IN200 : Les Fonctions

Sandrine Vial
sandrine.vial@uvsq.fr

Février 2016

Les Tableaux

- Ensemble d'éléments de même type désignés par une **unique** variable .
- Chaque élément est repéré par sa position dans le tableau

Un tableau d'entiers

12	34	45	5	234	6	-34	9	10

Déclaration

Pour définir un tableau il faut :

- un nom de variable,
- un type pour les éléments,
- une taille fixe : nombre (entier) d'éléments (de cases) dans le tableau.
- Par exemple :
  - Un tableau qui s'appelle `T` et qui contient 20 entiers :

    ```
    int T[20];
    ```

  - Un tableau qui s'appelle `TP` et qui contient 30 points :

    ```
    POINT TP[30];
    ```

Identification des éléments

- Un élément est accessible avec sa position dans le tableau.
- La première position dans le tableau porte le numéro **0**.
- Si on a un tableau de taille N, les éléments sont stockés entre les positions (indices) 0 et $N - 1$.

Un tableau de 9 entiers

	12	34	45	5	234	6	-34	9	10
Indices	0	1	2	3	4	5	6	7	8

Accès à un élément du tableau `T` déclaré par `int T[10];`

- `T[0]` : premier élément du tableau `T`.
- `T[1]` : deuxième élément du tableau `T`.
- `T[9]` : dernier élément du tableau `T`.

Attention : `T[10]` n'existe pas !

Utilisation

- Une fois déclaré un tableau peut être vu comme une suite de variables indépendantes.

```
int T[10];
```

- Ici on a 10 variables entières :

```
T[0], T[1], T[2], ..., T[9]
```

- Utilisation comme une variable habituelle :
  - `T[5] = 10 * 2 - 45;`
  - `T[4] = T[4] + 2;`
  - `T[2] = T[3] - 1;`

Initialisation des éléments d'un tableau

- Boucle pour initialiser chacune des valeurs

```
int main()
{
    int T[20];
    int i;

    for (i=0 ; i<20 ; i++)
    {
       T[i] = 0;
    }
}
```

Affichage des éléments d'un tableau

- Boucle pour afficher chacune des valeurs

```
int main()
{
    int T[20];
    int i;

    for (i=0 ; i<20 ; i++)
    {
       write_int(T[i]);
       write_text(" ");
    }
    writeln();
}
```

Les fonctions

- Permet de structurer le code
- Permet de nommer une partie du code et de pouvoir faire appel à cette partie à différentes étapes du code.

Les fonctions : un exemple

```
POINT a,b,c;

a.x = 50; a.y = 20;
b.x = 100; b.y = a.y;
c.x = 70; c.y = 100;
draw_line(a,b,bleu);
draw_line(b,c,vert);
draw_line(c,a,orange);
```

```
a.x = a.x + 100;
b.x = b.x + 100;
c.x = c.x + 100;
draw_line(a,b,bleu);
draw_line(b,c,vert);
draw_line(c,a,orange);
```

Les fonctions

Une fonction peut :

- modifier l'affichage
- avoir des effets sur les variables globales
- utiliser les paramètres et les variables locales pour effectuer des calculs
- renvoyer le résultat de ces calculs

Les fonctions

Une fonction a des paramètres, peut avoir des effets sur les variables globales et renvoie le résultat d'un calcul sur les paramètres, les variables locales et les variables globales.

```
type Nom_fonction(paramètres)
{
....
}
```

Les fonctions : le corps

Le corps d'une fonction contient :

- Des déclarations de variables locales : utilisables uniquement dans le corps de la procédure.
- Des instructions : elles portent sur les variables locales, les variables globales et sur les paramètres.
- Une instruction de renvoi de résultat si le type de la fonction est différent de `void`

Les fonctions : les paramètres effectifs

Appel de la fonction en lui donnant en paramètre des valeurs :

```
Nom_Fonction(5,6);
```

Attention : les paramètres effectifs doivent "correspondre"" aux paramètres formels :

- le *i*ème paramètre formel prend la valeur du *i*ème paramètre effectif.
- Mêmes types.

Les fonctions : un exemple

```
void dessine_triangle(POINT p1, POINT p2, POINT p3)
{
   draw_line(p1,p2,bleu);
   draw_line(p2,p3,vert);
   draw_line(p3,p1,orange);
 }

int main()
{
  POINT a,b,c;

  a.x = 50; a.y = 20;
  b.x = 100; b.y = a.y;
  c.x = 70; c.y = 100;
  dessine_triangle(a,b,c);
  a.x = a.x + 100;
  b.x = b.x + 100;
  c.x = c.x + 100;
  dessine_triangle(a,b,c);
}
```

Les fonctions : l'instruction `return`

- Permet de renvoyer le résultat d'un calcul à la fonction appelante.
- Ce qui suit l'instruction `return` doit être du même type que dans la signature de la fonction.
- Cette instrution termine l'exécution de la fonction. Toutes les instructions suivantes ne sont pas exécutées.

Les fonctions : un exemple

```
float moyenne(int a, int b, int c)
{
   float s;
   s = (a + b + c)/3.0;
   return s;
}
int main()
{
   int a = 10;
   int b = 20;
   int c = 40;
   float d;

   d = moyenne(a,b,c);
}
```

Mécanisme de passage de paramètres

- A l'appel de la fonction, les paramètres effectifs sont évalués (ont une valeur).
- Les paramètres formels sont alloués avec comme valeur initiale, une **COPIE** des valeurs des paramètres effectifs
- A l'intérieur de la fonction, on travaille donc sur une copie des valeurs.

Les fonctions

```
void decale_100(POINT p1, POINT p2)
{
    p1.x = p1.x + 100;
    p2.x = p2.x + 100;
    draw_rectangle(p1,p2,rouge);
}
int main()
{
POINT a,b;

  init_graphics(600,300);

  a.x = 50; a.y = 20;
  b.x = 100; b.y = 120;
  draw_rectangle(a,b,bleu);
  decale_100(a,b);
  draw_rectangle(a,b,noir);
  wait_escape();
  return 1;
}
```

Portée et Visibilité des variables

- Les variables :
  - Déclarées au sein d'un bloc
  - Utilisables uniquement dans le bloc dans lequel elles ont été déclarées
  - Durée de vie : celle du bloc

Visibilité des variables

- Les variables locales :
  - Déclarées au sein d'une fonction
  - Utilisables uniquement dans la fonction dans laquelle elles ont été déclarées
  - Durée de vie : celle de la fonction
- Les variables globales :
  - Déclarées en dehors de toute fonction
  - Utilisables dans n'importe quelle fonction
  - Durée de vie : celle du programme

Variables locales

```
int est_dans_cercle(POINT p, POINT centre, int r)
{
    int d;

    d = distance(p,centre);
    if (d <= r)
       return 1;
    else
       return 0;
}

int main()
{
    POINT C,P;
    int e;
    C = wait_clic();
    draw_fill_circle(C,50,rouge);
    P = wait_clic();
    e = est_dans_cercle(P,C,50)
    if (e == 1)
       draw_fill_circle(C,50,bleu);
    else
       draw_fill_circle(P,50,rouge);
}
```

Variables locales

```
void echange(int a, int b)
{
    int c;
    write_int(a); write_text(" "); write_int(b); writeln();
    c = a;
    a = b;
    b = c;
    write_int(a); write_text(" "); write_int(b); writeln();
}

int main()
{
  int a,b;

  init_graphics(600,300);
  a = 10;
  b = 20;
  write_int(a); write_text(" "); write_int(b); writeln();
  echange(a,b);
  write_int(a); write_text(" "); write_int(b); writeln();
  wait_escape();
  return 1;
}
```

Les variables globales

```
int a,b;

void echange()
{
    int c;
    write_int(a); write_text(" "); write_int(b); writeln();
    c = a;
    a = b;
    b = c;
    write_int(a); write_text(" "); write_int(b); writeln();
}

int main()
{

  init_graphics(600,300);
  a = 10;
  b = 20;
  write_int(a); write_text(" "); write_int(b); writeln();
  echange();
  write_int(a); write_text(" "); write_int(b); writeln();
  wait_escape();
  return 1;
}
```

Les variables globales

Avantage

On peut utiliser/modifier une variable globale n'importe où dans le code

Inconvénients

- On peut utiliser/modifier une variable globale n'importe où dans le code
- Les procédures sont moins réutilisables.

Une solution : les variables globales

Et si une variable globale porte le même nom qu'une variable locale ?

```
int a;

void modifie(int b)
{
   int a;

   b = b + 10;
   a = b * 2;
}

int main()
{
   int b;

   a = 10;
   b = 30;
   modifie(b);
   modifie(a);
}
```