Les Tableaux

Utilisation de beaucoup (mais vraiment beaucoup) de variables

Certains programmes ou problèmes de l'informatique font appel à énormément de données et de variables. Or, nous avons vu que les variables se définissent individuellement, en précisant leur type et leur nom. Même en utilisant une définition de variables multiples, le programme peut devenir long. Prenons encore une fois un exemple : on veut faire un programme d'analyse de relevés météorologiques qui calcule la moyenne des températures pour 10 villes de France sur un mois, sachant que pour chaque ville, il y a 3 relevés quotidiens. Il faut donc, en tout, relever 3 x 10 x 30 : 900 températures, soit 900 variables à définir !

Approche naïve

Pour la définition, il faudrait donc choisir 900 noms différents, ce qui est faisable en appelant les variables `temp_1`, `temp_2`,..., `temp_900`. En comptant qu'il faut 2 secondes pour écrire le nom d'une variable, la définition de toutes les variables prendrait (sans aucune erreur de frappe)…30 minutes.

Quant à la saisie des variables, il faudrait écrire 900 lignes de saisie :

```
saisir(temp_1);
saisir(temp_2);
…
saisir(temp_900);
```

idem pour le calcul de la température moyenne :

```
moyenne ← (temp_1 + temp_2 + … + temp_900)/900.0;
```

Attention, les points de suspension ne sont pas compris par l'ordinateur, ils sont juste utilisés dans l'exemple pour éviter d'écrire la ligne en entier !

Cette approche est donc particulièrement longue et inefficace ! Pourtant, nombre de programmes utilisent énormément de variables, bien souvent plus que les 900 nécessaires à l'exemple considéré ici.

On peut par contre remarquer que l'ensemble de toutes les variables nécessaires pour ce programme ont quelque chose en commun : le type. En effet, ce sont toutes des variables de type **entier**.

Il serait donc intéressant de pouvoir définir rapidement et efficacement 900 variables entières, et de pouvoir les utiliser efficacement.

Regroupement

L'idée est de ne plus définir les variables individuellement, mais de définir directement un ensemble ou un regroupement de variables qui ont toutes le même type. C'est le principe même d'un tableau en informatique : on définit directement un regroupement d'un certain nombre de variables de même type. Définir un tableau, c'est demander à l'ordinateur : "je voudrais un regroupement de *N* variables du type voulu". (*N* est la taille du regroupement ou tableau). Pour que l'ordinateur puisse manipuler les variables contenues dans le tableau, il faut lui donner un nom, en suivant les mêmes règles de nommage que pour les identificateurs de variable.

Présentation d'un tableau

Ce qu'est un tableau

Un tableau est un regroupement de plusieurs variables d'un type choisi, on parlera donc de tableau d'entiers, tableaux de réels, tableaux de caractères.

Ce que n'est pas un tableau

Il ne faut pas confondre le tableau, qui est un regroupement, avec une variable du type des éléments du tableau. Ainsi :

- Un tableau contenant des entiers n'est pas un entier
- Un tableau contenant des réels n'est pas un réel
- Un tableau contenant des caractères n'est pas un caractère.

On ne peut donc pas manipuler un tableau en utilisant les opérateurs arithmétiques et logiques que nous avons rencontré jusqu'à présent. Il est très important de se rappeler cela, c'est un bon moyen d'éviter les erreurs lorsque l'on écrit un programme.

Définition d'un tableau : la notation [].

Pour définir un tableau, on doit indiquer : le type des variables qu'il contient, son nom, ainsi que le nombre de variables que contient le tableau. Aucune autre information n'est nécessaire.

Syntaxe : type nom_du_tableau[nombre_de_variables];

Quelques exemples :

- Définition d'un tableau nommé des_entiers qui contient 50 variables de type entier :

```
entier des_entiers[50];
```

Cela remplace avantageusement une définition individuelle de 50 variables de type entier !

- Définition d'un tableau nommé ens_reels qui contient 2 variables de type réel :

```
reel ens_reels[2];
```

- Définition d'un tableau nommé du_texte qui contient 1500 variables de type caractère :

```
caractere du_texte[1500];
```

rappel : le tableau des_entiers n'est pas un entier, on ne peut donc pas écrire :

```
des_entiers ← 5;          // ERREUR
```

de même pour les tableaux ens_reels et du_texte : les lignes suivantes sont des sources d'erreur :

```
ens_reels ← -0.454545E-27;      // ERREUR
du_texte ← 'W';                 // ERREUR
```

Accès aux variables : indice

L'intérêt principal d'un tableau ne réside pas uniquement dans le fait que l'on puisse déclarer en une seule fois un grand nombre de variables, il faut que le tableau offre un moyen simple de manipuler les variables qu'il contient.

On considère que les variables contenues dans le tableau sont numérotées : on peut accéder à une variable en indiquant son numéro (un peu comme son dossard !), qui est un nombre entier**. Le nom de la variable portant le numéro *n* est le nom du tableau suivi de *n* entre crochets**. Il ne faut surtout pas le confondre avec la définition du tableau !

Règle de numérotation : en considérant un tableau comportant N variables, les numéros des variables dans le tableau vont de 0 à N-1;

Représentation d'un tableau en algorithmique :

Pour mieux visualiser les variables et les traitements, un tableau sera représenté comme un ensemble de cases (chaque case étant une variable contenu dans le tableau),

numéroté implicitement de la gauche vers la droite : la case de gauche sera la variable contenue dans le tableau et portant le numéro 0, la case située à droite de la première sera la variable contenue dans le tableau et portant le numéro 1, la case située à l'extrémité droite porte le numéro N-1 (c'est à dire, le dernier numéro disponible).

Exemples commentés : manipulation d'entiers regroupés dans un tableau.

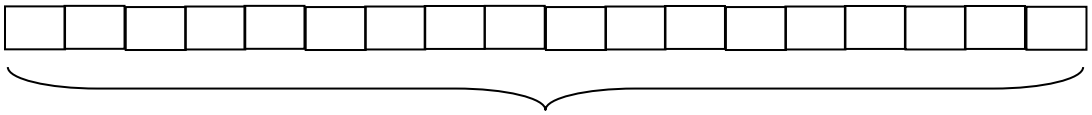
Un tableau de 20 entiers est défini par la ligne suivante :

```
entier tablo_ent[20];
```

Il s'agit d'une définition, qui signifie : `tablo_ent` est un tableau ou regroupement de 20 variables, qui sont chacune du type `entier`.

Sa représentation est la suivante :

Tablo_ent



20 cases représentant les 20 variables

Les noms des variables regroupées (contenues) dans le tableau sont :

```
tablo_ent[0] // celle qui porte le numéro 0
tablo_ent[1] // celle qui porte le numéro 1
```

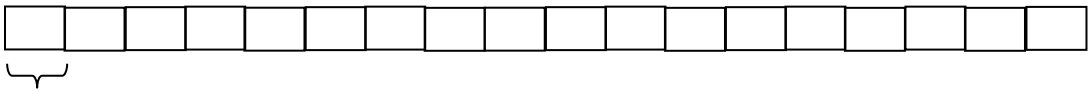
…

jusqu'à : `tablo_ent[19] // celle qui porte le numéro 19`

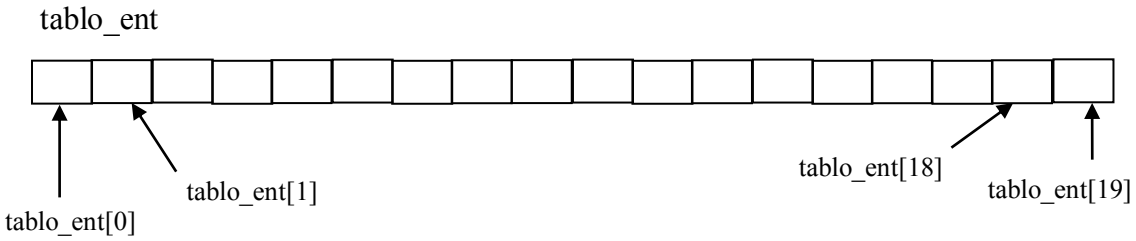
le numéro que porte une variable dans un tableau est aussi appelé son **indice** dans le tableau : un **indice** est toujours une valeur entière !

représentation des variables et des indices pour le tableau :

tablo_ent



Cette case correspond à la variable d'indice 0, on peut la nommer tablo_ent[0]



un rappel sur les types :

`tablo_ent` n'est pas un entier : c'est un tableau d'entiers

`tablo_ent[0]` est un entier, ainsi que `tablo_ent[1],…, tablo_ent[19]`.

Ce que l'on ne peut pas faire

Les instructions suivantes ne sont pas correctes (nous reprenons ici le tableau d'entiers tablo_ent défini ci-dessus) :

```
tablo_ent ← -1;
tablo_ent[] ← 4;
tablo_ent[] ← 1, 5, -7;
afficher(tablo_ent);
afficher(tablo_ent[]);
saisir(tablo_ent);
```

Si vous avez un doute quant à la validité d'une instruction concernant les tableaux, pensez systématiquement à vérifier les types des variables et des expressions que vous écrivez, c'est un indice très précieux qui vous évitera beaucoup de soucis : appliquons ceci aux instructions précédentes, cela nous permettra d'expliquer pourquoi elles ne sont pas correctes :

- `tablo_ent ← -1;`
  tablo_ent n'est pas un entier, -1 est un entier : il y a incompatibilité !
- `tablo_ent[] = 4;`
  l'écriture tablo_ent[] n'a pas de sens ici : un nom de tableau avec des crochets sans indice n'est pas une variable entière, et 4 est un entier : il y a incompatibilité !
- `tablo_ent[] = 1, 5, -7;`
  1,5,-7; on ne peut pas affecter quelquechose avec une liste, de plus tablo_ent[] n'a pas de sens.
- `afficher(tablo_ent);`

afficher permet d'afficher seulement des entiers, des reels et des caractères; or tablo_ent n'est d'aucun de ces types.

- `afficher(tablo_ent[]);`

l'écriture tablo_ent[] ne désigne ni un entier, ni un réel, ni un caractère, on ne peut donc pas utiliser afficher().

- `saisir(tablo_ent);`

même raison que pour afficher : tablo_ent n'est pas un entier, or on ne peut saisir que des entiers, des réels ou des caractères.

Pensez à formuler ce que vous voulez faire si vous avez un doute sur la formulation en langage algorithmique ou en langage informatique : si vous avez une idée claire de ce que vous voulez, vous arriverez bien plus facilement à retrouver une formulation correcte. De plus, si vous êtes bloqué, le fait de poser clairement votre question, à un camarade, à un chargé de TD, vous permettra d'obtenir très rapidement une réponse satisfaisante et claire !

Ce que l'on peut faire

On peut écrire les instructions suivantes :

```
tablo_ent[0] = -1257;
tablo_ent[1] = 17;
tablo_ent[19] = 0;
tablo_ent[0] = tablo_ent[3] + 1;
tablo_ent[14] = 3* tablo_ent[5] % (2/3*(tablo_ent[16]+4));
afficher(tablo_ent[5]+3-tablo_ent[12]);
saisir(tablo_ent[8]);
```

car tablo_ent suivi d'un entier entre crochets est le nom d'une variable de type entier contenue dans le tableau. On peut donc appliquer à cette variable tous les opérateurs connus : opérateurs arithmétiques, afficher et saisir.

Indices hors-limites :

Attention toutefois à la valeur des indices, ils ne peuvent pas être négatifs et ne peuvent pas dépasser *N*-1 si le tableau contient *N* variables. Dans le cas du tableau utilisé dans notre exemple, tablo_ent, défini par : `entier tablo_ent[20];` , on ne peut pas écrire :

```
tablo_ent[-1]; // car l'indice est négatif;
tablo_ent[20]; // car le plus grand indice autorisé est 19
```

Ce genre d'erreurs fait en général 'planter' le programme sur la machine.

Indices variables :

Autre exemple : tableau contenant un autre type que des entiers, utilisation d'indices variables.

Bien évidemment, tout ce que l'on vient de dire est valable pour les tableaux contenant autre chose que des entiers, mais un exemple illustrera mieux cela. Nous allons traiter le cas d'un tableau contenant des caractères : nommons-le `carac` par exemple (il n'y aucune raison pour que 'tab' apparaisse en toutes lettres dans l'identificateur du tableau); ce tableau contiendra 80 variables.

Voici sa définition :

```
caractere carac[80];
```

on ne peut toujours pas faire :

```
carac ← '%'; // carac n'est pas de type caractere
```

ou

```
carac ← 0; // pour la même raison que l'instruction précédente
```

mais on peut très bien écrire :

```
carac[0] ← '(';
carac[79] ← carac[0] + 2;
```

si l'on fait : `afficher(carac[0]);` on obtient à l'écran :

```
(
```

Un tableau est en fait un outil encore plus souple que ce que nous avons vu jusqu'à présent, car on n'est pas limité à l'emploi de constantes pour les indices des variables dans les tableaux.

Dans les exemples présentés jusqu'ici, on a toujours utilisé des constantes pour indiquer l'indice d'une variable dans un tableau, mais ce n'est nullement une obligation. On peut en réalité utiliser une **expression qui donne une valeur entière comme indice**. (rappelez-vous, une expression est une formule compréhensible et qui donne un résultat).

Pour l'instant, nous avons utilisé une constante, qui est bien une expression, mais on peut aussi utiliser une variable entière comme indice : c'est un cas qui se produit très souvent, comme nous le verrons dans les exemple présentant les algorithmes les plus employés pour le traitement des tableaux.

Ainsi, si on définit une variable entière, on peut s'en servir comme indice.

Voici un exemple de programme utilisant une variable comme indice, en reprenant le tableau de caractères nommé carac :

Nous allons détailler ligne par ligne le fonctionnement de ce programme, en vérifiant à chaque fois le contenu des variables que nous utilisons.

```
programme indice_et_tableau
caractère carac[80];
entier position;

position ← 4;
afficher("entrez un caractere :");
saisir(carac[position]);
position ← position+1;
carac[position] = 'T';
position ← position -1;
afficher("vous avez entrez :",carac[position],"\n");
afficher("on a range dans la variable suivante
:",carac[position+1]);
```

On peut même écrire :

```
carac[3*2/5+6%3+1] = 'T';
```

Cela ne pose aucun problème, car est une expression dont le résultat est un nombre entier, et donc peut servir d'indice.

Initialiser un tableau

On peut initialiser un tableau lors de sa définition en fournissant une liste de valeurs, séparées par des virgules et encadrée par des accolades '{' et '}'.

Un exemple : soit un tableau de 4 réels que l'on veut initialiser avec les valeur suivantes : -1; -0,5; -1,054.10^{23}; 4,2.

La ligne suivante :

```
réel tab_virg[4] ← {-1.0, -.5, -1.054E+23, 4.2};
```

permet de faire : la définition du tableau, et d'initialiser les 4 variables qu'il contient.

On peut également initialiser un tableau en donnant moins de valeurs que le tableau peut en contenir : dans ce cas, les premières variables du tableau sont initialisées avec les valeurs données, et les suivantes sont initialisés à 0 (selon le type : 0 pour les entiers et les caractères, 0.0 pour les réels).

Un exemple :

```
entier des_entiers[10] = {0,5,6};
```

permet de définir le tableau nommé des_entiers, et de stocker les valeurs 0, 5 et 6 dans les variables des_entiers[0], des_entiers[1], des entiers[2]. Toutes les autres sont initialisés à 0.

En pratique : utilisation du tableau

Taille maximale / taille utile

Lorsque l'on définit un tableau, on doit préciser, entre les crochets, le nombre maximum de variables que peut contenir ce tableau, ce nombre doit être une constante, et ne peut pas changer lors du programme. C'est la **taille maximum** du tableau.

On ne pourra donc pas utiliser de variables supplémentaires dans ce tableau s'il est déjà rempli (c'est à dire, si toutes les variables sont déjà utilisées). On ne peut redéfinir un tableau en cours de programme. Il faut donc prévoir un tableau suffisamment grand dès le départ, sous peine d'avoir à modifier le programme pour augmenter la taille du tableau.

Ce problème est d'autant plus gênant que l'on ne sait pas forcément, lorsqu'on écrit un programme, combien de variables d'un tableau on va utiliser, par exemple un programme qui doit 'saisir un certain nombre de valeurs et en faire la moyenne'. On ne sait pas à l'avance quel est ce 'certain nombre'. Dans cet exemple, on pourra choisir de créer un tableau contenant 100 variables, ce qui devrait être suffisant, mais ce n'est pas une certitude absolue.

Quel est le nombre de variables utilisées dans un tableau ?

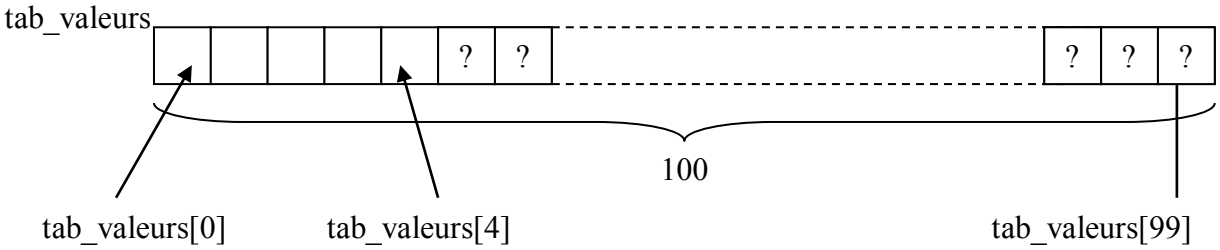
Puisque l'on doit prévoir un nombre de variables suffisamment élevé, il est fort probable que l'on n'utilisera pas toutes les variables du tableau. Dans l'exemple précédent d'un tableau de 100 variables, il se peut que l'on n'en utilise que 5, ou 8, ou 56, selon ce que l'utilisateur choisira.

Or si l'on veut calculer la moyenne des valeurs entrées, il faudra diviser la somme des valeurs entrées par le nombre des valeurs entrées (c'est à dire le nombre de variables utilisées) et non pas par la taille maximum du tableau !

Illustration du problème :

On suppose que l'on a un tableau de 100 variables de type réel, nommé tab-valeurs et que l'utilisateur en a saisi 5. Ce tableau aura donc été déclaré par : `réel tab_valeurs[100];` et seules les variables d'indice 0 à 4 contiennent des valeurs intéressantes. Les variables d'indice 5 à 99 sont inconnues : elles ne sont pas initialisées, **on ne peut même pas supposer qu'elles valent 0.0**, ce qui est important.

Utilisons le schéma du tableau pour visualiser cette situation :



On voit bien sur se schéma que rien, à part la valeur, ne distingue une variable utilisée d'une variable non utilisée ! Et comme on ne dispose d'aucune information sur les valeurs des variables non utilisées (d'où le ? noté dans les variables correspondantes), on ne peut pas 'détecter' la limite entre variables utilisées et non utilisées.

La seule solution restante est de stocker, dans une nouvelle variable, le nombre de variables utilisées dans le tableau. Cette nouvelle variable sera donc de type entier.

Ainsi, pour le tableau tab_valeurs de notre exemple, on peut stocker le nombre de variables utilisées (ici, 5) dans une nouvelle variable que l'on peut appeler `taille` par exemple (comme toute autre variable, on peut la nommer comme on veut, x ou toto par exemple, mais autant choisir un nom en rapport avec son rôle).

Ce nombre de variables utilisées dans un tableau porte le nom de **taille utile**, par opposition à la taille maximum donnée lors de la définition du tableau.

A retenir : un tableau possède :

- une **taille maximum**, qui est la valeur donnée entre les crochets lors de la définition de la variable, cela correspond au **nombre de variables que l'on peut utiliser** dans le tableau;
- une **taille utile**, stockée dans une autre variable entière, qui correspond au **nombre de variables effectivement utilisées** dans le tableau.

Pour calculer la moyenne des 5 valeurs utiles stockées dans le tableau tab_valeurs, on fera la somme : `tab_valeurs[0] + tab_valeurs[1] + tab_valeurs[2] + tab_valeurs[3] + tab_valeurs[4];` et on divisera par la variable `taille` qui vaut 5. Le résultat sera donc bien celui que l'on attend.

Dès que l'on définit un tableau, on doit aussi définir une variable entière qui contiendra sa taille utile.

Algorithmes classiques : la relation tableau / boucles

Lorsqu'on manipule un tableau, on utilise de manière assez systématique des boucles tant...que, faire...tant que ou pour, car on doit accéder à plusieurs variables du tableau avec un indice qui évolue. Nous allons prendre dans les paragraphes suivantes des exemples simples dans lesquels nous allons détailler le choix des boucles en fonction des opérations à faire.

Affichage

L'opération à effectuer est assez simple : afficher les unes après les autres les variables contenues dans un tableau nommé `t_val`. On suppose que la taille utile du tableau est stockée dans une variable nommée `nb_val`, qui contient la bonne valeur.

L'opération d'affichage se résume donc à : `afficher(t_val[0]);` puis `afficher(t_val[1]);`...jusqu'à `afficher(t_val[nb_val-1]);` rappel : les indices commencent à 0 !

Cela revient donc à faire l'opération : `afficher(t_val[compteur]);` avec `compteur` qui est une variable entière évoluant de 0 à nb_val –1, de 1 en 1.

On connaît donc à l'avance le nombre de fois où l'on répètera l'opération : on utilise une boucle pour.

Le programme :

```
programme affiche_tableau

entier des_entiers[50];
entier taille;
entier compteur;

// on ne montre pas les instructions de remplissage des variables du
// tableau, on suppose que le tableau contient maintenant des
// variables et que sa taille utile est correcte
```

```
pour compteur de 0 à taille-1
{
      afficher(des_entiers[compteur]);
      afficher("\n"); // pour passer à la ligne entre chaque valeur

}
```

Saisie

Le but de cette opération est de remplir les variables du tableau au fur et à mesure avec des valeurs saisies au clavier par l'utilisateur. On suppose qu'on ne connaît pas à l'avance le nombre ce valeurs à saisir (ceci est un indice très important pour déterminer le type de boucle à employer). La différence avec l'exemple précédent, c'est que l'on ajoute des valeurs dans les variables du tableau, et donc la taille utile du tableau sera modifiée.

Au départ, aucune variable du tableau n'est utilisée (il faut ici éviter de dire : 'le tableau est vide', car un tableau n'est pas vide : les variables qu'il contient ne sont pas initialisées, ou pas utilisées, mais contiennent toujours une valeur !), et donc, on en déduit que la taille utile du tableau est égale à 0.

Lorsque l'on saisit une variable du tableau, on l'utilise, donc la taille utile du tableau augmente de 1. Reste à savoir maintenant quel est l'indice de la variable que l'on saisit !

Si la taille utile est égale à 0 : il n'y aucune variable utilisée, donc on peut saisir la première variable du tableau : elle porte l'indice 0. La taille utile devient donc égale à 1.

Si la taille est égale à 1 : il y a une variable utilisée dans le tableau, on peut donc saisir la variable suivante : elle porte l'indice 1. La taille utile devient égale à 2.

Et ainsi de suite, on en déduit donc que l'indice de la prochaine variable à saisir dans un tableau est égal à la taille utile du tableau.

Puisqu'on ne sait pas combien de variables on va saisir, on utilise une boucle tant...que; il faut aussi prévoit un moyen d'arrêter de saisir des valeurs : on proposera à l'utilisateur, après chaque saisie, de continuer, sous la forme d'un message auquel il répondra par 'o' ou 'n' (donc par un caractère).

Il existe enfin une dernière précaution à prendre : ne pas dépasser la taille maximale du tableau ! il faudra donc systématiquement tester, avant de saisir une variable, qu'il y a bien de la place dans le tableau pour la stocker.

Voici donc un programme de saisie de valeurs à ranger dans un tableau:

```
programme saisie_valeurs
réel tab_r[100];
entier taille_tab;    // on stocke la taille utile du tableau
caractère reponse;    // pour stocker la réponse de l'utilisateur

taille_tab ← 0;       // au départ : pas de variables utilisées

reponse='o';          // on donne la valeur 'o' à cette variable
                      // pour que la première saisie se fasse

tant que ((taille_tab < 100) && (reponse = 'o'))
{
      afficher("entrez une valeur :");
      saisir(&tab_r[taille_tab]);
      taille_tab ← taille_tab + 1;
      afficher("voulez-vous continuer (o/n) ?");
      saisir(&reponse);
}
```

les recherches dans les tableaux

les tableaux peuvent stocker beaucoup de données, et on doit parfois extraire une de ces données, ou savoir si elle est déjà présente dans le tableau, ou même savoir à combien d'exemplaires (ou occurrences) elle se trouve dans le tableau. Nous allons donc aborder plusieurs problèmes simples de recherche pour lesquels nous allons établir les algorithmes.

minimum / maximum

premier type de recherche : quelle est la valeur minimum ou maximum présente dans un tableau ?

Comme dans 99 % des cas, un traitement de variables stockées dans un tableau implique l'utilisation d'une boucle ou répétition. En se posant quelques questions simples, on peut facilement déterminer le type de boucle à employer. Une recherche de minimum ou de maximum ne nécessite qu'un parcours du tableau (ce n'est pas forcément le cas pour d'autres traitements). Est-il nécessaire de parcourir tout le tableau (toutes les variables utilisées) pour la recherche du minimum / maximum ?

Oui, car ce minimum / maximum peut très bien se trouver dans la dernière variable utilisée, et donc, on ne peut pas être certain de le rencontrer avant cette variable !

Exemple traité : recherche de la valeur minimum

Pour trouver quelle est la valeur minimum entre plusieurs valeurs, on utilise le raisonnement par récurrence que voici :

Le minimum de deux nombres est trouvé en employant l'opérateur < ou > avec un test si.

Le minimum de trois nombres est le minimum entre le troisième nombre et le minimum des deux premiers. (que l'on sait trouver).

Le minimum de quatre nombres est le minimum entre le quatrième nombre et le minimum des trois premiers nombres (que l'on sait maintenant trouver), et ainsi de suite.

Illustrons le principe de cette recherche par étapes avec un tableau contenant des valeurs entières arbitraires :

2	8	1	4	6	?	?

Nous allons stocker la plus petite valeur déjà rencontrée dans une variable nommé `mini`. Au cours du parcours, cette variable `mini` sera comparée aux variables du tableau, et si la variable a une valeur plus petite que celle de `mini`, celle ci prend la valeur de la variable du tableau.

Il ne reste qu'une chose à faire avant de commencer l'algorithme : quelle est la valeur initiale de `mini` ? Le choix d'une valeur arbitraire est risqué, car il se peut que toutes les valeurs stockées dans les variables du tableau soient supérieures à la valeur initiale de `mini`, auquel cas on n'obtiendra pas le bon résultat !

La solution la plus sure est d'initialiser `mini` avec la valeur de la première variable du tableau.

Donc, sur notre exemple, mini est initialisée à 2. On procède ensuite étape par étape :

Comparer mini et la première variable du tableau : cette variable n'est pas plus petite (elle vaut aussi 2), donc mini ne change pas.

Comparer mini et la deuxième variable du tableau : celle-ci vaut 8, donc n'est pas plus petite que mini, donc mini ne change pas.

Comparer mini et la troisième variable du tableau : celle-ci vaut 1, donc est plus petite que mini, qui vaut 2 (2 est effectivement la plus petite valeur rencontrée jusqu'ici). mini prend alors la valeur 1.

De même pour les comparaisons avec les deux valeurs suivantes stockées dans le tableau : 4 est plus grand que mini, donc mini ne change pas; 6 est plus grand que mini, donc mini ne change pas.

Au résultat : le minimum vaut 1, ce qui est bien le résultat attendu.

```
programme rech_mini

entier tab_val[7] ← {2,8,0,1,6};
entier taille, compt;
entier val_mini;

taille ← 5;     // nombre de variables utilisées
val_mini ← tab_val[0];

pour compt de 0 à taille-1 faire
{
      si (tab_val[compt] < mini) alors
      {
            mini ← tab[compt];
      }
      // sinon, il n'y a rien à faire, puisque la valeur de mini
      // est toujours correcte.
}
afficher("le minimum des éléments du tableau vaut :",mini);
```

recherche de l'indice

Plutôt que de faire une recherche de la valeur du minimum, on peut rechercher son indice : la recherche suit exactement le même principe que précédemment, sauf que l'on stocke l'indice de la plus petite valeur rencontrée plutôt que cette valeur elle-même. Le programme devient alors :

```
programme rech_ind_mini

entier tab_val[7] ← {2,8,0,1,6};
entier taille, compt;
entier ind_mini;

taille ← 5;     // nombre de variables utilisées
ind_mini ← 0;   // au départ, on considère que l'indice du minimum
      // est 0, ce qui revient à dire que le minimum est tab_val[0]

pour compt de 0 à taille-1 faire
{
      si (tab_val[compt] < tab_val[ind_mini])
      {
            ind_mini ← compt;
      }
}
```

```
afficher("a l'indice ", ind_mini, " se trouve la valeur ", tab_val[ind_mini]);
```

la recherche de l'indice a donc un avantage sur celle de la valeur : tout en prenant le même temps (ce n'est pas plus compliqué), elle indique l'indice de la variable du tableau où se trouve le minimum. On peut donc facilement retrouver la valeur de ce minimum en consultant la variable. Le fait de rechercher l'indice plutôt que la valeur peut être très utile, nous en verrons une illustration dans les méthodes de tri de tableaux.

Adaptation pour la recherche du maximum des valeurs stockées dans un tableau : il suffit de remplacer, lors du test, l'opérateur < par l'opérateur >, et le tour est joué. N'oubliez pas cependant que les noms des variables risquant de ne plus être adaptés (la variable mini qui contient la valeur maximum, c'est possible, mais pas recommandé !).

présence d'une valeur

Autre type de recherche : savoir si oui ou non, une valeur (saisie par l'utilisateur par exemple) est stockée dans un tableau. Cela peut servir dans un tableau sans doublons (on ne veut pas stocker deux fois la même valeur) : on saisit une valeur, on teste si elle se trouve dans le tableau, et si elle ne s'y trouve pas, on la stocke.

Comme toujours, il faudra utiliser une répétition, car on traite un tableau de variables. A-t-on systématiquement besoin d'examiner toutes les variables utilisées dans le tableau ?

Non, car dès que l'on trouve la valeur recherchée, on sait que la réponse à la recherche sera oui. Il est alors inutile de poursuivre la recherche. On emploiera donc une répétition de type tant que ou faire … tant que. Quelles sont les conditions auxquelles on arrête la recherche de la valeur voulue ?

- La valeur est stockée dans une des variables du tableau : un simple test suffit pour s'en rendre compte
- La valeur n'est stockée dans aucune des variables du tableau : cela est détectable si l'on arrive à la fin du tableau (en terme de taille utile et non de taille maximum) sans avoir trouvé la valeur.

On continuera donc la recherche dans le cas contraire, c'est à dire si : la valeur n'a pas été trouvée et si la fin du tableau n'a pas été atteinte.

On peut donc résumer l'algorithme de recherche de la manière suivante, en employant une variable entière nommée `trouve` qui indique si la valeur demandée a été trouvée ou non.

```
programme rech_val

entier tab_ent[10]←{6,5,12,0,-8,7};
entier tai, trouve, indice;

// tai : taille utile du tableau, trouve : indique si la valeur a
// été trouvée , cette variable vaut 0 pour non et 1 pour oui.
// indice : indice de la variable du tableau que l'on teste.

entier val_rech; // valeur à rechercher dans le tableau

afficher("entrez la valeur a rechercher :");
saisir(val_rech);

tai ← 6;  // il y a 6 variables utilisées sur les 10 possibles
trouve ← 0; // au départ : pas encore trouvé
indice ← 0; // on commence par la variable d'indice 0

tant que ((trouve = 0) ET (indice < tai)) faire
{
     si (tab_ent[indice] = val_rech) alors
     {
          trouve ← 1;
     }

     indice ← indice + 1;
}

si (trouve=1) alors
{
     afficher("la valeur est stockée dans le tableau\n");
}
sinon
{
     afficher("la valeur n'est pas stockée dans le tableau\n");
}
```

De même qu'avec la recherche du minimum / maximum, le fait de savoir si la valeur est stockée peut s'avérer insuffisant, connaître son indice éventuel peut être intéressant. La réponse obtenue par le programme, de type oui ou non, peut également être l'indice auquel est stocké cette valeur. Si elle ne s'y trouve pas, la réponse sera alors –1 (car un indice ne peut pas prendre cette valeur). Le programme peut donc être facilement modifié en conséquence.

nombre d'occurrences d'une valeur

Autre possibilité de recherche : connaître le nombre d'exemplaires d'une valeur dans un tableau. Cette recherche est assez proche de la précédente, mais le type de la boucle change, car il ne s'agit plus d'une répétition de type tant que ou faire…tant que. En effet, pour connaître le nombre d'exemplaires d'une valeur, il faut traiter le tableau dans son intégralité,

car la dernière variable utilisée peut contenir la valeur recherchée. Le fait de rencontrer cette valeur lors du parcours ne permet pas directement de conclure sur le nombre d'exemplaires de cette valeur. La répétition à utiliser est donc de type pour. La variable `trouve` de l'exemple précédent sera remplacée par une variable entière nommée `nb_ex` dans laquelle sera stocké le nombre d'exemplaires.

Il suffit d'augmenter cette variable `nb_ex` de 1 à chaque fois que la valeur voulue se trouve dans une des variables utilisées du tableau en cours de traitement. Au début du traitement, cette variable `nb_ex` doit être initialisée à 0.

les tris et les tableaux triés

tri

Le problème du tri est présent dans de très nombreux programmes et applications, et nous allons traiter en détail quelques méthodes de tris. Il existe en effet plusieurs méthodes, qui se distinguent les unes des autres par leur rapidité et leur simplicité. A peu de choses près, on peut dire que les méthodes de tri les plus simples à expliquer sont malheureusement les moins rapides ! Nous aurons l'occasion, plus tard au cours de l'année, de revenir sur des méthodes de tri plus complexes mais optimales (c'est à dire, les plus rapides possibles).

Trier un tableau, c'est ranger les variables qu'il regroupe selon un ordre croissant ou décroissant. On veut donc qu'à la fin d'un tri par ordre croissant (resp. décroissant), la variable ayant la valeur la plus faible (resp. élevée) se trouve dans la variable d'indice 0, et que celle ayant la valeur la plus élevée (resp. faible) se trouve dans la variable d'indice taille-1, où la variable taille indique la taille utile du tableau.

Nous utiliserons lors de nos exemples, des tableaux d'entiers, car ils se représentent facilement, mais les méthodes seront applicables aux nombres réels et aux caractères.

Le détail des méthodes proposées pour le tri est très intéressante d'un point de vue algorithmique, car elle permet de mettre en algorithme des idées assez simples.

Le tri à bulles

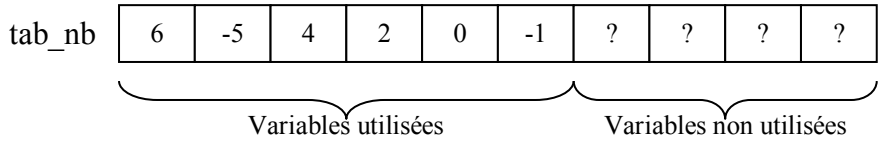
C'est la méthode la plus simple, mais aussi une des moins efficaces, cependant nous verrons que l'on peut facilement l'améliorer en réfléchissant un minimum.

Le principe du tri à bulles est de parcourir le tableau du début à la fin et à comparer entre elles deux variables d'indices consécutifs : si ces variables ne sont pas rangées dans le

bon ordre, on les intervertit (on échange les valeurs stockées dans ces variables). On doit effectuer ce parcours de tout le tableau autant de fois que le tableau contient de variables utilisées.

Illustration :

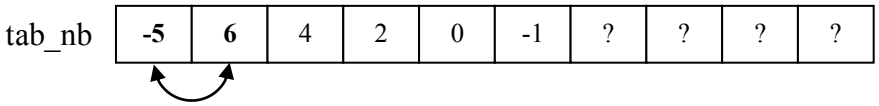
Soit au départ le tableau tab_nb, contenant au maximum 10 variables entières, et qui est initialisé avec les valeurs suivantes : 6,-5,4,2,0,-1. On souhaite le trier par ordre croissant.



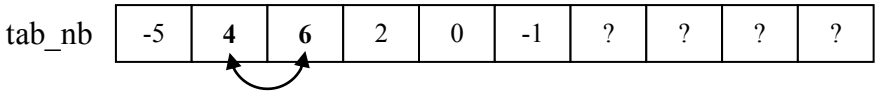
Il y a 6 variables utilisées, donc le tableau doit être parcouru 6 fois.

Premier parcours : on va comparer les deux premières variables du tableau tab_nb :

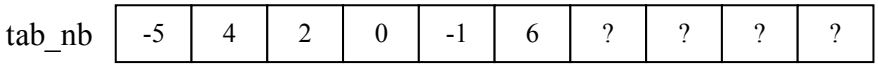
tab_nb[0] et tab_nb[1]. Sont elles rangées dans le bon ordre (ordre croissant) ? non, donc on les intervertit : on doit mettre la valeur –5 dans tab_nb[0] et la valeur 6 dans tab_nb[1].



On compare ensuite les valeurs de tab_nb[1] et tab_nb[2], et on procède au même test, et éventuellement à l'échange des valeurs si elles ne sont pas correctement classées.



Et ainsi de suite jusqu'à tester tab_nb[4] et tab_nb[5], on obtient à la fin de ce premier parcours :



Les autres parcours se déroulent exactement de la même manière.

Détermination de l'algorithme :

On devra répéter un nombre de fois connu à l'avance un parcours complet du tableau : utilisation d'une boucle pour

Pour chaque parcours : on doit aller de la première variable utilisée à la dernière variable utilisée pour faire les comparaisons et les échanges éventuels de valeurs. On sait donc également à l'avance le nombre de fois où l'on va répéter ces tests et échanges : on utilise également une boucle pour.

Nous pouvons déjà écrire une première version simple de l'algorithme :

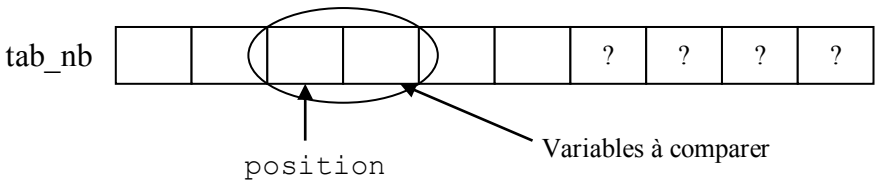
```
programme tri_a_bulles_croissant

entier tab_nb[10] ← { 6,-5,4,2,0,-1};
entier taille_utile;
entier tour, position;

taille_utile = 6;

pour tour de 0 à taille_utile-1 faire
{
      pour position de 0 à taille_utile-1 faire
      {
            si deux variables consécutives du tableau sont mal
classées
            {
                  échanger les valeurs de ces deux variables
            }

      }
}

// on peut de plus afficher le résultat obtenu à la fin du programme
```

Regardons en détail le test de variables successives et les échanges de valeurs de variables du tableau.

A chaque valeur de la variable `position`, qui sert à parcourir le tableau, on peut associer les indices des variables du tableau qui seront comparées entre elles :

Lorsque position vaut	On compare les variables d'indices
0	0 et 1
1	1 et 2
2	2 et 3
3	3 et 4
4	4 et 5
5	5 et **6 attention !**

Ce que l'on peut représenter par :



On constate donc que lorsque la variable `position` a une certaine valeur, on teste les valeurs des variables du tableau dont les indices sont égaux à `position` et `position+1`. On constate également que la variable `position` doit aller de 0 à `taille_utile-2` (et non `taille_utile-1` comme on pouvait le penser au départ).

Réalisation du test : pour tester si les valeurs sont classées par ordre croissant, on utilise l'opérateur < ou >.

La condition (tab_nb[position] > tab_nb[position+1]) est vraie si les deux variables ne sont pas rangées dans l'ordre croissant. Dans ce cas, on fait l'échange des valeurs en utilisant une variable temporaire.

D'où le programme complet suivant :

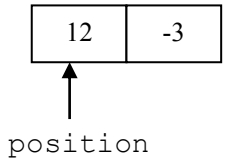
```
programme tri_a_bulles_croissant

entier tab_nb[10] ← { 6,-5,4,2,0,-1};
entier taille_utile;
entier tour, position, echange;

taille_utile ← 6;

pour tour de 0 à taille_utile-1 faire
{
      pour position de 0 à taille_utile-2 faire
      {
            si (tab_nb[position] > tab_nb[position+1]) alors
            {
                  echange ← tab_nb[position];
                  tab_nb[position] ← tab_nb[position+1];
                  tab_nb[position+1] ← echange;
            }

      }
}
// affichage du résultat pour vérification

pour compteur de 0 à taille_utile-1 faire
{
      afficher("tab_nb[", compteur,"] vaut",tab_nb[compteur],"\n");
}
```

quelques explications sur l'échange des valeurs des variables :

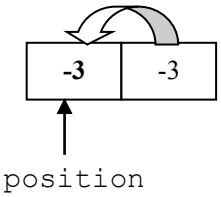
A quoi sert l'utilisation d'une variable temporaire ? il suffit pour répondre à cette question d'essayer d'écrire l'échange sans cette variable. On pourrait, en effet, penser à écrire l'échange de la manière suivante :

```
tab_nb[position] ← tab_nb[position+1];
tab_nb[position+1] ← tab_nb[position];
```

Prenons un exemple avec deux valeurs arbitraires pour illustrer ce qui se passe dans le tableau si l'on emploie ces instructions :

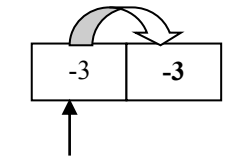


effet de l'instruction : `tab_nb[position] ← tab_nb[position+1];`



On constate bien que la valeur 12 est écrasée par la valeur –3 ! il aurait donc fallu la ranger dans une variable temporaire avant.

effet de l'instruction : `tab_nb[position] ← tab_nb[position+1];`



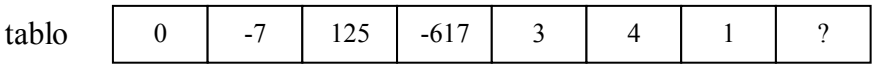
Il est de toute façon trop tard pour récupérer la valeur 12 qu'il aurait fallu stocker dans `tab_nb[position+1]` !

Le tri par sélection ou extraction

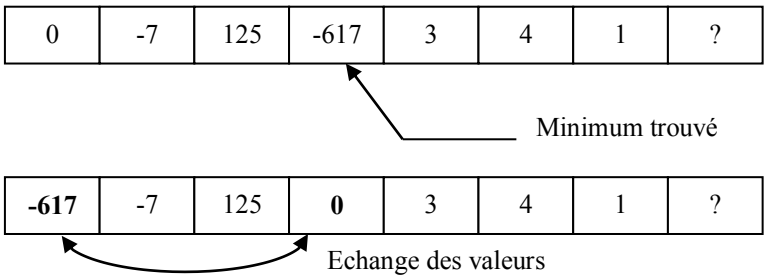
Une autre méthode de tri consiste à ranger les éléments un par un : on cherche le minimum dans le tableau, puis on échange sa valeur avec celle de la première variable du tableau, ensuite on recherche le minimum dans le tableau privé de sa première variable (puisqu'elle est bien triée), et on échange sa valeur avec celle de la deuxième variable du tableau, et ainsi de suite jusqu'à ce que le tableau soit entièrement trié.

Illustration : soit un tableau contenant au maximum 8 entiers et nommé `tablo`. Il sera défini par : entier tablo[8];

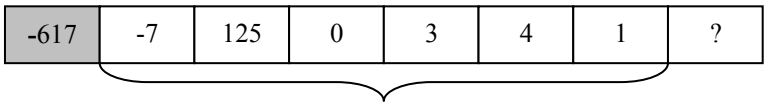
On considère qu'il est initialisé avec les valeurs : 0, -7,125,-617,3,4,1. Il y a donc une variable inutilisée, et la taille utile du tableau vaut 7, et stockée dans une variable nommée `tai_tab`.



On recherche la valeur minimum de tout le tableau : c'est –617, et on l'échange avec la variable d'indice 0 du tableau (la première variable du tableau).

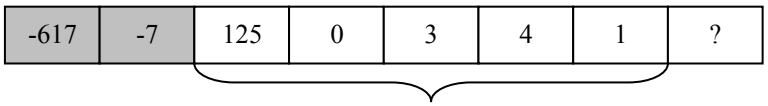


On n'a plus à se préoccuper de cette valeur, puisqu'elle est définitivement bien classée. Pour illustrer cela, nous noterons la variable correspondante avec un fond gris.



Deuxième étape : répétition de la première étape sur la partie du tableau repérée par une accolade : le tableau privé de son premier élément.

Le minimum trouvé est –7, on l'échange avec la variable situé à l'indice 1 du tableau.



A la fin de la deuxième étape, le tableau se trouve dans cet état, on recommence sur la partie du tableau repérée par une accolade, et ainsi de suite !

Réalisation de l'algorithme : on doit effectuer une recherche de minimum et un échange de variables autant de fois qu'il y a d'éléments dans le tableau (en fait une fois de moins suffit, nous verrons pourquoi). On sait donc à l'avance quel type de boucle utiliser : une répétition pour. On aura donc une variable de type compteur, que l'on peut appeler `num`, qui ira de 0 à `tai_tab - 1`. Cette variable contient en quelque sorte le numéro de l'étape de traitement du tableau : recherche de minimum + échange de valeurs.

Nous avons constaté que, pour la recherche du minimum, l'indice de la variable ou débute la recherche dépend de l'étape, donc de la valeur de la variable `num`.

Lorsque `num` vaut	La recherche du minimum commence à la variable d'indice
0	0
1	1
2	2
3	3
4	4
5	5
6	6

Cet indice de début de recherche de minimum est donc `num`. L'indice de fin de la recherche du minimum est `tai_tab-1`, car on doit chercher le minimum jusqu'à la dernière variable utilisée dans le tableau. L'algorithme de recherche du minimum a, quant à lui, déjà été traité lors de la section précédente.

Echange de valeurs entre le minimum trouvé et une variable du tableau.

Une fois le minimum trouvé pour une certaine étape, il reste à échanger sa valeur avec celle d'une variable du tableau. Quel est l'indice de cette variable ? Un tableau peut encore nous être très utile :

Lorsque `num` vaut	L'indice de la variable à échanger avec le minimum est :
0	0
1	1
2	2
3	3
4	4
5	5
6	6

L'indice de la variable à échanger est donc égal à `num`. Mais, pour procéder à l'échange, on doit connaître également l'indice de la variable contenant ce minimum !

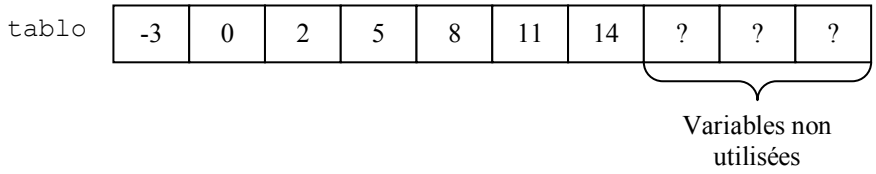
Insérer une valeur dans un tableau déjà trié

Lorsque l'on ajoute une valeur dans un tableau, il est assez simple de la stocker dans la première variable non utilisée. C'est ce qui est fait par exemple lors de la saisie de valeurs. Si le tableau en question est déjà trié, le fait d'ajouter une valeur quelconque à la suite des variables triées peut faire que les tableau n'est plus trié.

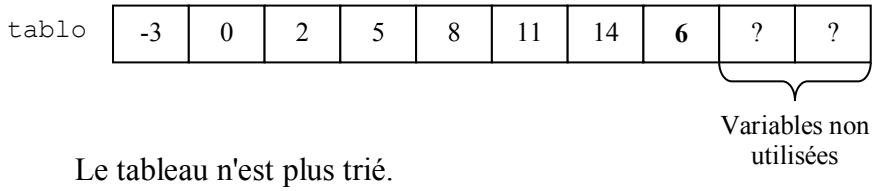
Il faudrait alors retrier tout le tableau, ce qui est inefficace. Ne peut-on pas directement insérer la valeur à ajouter à sa bonne place pour que le tableau reste trié ?

Regardons sur un schéma la manière dont nous pourrions procéder :

Soit le tableau d'entiers `tablo` trié par ordre croissant : on veut y insérer la valeur 6 située dans une variable nommé `valeur`. La taille utile de tablo est stocké dans une variable nommée `taille`.



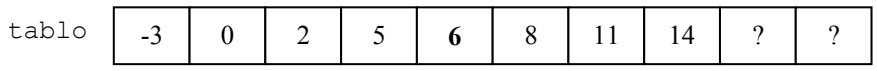
Insertion de la valeur 6 en fin de tableau :



Le tableau n'est plus trié.

On aimerait insérer la valeur 6 directement au bon endroit, c'est à dire qu'on voudrait connaître l'indice de la variable du tableau dans laquelle on veut stocker cette valeur.

Le résultat attendu est le suivant :



La valeur 6 devrait être rangée dans la variable d'indice 4. Or il se trouve qu'il y a déjà une valeur stockée dans la variable portant cet indice, c'est la valeur 8. Dans le schéma du tableau résultat, cette valeur se situe dans la variable d'indice 5. Elle a donc été décalée vers la 'droite' du tableau, idem pour les valeurs 11 et 14. Toutes les valeurs supérieures à la valeur à ranger doivent être décalées d'une position vers la droite pour libérer une variable.

Ainsi, pour ranger une valeur directement à sa bonne place dans un tableau trié, il faut :

a) déterminer l'indice de la variable dans laquelle on doit stocker la valeur à insérer;

b) décaler les variables d'indice supérieur vers la droite du tableau;

c) insérer la valeur voulue à la bonne place.

Il existe un cas particulier que l'on peut traiter simplement : celui où la valeur doit être rangée à la fin du tableau.

Reprenons ces différents points un par un pour établir l'algorithme d'insertion.

a) détermination de l'indice de la variable :

Pour savoir quel est l'indice de la variable où doit être stockée la valeur à insérer, il suffit de comparer cette valeur avec celles déjà stockées dans le tableau. Si ces valeurs sont triées par ordre croissant, alors on recherchera cet indice en parcourant le tableau depuis le début et en se déplaçant tant que la valeur à stocker est plus grande que celle de la variable du tableau en cours de traitement.

b) décaler les variables d'indice supérieur vers la droite du tableau :

Lorsque l'indice a été déterminé, appelons-le indice_trouve, on doit déplacer toutes les

c) insertion de la valeur voulue à la bonne place :

```
programme insere_val_trie

entier tablo[10];     // tableau contenant les valeurs
entier val_insere;    // la valeur à insérer
entier compteur;      // un compteur pour traiter le tableau
entier place;         // pour stocker l'indice trouvé pour insérer la
                      // valeur

entier taille_ut;     // taille utile du tableau

// initialisation du tableau

tablo[0] ← -3;
tablo[1] ← 0;
tablo[2] ← 2;
tablo[3] ← 5;
tablo[4] ← 8;
tablo[5] ← 11;
tablo[6] ← 14;

taille_ut ← 7;

afficher("entrez la valeur a inserer :");
saisir(val_insere);

// recherche de l'indice
place ← 0;

tant que ((place < taille_ut) ET (tablo[place] < val_insere))
{
     place ← place + 1;
}

// testons si la fin du tableau est atteinte

si (place = taille_ut) alors
```

```
{
// on insère directement à la dernière place
     tablo[place] ← val_insere;
}
sinon
{
// on doit décaler les variables.
// en faisant très attention au sens du décalage

     pour compteur de taille_ut à place faire
     {
          tablo[compteur] ← tablo[compteur-1];
     }
}

// insertion de la valeur

tablo[place] ← val_insere;

taille_ut ← taille_ut + 1;
```

remarque : on trouve 2 fois l'instruction : `tablo[place] ← val_insere;`. En fonction du placement respectif de ces instructions dans le programme, n'est-il pas possible de simplifier le programme ?

Supprimer une valeur dans un tableau (déjà trié)

La suppression d'une valeur dans un tableau, qu'il soit trié ou non, pose également un problème de décalages de valeurs. En effet, puisque l'on utilise uniquement un entier pour indiquer combien de variables sont utilisées, on doit supposer que ce sont les variables d'indice les plus petits. Il ne faut donc pas qu'il y ait de 'trous' dans le tableau, c'est à dire des variables inutilisées au milieu des variables utilisées, car il est impossible dans ce cas de les différencier.

Illustration : on cherche à supprimer la valeur 4 du tableau suivant :

5	4	1	0	5	?	?	?

(taille utile 5, taille maximum 8)

On ne peut supprimer la variable et réduire directement le tableau. Schématiquement, une solution naïve serait la suivante : on voudrait obtenir :

5	?	1	0	5	?	?	?

(taille utile ?, taille maximum 8)

Mais à quoi correspond ce point d'interrogation ? Il ne signifie pas que la variable n'est pas utilisée, il indique juste que la variable contient une valeur inconnue qui ne nous intéresse pas. Le seul moyen d'indiquer qu'une variable est utilisée est la taille utile !

Ce que l'on doit donc obtenir ressemble plutôt à ceci :

5	1	0	5	?	?	?	?

(taille utile 4, taille maximum 8)

Il faut donc repérer dans le tableau l'indice de la variable qui stocke la valeur indésirable, puis décaler vers la gauche toutes les variables utilisées situées à droite de celle-ci.

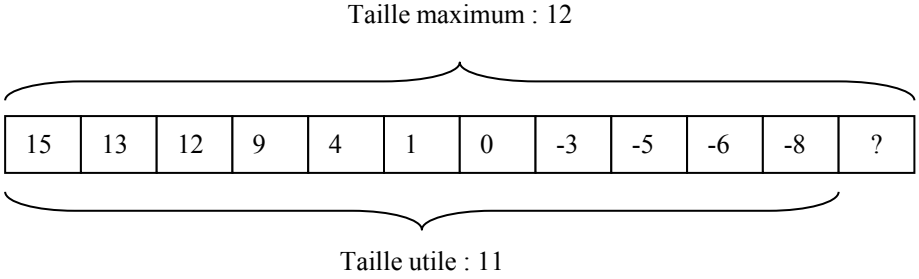
De même que l'on doit partir de la droite (indices élevés) pour faire des décalages vers la droite, on doit partir de la gauche (indices faibles) pour faire des décalages vers la gauche.

Recherche rapide dans un tableau trié

Recherche par dichotomie.

Lorsque l'on recherche une valeur dans un tableau dont les variables sont déjà triées, il existe une méthode dont les performances sont très bonnes : c'est la méthode par dichotomie (ou partage). En effet, lorsque le tableau n'est pas trié, la valeur recherchée peut être stockée dans n'importe quelle variable, et il faut donc impérativement parcourir tout le tableau. Si les valeurs sont triées (par ordre décroissant par exemple), une comparaison entre la valeur située au milieu du tableau et la valeur recherchée donne des informations supplémentaires : on saura dans quelle moitié du tableau recherche la valeur.

Exemple : soit le tableau trié contenant les valeur suivantes, et soit x la valeur à rechercher dans le tableau.



Le premier test consiste à comparer x et la valeur située dans la variable située 'au milieu' du tableau, c'est à dire sur l'exemple proposé : 1

Si $x = 1$, alors on a trouvé directement la valeur;

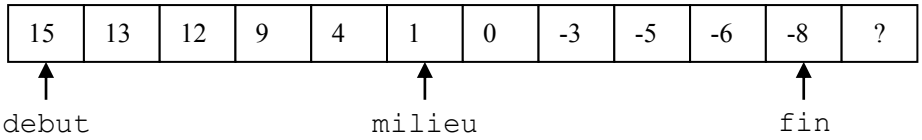
Si x > 1 alors on sait que l'on doit rechercher dans la partie gauche du tableau puisque c'est dans cette partie que sont stockées les valeurs supérieures à 1

Si x < 1 alors on sait que la recherche doit se faire dans la partie droite du tableau puisqu'elle ne contient que les valeurs inférieures à 1.

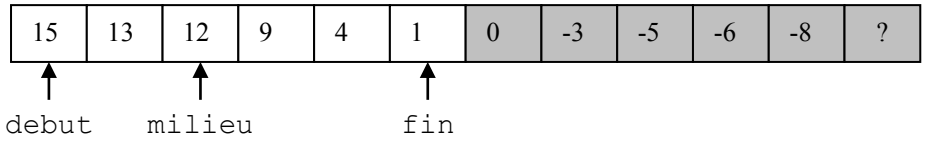
Dans les deux derniers cas, on applique de nouveau le principe de la recherche par dichotomie à la partie du tableau sélectionnée.

Exemple de recherche avec $x = 13$. Les indices limitant la zone du tableau où s'effectue la recherche sont nommé `debut` et `fin`, et sont matérialisés sur les schémas.

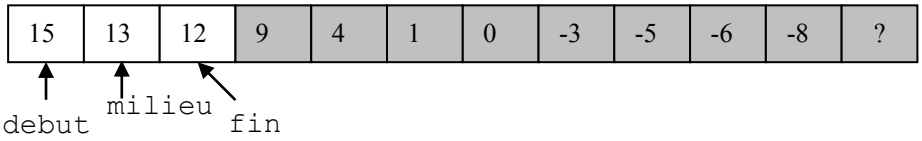
L'indice de la variable avec laquelle est comparée la valeur à rechercher est nommé `milieu`.



1er test : 13 > 1, la recherche se poursuivra dans la moitié gauche du tableau, les variables qui ne sont plus concernées par la recherche sont grisées.



2ème test : 13 > 12, la recherche se poursuivra dans la moitié gauche du tableau traité.



3ème test : 13 = 13, donc on a trouvé l'élément, au bout de trois tests.

Pour l'efficacité de cette méthode, la règle générale est que l'on divise par deux la zone du tableau à explorer à chaque étape : on est donc sur de traiter un tableau contenant 2^n valeurs en *n* étapes. Pour n = 10, cela donne une idée de l'efficacité : on traite un tableau avec 1024 valeurs en 10 étapes au maximum !

Comment détecter, avec cette méthode, qu'une valeur ne se trouve pas dans le tableau ? Lorsque les indices de limites de zone de recherche, `debut` et `fin`, sont égaux, et que la valeur stockée dans la variable correspondante n'est pas la valeur recherchée.

Quelques remarques sur la valeur de l'indice milieu : l'indice `milieu` est calculé comme la moyenne des indices `debut` et `fin`, mais le résultat ne donne pas toujours un nombre entier : en effet, lors de la deuxième étape de l'exemple, `debut` vaut 0 et `fin` vaut 5. Donc `milieu` devrait valoir 2,5; ce qui ne convient pas pour un indice. La valeur choisie pour milieu est donc la valeur tronquée (qui est obtenue par la division entière) du résultat attendu : 2 dans l'exemple.

Voici un algorithme sommaire de recherche par dichotomie :

Initialiser les indices debut et fin
Saisir (ou initialiser) la valeur à rechercher
Tant que la valeur n'a pas été trouvée et que debut < fin
    Calculer milieu à partir de debut et fin
    Si la valeur à rechercher est égale à la valeur de la variable d'indice milieu
        Alors on a trouvé la valeur
    Sinon si la valeur est supérieure à la valeur de la variable d'indice milieu
        Alors modifier debut pour rechercher dans le demi-tableau de droite
    Sinon
        Modifier fin pour rechercher dans le demi-tableau de gauche

Comparaison de tableaux

Comparer deux tableaux n'est pas aussi simple que l'on peut le penser. En effet, puisqu'on ne peut manipuler que les variables individuellement dans le tableau, on doit également faire des comparaisons individuellement.

Que signifie d'ailleurs 'comparer deux tableaux' ? est-ce que cela signifie qu'ils ont le même nom ? Ce ne peut pas être cela, car deux variables ne peuvent pas porter le même nom, l'ordinateur n'arriverait pas à s'y retrouver.

Est-ce que cela signifie qu'ils comportent les mêmes variables ? cela n'est pas possible, car puisqu'ils ont des noms différents, leurs variables portent elles aussi des noms différents.

Est-ce que cela signifie qu'ils stockent les mêmes valeurs ? oui, très probablement, et donc, pour comparer les tableaux, il faut comparer les valeurs stockées dans leurs variables

respectives. Une condition à ceci est que les deux tableaux aient la même taille utile, mais pas forcément la même taille maximale.

Il suffit donc d'utiliser une boucle tant que (pourquoi ne pas utiliser une boucle pour d'ailleurs ?) pour effectuer cette comparaison.

Les tableaux de caractère : manipulation de textes

Les tableaux de caractère ne jouent pas le même rôle, vis à vis de l'utilisateur, que les autres types de tableaux : ceux qui contiennent des entiers et ceux qui contiennent des réels. Bien entendu, on est tout à fait libre d'appliquer aux tableaux de caractère toutes les manipulations évoquées précédemment : affichage, saisie, tris, recherches, puisqu'en définitive, les caractères sont considérés par l'ordinateur comme des nombres entiers.

Cependant, il peut aussi être utile de manipuler des tableaux de caractères comme du texte, et non plus caractère par caractère.

Saisie de texte

Un exemple est la saisie d'un texte : doit-on saisir caractère par caractère et ranger toutes les valeurs dans un tableau de caractère ? Cela impliquerait, entre autres, de valider la saisie de chaque caractère par un appui sur la touche entrée…Il serait utile, dans cette situation, de disposer d'une instruction spécifique au texte, qui permettrait de le saisir et qui s'occuperait de ranger les caractères correspondants dans le tableau.

On utilise pour cela l'instruction LIRE, qui s'utilise de la manière suivante

Utilisation : LIRE(tableau de caractères);

Exemple d'utilisation :

```
caractere message[25];
lire(message);
```

Permettra de saisir du texte, et la validation se fait par un seul appui sur la touche entrée après la saisie du texte, ce qui est tout de même plus confortable ! L'instruction LIRE ne fait par contre aucun contrôle sur le nombre de caractères saisi, il faut donc faire attention à ne pas dépasser la taille maximale du tableau !

Et la taille utile dans tout ceci ?

Petit souci : en gérant la saisie caractère par caractère, on peut augmenter la taille utile de 1 à chaque valeur rangée dans une variable du tableau, mais l'instruction LIRE ne met pas à jour cette variable.

L'ordinateur arrive par contre à calculer cette taille utile en utilisant l'instruction LONGUEUR_TEXTE.

Utilisation : longueur_texte(tableau de caractères); est une valeur entière qui indique la taille utile du tableau de caractères. On n'a donc plus besoin de la stocker.

Programme exemple : saisie de texte et affichage des caractères comme avec un tableau classique.

```
Programme texte

caractere message[100];
entier compteur;

afficher("entrez votre texte (100 caracteres maximum) :");
lire(message);

pour compteur de 0 à longueur_texte(message)-1 faire
{
     afficher(message[compteur]); // affiche un seul caractère
}

afficher("votre texte contient ",longueur_texte(message), "
caracteres");
```

Affichage d'un texte

Il serait, de même qu'avec la commande LIRE, utile de disposer d'une commande d'affichage qui n'obligerait pas à utiliser une boucle pour ou tant que, comme dans l'exemple précédent. Cette command existe, il s'agit de : ECRIRE.

Utilisation : ECRIRE(tableau_de_caractères);

Exemple d'utilisation :

```
caractere dutexte[100];
afficher("entrez votre texte :");
lire(dutexte);
afficher("vous avez saisi :");
ecrire(dutexte);
```

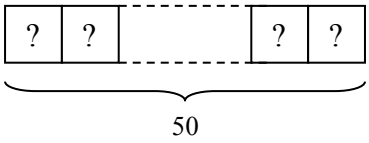
Fin de texte

Il est possible d'utiliser un caractère spécial (dont le code ASCII est égal à 0), pour indiquer que les variables suivantes du tableau ne font pas partie du texte à traiter. Nous noterons par EOT (pour : End Of Text) ce caractère spécial. C'est grâce à ce caractère EOT que l'on peut obtenir directement la taille utile d'un texte, ou qu'il peut être affiché en ne tenant pas compte des variables intuilisées du tableau de caractères correspondant.

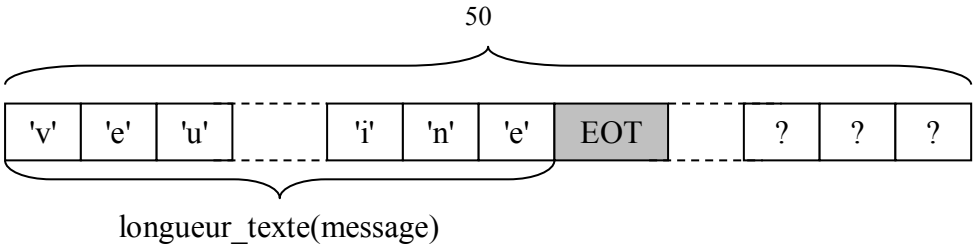
Lorsque la commande LIRE est utilisée, le caractère EOT est placé automatiquement à la fin du tableau (c'est à dire, après les caractères saisis).

Exemple :

```
caractere message[50];
lire(message);
```



si l'utilisateur entre le texte "veuillez eteindre la machine", alors le tableau `message` contiendra :

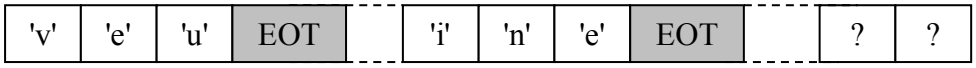


Le caractère EOT n'étant pas imprimable (il n'apparaît jamais à l'écran), il n'est pas compté dans la taille utile du tableau de caractères, mais il doit être présent dans le tableau pour marquer la fin de texte.

Que se passe-t-il si l'on écrit l'instruction suivante :

```
message[3] ← EOT;
```

le tableau est modifié de la manière suivante :



longueur_texte compte les caractères jusqu'à rencontrer le caractère EOT : `longueur_texte(message)` vaut maintenant 3.

```
ecrire(message);
```

affichera :

```
veu
```

Il arrivera que parfois, on ait besoin d'ajouter 'à la main' ce caractère EOT pour signifier la fin d'un texte stocké dans un tableau de caractères.

Copie de texte

Puisqu'un texte est un tableau de caractères, on ne peut pas copier en une instruction tout son contenu dans un autre tableau.

Rappel :

Soient tab1 et tab2 deux tableaux de caractères (définis par `caractere tab1[20], tab2[34];` par exemple), on ne peut pas écrire la ligne suivante : `tab2 ← tab1;`).

On dispose heureusement d'une commande COPIER qui recopie, caractère par caractère, le contenu d'un tableau (la source) dans un autre tableau (la destination), jusqu'au caractère EOT (qui est lui aussi recopié). Le tableau source et le tableau destination n'ont pas forcément la même taille maximale, il suffit qu'ils soient tous les deux suffisamment grands pour contenir le texte qui est recopié.

Utilisation : COPIER(tableau_destination, tableau_source); (attention à l'ordre des tableaux à fournir pour cette commande);

Exemple d'utilisation :

```
caractere tex1[30], tex2[150];

afficher("saisissez un texte :");
lire(tex1);
copier(tex2, tex1); // recopie les caracteres de tex1 dans tex2
écrire(tex2);
```

Concaténation de texte

Une opération classique avec les textes en informatique consiste à créer un texte en accolant (ou en concaténant) deux textes existants l'un à la suite de l'autre. Certains langages assez évolués permettent de réaliser cette opération grâce à l'opérateur d'addition +, dont l'utilisation dans ce cas s'adapte à la nature (au type) des données à additionner. Avec le langage algorithmique (ainsi qu'avec le langage C), l'utilisation de cet opérateur d'addition pour le texte est souvent source de confusion, et peut également avoir un autre résultat que celui d'accoler les textes concernés. Pour effectuer cette opération, qui s'avère souvent utile, on utilise une commande : CONCAT.

Utilisation : CONCAT(tableau1, tableau2);

Effet : ajoute le texte (toujours caractère par caractère) contenu dans tableau2 à la suite des caractères dans le tableau1. Le tableau1 est modifié !

Exemple :

```
caractere tab1[100], tab2[50];

copier(tab1, "ceci est le debut ");
copier(tab2, ", et voici la fin !");
concat(tab1,tab2);

afficher(tab1);
```

```
ceci est le debut, et voici la fin !
```

Que faire si l'on veut garder inchangé le texte contenu dans le tableau 1 ? il faut recopier ce texte dans un autre tableau, avant de lui appliquer la commande concat.

Exemple :

```
caractere tex_orig[100], tex_ajout[100], tex_copie[100];

afficher("entrez votre texte");
lire(tex_orig);
copier(tex_copie, tex_orig);

afficher("entrez le texte a ajouter :");
lire(tex_ajout);
concat(tex_copie, tex_ajout);

afficher("textes saisis :\n", tex_orig, "\n",tex_ajout,"\n");
afficher("texte obtenu:",tex_copie);
```

```
entrez votre texte : c'est la premiere partie
entrez le texte a ajouter : il y en a une deuxieme
textes saisis:
c'est la premiere partie
il y en a une deuxieme
texte obtenu : c'est la premiere partieil y en a une deuxieme
```

Comparaison de textes

Comparer deux textes, c'est comparer un à un les caractères contenus dans les tableaux où ces textes sont stockés. Utiliser un opérateur d'égalité entre deux tableaux n'a pas de sens en algorithmique. Plutôt que de faire cette comparaison à l'aide d'une boucle (ce qui est faisable mais guère pratique), il est possible d'utiliser la commande COMP.

Utilisation : COMP(tableau1, tableau2); Cette commande agit comme un test, c'est à dire qu'elle indique si l'égalité entre les caractères constituant les textes est VRAI ou FAUSSE, ce que l'on peut également traduire par 1 ou 0.

Exemples d'utilisation :

```
programme utilise_comp
caractere texte1[20];
caractere texte2[20];

afficher("entrez un premier texte :");
lire(texte1);
afficher("entrez le deuxieme texte :");
lire(texte2);

si (comp(texte1,texte2)) alors
{
     afficher("vous avez saisi deux fois le meme texte !");
```

```
}
sinon
{
      afficher("les textes sont différents");
}
```

Recherche d'un texte dans un autre texte

Cet exemple sera traité par plusieurs exercices en TD.

Les tableaux à plusieurs dimensions

Un tableau ressemble en fait à une matrice à une dimension, on peut choisir la variable à l'aide d'un indice. On peut utiliser plus d'une dimension, c'est à dire choisir une variable à l'aide 2, 3, … indices.

Un tableau à deux dimensions ressemble à une matrice classique, ou à un tableau à deux dimensions dans lequel on se repère avec deux coordonnées (une pour les lignes, une pour les colonnes). Le tableau à deux dimensions est souvent utilisé pour la programmation de certains jeux utilisant un plateau en deux dimensions, comme les dames, les échecs, le go, etc…

Nous allons traiter dans cette section des tableaux à 2 dimensions, mais tout ce qui sera montré est valable pour des tableaux à plus de deux dimensions (on peut facilement créer sur le même principe un tableau à 5 dimensions, même si sa visualisation est loin d'être évidente).

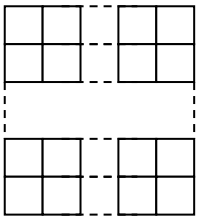
Qu'est-ce qu'un tableau à deux dimensions ?

Un tableau à deux dimensions est un tableau dont chacune des variables est elle-même un tableau. Soit `tab` un tableau à deux dimensions, il est possible de le représenter ainsi :

`tab`

Mais aussi de la manière suivante, ce qui est plus confortable :

`tab`



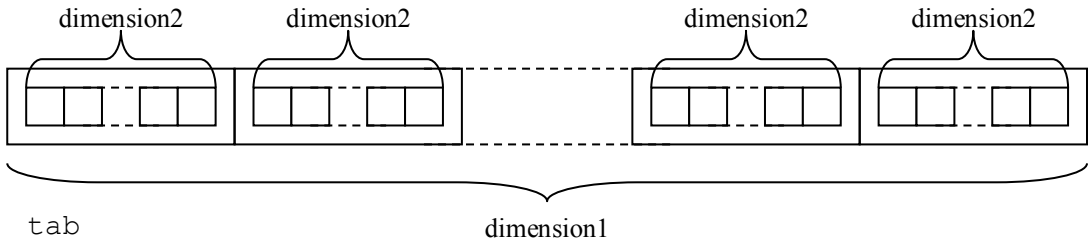
Définition d'un tableau à deux dimensions

On définit un tableau à deux dimensions en indiquant les deux tailles maximum entre crochets, de manière similaire à celle employée pour définir un tableau à une dimension :

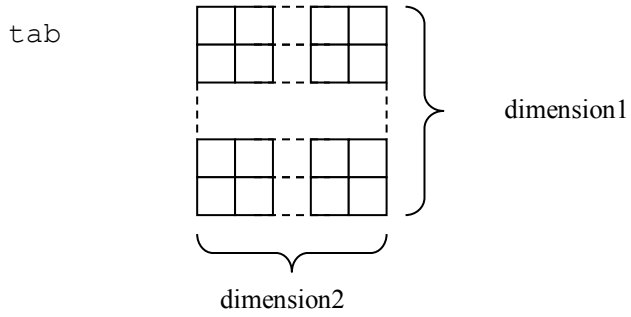
```
type nom_du_tableau[dimension1][dimension2];
```

le type est le type des variables stockées, c'est en général un type de base: entier, réel ou caractère.

Reportons ces valeurs dimension1 et dimension2 sur les schémas précédents :



ou encore :



dans cette dernière représentation, on peut donc associer la dimension1 au nombre de lignes de ce tableau, et dimension 2 au nombre de colonnes.

Prenons un exemple concret, avec la définition d'un tableau à deux dimensions comportant des valeurs réelles :

```
réel tab_2d[6][14];
```

Pour savoir comment on peut le manipuler, intéressons-nous aux types :

`tab_2d` n'est pas un réel.

la variable `tab_2d[i]`, pour *i* compris entre 0 et 5 (indices valides); n'est pas non plus un réel : c'est un tableau à une dimension contenant des variables de type réel. Il faut donc appliquer un deuxième indice pour accéder à une variable réelle.

La variable `tab_2d[i][j]`, pour *i* compris entre 0 et 6 et *j* compris entre 0 et 13 (indices valides) est un réel, et peut donc être manipulée comme tel. On peut écrire :

```
saisir(tab_2d[i][j]);
```

mais on ne peut pas écrire :

```
saisir(tab_2d);

saisir(tab_2d[i]);
```

car l'instruction saisir n'accepte que des variables dont le type est entier, réel ou caractère.

traitement avec des boucles : initialisation d'un tableau à deux dimensions.

Pour initialiser toutes les variables d'un tableau à deux dimensions, on doit utiliser deux boucles imbriquées l'une dans l'autre, ce que l'on peut traduire par :

Pour chaque ligne, initialiser individuellement toutes les variables de la ligne.

Exemple avec le tableau tab_2d, que l'on veut initialiser en stockant la valeur 1.414 dans toutes les variables du tableau. On considère de plus que le tableau doit être utilisée dans son intégralité (les tailles utiles sont égales aux tailles maximum).

```
programme initialise_2d

reel tab_2d[6][14];
entier indice1, indice2;
entier utile1, utile2;

utile1 ← 6;
utile2 ← 14;

pour indice1 de 0 à utile1-1 faire
{
      pour indice2 de 0 à utile2-1 faire
      {
            tab_2d[indice1][indice2] ← 1.414;
      }
}
```

autre exemple : un programme utilisant un tableau à deux dimensions pour représenter un damier. On utilise un tableau à deux dimensions contenant des entiers, en choisissant une convention pour représenter les cases blanches et noires, par exemple 0 pour une case noire et 1 pour une case blanche.

```
programme damier

entier damier[10][10];     // le tableau 2D représentant le damier
entier taille1, taille2;   // tailles utiles
entier blanc, noir;        // conventions pour représenter blanc et
                           // noir
```

```
entier ligne, colonne;     // indices pour parcourir le tableau

caractere case_blanche;

case_blanche ← 219; // code ascii d'une case blanche

taille1 ← 10;
taille2 ← 10;

blanc ← 1;
noir ← 0;

pour ligne de 0 à taille1-1 faire
{
      pour colonne de 0 à taille2-1 faire
      {
            damier[ligne][colonne] ← (ligne+colonne) % 2;
            // cela alterne des 0 et des 1
      }
}

pour ligne de 0 à taille1-1 faire
{
      pour colonne de 0 à taille2-1 faire
      {
            si (damier[ligne][colonne] = 0) alors
            {
                  afficher(" "); // espace pour les cases noires
            }
            sinon
            {
                  afficher(case_blanche);
            }
      }
      afficher("\n"); // passage à la ligne en fin de ligne
}
```

résultat :

Pour représenter des pions, on pourra choisir par exemple, une troisième valeur. Sachant que les pions se trouvent uniquement sur des cases d'une seule couleur (noire ou blanche), on peut donc décider de représenter les cases noires vides par la valeur 0, les cases blanches par la valeur 1, les cases noires occupées par des pions noirs par la valeur 2, et enfin les cases noires occupées par des pions blancs par la valeur 3.

RESUME :

Un tableau a une taille maximale : c'est la valeur fournie entre crochets lors de sa définition. Il ne peut contenir plus de variables que cette taille maximale

Un tableau a également une taille utile, correspondant au nombre de variables effectivement utilisées. Ces variables utilisées.

On accède à une variable d'un tableau par son indice, qui correspond à son numéro d'ordre dans le tableau.

Pour un tableau de taille maximum N, les indices vont de 0 à N-1.

L'ordinateur ne contrôle pas à l'avance la valeur d'un indice : si un indice est négatif ou supérieur à la limite autorisée, le programme risque de planter.

Les tableaux de caractère peuvent être traités de manière spéciale, même s'ils restent des tableaux, et à ce titre peuvent être manipulés comme tel.