ANNEE UNIVERSITAIRE 2002 – 2003
COURS : ALGORITHMIE ET PROGRAMMATION
AUTEUR : FABRICE GONTARD

TABLEAUX

1. ROLE D'UN TABLEAU

Un tableau est un outil qui permet de regrouper sous le même nom un ensemble de variables de même type.

Dans quel but ? Pour répondre à cette question, prenons un exemple que nous avons déjà analysé dans le cours précédent « Algorithmie et programme ». On demande à l'utilisateur de saisir une série de nombres entiers positifs ou négatifs. La saisie s'arrête lorsque l'utilisateur tape la valeur 0 (zéro). Lorsque la saisie est terminée, le programme affiche la somme des valeurs.

```
Programme FaireSomme
Var
    ValeurLue NOMBRE /* Valeur saisie au clavier */
    LaSomme : NOMBRE /* Contient la somme des valeurs lues */

Début
    LaSomme ← 0

    Afficher « Entrez une série de valeurs »
    Saisir ValeurLue

    Tant que ValeurLue <> 0 Faire
        LaSomme ← LaSomme + ValeurLue
        Saisir ValeurLue
    Fin tant que

    Afficher « La somme des valeurs saisies est », LaSomme
Fin
```

Modifions l'énoncé de ce programme. On souhaite maintenant afficher à la fin du programme la liste des valeurs saisies, en plus de la somme. (Pour simplifier le problème, nous allons nous limiter à la saisie de 5 valeurs).

Précédemment, il était possible d'utiliser une seule variable pour la saisie des valeurs, car à chaque tour de boucle nous ne souhaitons pas conserver la valeur saisie, elle pouvait être réutilisé pour une nouvelle saisie après l'opération de cumul. Dans notre deuxième cas, ce n'est plus possible, il faut déclarer autant de variables qu'il y a de valeurs à mémoriser, soit 5 variables. Voici le programme modifié.

```
Programme FaireSomme
Var
    Valeur1, Valeur2, Valeur3, Valeur4, Valeur5 : NOMBRE
    LaSomme : NOMBRE /* Contient la somme des valeurs lues */
```

```
Début
    LaSomme ← 0

    Afficher « Entrez la première valeur »
    Saisir Valeur1
    Afficher « Entrez la deuxième valeur »
    Saisir Valeur2
    Afficher « Entrez la troisième valeur »
    Saisir Valeur3
    Afficher « Entrez la quatrième valeur »
    Saisir Valeur4
    Afficher « Entrez la cinquième valeur »
    Saisir Valeur5

    LaSomme ← Valeur1 + Valeur2 + Valeur3 + Valeur4 + Valeur5

    Afficher « Voici la liste des valeurs saisies »
    Afficher Valeur1, Valeur2, Valeur3, Valeur4, Valeur5
    Afficher « et la somme des valeurs saisies est », LaSomme
Fin
```

Nous imaginons aisément que si nous devions écrire ce même programme pour 50 valeurs, il deviendrait vite complexe. En effet, il faudrait déclarer 50 variables, ce qui est impensable. Il faut donc utiliser un autre outil de l'algorithmie.

La solution consiste à réserver 50 emplacements mémoires différents, mais consécutifs, repérés par le même nom. Nous obtenons ainsi une variable unique, ne nécessitant qu'une déclaration simple et pouvant mémoriser un ensemble de valeurs différentes mais de même nature.

Il s'agit d'une variable **tableau**. Sans entrer immédiatement dans le détail, transformons le programme précédent en utilisant ce nouvel outil.

```
Programme FaireSomme
Var
   LesValeurs : Tableau (50) de NOMBRE /* Déclaration d'un tableau de 50 cases de type
                                          NOMBRE*/
   LaSomme : NOMBRE /* Contient la somme des valeurs lues*/
   Indice : NOMBRE /*Variable repérant une case dans le tableau*/
   ValLue : NOMBRE /*Contient la valeur saisie au clavier*/
Début
   LaSomme ← 0

   Afficher « Entrez 50 valeurs »
   Pour Indice ← 1 à 50
      Saisir ValLue
      LesValeurs [Indice] ← ValLue
      LaSomme ← LaSomme + LesValeurs[Indice]
   Fin pour
```

```
    Afficher « Voici la liste des valeurs saisies »
    Pour Indice ← 1 à 50
        Afficher LesValeurs[Indice]
    Fin pour
    Afficher « et la somme des valeurs saisies est », LaSomme
Fin
```

Nous voyons immédiatement que l'algorithme se simplifie en autorisant l'utilisation d'instructions de répétition (Pour) mais aussi au niveau des déclarations de variables, car nous n'avons pas eu à déclarer une longue.
Pour terminer notre exemple, il est possible de simplifier encore le programme en effectuant la saisie directement dans la case du tableau sans variable intermédiaire. Voici les instructions de la boucle **Pour** modifiée :

```
Pour Indice ← 1 à 50
        Saisir LesValeurs [Indice]
        LaSomme ← LaSomme + LesValeurs[Indice]
Fin pour
```

Remarque : Dans le programme précédent, nous avons utilisé une boucle **Pour** pour parcourir le tableau mais nous aurions pu également utiliser un **Tant que**. Nous aurions pu obtenu le résultat suivant :

```
Programme FaireSomme
Var
    LesValeurs : Tableau (50) de NOMBRE /* Déclaration d'un tableau de 50 cases de type
                                           NOMBRE*/
    LaSomme : NOMBRE /* Contient la somme des valeurs lues*/
    Indice : NOMBRE /*Variable repérant une case dans le tableau*/
    ValLue : NOMBRE /*Contient la valeur saisie au clavier*/
Début
    LaSomme ← 0

    Afficher « Entrez 50 valeurs »
    Indice ← 1
    Tant que Indice <= 50 Faire
        Saisir Les Valeurs[Indice]
        LaSomme ← LaSomme + LesValeurs[Indice]
        Indice ← Indice + 1
    Fin Tant que

    Afficher « Voici la liste des Valeurs saisies »
    Indice ← 1
    Tant que Indice <= 50 Faire
        Afficher LesValeurs[Indice]
        Indice ← Indice + 1
    Fin Tant que

    Afficher « et la somme des valeurs saisies est », LaSomme
Fin
```

2. DEFINITION D'UN TABLEAU

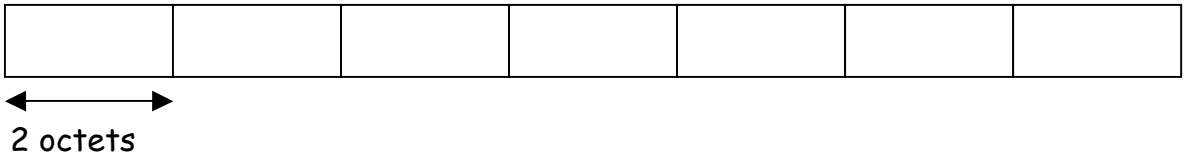
Un tableau est un ensemble d'éléments de même nature, placés de manière contiguë en mémoire. C'est le seul cas où l'on st sûr d'avoir des éléments contigus en mémoire. En effet, deux variables déclarées l'une après l'autre ne seront pas forcément placées de manière contiguë en mémoire.

Exemple : Soit la déclaration suivante.

LeTableau : Tableau (8) de NOMBRE.

L'utilisation déclarer un tableau de nom LeTableau possédant 8 cases de type NOMBRE, chacune de deux octets.

LeTableau



Remarques :

- Chaque case du tableau possède une taille identique, donnée par le type des éléments contenus dans le tableau ;
- Un tableau ne peut pas contenir des éléments de natures différentes. Tous les éléments d'un tableau sont forcément du même type. On parlera d'un tableau d'entiers, d'un tableau de caractères, etc. ;
- Un tableau possède une taille fixe, connue dès le départ de l'algorithme. Cette taille **ne pourra plus changer** au cours du déroulement de l'algorithme ;
- Un tableau peut ne pas être entièrement rempli, mais il ne pourra jamais contenir plus d'éléments que le nombre prévu lors de la déclaration.

Pour déclarer un tableau, il faut employer la syntaxe suivante :

NomTableau : Tableau (Taille) de TypeElément

Il faut donc préciser le nom du tableau, sa taille en plaçant l'information entre parenthèses et enfin le type d'éléments contenus dans chacune des cases du tableau.

Exemple : TabEntiers : Tableau (50) de NOMBRE

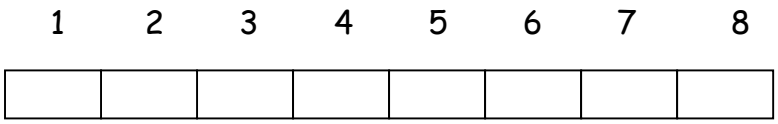
Nous déclarons un tableau de nom TabEntiers d'une taille de 50 cases, chaque case contenant un élément de type NOMBRE.

3. ATTEINDRE UNE CASE D'UN TABLEAU

Nous avons vu précédemment que toutes les cases d'un tableau portent le même nom. Par conséquent, la seule manière de les différencier est de leur attribuer un numéro en partant de 1 puis en incrémentant de un en un . les cases seront donc numérotées de 1 à n (N représentant la **dimension** du tableau). En fait, on dira plutôt qu'un tableau possède un ensemble d'**indices**. A chaque valeur de l'indice ne correspond qu'une et une seule case du tableau, donc un seul élément.

Reprenons notre déclaration :

LeTableau : Tableau (8) de NOMBRE



LeTableau

Le Tableau déclaré précédemment possède donc 8 cases numérotées (indicées) de 1 à 8.

Maintenant que nous avons défini la notion d'indice, pour atteindre l'une des cases d'un tableau, il suffit de préciser le nom ainsi que l'indice de la case voulue entre crochets.

Exemples :

LeTableau[2]	Indique la deuxième case du tableau LeTableau.
LeTableau[i]	Indique la ième case du tableau LeTableau. La valeur de i et l'indice maximal du tableau.
LeTableau[2] ← 5	La deuxième case du tableau LeTableau reçoit la valeur 5.
LaValeur ← LeTableau[2]	La variable LaValeur reçoit le contenu de la deuxième case du tableau LeTableau.
LeTableau[i] ← 105	La ième case du tableau LeTableau reçoit la valeur 105. La valeur de i doit être comprise entre 1 et l'indice maximal du tableau.
~~LeTableau[i] ← 'A'~~	Impossible car le tableau attend des nombres et non des caractères.

Remarques : Chaque langage de programmation possède sa propre convention pour indicer un tableau : le langage Pascal commence à indicer un tableau à partir de 1 ; le langage Basic à partir de 1 ou 0 ; le langage ADA à partir d'une valeur quelconque ; le langage C à partir de 0.

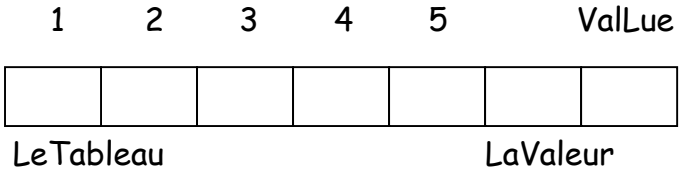
Attention : Le dépassement des limites d'un tableau n'est pas forcément contrôlé par tous les langages de programmation. En effet que se passe-t-il si je demande d'atteindre le vingtième case d'un tableau qui n'en contient que 8 ?

La plupart des langages contrôlent ce dépassement et déclenchent une erreur qui généralement arrête le programme, mais le langage C, par exemple, n'effectue aucun contrôle.

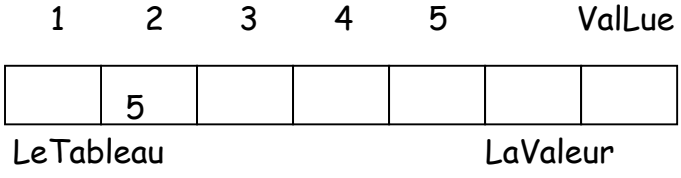
Exemple de problème sur la limite des tableaux : Nous considérons que le langage utilisé ne fait aucun contrôle sur la limite des tableaux.

LeTableau : Tableau (5) de NOMBRE
LaValeur : NOMBRE
ValLue : NOMBRE

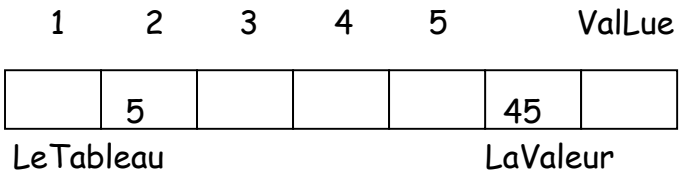
Supposons que l'emplacement mémoire des variables LaValeur et ValLue se trouve juste derrière le tableau. Ce cas est, en effet, tout à fait envisageable.



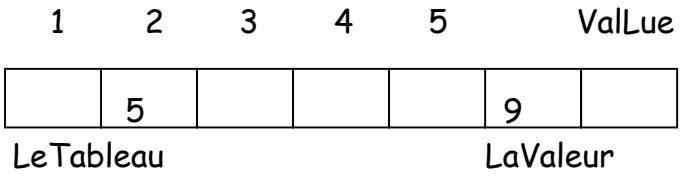
LeTableau[2] ← 5
La deuxième case du tableau LeTableau reçoit la valeur 5.



LaValeur ← 45
La variable LaValeur reçoit la valeur 45.



LeTableau[6] ← 9
La sixième case du tableau LeTableau reçoit la valeur 9.



La variable LaValeur a été modifiée parce que malheureusement son emplacement en mémoire est juste derrière le tableau et qu'il n'y a pas de contrôle.

Que vaut maintenant la variable LaValeur ?
Elle ne vaut plus la valeur que l'on s'attend à y trouver. Ce programme est difficilement debuggable car l'erreur ne se produit pas à chaque fois. Il suffit qu'un jour le système réserve l'emplacement de la variable LaValeur ailleurs et tout se passe bien.

C'est pourquoi, en algorithmie, il faut toujours s'habituer à contrôler les limites d'un tableau.

Exemple : On demande à l'utilisateur de saisir une série de nombres entiers positifs ou négatifs. A chaque saisie, on place la valeur lue dans une case d'un tableau déclaré auparavant. La saisie s'arrête lorsque l'utilisateur tape la valeur 0 (zéro) ou lorsque le tableau est rempli.

Lorsque la saisie est terminé, le programme affiche le contenu du tableau.

```
Programme RemplirTableau
Var
    LeTableau : Tableau (20) de NOMBRE
    ValeurLue : NOMBRE /* Contient la valeur lue */
    Indice : NOMBRE /* Permet de se déplacer dans le tableau */
    i NOMBRE /* Variable de boucle */
Début
    /* Initialiser la variable d'indice */
    Indice ← 1 /* Le tableau est indicé à partir de 1 */

    /* Saisir le contenu du tableau */
```

```
    Afficher « Entrez une liste de nombres (zéro pour terminer) : »
    Saisir ValeurLue

    Tant que ValeurLue <> 0 Et Indice <= 20 Faire
        LeTableau [Indice] ← ValeurLue /* On place la valeur dans le tableau */
        Indice ← Indice + 1 /* On passe à la casse suivante */
        Saisir ValeurLue
    Fin Tant que

    /* Afficher le résultat */
    Afficher « Le contenu du tableau est : »
    Pour i ← 1 à Indice –1
        Afficher LeTableau[i]
    Fin Pour
Fin
```

Remarque : Dans la boucle **Pour** permettant d'afficher le contenu du tableau, nous allons de 1 à **indice –1**, pourquoi ? La réponse est simple, au début du programme, nous avons initialisé la variable **Indice** à 1, c'est-à-dire à la valeur d'indice de la première case du tableau, au départ vide. Ensuite, à chaque fois que nous ajoutons un élément au tableau, nous incrémentons la variable **Indice**. En conclusion, elle contient toujours la valeur d'indice de la PROCHAINE case à remplir. En fait, elle a un temps d'avance. Donc la dernière case remplie est celle juste avant, à savoir **Indice –1**, d'où les valeurs des bornes de l'instruction **Pour**.

En complément de la remarque précédente, nous allons réécrire le programme **RemplirTableau** en initialisant la variable d'indice à 0 et voir les différences.

```
Programme RemplirTableauVersion2
Var
    LeTableau : Tableau (20) de NOMBRE
    ValeurLue : NOMBRE /* Contient la valeur lue */
    Indice : NOMBRE /* Permet de se déplacer dans le tableau */
    i NOMBRE /* Variable de boucle */
Début
    /* Initialiser la variable d'indice */
    Indice ← 0/* Le tableau est indicé à partir de 0*/

    /* Saisir le contenu du tableau */
    Afficher « Entrez une liste de nombres (zéro pour terminer) : »
    Saisir ValeurLue

    Tant que ValeurLue <> 0 Et Indice < 20 Faire
        Indice ← Indice +1 /* On se place sur la case à remplir */
        LeTableau[Indice] ← ValeurLue /* on place la valeur dans le tableau */
        Saisir ValeurLue
    Fin Tant que

    /* Afficher le résultat */
    Afficher « Le contenu du tableau est : »
    Pour i ← 1 à indice
```

```
        Afficher LeTableau[i]
    Fin pour
Fin
```

Dans cet exemple, nous avons initialisé la variable **Indice** à zéro. Ensuite, à chaque fois que nous ajoutons un élément au tableau, nous commençons par incrémenter la variable **Indice** avant de placer la valeur lue, afin de nous positionner sur la bonne case. En conclusion, la variable d'indice contient toujours la valeur d'indice de la dernière case remplie, c'est pourquoi les bornes de la boucle **Pour** d'affichage du contenu du tableau vont de 1 à **Indice**.

Remarque : Entre les deux exemples, il y a encore une autre différence, liée au choix de la valeur d'initialisation, c'est la condition du **Tant que** et plus précisément la condition du contrôle de dépassement du tableau.

4. NOTION DE CHAINE DE CARACTERES

Une chaîne de caractères n'est ni plus ni moins qu'un tableau de caractères. Elle se déclare de la manière suivante :

NomChaine : Tableau(Taille) de CARACTERE

Exemple :

LaChaine : Tableau(80) de CARACTERE

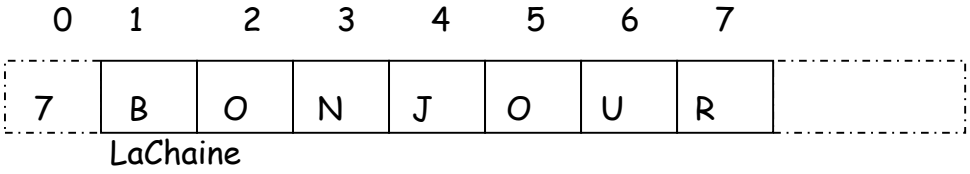
Cette déclaration réserve en mémoire un tableau de 80 caractères. Celui-ci a pour nom **LaChaine**.

Remarque : Certains langages de programmation possèdent un type spécifique pour les chaînes de caractères. Il s'agit en général du type **STRING**.

Une chaîne de caractères se différencie des autres types de tableau par le fait qu'il existe un mécanisme permettant de matérialiser la fin de la chaîne de caractères. Elle effet, ce n'est pas parce que déclarez un tableau de 80 caractères que vous devez forcément y saisir une phrase de 80 caractères. Elle peut en avoir moins mais jamais plus.

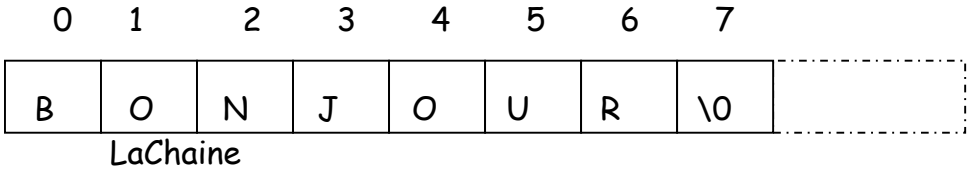
Dans le cas particulier des chaînes de caractères, les langages ont différentes manières de connaître la fin d'une chaîne de caractères. En fait, il existe deux mécanismes distincts :

En langage Pascal, le système ajoute une case d'indice 0 qui contient la taille de la chaîne de caractères.



Ceci n'autorise que des chaînes de caractères d'au maximum 255 caractères. En effet, la taille de chaque case du tableau est de 1 octet (car c'est un tableau de caractères). Donc chaque case ne peut contenir que des valeurs allant de 0 à 255, qui correspondent à des valeurs de la table ASCII, même la case d'indice 0.

En langage C, la fin de la chaîne de caractère est matérialisée par un caractère particulier, le \0 (dit **Backslash Zéro**) caractère de code ASCII 0.



Cela permet donc de créer des chaînes de caractères de n'importe quelle taille, non limitée à 255 caractères. Attention, il y a aucun contrôle du dépassement de la taille du tableau en C.
De plus, dans le dernier cas, il faut compter l'emplacement de la case qui contiendra la caractère **\0** dans la taille du tableau spécifié. Ainsi un tableau de 80 caractères ne permettra de contenir qu'au maximum 79 caractères.

Pour l'instant, dans le contexte de l'algorithme, nous ferons abstraction du problème précédent. Pour cela, nous allons considérer que nous disposons d'une fonction de non **Longueur ( )** qui retourne la taille de la chaîne de caractères passée en paramètre.

Exemple d'utilisation de la fonction Longueur ( ) :

LaChaine : Tableau (80) de CARACTERE
LaTaille : NOMBRE
LaChaine ← « Bonjour »
LaTaille ← Longueur (LaChaine)

La variable LaTaille contient donc la valeur 7.

Voici un programme complet utilisant cette nouvelle fonction.

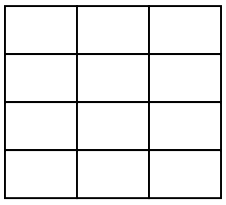
```
Programme CalculLongueur
Var
        LaChaine : Tableau (80) de CARCTERE /* Déclaration d'un tableau de 80 cases de
                                                Type CARACTERE */
        LaTaille : NOMBRE /* Contient la taille de la chaîne lue */
Début
        /* Saisir la chaîne de caractères */
        Afficher « Entrez une phrase »
        Saisir LaChaine

        /* Calculer la taille */
        LaTaille ← Longueur (LaChaine)

        /* Afficher la taille */
        Afficher « La taille de la chaîne est de » LaTaille, « caractère(s) »
Fin
```

5. TABLEAUX A PLUSIEURS DIMENSIONS

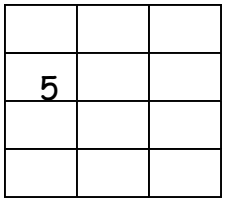
Pour déclarer un tableau à plusieurs dimensions, il faut utiliser la syntaxe suivante :

NomTableau : Tableau (Dimension1, Dimension2, …) de TypeElément

Exemple d'un tableau (4,3) de NOMBRE */* Matrice 4 lignes 3 Colonnes */*



Pour atteindre une case de ce tableau, il suffit de donner l'indice de la ligne et l'indice de la colonne à atteindre. Ainsi l'instruction **Matrice [2,1] = 5** permet de placer la valeur 5 dans la case se trouvant ligne 2, colonne1.



6. EXERCICES

6.1 Recherche de la plus grande valeur dans un tableau d'entiers

Enoncé : On demande à l'utilisateur de saisir une série de nombres entiers positifs ou négatifs. A chaque saisie, on place la valeur lue dans une case d'un tableau déclaré auparavant. La saisie s'arrête lorsque l'utilisateur tape la valeur 0 (zéro) ou lorsque le tableau est rempli.

Lorsque la saisie est terminé, le programme recherche la plus grande valeur ainsi que le nombre de fois où elle apparaît dans un tableau.

6.2 Répartition des valeurs lues entre deux tableaux

Enoncé : On dispose de deux tableaux **TabPositifs** et **TabNégatifs**. Le premier ne peut contenir que des valeurs positives et le deuxième que des valeurs négatives. On demande à l'utilisateur de saisir une série de nombres entiers positifs ou négatifs.

A chaque saisie, on place les valeurs positives dans le tableau des positifs et les valeurs négatives dans le tableau des négatifs. La saisie s'arrête lorsque l'utilisateur tape la valeur 0 (zéro) ou lorsque l'un des deux tableaux rempli.

Lorsque la saisie est terminée, le programme affiche le contenu des deux tableaux.

6.3 Répartition des valeurs positives et négatives dans un tableau unique

Enoncé : On demande à l'utilisateur de saisir une série de nombres entiers positifs ou négatifs. A chaque saisie, on place les valeurs négatives au début d'un tableau d'entiers et les valeurs positives dans le même tableau, mais à partir de la fin. La saisie s'arrête lorsque l'utilisateur tape la valeur 0 (zéro) ou lorsque le tableau est rempli.

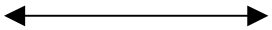
Lorsque la saisie est terminée, le programme affiche le contenu du tableau.

Remarque : Les listes des nombres négatifs et des nombres positifs n'ont pas besoin d'être triées. Les valeurs sont placées dans l'ordre de saisie.

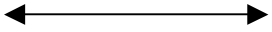
6.4 Palindrome

Enoncé : On veut afficher à l'écran si un mot lu au clavier est un palindrome ou non. La définition d'un palindrome est un mot qui se lire, de manière identique, dans les deux sens.

RADAR



ELLE



Principe : L'algorithme fait évoluer deux indices, l'un positionné au début du mot et l'autre à la fin du mot, de manière à comparer les deux caractères « pointés » par les deux indices. En cas d'égalité sur les deux caractères, on incrémente l'indice de début et on décrémente l'indice de fin, puis on recompare les deux caractères « pointés » et ainsi de suite.

L'algorithme s'arrête lorsqu'il trouve deux caractères différents ou lorsque les deux indices arrivent au milieu du mot. Si le mot a une longueur paire, l'algorithme s'arrête, si tout va bien jusque là, lorsque l'indice de début devient supérieur à l'indice de fin. En revanche, si le mot a une longueur impaire, l'algorithme s'arrête, si tout va bien jusque là, lorsque l'indice de début devient égal à l'indice de fin.

En résumé, on compare les caractères tant qu'ils sont égaux et que l'indice de début reste strictement inférieur à l'indice de fin.

6.5 Anagramme

Enoncé : On veut afficher à l'écran si deux mots lus au clavier sont des anagrammes ou non. Par définition, deux mots sont des anagrammes, si et seulement si, toutes les lettres du premier mot se retrouvent dans le deuxième mot et vice-versa.

CHIEN
NICHE

Principe : L'algorithme consiste à prendre une lettre dans le premier mot et à rechercher dans le deuxième mot. Si l'algorithme trouve cette lettre, il la remplace par une étoile pour éviter de la recompter une deuxième fois par la suite, lors d'une autre recherche, puis il recommence avec la lettre suivante du

premier mot. En revanche, si cette lettre n'est pas trouvée, l'algorithme doit s'arrêter.

Notion de flag ou drapeau : Un flag, encore appelé drapeau, est une variable de type BOOLEEN (Vrai/Faux) permettant de savoir si oui ou non un bloc d'instructions s'est exécuté. On peut ainsi optimiser le programme en autorisant une sortie avant terme d'une instruction **Tant que**.

6.6 Occurrence

Enoncé : On demande à l'utilisateur de saisir une phrase et une lettre. Le programme affiche le nombre de fois où la lettre apparaît dans la phrase. On compte donc le nombre d'occurrences de la lettre dans la phrase.

6.7 Suppression d'une valeur dans la liste de nombres

Enoncé : On demande à l'utilisateur de saisir une phrase et une lettre. Le programme supprime chaque occurrence du caractère dans la phrase.

Dans cette exercice, la caractère à supprimer est remplacé par un espace.

6.8 Suppression d'une valeur dans une liste de nombres

Enoncé : On demande à l'utilisateur de saisir une série de nombres entiers positifs ou négatifs. A chaque saisie, on place la valeur lue dans une case d'un tableau déclaré auparavant. La saisie s'arrête lorsque l'utilisateur tape la valeur 0 (zéro) ou lorsque le tableau est rempli.

Dans un deuxième temps, l'utilisateur saisit une valeur puis le programme recherche toutes les occurrences de cette valeur dans le tableau. Chaque occurrence est supprimée du tableau en avançant d'une case toutes les valeurs qui suivent.

6.9 Inversion d'un mot

Enoncé : L'utilisateur saisit un mot. Le programme inverse ensuite les caractères de ce mot. Attention, on n'utilise qu'un seul tableau pour réaliser l'opération.

Principe : L'algorithme va faire évoluer deux indices, l'u, positionné au début du mot et l'autre à la fin du mot, de manière à échanger le contenu des deux cases

« pointées » par les deux indices. Ensuite, on incrémente l'indice de début et on décrémente l'indice de fin, puis on recommence l'opération et ainsi de suite.

L'algorithme s'arrête lorsque les deux indices arrivent au milieu du mot. Si le mot a une longueur paire, l'algorithme s'arrête lorsque l'indice de début devient supérieur à l'indice de fin. En revanche, si le mot a une longueur impaire, l'algorithme s'arrête lorsque l'indice de début devient égal à l'indice de fin.

En résume, on effectue les échanges tant que l'indice de début reste strictement inférieur à l'indice de fin.

6.10 Inversion d'un mot avec deux tableaux

Enoncé : L'utilisateur saisit un mot. Le programme inverse ensuite les caractères de ce mot. Dans cette version, on utilise un deuxième tableau pour réaliser l'opération.

6.11 Transfert de valeurs

Enoncé : On demande à l'utilisateur de saisir deus séries de nombres entiers qui seront placées dans deux tableaux différents. La saisie dans l'une et l'autre des séries s'arrête lorsque l'utilisateur tape la valeur 0 (zéro) ou lorsque le tableau est rempli.

Dans un deuxième temps, le programme compare les deux valeurs situées au même indice dans les deux tableaux et place, dans un troisième tableau, la plus grande des deux. Cette opération se fera sur l'ensemble des valeurs des deux tableaux.

Si l'un des tableaux contient plus de valeurs que l'autre, on continue l'opération en plaçant dans le troisième tableau le reste des valeurs sans comparaison.

Exemple :

Premier tableau :	10 15 7 25 50 25 14 23
Deuxième tableau :	14 45 1 47 23 12 14 24 25 30 14 45
Troisième tableau :	14 45 7 47 50 25 14 24 25 30 14 45

6.12 Comparaison de deux chaînes de caractères

Enoncé : L'utilisateur saisit deux chaînes de caractères. Le programme affiche si oui ou non la deuxième est contenue dans la première. Attention , la deuxième chaîne de caractères peut être contenue dans un mot de la première chaîne de caractères.

Exemple :

Première phrase : Bonjour Monsieur
Deuxième phrase : jour

Réponse : La deuxième chaîne se trouve dans la première

6.13 Le mot le plus long

Enoncé : L'utilisateur saisit une chaîne de caractères. Le programme affiche le mot le plus long contenu dans la phrase.

Exemple :

Phrase saisie : Bonjour, bienvenue dans ce cour
Réponse : Le mot plus long est : bienvenue

6.14 Tri par insertion

Enoncé : On demande à l'utilisateur de saisir une série de nombres entiers ou négatifs. A chaque saisie, on insère la nouvelle valeur au bon endroit dans un tableau déclaré auparavant, de manière a ce que le contenu de ce dernier soit toujours trié. La saisie s'arrête lorsque l'utilisateur tape la valeur 0 (zéro) ou lorsque le tableau est rempli.

Principe : A chaque lecture d'un nouveau nombre, on parcourt les valeurs qui se trouvent déjà dans le tableau concerné jusqu'à trouver une valeur plus grande ou jusqu'à atteindre la fin de la liste des nombres déjà traités. Dès que l'on a trouvé une place, on décale toutes les valeurs nécessaires, de manières à libérer l'emplacement et y insérer la nouvelle valeur. Le tableau continue ainsi à être ordonné.

6.15 Tri bulle

Enoncé : L'utilisateur saisit une liste de nombres entiers au clavier. Chaque valeur est stockée dans un tableau d'entiers. La lecture de la liste s'arrête lorsque

l'utilisateur frappe la valeur numérique 0 (zéro) ou lorsque le tableau d'entiers est rempli. Dans un deuxième temps, l'algorithme trie la liste en utilisant le principe du tri à bulle.

Principe : Ce tri à prendre une série de nombres, puis en comparant les valeurs de la série 2 par 2, il effectue des permutations éventuelles de manière à amener la valeur la plus grande de la liste à la fin de la série. On recommence avec une sous-série correspondant à la série précédente moins le dernier élément car il est déjà en place. Le tri s'arrête lorsque la sous-série n'est plus composée que d'un élément.

Exemple :

14	**4**	7	8	10	9	7	12	9	1 re série
4	**14**	**7**	8	10	9	7	12	9	
4	7	**14**	**8**	10	9	7	12	9	
4	7	8	**14**	**10**	9	7	12	9	
4	7	8	10	**14**	**9**	7	12	9	
4	7	8	10	9	**14**	**7**	12	9	
4	7	8	10	9	7	**14**	**12**	9	
4	7	8	10	9	7	12	**14**	**9**	
4	7	8	10	9	7	12	9	**14**	2 e série

4	**7**	8	10	9	7	12	9	
4	**7**	**8**	10	9	7	12	9	
4	7	**8**	**10**	9	7	12	9	
4	7	8	**10**	**9**	7	12	9	
4	7	8	9	**10**	**7**	12	9	
4	7	8	9	7	**10**	**12**	9	
4	7	8	9	7	10	**12**	**9**	
4	7	8	9	7	10	9	**12**	3 e série

4	**7**	8	9	7	10	9
4	**7**	**8**	9	7	10	9
4	7	**8**	**9**	7	10	9
4	7	8	**9**	**7**	10	9
4	7	8	7	**9**	**10**	9
4	7	8	7	9	**10**	**9**

4 7 8 7 9 9 **10** 4 e série

4 7 8 7 9 9
4 **7 8** 7 9 9
4 7 **8 7** 9 9
4 7 7 **8 9** 9
4 7 7 8 **9 9**
4 7 7 8 9 **9** 5 e série

4 7 7 8 9
4 **7 7** 8 9
4 7 **7 8** 9
4 7 7 **8 9**
4 7 7 8 **9**

Il n'y a eu aucune permutation dans cette série, il est donc inutile de poursuivre l'algorithme. Il faut utiliser un flag ou drapeau, pour vérifier s'il y a eu une permutation dans une série.

4 7 7 8 9 9 10 12 14 Contenu du tableau après le tri.

6.16 Tri shell

Enoncé : L'utilisateur saisit une liste de nombres entiers au clavier. chaque valeur est stockée dans un tableau d'entiers. La lecture de la liste s'arrête lorsque l'utilisateur frappe la valeur numérique 0 (zéro) ou lorsque le tableau d'entiers est rempli.
Dans un deuxième temps, l'algorithme trie la liste en utilisant le principe du tri Shell.

Principe : Ce tri consiste à prendre une série de nombres et une valeur de « pas » de comparaison. Puis en comparant les valeurs de la série 2 par 2 (la valeur courante et la valeur qui se trouve « un pas » plus loin dans la tableau), il effectue des permutations éventuelles.
S'il y a permutation, on compare la nouvelle valeur se trouvant à la position courante avec la valeur se trouvant « un pas » places avant.
S'il y a de nouveau