Rappel sur les constantes

Différences par rapport aux variables

- valeur déterminée avant la compilation
- valeur non modifiable

Déclaration

Pseudo :

```
Constante pi : réel 3,14159
```

C-- :

```
const float pi = 3.14159;
```

Utilisation

Comme une variable dans les expressions, mais on ne peut rien leur affecter.

Les tableaux

Le but

Manipuler :

- un ensemble de N variables du même type
- repérées par un seul identificateur

Exemple

$t[1] \cdots t[5]$ au lieu de a,b,c,d,e

→ Permet d'écrire une boucle `Pour`

Remarque : Le nombre N doit être connu à l'avance

Déclaration

Taille souvent définie en constante

Exemple : tableau de `NbCases` entiers

```
  Constante NbCases : entier 100
  Variable  tab : tableau d'entiers
                  indicé de 1 à NbCases
```

Remarque : Préciser les indices extrêmes

On peut également définir un type

Exemple

```
  Constante NbCases : entier 10000
  type tableau_entiers = tableau d'entiers
                         indicé de 1 à NbCases
  Variables tab1, tab2 : tableau_entiers
```

Accès

```
tab[10]
tab[a + 1]
tab1[tab2[a]]
```

Exemple : dans une boucle

```
Pour i <- 1 à NbCases Faire
  tab[i] <- 1
Fin Pour
```


A propos des indices

Ne commencent pas toujours à 0 :

Exemple

```
Constantes
  première : entier -3
  dernière : entier 3
Variables
  tab : tableau de réels
        indicés de première à dernière
       // Nombre de cases :
       // (dernière-première+1)
  i : entier    // indice de boucle
Pour i <- première à dernière Faire
  tab[i] <- sqrt(abs(i))
Fin Pour
```

Exemple : par des caractères

Chaque caractère a un code propre ('A' = 65 par exemple)

→ on peut avoir un tableau de 26 cases de type réel indicé par 'A' à 'Z'

```
  tab['a'] <- tab['b'] + 2,5
```

Taille variable

On peut définir comme constante une borne pour la taille, et n'utiliser ensuite qu'une partie du tableau.

Exemple

```
Constante TailleMax : entier 1000
Variable
  tab : tableau d'entiers
        indicé de 1 à NbCases
  taille : entier;
...
afficher "taille du tableau ? (< 1000)
lire taille
Pour i <- 1 à taille Faire
  tab[i] <- i
Fin Pour
```


Tableaux en C

Déclaration et accès

- spécifier le type, l'identificateur et la taille N
- les indices vont de 0 à $N-1$

```
const int Taille=21;
int tab[Taille];
```

ou, avec une définition de type :

```
typedef int tableau_entiers[Taille];
tableau_entiers tab;
```

Initialisation

Au moment de la déclaration :
`int tab[Taille]={1,2,3};` ou
`tableau_entiers tab={1,2,3};`
Ici, les trois premières cases sont initialisées à 1, 2 et 3 respectivement, les suivantes à 0.

Initialisation dans le programme :

```
int i;
for (i=0; i<N; i++) {
  tab[i] = 1;
}
```

Toutes les cases sont initialisées à 1.

Remarque : $i \leq N-1$ est équivalent à $i < N$

Tableaux en C (suite)

Translation

Si les indices ne partent pas de 0, faire la translation :

Exemple

```
const int premiere=-3, derniere=3;
typedef float tableau_reels[derniere-premiere+1];

main(){
  tableau_reels tab;
  int i;

  for (i=premiere; i<=derniere; i++)
    tab[i-premiere] = sqrt(abs(i));
}
```


Saisie, affichage

Avec une boucle `Pour`

- En pseudo, plus tard on pourra aussi écrire directement :
  `lire tab`
  `afficher tab`
- Des fonctions à écrire en TP pour le C-- :

```
void lire_tab(tableau_entiers tab,
              int premiere, int derniere)

void affiche_tab(tableau_entiers tab,
                 int premiere, int derniere)
```

Exemple : calcul du minimum

```
Constante TailleMax : entier 1000
Type TabR : tableau de reels,
            indicé de 1 à TailleMax
            // Nombre de cases : TailleMax

Fonction min(t : TabR, a et b : entiers)
  resultat réel
Variables
  i : entier // indice de boucle
  m : reel // minimum temporaire
Debut
  m <- t[a]
  Pour i <- a + 1 jusqu'à b Faire
    Si t[i] < m Alors
      m <- t[i]
    Fin Si
  Fin Pour
  Retourner m
Fin
```

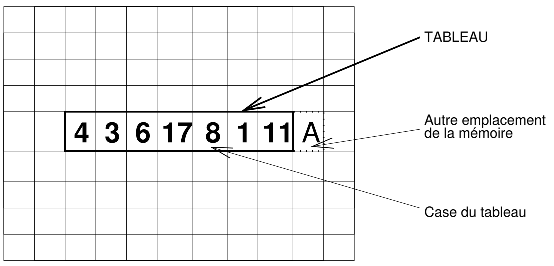

Fonctions et tableaux en C - -

Un tableau ne se comporte pas comme une variable ordinaire.

Représentation interne

- déclarer un tableau = réserver N cases *à la suite* en mémoire
- le compilateur ne connaît que l'adresse de la première case
- le type du tableau lui permet de connaître la taille d'une case (un réel prend plus de place qu'en entier par exemple) et donc d'aller à la case i :

$$\text{adresse}(i) = \text{adresse}(0) + i * \text{taille}(\text{type})$$



- **le compilateur ne connaît pas la fin du tableau : vous devez vous arrêter à temps !**

Fonctions et tableaux en C - - (suite)

En utilisant un type défini auparavant

```
typedef float tableau_reels[Taille];

float min(tableau_reels) {
  ...
}
```

Attention

- Travail direct sur les tableaux passés en paramètre : ils ne seront pas recopiés.
- Si on modifie une valeur du tableau dans la fonction, elle restera modifiée *après* la fin de la fonction.

Exemple de programme complet

```
#include <iostream.h>
#include <math.h>

const int Taille_Max = 100;
typedef float tabF[Taille_Max];

float min(tabF t, int taille) {
  int i;
  float m;
  m = t[0];
  for (i=1; i<taille; i++)
    if (t[i] < m)
      m = t[i];
  return m;
}

int main() {
  tabF tab;
  int i, taille;
  cout << "Taille ? ";
  cin >> taille;
  for (i=0; i<taille; i++)
    tab[i] = sin(i);
  cout << "Le minimum vaut : "
       << min(tab, taille) << endl;
}
```

Exercices en Cours/TD

Exercice 1 ⋄ Ecrire une fonction `max`, qui renvoie le maximum d'un tableau de réels indicés de a à b

Exercice 2 ⋄ Ecrire une fonction `moyenne`, qui renvoie la moyenne d'un tableau de réels indicés de a à b

Exercice en Cours/TD (suite)

Recherche dans un tableau

Soit un tableau d'entiers T.

Exercice 3 ⋄ Ecrire une fonction qui cherche si une valeur v apparaît dans le tableau T.

Exercice 4 ⋄ On considère de plus que ce tableau est trié dans l'ordre croissant. Réécrire la fonction précédente, pour accélérer l'algorithme de recherche précédente en utilisant une recherche dichotomique :
on utilise deux variables `petit` et `grand`, désignant respectivement la première et la dernière case du tableau. L'algorithme procède comme suit :

- Une troisième variable `milieu` est fixée à $\frac{\texttt{petit}+\texttt{grand}}{2}$.
- Si la case désignée par `milieu` contient la valeur v, l'algorithme s'arrête et renvoie `milieu`.
- Si la case contient une valeur supérieure à v, on doit chercher v dans la première moitié du tableau, donc on donne à `grand` la valeur de `milieu`.
- Sinon, on doit chercher dans la deuxième moitié, donc on donne à `petit` la valeur de `milieu`.
- Si on n'a pas trouvé, on répète alors les opérations ci-dessus.

Attention : il faut aussi prévoir le cas où v ne se trouve pas dans le tableau.

Exercices en TD/TP

Produit scalaire

Soit deux vecteurs $X = (x_1 \cdots x_n)$ et $Y = (y_1 \cdots y_n)$, à coefficients réels.

Exercice 5 ⋄ Calculer le produit scalaire $X.Y = \sum_{i=1}^{n} x_i y_i$.

Polynôme

On peut représenter un polynôme de degré n : $P(x) = \sum_{i=0}^{n} a_i x^i$ sous forme de tableau, en stockant dans un tableau de réels les coefficients $a_0, a_1, \cdots, a_n$.

Exercice 6 ⋄ Ecrire une fonction qui renvoie la valeur (approchée) d'un polynôme P appliqué à x.

Exercice 7 ⋄ Ecrire une fonction qui renvoie vrai si x est racine du polynôme P.

Attention : les variables de type "réel" peuvent subir un arrondi lorsqu'elles sont stockées en mémoire. Par conséquent :

- la plupart du temps, les valeurs calculées sont des valeurs approchées,
- plutôt que de tester l'égalité avec 0 d'une expression de type "réel", il faut tester si cette expression est inférieure à une valeur très petite, définie par exemple comme constante du programme.

Exercices en TD/TP (suite)

Cryptage et miroir

Dans les exercices ci-dessous, on utilise le fait que les valeurs d'un tableau peuvent être modifiées dans une fonction Ce mécanisme sera vu de manière plus précise dans les cours suivants.

Soit un tableau de 26 caractères, indicé par 'a' à 'z' qui contient pour chaque lettre une autre lettre de substitution en vue du codage d'un message.

Par exemple, si `code['b']` vaut `'a'` et `code['o']` vaut `'g'`, le mot `"bob"` s'écrit `"aga"`.

Exercice 8 ⋄ Définir un type `tabcode` dont chaque variable pourra servir à stocker un tel code, et un type `message` dont chaque variable contiendra un message de taille arbitraire (limitée par une constante).

Exercice 9 ⋄ Ecrire une fonction de saisie du code, qui refuse à l'utilisateur de mettre deux fois le même caractère.

Exercice 10 ⋄ Ecrire une fonction qui applique ce code pour crypter un message stocké dans un tableau de caractères. Ecrire la fonction qui décrypte un message crypté.

Exercice 11 ⋄ Pour améliorer le cryptage, on se propose de plus d'inverser la séquence cryptée par rapport à la séquence en clair. Modifier les fonctions de codage et décodage.

Exercice 12 ⋄ Trouvez une autre façon de compliquer ce code, tout en rendant possible le décodage.