IN200 : Les Fonctions (la suite)

Sandrine Vial
sandrine.vial@uvsq.fr

Février 2016

Les fonctions : un exemple

Les fonctions sont évaluées :
elles peuvent être incluses dans un calcul arithmétique.

```
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) + moyenne(45,67,89);
}
```

Les fonctions : un exemple

Les fonctions sont évaluées :
elles peuvent être passées en paramètres de fonctions.

```
int max(int a, int b)
{
   if (a > b)
      return a;
   else
      return b;
}
```

```
int min(int a, int b)
{
   if (a < b)
      return a;
   else
      return b;
}
```

```
int main()
{
   int a = 10;
   int b = 20;
   int c = 5;
   int  d;

   d = min(max(a,b),c);

}
```

Tableaux et fonctions

- Paramètre formel :
  `void f(int T[10])` ou `void f(int T[])`
- Paramètre effectif `f(T);`

Tableaux et fonctions

```
void Initialise_Tableau(int T[], int N)
{
   int i;

   for(i = 0; i < N; i++)
   {
     T[i] = 0;
   }
}
void Affiche_Tableau(int T[], int N)
{
   int i;

   for(i = 0; i < N; i++)
   {
     write_int(T[i]); write_text(" ");
   }
}
int main()
{
    int T[10];

    Initialise_Tableau(T,10);
    Affiche_Tableau(T,10);
}
```

Passage de paramètres

Comment faire pour pouvoir modifier la valeur d'une variable passée en paramètres ?

- Comment faire dans la signature de la fonction ?
- Comment faire dans le corps de la fonction ?
- Comment faire dans l'appel de la fonction ?

Passage de paramètres

Une variable est caractérisée par :

- son type
- son nom
- sa valeur
- son adresse

Dans le corps d'une fonction ou dans les paramètres effectifs d'une fonction :

Adresse de la variable a : `&a`

Signature de la fonction

- Déclaration d'un paramètre formel qui soit une adresse.
- Les adresses ont des types particuliers.
- Une variable qui contient une adresse est un **pointeur**

- `a` : Variable qui contient l'adresse d'un entier :

```
int *a;
```

- `b` : Variable qui contient l'adresse d'un POINT :

```
POINT *b;
```

Signature de la fonction

```
void echange(int *adr_a, int *adr_b)
{
    int c;
    ....
}

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();
}
```

Dans le corps de la fonction

- Comment modifier le contenu d'une variable lorsque l'on connait son adresse ?
- On utilise l'opérateur $*$:
- Si l'on a déclaré `int *x;` :
  - `*x` est la valeur qui est contenue à l'adresse x.
  - `*x` est de type `int`.
  - `*x` s'utilise comme un entier dans le corps de la fonction.

Dans le corps de la fonction

```
void echange(int *adr_a, int *adr_b)
{
    int c;

    c = *adr_a;
    *adr_a = *adr_b;
    *adr_b = c;
}

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();
}
```

Utilisation des adresses dans les paramètres

Avantages

- Gestion des accès à une variable.
- Les fonctions ne peuvent pas aller modifier des variables dont elles ne connaissent pas l'adresse.

Inconvénient

- Passer en paramètres toutes les variables nécessaires à une fonction.

Mécanisme de passage de paramètres

Le passage de paramètres reste `identique` que les paramètres soient des valeurs ou des adresses :

- 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.

Fonctions et Tableaux

Un tableau T de 10 points est déclaré : `POINT T[10];`

- Tous les éléments sont stockés les uns à la suite des autres en mémoire.
- A partir de l'adresse du premier élément, on a accès à tous les autres.
- L'**adresse** du premier élément du tableau est `T`.
- `*T` est de type `POINT`.
- `T[2]` est un `POINT` qui est stocké à l'adresse `T + 2`.

Fonctions et Tableaux

- **Signature d'une fonction** : Le tableau est représenté par son adresse.

```
int dans_tableau_cercle(POINT P, int N, POINT TCentre[])
```

- **Appel de la fonction** :L'appel se fait avec l'adresse du tableau.

```
k = dans_tableau_cercle(p,N,TCentre);
```

Structures de données

- Permet de regrouper sous un seul nom un ensemble de variables de types différents.
- Chaque variable est appelée un **champ**
- Plusieurs étapes distinctes :
  1. Définition un modèle de structure.
  2. Déclaration de variables de type structuré.
  3. Utilisation de variables de type structuré.

Définition d'un modèle de structure

```
struct nom_de_la_structure
{
    Liste des champs de la structure
};
```

Un article qui a un numéro, un prix, une quantité :

```
struct article
{
  int numero;
  float prix;
  int qte;
};
```

Déclaration et Utilisation d'une variable structurée

- Déclaration de la variable A de type structuré article :

  `struct article A;`

- Utilisation d'une variable structurée : opérateur `.`
  - `A.numero` donne accès au champ `numero` de la variable `A`
  - `A.numero` est une variable de type `int`

```
struct article A;

A.numero = 23;
A.prix = 45.67;
A.prix = A.prix * 1.1;
```

Utilisation globale

- Possibilité de copier une variable de type structuré dans une autre variable de même type.

```
struct article A, B;

B.numero = A.numero;
B.qte = A.qte;
B.prix = A.prix;
```

équivaut à

```
struct article A, B;

B = A;
```

Opérateur `typedef`

- Permet de définir des équivalences (synonymes) entre types.
  - `typedef int entier;`
  - `typedef struct article ARTICLE;`
- Déclaration de variables :

```
int n;
struct article A;
```

équivaut à

```
entier n;
ARTICLE A;
```

Une utilisation : le type POINT

- Dans `graphics.h` :

```
typedef struct point {int x,y;} POINT;
```

- Utilisation dans vos codes :

```
POINT p1, p2;

p1.x = 100;
p1.y = 200;

p2 = p1;
```