour
```

5.11.5. Le tri alphabétique

Le programme consiste à saisir des mots (au maximum 10) de 20 caractères maximum et de les insérer dans un tableau dans l'ordre alphabétique. Puis d'afficher ensuite ce tableau.
Le tableau résultat est du type **TABLEAU CAR** [10,20].

A vos neurones

Corrigé

```
ENTIER nbMots                   //  nombre de mots saisis
ENTIER nbLignes = 10            //  nombre maxi de saisies possibles
ENTIER nbColonnes = 20          // taiile maxi des mots saisis

TABLEAU CAR  strLettre[nbLignes ][ nbColonnes ]  // tableau pour le tri à 2 dimensions
TABLEAU CAR strMot [nbColonnes ]                 // tableau pour mot saisi

ENTIER nLigne                   //  pointeur se déplaçant de ligne en ligne
ENTIER nColonne                 //  pointeur se déplaçant de colonne en colonne
ENTIER nIndice, nLongMot
BOOLEEN bPlusPetit

ChargTableau                            // appelle d'une procédure de chargement du tableau

POUR nbmots ALLANT DE 1 A nbLignes
        AFFICHER "entrer le mot suivant"
        LIRE strMot
        bPlusPetit ⇐ VRAI
        nLigne ⇐ 1
        TANTQUE (bPlusPetit = vrai ET (nLigne < nbmots) )
                nColonne⇐ 1
                TantQue ((strLettre(nLigne, nColonne) = strMot(nColonne))
                                ET nColonne <= 20)
                        nColonne ⇐ nColonne +1
                FINTQ
                SI (strLettre(nLigne, nColonne) < strMot(nColonne))
                ALORS
                        nLigne ⇐nLigne + 1
                SINON
                        bPluspetit ⇐ FAUX
                FINSI
        FINTQ
        SI (nLigne < nbMots   ET nbMots <> 0
                POUR nIndice ALLANT DE nbmots A nLigne PAS -1
                        POUR nColonne ALLANT DE 1 A nbColonnes
                          strLettre(nIndice, nColonne) ⇐ strLettre(nIndice -1, nColonne)
                        FinPour
                FinPour
        FINSI
```

```
                POUR nColonne ALLANT DE 1 A 20
                        strLettre(nLigne,nColonne) ⇐ strMot(nColonne)
                FinPour
        FinPour

        POUR nLigne ALLANT de 1 A nbMots
                AFFICHER strLettre(nLigne)
        FinPour
```

5.12. Le calcul des heures

Le programme réalise l'addition de deux données exprimées en **HH :MM:SS** et affiche le résultat sous la même forme.

A vos neurones

Corrigé

```
ENTIER nIndice,,nHeureTotal, nMinuteTotal, nSecondeTotal
TABLEAU ENTIER nHeure[2], nMinute[2], nSeconde[2]

Pour nIndice allant de 1 a 2
                    AFFICHER "entrer l'heure"
                    LIRE nHeure(nIndice )
                    AFFICHER "entrer les minutes"
                    LIRE nMinute(nIndice )
                    AFFICHER "entrer les sécondes"
                    LIRE nSeconde(nIndice)
FinPour

nSecondeTotal ⇐ nSeconde( 1 )+ nSeconde ( 2 )
SI (nSecondeTotal >= 60)
        nSecondeTotal ⇐ nSecondeTotal -  60
        nMinuteTotal⇐  1
SINON
        nMinuteTotal⇐  0
FINSI
nMinuteTotal ⇐ nMinuteTotal+  nMinute ( 1 )+ nMinute ( 2 )
SI (nMinuteTotal >= 60)
        nMinuteTotal ⇐ nMinuteTotal -  60
        nHeureTotal⇐  1
SINON
        nHeureTotal⇐  0
nHeureTotal ⇐ nHeureTotal +  nHeure ( 1 )+ nHeure ( 2 )

AFFICHER H(1) ":" M(1) ":" S(1)
AFFICHER "+"
AFFICHER H(2) ":" M(2) ":" S(2)
AFFCHER "="
AFFICHER "Le résultat est" + nHeuresTotal +  " heures "
                                + nMinuteTotal +  " minutes et "
                                 + nSecondeTotal " secondes "
```

5.13. Le jeu du pendu

Écrire le programme du jeu du pendu.
Le principe est le suivant :
Un premier joueur choisit un mot de moins de 10 lettres.
Le programme affiche _ _ _ _ _ _ _ _ avec un _ par lettre.
Le deuxième joueur propose des lettres jusqu'à ce qu'il ait trouvé le mot ou qu'il soit pendu (11 erreurs commises).
A chaque proposition le programme réaffiche le mot avec les lettres découvertes ainsi que les lettres déjà annoncées et le nombre d'erreurs.

En deuxième partie:
Réécrire le jeu du pendu en utilisant des fonctions et/ou procédures.

A vos neurones

Corrigé

```
TABLEAU CAR cMot[10], cDevine[10]
TABLEAU CAR cLettres[26]
ENTIER nblettres, nbechecs, nIndice1, nbtrouves
BOOLEEN bGagne, bAbsent

AFFICHER "entrer le mot a deviner"
LIRE cMot
 //  recherche longueur du mot à trouver
i ⇐ 1
TANTQUE (cMot(nIndice1) <> ' ')
        nIndice1 ⇐ nIndice1+1
FINTQ
// met des tirets pour chaque lettre du mot à trouver
nblettres ⇐ nIndice1 - 1;
POUR nIndice1 ALLANT DE 1  A nblettres
        devine(nIndice1) ⇐ '_'
FinPour
// analyse si lettre est présente dans le mot à trouver
```

```
nbEchecs ⇐0
nbtrouves  ⇐ 0
bGagne ⇐ faux
TANTQUE  nbEchecs < 11 ET bGagne = faux
        AFFICHER devine
        AFFICHER lettres  "nombre d'echecs " nbechecs
        AFFICHER "entrer une lettre"
        LIRE cLettre
        lettres(nbtrouve+nbechecs+1) ⇐ cLettre
        bAbsent ⇐VRAI
        POUR nIndice1 ALLANT DE 1 A nblettres
                SI mot(nIndice1) = cLettre
                        devine(nIndice1) = mot(nIndice1)
                        nbtrouves ⇐ nbtrouves + 1
                        bAbsent ⇐FAUX
                        SI (nbtrouve = nblettres)
                                bGagne ⇐ VRAI
                        FINSI
                FINSI
        FinPour
        SI (bAbsent) = vrai
                nbechecs ⇐ nbechecs + 1
        FINSI
FINTQ
SI (gagne)
        AFFICHER "vous avez gagne"
SINON
        AFFICHER "vous êtes pendu" FINSI
```

5.14. Le crible d'Erathostène

Cet algorithme permet d'afficher progressivement la liste des nombres premiers inférieurs à une valeur donnée : **MAX**.

Pour ce faire, on construit un tableau de **MAX** éléments, vide au départ, que l'on parcourt.
Chaque fois que la case est vide cela signifie que l'indice du tableau est un nombre premier, on l'affiche puis on remplit avec une valeur quelconque toutes les cases du tableau indicées par un multiple de l'indice courant.

exemple pour **MAX** = 10
tableau au départ = 0 0 0 0 0 0 0 0 0 0
indice :

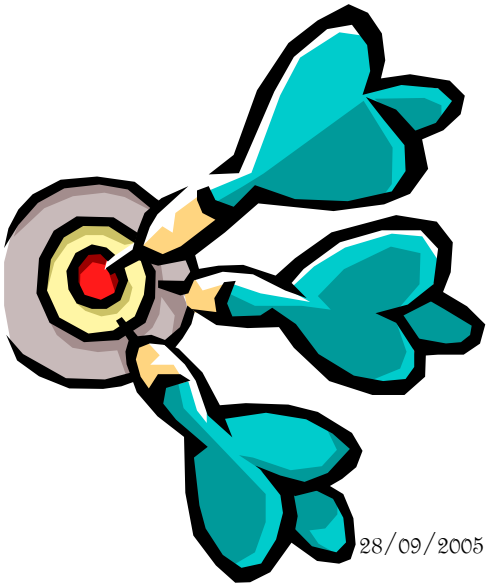
1 1 est un nombre premier (je ne marque rien !)
2 2 est un nombre premier ==>je marque 1 pour tableau (2)
je marque 1 pour tableau (2 + 2)
je marque 1 pour tableau (2 +2 +2)

tableau = 0 1 0 1 0 1 0 1 0 1
3 3 est un nombre premier ==>je marque 1 pour tableau(3)
je marque 1 pour tableau (3 + 3)
je marque 1 pour tableau (3 +3 +3)

tableau = 0 1 1 1 0 1 0 1 1 1
4 4 n'est pas un nombre premier car il est déjà à 1
5 5 est un nombre premier ==> je marque etc....

Écrire un programme qui demande un nombre et affiche tous les nombres premiers inférieurs au nombre donné.

A vos neurones



6. Quelques corrigés en langage Java

Remarque : ces classes programmes sont perfectibles : il y manque l'aspect contrôle des informations saisies volontairement ignoré ici.
D'autre part le \r est parfois à remplacé par \n si Jbuilder 4 et plus.

6.1. Alternative

```
package TestAlgo;
import java.lang.*;

public class Alternative
{
   public static String lire() throws java.io.IOException
          {
                String strSaisie = "";
                char C;
                while((C=(char)System.in.read()) != '\r')   // ou \n
                 {
      if (C != '\n')
         {
                        strSaisie = strSaisie + C;
         }
                 }
                return strSaisie;
       }
   public static int convertStringInt(String strEntier)
   {
      Integer intConv = new Integer(strEntier);
      int nConv = intConv.intValue();
           return nConv;
   }
   public static float convertStringFloat(String strEntier)
   {
      Integer intConv = new Integer(strEntier);
      float nConv = intConv.floatValue();
           return nConv;
```

```
    }

    /**
    * Point d'entrée principal de l'application.
    */
            public static void main (String[] args) throws java.io.IOException
            {
                    String strRepMarie;
    int    nEnfants;
    double nParticipation;
    float  nSalaire;

    System.out.println("L'employ\u201A est-il mari\u201A  (O/N) ?");
....strRepMarie = lire();
    System.out.println("Combien a-t-il d'enfants ?");
                    String strEnfants = lire();
    nEnfants = convertStringInt(strEnfants);
     System.out.println("Quel est son salaire ?");
   String strSalaire = lire();
    nSalaire = convertStringFloat(strSalaire);

    if (strRepMarie.equals("N") == true)
    {
       nParticipation = 0.2;
    }
    else
       nParticipation = 0.25;
    nParticipation = nParticipation + (nEnfants * 0.1);
    if (nSalaire <= 6000)
       nParticipation = nParticipation + 0.1;
    if (nParticipation > 0.5)
    {
      nParticipation = 0.5;
          }
    System.out.println("la participation est de : " + nParticipation);
    System.out.println("taper n'importe quoi pour finir");
    String strBidon = lire();
    }
}
```

6.2. Alternative2

```
package TestAlgo;
import java.lang.*;

public class Alternative2
{
   public static String lire() throws java.io.IOException
           {
                   String strSaisie = "";
                   char C;
                   while((C=(char)System.in.read()) != '\r')  //  ou \n
                    {
        if (C != '\n')
           {
                           strSaisie = strSaisie + C;
           }
                    }
                   return strSaisie;
         }

   public static int convertStringInt(String strEntier)
   {
      Integer intConv = new Integer(strEntier);
      int nConv = intConv.intValue();
            return nConv;
   }
   public static float convertStringFloat(String strEntier)
   {
      Integer intConv = new Integer(strEntier);
      float nConv = intConv.floatValue();
            return nConv;
   }

  /**
  * Point d'entrée principal de l'application.
  */
         public static void main (String[] args) throws java.io.IOException
         {
                   String strRepMarie;
  int    nEnfants;
```

```
double nParticipation;
float  nSalaire;
boolean bSuite;

bSuite = true;
while (bSuite == true)
{
                System.out.println("L'employ\u201A est-il mari\u201A ?  (O/N) ?");
                   strRepMarie = lire();

   System.out.println("Combien a-t-il d'enfants ?");
                      String strEnfants = lire();
   nEnfants = convertStringInt(strEnfants);


                System.out.println("Quel est son salaire ?");
      String strSalaire = lire();
   nSalaire = convertStringFloat(strSalaire);

   if (strRepMarie.equals("N") == true)
   {
     nParticipation = 0.2;
   }
   else
     nParticipation = 0.25;
   nParticipation = nParticipation + (nEnfants * 0.1);
   if (nSalaire <= 6000)
     nParticipation = nParticipation + 0.1;
   if (nParticipation > 0.5)
     nParticipation = 0.5;

System.out.println("la participation est de : " + nParticipation);
   System.out.println("Voulez-vous continuer ? O / N  ?");
   String strBidon = lire();
   if (strBidon.equals("N") == true)
      bSuite = false;
  }
 }
}
```

6.3. Boucle

```
package TestAlgo;

public class Boucle
{
  public static String lire() throws java.io.IOException
        {
                String strSaisie = "";
                char C;
                while((C=(char)System.in.read()) != '\r') //  ou \n
                 {
                if (C != '\n')
                {
                        strSaisie = strSaisie + C;
                }
           }
        return strSaisie;
        }

   public static int convertStringInt(String strEntier)
   {
      Integer intConv = new Integer(strEntier);
      int nConv = intConv.intValue();
            return nConv;
   }
  /**
  * Point d'entrée principal de l'application.
  */
        public static void main (String[] args) throws java.io.IOException
        {
   double  nSomme;
   double  nMoyenne;
   int     nNombreFrappes;
   int     nSaisie;

   nSomme = 0;
   nNombreFrappes = 0;
   nSaisie = 99999;
```

```
while (nSaisie != 0)
{
   System.out.println("entrer un entier?");
   String strEntier = lire();
   nSaisie = convertStringInt(strEntier);

         if (nSaisie != 0)
         {
               nNombreFrappes  = nNombreFrappes + 1;
               nSomme  = nSomme + nSaisie;
         }
   else
   {
      nMoyenne  = (nSomme / nNombreFrappes);
      System.out.println("La somme = " + nSomme + " La moyenne = " + nMoyenne);
      System.out.println("taper n'importe quoi pour finir");
      String strBidon = lire();
   }
 }
}
}
```

6.4. Statistiques

```
package TestAlgo;
import java.lang.*;

public class Statistiques
{
  public static String lire() throws java.io.IOException
          {
                  String strSaisie = "";
                  char C;
                  while((C=(char)System.in.read()) != '\r')   // ou \n
                   {
        if (C != '\n')
        {
                  strSaisie = strSaisie + C;
        }
                   }
                  return strSaisie;
          }

   public static int convertStringInt(String strEntier)
   {
      Integer intConv = new Integer(strEntier);
      int nConv = intConv.intValue();
            return nConv;
   }
  /**
   * Point d'entrée principal de l'application.
   */
          public static void main (String[] args) throws java.io.IOException
          {
   int     nMini;
   int     nMaxi;
   int     nSomme;
   double   nMoyenne;
   int     nIndice;
   int     nbNombre;
   int     nSaisie;
```

```
int     nTab[] = new int[50];

System.out.println("entrer un entier?");
String strEntier = lire();
nSaisie = convertStringInt(strEntier);

nbNombre = 0;
while (nSaisie != 0 & nbNombre < 50)
{
        if (nSaisie != 0)
        {
                nbNombre = nbNombre + 1;
                nTab[nbNombre] = nSaisie;
        }
   System.out.println("entrer un entier?");
   strEntier = lire();
   nSaisie = convertStringInt(strEntier);
}
nSomme = 0;
nMini  = nTab[1];
nMaxi  = nTab[1];
for (nIndice = 1; nIndice <= nbNombre; nIndice++)
{
     nSomme = nSomme + nTab[nIndice];
     if (nTab[nIndice] < nMini)
        nMini = nTab[nIndice];

     if (nTab[nIndice] > nMaxi)
        nMaxi = nTab[nIndice];
}
nMoyenne  = ((double)nSomme / nbNombre);
 System.out.println("La somme = " + nSomme + " La moyenne = " + nMoyenne);
System.out.println("Le mini =  " + nMini  + " Le maxi =    " + nMaxi);
System.out.println("taper n'importe quoi pour finir");
String strBidon = lire();
 }
}
```

6.5. Contraction

```
package TestAlgo;
public class Contraction
{
  public static void main (String[] args) throws java.io.IOException
  {
   String strSaisie;
   char[] strPhrase1 = new char[100];
   byte[] strResultat = new byte[100];
   char   strOctet;
   char   strFin = '.';
   int    nIndice1;
   int    nIndice2;

   System.out.println("Saisir la phrase termin\u201Ae par un point");
   strSaisie = Alternative.lire();
   strPhrase1 = strSaisie.toCharArray();
   System.out.println("Saisir la lettre à retirer");
   strSaisie = Alternative.lire();
   strOctet  = strSaisie.charAt(0);
   nIndice1 = 0;
   nIndice2 = 0;
   while (strPhrase1[nIndice1] != strFin )
   {
      if (strPhrase1[nIndice1] != strOctet)
      {
         strResultat[nIndice2] = (byte)strPhrase1[nIndice1];
         nIndice2 = nIndice2 + 1;
      }
      nIndice1 = nIndice1 + 1;
   }
   String strAffiche = new String(strResultat);
   System.out.println("Le r\u201Asultat est   " + strAffiche);
   System.out.println("taper n'importe quoi pour finir");
   String strBidon = Alternative.lire();
  }
}
```

6.6. Doublons

```
package TestAlgo;
public class Doublons
{
  public static void main (String[] args) throws java.io.IOException
  {
   String strSaisie;
   char[] strPhrase1 = new char[100];
   byte[] strResultat = new byte[100];
   char   strOctet;
   char   strFin = '.';
   int    nIndice1;
   int    nIndice2;
   System.out.println("Saisir la phrase termin\u201Ae par un point");
   strSaisie = Alternative.lire();
   strPhrase1 = strSaisie.toCharArray();
   strOctet  = ' ';
   nIndice1 = 0;
   nIndice2 = 0;
   while (strPhrase1[nIndice1] != strFin )
   {
      if (strPhrase1[nIndice1] != strOctet)
      {
         strResultat[nIndice2] = (byte)strPhrase1[nIndice1];
         nIndice2 = nIndice2 + 1;
         strOctet = strPhrase1[nIndice1];
      }
      nIndice1 = nIndice1 + 1;
   }
   String strAffiche = new String(strResultat);
   System.out.println("Le r\u201Asultat est   " + strAffiche);
   System.out.println("taper n'importe quoi pour finir");
   String strBidon = Alternative.lire();
  }
}
```

6.7. Equivalence

```
package TestAlgo;
public class Equivalence
{
  public static void main (String[] args) throws java.io.IOException
  {
   String strSaisie;
   char[] strPhrase1 = new char[100];
   char[] strPhrase2 = new char[100];
   char   strOctet;
   char   strFin = '.';
   int    nIndice1;
   int    nIndice2;
   boolean bOk;
   int    nLong1;
   int    nLong2;

   System.out.println("Saisir la premi\u0160re phrase termin\u201Ae par un point");
   strSaisie = Alternative.lire();
   nLong1 = strSaisie.length();
   strPhrase1 = strSaisie.toCharArray();

   System.out.println("Saisir la seconde phrase termin\u201Ae par un point");
   strSaisie = Alternative.lire();
   nLong2 = strSaisie.length();
   strPhrase2  = strSaisie.toCharArray();

   bOk = true;
   nIndice1 = 0;
   nIndice2 = 0;
```

```
        while (bOk == true & nIndice1 < nLong1 & nIndice2 < nLong2)
        {
            strOctet = strPhrase1[nIndice1];
            if (strPhrase2[nIndice2] == strOctet)
            {
        // utilisation de && pour sortir des que < nLong2 sans évaluer le second terme
                while( nIndice2 < nLong2 && strPhrase2[nIndice2] == strOctet )
                {
                    nIndice2 = nIndice2+1;
                }
            }
            else
                bOk = false;

           while(nIndice1 < nLong1 && strPhrase1[nIndice1] == strOctet )
           {
              nIndice1 = nIndice1+1;
           }

        }
        if (bOk)
            System.out.println("les phrases sont \u201Aquivalentes");
        else
            System.out.println("les phrases ne sont pas \u201Aquivalentes");

        System.out.println("taper n'importe quoi pour finir");
        String strBidon = Alternative.lire();

      }
    }
```

6.8. Eparpillement

```
public class Eparpillement
{
  public static void main (String[] args) throws java.io.IOException
  {
   String strSaisie;
   char[] strPhrase1  = new char[100];
   char[] strPhrase2  = new char[100];
   char    strFin = '.';
   int     nIndice1;
   int     nIndice2;
   int     nLong1;
   int     nLong2;

   System.out.println("Saisir la phrase termin\u201Ae par un point");
   strSaisie = Alternative.lire();
   nLong1 = strSaisie.length();
   strPhrase1 = strSaisie.toCharArray();

   System.out.println("Saisir le mot recherch\u201A");
   strSaisie = Alternative.lire();
   nLong2 = strSaisie.length();
   strPhrase2  = strSaisie.toCharArray();

   nIndice1 = 0;
   nIndice2 = 0;
   while (nIndice2 < nLong2 & nIndice1 < nLong1)
   {
      if (strPhrase1[nIndice1] == strPhrase2[nIndice2])
      {
         nIndice2 = nIndice2 + 1;
      }
      nIndice1 = nIndice1 + 1;
   }
```

```
        if (nIndice2 < nLong2)
            System.out.println("le r\u201Asultat est faux");
        else
            System.out.println("le r\u201Asultat est vrai");

        System.out.println("taper n'importe quoi pour finir");
        String strBidon = Alternative.lire();
    }
}
```

6.9. Inversion

```
public class Inversion
{
  public static void main (String[] args) throws java.io.IOException
  {
   String strNomPrenom;
   char[] strOctet = new char[30];
   char[] strNom   = new char[30];

   int     nIndice;
   int     nCurseur;
   int     nCurseur2;
   int     nFinPrenom;
   int     nDebutPrenom;

   System.out.println("Entrez un nom et un pr\u201Anom s\u201Apar\u201A par un espace");
   strNomPrenom = Alternative.lire();
   nCurseur = strNomPrenom.length()- 1;
   strOctet = strNomPrenom.toCharArray();

// recherche de la fin du prénom
   while (nCurseur >= 0 & strOctet[nCurseur] == ' ')
   {
      nCurseur--;
   }
   if (nCurseur == 0)
   {
      System.out.println("Vous n'avez rien saisi");
   }
   else
   {
      nFinPrenom = nCurseur;
      // recherche début du prénom
```

```
            while (nCurseur >= 0 & strOctet[nCurseur] != ' ')
            {
               nCurseur--;
            }
            if (nCurseur == 0)
            {
                System.out.println("Vous n'avez saisi que le pr\u201Anom");
            }
            else
            {
               nDebutPrenom = nCurseur +1;
               nCurseur2    = 0;
               for (nIndice= nDebutPrenom; nIndice <= nFinPrenom; nIndice++)
               {
                  if (strOctet[nIndice] == '-')
                     strNom[nCurseur2] = ' ';
                  else
                     strNom[nCurseur2] = strOctet[nIndice];
                  nCurseur2++;
               }
               nCurseur = nDebutPrenom - 2;
               strNom[nCurseur2] = ' ';
               for (nIndice = 0; nIndice <= nCurseur; nIndice++)
               {
                  nCurseur2++;
                  strNom[nCurseur2] = strOctet[nIndice];
               }
            }
         }
         strNomPrenom = new String(strNom);
         System.out.println("le résultat est : '" + strNomPrenom);
         System.out.println("taper n'importe quoi pour finir");
         String strBidon = Alternative.lire();
      }
   }
```

6.10. Palindrome

```
public class Palindrome
{
  public static void main (String[] args) throws java.io.IOException
  {
   String strSaisie;
   char[] strPhrase1 = new char[100];
   char   strOctet;
   char   strFin = '.';
   int    nIndice1;
   int    nIndice2;
   boolean bOk;

   System.out.println("Saisir la phrase termin\u201Ae par un point");
   strSaisie = Alternative.lire();
   nIndice2 = strSaisie.length() - 2;
   strPhrase1 = strSaisie.toCharArray();

   bOk = true;
   nIndice1 = 0;
   while (bOk && nIndice1 <= nIndice2)
   {
      if (strPhrase1[nIndice1] == ' ')
      {
         while (strPhrase1[nIndice1] == ' ' && nIndice1 <= nIndice2)
         {
            nIndice1++;
         }
      }

      if (strPhrase1[nIndice2] == ' ')
      {
         while (strPhrase1[nIndice2] == ' ' && nIndice1 <= nIndice2)
         {
            nIndice2--;
         }
      }
```

```
            if (strPhrase1[nIndice1] == strPhrase1[nIndice2])
            {
               nIndice1++;
               nIndice2--;
            }
            else
               bOk = false;
         }
         if (bOk)
            System.out.println("la phrase est un palindrome");
         else
            System.out.println("la phrase n'est pas un palindrome");

         System.out.println("taper n'importe quoi pour finir");
         String strBidon = Alternative.lire();
      }
   }
```

6.11. Cryptage

```
public class Cryptage
{
   public static void main (String[] args) throws java.io.IOException
   {
   int nIndice1;
   int nIndice2;
   int nRang;

   String strSaisie;
   char strPhrase1[]  = new char[100];
   char strAlphabet[] = new char[26];
   char strResultat[] = new char[100];
   char    strFin = '.';

   System.out.println("Saisir la phrase termin\u201Ae par un point");
   strSaisie = Alternative.lire();
   strPhrase1 = strSaisie.toCharArray();

   strSaisie   = "abcdefghijklmnopqrstuvwxyz";
   strAlphabet = strSaisie.toCharArray();

   nIndice1 = 0;
   nRang    = 1;

   while (strPhrase1[nIndice1] != strFin)
   {
      if (strPhrase1[nIndice1] != ' ')
      {
         nIndice2 =0;
         while (strPhrase1[nIndice1] != strAlphabet[nIndice2])
         {
            nIndice2++;
         }
```

```
                nIndice2 = nIndice2 + nRang;
                if (nIndice2 > 25)
                    nIndice2 = nIndice2 - 26;
                strResultat[nIndice1] = strAlphabet[nIndice2];
            }
            else
            {
                strResultat[nIndice1] = ' ';
                nRang++;
            }
            nIndice1++;
        }
        strSaisie = new String(strResultat);
        System.out.println("le r\u201Asultat est : " + strSaisie);

        System.out.println("taper n'importe quoi pour finir");
        String strBidon = Alternative.lire();
        }
    }
```

6.12. Comptage

```
public class Comptage
{
public static void main (String[] args) throws java.io.IOException
{
   String strSaisie;
   char[] strPhrase1 = new char[100];
   char[] strTermine = new char[10];
   char[] strMot     = new char[26];

   char   strFin = '.';
   int    nIndice1;
   int    nIndice2;
   int    nIndice3;
   int    nIndice4;
   int    nLongPhrase;
   int    nLong;
   int    nLongMot;
   int    nCpt;
   boolean bOk;

   System.out.println("Saisir une phrase termin\u201Ae par un point");
   strSaisie   = Alternative.lire();
   nLongPhrase = strSaisie.length();
   strPhrase1  = strSaisie.toCharArray();
   System.out.println("Entrer la terminaison");
   strSaisie   = Alternative.lire();
   nLong       = strSaisie.length();
   strTermine  = strSaisie.toCharArray();

   nCpt = 0;
   nIndice1 = 0;
   nLongMot = 25;
  while (nIndice1 < nLongPhrase && strPhrase1[nIndice1] != strFin)
   {
      for (nIndice2 = 0; nIndice2 < nLongMot ; nIndice2++)
      {
```

```
                    strMot[nIndice2] = ' ';
                }
                nIndice2 = 0;
                while(strPhrase1[nIndice1] != ' ' && strPhrase1[nIndice1] != strFin)
                {
                    strMot[nIndice2] = strPhrase1[nIndice1];
                    nIndice1++;
                    nIndice2++;
                }
                nLongMot = nIndice2;
                if (nLongMot > 0)
                {
                    if (nLong <= nLongMot)
                    {
                        bOk = true;
                        nIndice3 = 0;
                        int nDepart = nLongMot - nLong;
                        for (nIndice4 = nDepart;nIndice4 <= nLongMot-1;nIndice4++)
                        {
                            if (strTermine[nIndice3] != strMot[nIndice4])
                                bOk = false;
                            nIndice3++;
                        }
                        if (bOk) nCpt++;
                    }
                }
                nIndice1++;
            }
            String strFinMessage;
            if (nCpt > 1)       strFinMessage = "mots";
            else                strFinMessage = "mot";
            System.out.println("le r\u201Asultat est : " + nCpt + " " + strFinMessage);
            System.out.println("taper n'importe quoi pour finir");
            String strBidon = Alternative.lire();
        }
    }
}
```

6.13. Dichotomie

```
public class Dichotomie
{
    static int nLong  = 20;
    static int[] nTab = new int[nLong];
    public static String lire() throws java.io.IOException
    {
                    String strSaisie = "";
                    char C;
                    while((C=(char)System.in.read()) != '\r')   ou \n
                    {
                            if (C != '\n')
                    {
                    strSaisie = strSaisie + C;
            }
    }
    return strSaisie;
    }
    public static int convertStringInt(String strEntier)
    {
      Integer intConv = new Integer(strEntier);
      int nConv = intConv.intValue();
              return nConv;
    }
    public static void chargTableau() throws java.io.IOException
    {
      System.out.println("indiquer le nombre de poste");
      String strSaisie = lire();
      nLong = convertStringInt(strSaisie);
      strSaisie = "";
      int nIndCharg = 0;
           for (nIndCharg = 0;nIndCharg < nLong; nIndCharg++)
           {
                    System.out.println("indiquer un entier de la liste bien placé");
                    strSaisie = lire();
                     nTab[nIndCharg] = convertStringInt(strSaisie);
          }
    }
```

```
public static void main (String[] args) throws java.io.IOException
    {
  int nAInserer;
  int nDebut = 0;
  int nFin;
  int nbElement;
  int nPosition = 0;
  int nIndice;

  chargTableau();

  System.out.println("indiquer l'entier à inserer");
  String strSaisie = lire();
  nAInserer = convertStringInt(strSaisie);

  nFin = nLong;
  nbElement = nLong;
  if (nFin != 0)
  {
    while (nbElement > 2)
    {
      nPosition = nbElement / 2;
      int nbElt     = nbElement * 2;
      if (nPosition != nbElt)
        nPosition++;
      nPosition =  nPosition + nDebut -1;

      if (nTab[nPosition] < nAInserer)
      {
        nDebut = nPosition;
      }
      else
      {
        nFin = nPosition;
      }
      nbElement = nFin - nDebut + 1;
    }
```

```
            if (nTab[nDebut] < nAInserer)
            {
               nPosition = nDebut + 1;
            }
            else
            {
               nPosition = nDebut;
            }
            for (nIndice = nLong - 1; nIndice >= nPosition; nIndice--)
            {
               nTab[nIndice + 1] = nTab[nIndice];
            }
         }
         nTab[nPosition] = nAInserer;
           System.out.println("le r\u201Asultat final est : " );

         for (nIndice = 0; nIndice < nLong+1; nIndice++)
         {
          System.out.print(nTab[nIndice]);
          System.out.print('\t');
         }
         System.out.print('\n');
         System.out.println("taper n'importe quoi pour finir");
         String strBidon = Alternative.lire();
      }
   }
```

6.14. TriSelection

```
public class TriSelection
{
    static int nLong  = 10;
    static int[] nTab = new int[nLong];

    public static void chargTableau() throws java.io.IOException
            {
                    String strSaisie = "";
      int nIndCharg = 0;
           for (nIndCharg = 0;nIndCharg < 10; nIndCharg++)
                        {
          System.out.println("indiquer un entier");
          strSaisie = Alternative.lire();
          nTab[nIndCharg] = Alternative.convertStringInt(strSaisie);
      }
          //          return strSaisie;
          }
          public static void main (String[] args) throws java.io.IOException
          {

    int nIndice1;
    int nIndice2;
    int nPetit;
    int nPosition = 0;

    chargTableau();
    System.out.println("les r\u201Asultats interm\u201Adaires sont : " );
```

```
        for (nIndice1 = 0; nIndice1 < nLong - 1; nIndice1++)
        {
          nPetit = nTab[nIndice1];
          for (nIndice2 = nIndice1; nIndice2 < nLong; nIndice2++)
          {
              if ( nTab[nIndice2] < nPetit)
              {
                 nPetit = nTab[nIndice2];
                 nPosition = nIndice2;
              }
          }
          for (nIndice2 = nPosition; nIndice2 > nIndice1; nIndice2--)
          {
              nTab[nIndice2] = nTab[nIndice2 - 1];
          }
          nTab[nIndice1] = nPetit;

          for (nIndice2 = 0; nIndice2 < nLong; nIndice2++)
           {
           System.out.print(nTab[nIndice2]);
           System.out.print('\t');
           }
        }
//    String strBidon = new String(nTab);
        System.out.println("le r\u201Asultat final est : " );
        for (nIndice1 = 0; nIndice1 < nLong; nIndice1++)
        {
           System.out.print(nTab[nIndice1]);
           System.out.print('\t');
        }
        System.out.println("taper n'importe quoi pour finir");
        String strBidon = Alternative.lire();

      }
};
```

6.15. Tri Bulle

```
public class TriBulle
{
    static int nLong  = 10;
    static int[] nTab = new int[nLong];

    public static void chargTableau() throws java.io.IOException
    {
       String strSaisie = "";
       int nIndCharg = 0;
          for (nIndCharg = 0;nIndCharg < 10; nIndCharg++)
          {
           System.out.println("indiquer un entier");
           strSaisie = Alternative.lire();
           nTab[nIndCharg] = Alternative.convertStringInt(strSaisie);
       }
          //            return strSaisie;
          }
          public static void main (String[] args) throws java.io.IOException
          {

    int nIndice1;
    int nIndice2;
    int nTampon;
    boolean bInversion = true;

    chargTableau();
    System.out.println("les r\u201Asultats interm\u201Adaires sont : " );
```

```
        while (bInversion)
        {
           bInversion = false;
           for (nIndice1 = 0; nIndice1 < nLong - 1; nIndice1++)
           {
            if (nTab[nIndice1] > nTab[nIndice1 + 1])
            {
                 nTampon = nTab[nIndice1];
                 nTab[nIndice1] = nTab[nIndice1 + 1];
                 nTab[nIndice1 + 1] = nTampon;
                 bInversion = true;
            }
            for (nIndice2 = 0; nIndice2 < nLong; nIndice2++)
            {
                 System.out.print(nTab[nIndice2]);
                 System.out.print('\t');
            }
           }
        }
//    String strBidon = new String(nTab);
        System.out.println("le r\u201Asultat final est : " );
        for (nIndice1 = 0; nIndice1 < nLong; nIndice1++)
        {
           System.out.print(nTab[nIndice1]);
           System.out.print('\t');
        }
        System.out.println("taper n'importe quoi pour finir");
        String strBidon = Alternative.lire();
      }
};
```

6.16. TriPermutation

```
public class TriPermut
{
    static int nLong  = 10;
    static int[] nTab = new int[nLong];

    public static void chargTableau() throws java.io.IOException
     {
        String strSaisie = "";
        int nIndCharg = 0;
             for (nIndCharg = 0;nIndCharg < 10; nIndCharg++)
                          {
            System.out.println("indiquer un entier");
            strSaisie = Alternative.lire();
            nTab[nIndCharg] = Alternative.convertStringInt(strSaisie);
        }
            //            return strSaisie;
            }
            public static void main (String[] args) throws java.io.IOException
            {

    int nIndice1;
    int nIndice2;
    int nIndice3;
    int nABouger;

    chargTableau();
    System.out.println("les r\u201Asultats interm\u201Adaires sont : " );
```

```
        for (nIndice1 = 0; nIndice1 < nLong - 1; nIndice1++)
        {
          if (nTab[nIndice1 +1] < nTab[nIndice1])
          {
            nABouger = nTab[nIndice1+1];
            nIndice2 = 0;
            while (nIndice2 < nIndice1 && nTab[nIndice2] < nTab[nIndice1 + 1])
            {
                nIndice2++;
            }
            for (nIndice3 = nIndice1 + 1; nIndice3 > nIndice2; nIndice3--)
            {
                nTab[nIndice3] = nTab[nIndice3 - 1];
            }
            nTab[nIndice2] = nABouger;

            for (nIndice2 = 0; nIndice2 < nLong; nIndice2++)
            {
                System.out.print(nTab[nIndice2]);
                System.out.print('\t');
            }
          }
        }

        System.out.println("le r\u201Asultat final est : " );
        for (nIndice1 = 0; nIndice1 < nLong; nIndice1++)
        {
          System.out.print(nTab[nIndice1]);
          System.out.print('\t');
        }
        System.out.println("taper n'importe quoi pour finir");
        String strBidon = Alternative.lire();
      }
    }
```

6.17. Tri Comptage

```
public class TriComptage
{
   static int nLong  = 10;
   static int[] nTab = new int[nLong];

   public static void chargTableau() throws java.io.IOException
  {
      String strSaisie = "";
      int nIndCharg = 0;
         for (nIndCharg = 0;nIndCharg < 10; nIndCharg++)
         {
          System.out.println("indiquer un entier");
          strSaisie = Alternative.lire();
          nTab[nIndCharg] = Alternative.convertStringInt(strSaisie);
         }
  }
  public static void main (String[] args) throws java.io.IOException
  {
    int nIndice1;
    int nIndice2 = 0;
    int [] nResultat = new int[nLong];
    int [] nPosition = new int[nLong];

    chargTableau();
    for (nIndice1 =0; nIndice1 < nLong; nIndice1++)
    {
       nResultat[nIndice1] = 0;
       nPosition[nIndice1] = 0;
       for (nIndice2 = 0; nIndice2 < nLong ; nIndice2++)
       {
          if (nTab[nIndice2] < nTab[nIndice1])
          {
               nPosition[nIndice1] = nPosition[nIndice1] + 1;
          }
       }
    }
```

```
        for (nIndice1 =0; nIndice1 < nLong; nIndice1++)
        {
           nIndice2 = nPosition[nIndice1];

           while (nResultat[nIndice2] != 0)
           {
              nIndice2++;
           }
           nResultat[nIndice2]  = nTab[nIndice1];
        }

        System.out.println("le r\u201Asultat final est : " );
        for (nIndice1 = 0; nIndice1 < nLong; nIndice1++)
        {
           System.out.print(nResultat[nIndice1]);
           System.out.print('\t');
        }
        System.out.println("taper n'importe quoi pour finir");
        String strBidon = Alternative.lire();
     }
  }
```

6.18. Tri Alphabétique

```
public class TriAlpha
{
  public static void main (String[] args) throws java.io.IOException
  {
    int nbMots;
    int nbLignes = 4;
    int nbColonnes = 20;
    char [][] strLettre = new char[nbLignes][nbColonnes];
    char []   strMot    = new char[nbColonnes];
    int nLigne;
    int nColonne;
    int nIndice;
    int nLongMot;
    boolean bPlusPetit;

    for (nbMots = 0; nbMots < nbLignes; nbMots++)
    {
       System.out.println("Entrez le mot suivant");
       String strSaisie = Alternative.lire();
       strMot = strSaisie.toCharArray();
       nLongMot = strSaisie.length();
       bPlusPetit = true;
       nLigne = 0;
// test si même lettre dès que non ==> est-ce plus grand ?
       while (bPlusPetit && nLigne < nbMots)
       {
          nColonne = 0;
          while(strLettre[nLigne][nColonne]  == strMot[nColonne]
                && nColonne < nbColonnes)
          {
              nColonne++;
          }
          if (strLettre[nLigne][nColonne]  < strMot[nColonne])
          {
              nLigne++;
          }
```

```
                else
                {
                    bPlusPetit = false;
                }
            }
// décalage des lignes plus grandes
            if (nLigne < nbMots & nbMots != 0)
            {
              for (nIndice = nbMots; nIndice > nLigne ; nIndice--)
              {
                 for (nColonne =0; nColonne < nbColonnes; nColonne++)
                 {
                   strLettre[nIndice][nColonne] = strLettre[nIndice-1][nColonne];
                 }
              }
            }
// placement du mot à la ligne qui lui revient
            for (nColonne =0; nColonne < nLongMot ; nColonne++)
            {
               strLettre[nLigne][nColonne] = strMot[nColonne];
            }
            for (nColonne =nLongMot; nColonne < nbColonnes ; nColonne++)
            {
               strLettre[nLigne][nColonne] = ' ';
            }
        }
       System.out.println("le r\u201Asultat final est : " );
       for (nLigne = 0; nLigne < nbLignes; nLigne++)
       {
          String strSaisie = new String(strLettre[nLigne]);
          System.out.print("le mot en position " + (nLigne+1) + " est : ");
          System.out.print('\t');
          System.out.println(strSaisie);
       }
       System.out.println("taper n'importe quoi pour finir");
       String strBidon = Alternative.lire();
      }
}
```

Fin du document