Introduction à la programmation
Travaux pratiques: séance 9
INFO0201-1

X. Baumans
(xavier.baumans@ulg.ac.be)

01/04/2014

Programme de la séance

- Fonctions et procédures
  $\rightarrow$ séries d'instructions paramétrables et ré-utilisables
  - Structuration du programme
  - Ré-utilisation de code
  - Modularité du code

Fonctions et procédures

Le **nombre de lignes** de code d'un programme moyennement complexe peut rapidement devenir très élevé. Pour structurer ce code on le découpe en sous-programmes appelés **fonctions**.

Ces **fonctions** peuvent être utilisées indépendamment du programme principal. On peut ainsi les utiliser à plusieurs endroits différents du même programme sans en ré-écrire le code (et éviter le copier/coller qui est source d'erreurs !). On peut même les ré-utiliser dans d'autres programmes (**modularité**).

Une fonction effectue une série d'instructions qui dépendent de la valeur de paramètres qu'on peut lui fournir. Elle retourne ensuite une valeur. Si elle ne retourne pas de valeur, on peut également lui donner le nom de **procédure** ou **routine**.

Fonctions et procédures : déclaration

Comment déclarer une procédure/fonction :

type nom_fonction(**type** *paramètre_1*, **type** *paramètre_2*, ...) ;

→ Il s'agit du **prototype** de la fonction

Fonctions et procédures : déclaration

Comment déclarer une procédure/fonction :

type nom_fonction(**type** *paramètre_1*, **type** *paramètre_2*, ...) ;

→ Il s'agit du **prototype** de la fonction

- **type** de la fonction : **int**, **float**, **double**, **char**, ...
  Définit le type de valeur que retournera la fonction. Dans le cas particulier d'une fonction (procédure) qui ne retourne pas de valeur, on utilise le mot-clé **void** pour le signaler

Fonctions et procédures : déclaration

Comment déclarer une procédure/fonction :

type nom_fonction(**type** *paramètre_1*, **type** *paramètre_2*, ...) ;

→ Il s'agit du **prototype** de la fonction

- **type** de la fonction : **int**, **float**, **double**, **char**, ...
  Définit le type de valeur que retournera la fonction. Dans le cas particulier d'une fonction (procédure) qui ne retourne pas de valeur, on utilise le mot-clé **void** pour le signaler
- **nom_fonction** : le plus représentatif possible, il servira à appeler la fonction dans le programme

Fonctions et procédures : déclaration

Comment déclarer une procédure/fonction :

type nom_fonction(**type** *paramètre_1*, **type** *paramètre_2*, ...) ;

→ Il s'agit du **prototype** de la fonction

- **type** de la fonction : **int**, **float**, **double**, **char**, ...
  Définit le type de valeur que retournera la fonction. Dans le cas particulier d'une fonction (procédure) qui ne retourne pas de valeur, on utilise le mot-clé **void** pour le signaler
- **nom_fonction** : le plus représentatif possible, il servira à appeler la fonction dans le programme
- **type** *paramètre_1* :
  - **type** : type de la variable qui est passée en paramètre
  - *paramètre_1* : nom de celle-ci, qui sera utilisé pour y accéder dans le corps de la fonction

Fonctions et procédures : déclaration

Comment déclarer une procédure/fonction :

type nom_fonction(**type** *paramètre_1*, **type** *paramètre_2*, ...) ;

→ Il s'agit du **prototype** de la fonction

- **type** de la fonction : **int**, **float**, **double**, **char**, ...
  Définit le type de valeur que retournera la fonction. Dans le cas particulier d'une fonction (procédure) qui ne retourne pas de valeur, on utilise le mot-clé **void** pour le signaler
- **nom_fonction** : le plus représentatif possible, il servira à appeler la fonction dans le programme
- **type** *paramètre_1* :
  - **type** : type de la variable qui est passée en paramètre
  - *paramètre_1* : nom de celle-ci, qui sera utilisé pour y accéder dans le corps de la fonction
- **type** *paramètre_2* : idem pour chaque paramètre, séparés par des virgules ","

Fonctions et procédures : déclaration et définition

La **déclaration** d'une fonction signale au compilateur l'existence d'une telle fonction quelque part dans le code du programme.

$\rightarrow$ Il est ensuite nécessaire de la définir. La **définition** de la fonction commence par le **prototype** de la fonction suivi des instructions que celle-ci doit accomplir, placées entre accolades **{...}**

```
1  // Définition de la fonction somme
2  double addition(double a, double b){
3
4    double somme;
5    somme = a + b;
6    return somme;
7  }
```

Le mot-clé **return** détermine la valeur retournée par la fonction. L'exécution de la fonction se termine dès que ce mot-clé est rencontré.

Fonctions et procédures

Toutes les fonctions utilisées dans le **main()** doivent être déclarées avant celui-ci.

Remarque : en effet, à un endroit donné du programme, n'existe que ce qui a été déclaré plus tôt dans le programme. Cela est valable pour les fonctions comme pour les variables.

Les fonctions peuvent donc être **définies** ailleurs (après le main) pour autant qu'elles aient été **déclarées** avant.

Portée des variables :

Les variables déclarées dans une fonction n'existent que durant l'exécution de la fonction (elles sont dites **locales**).
Les valeurs des variables passées en paramètres sont copiées dans des variables locales de la fonction (qui portent le nom qui leur a été attribué lors de la définition de la fonction).
De ce fait, les variables qui ont servi à appeler la fonction ne sont pas modifiées par la fonction.

Fonctions et procédures : exemple en pratique

```
1  // Définition de la fonction puissance entière > 0
2  double power(double base, int exposant){
3
4    double puissance = 1; // si exposant == 0, return 1
5    for(  ; exposant > 0; exposant--) // pas d'init
6     puissance = puissance * base;
7    return puissance;
8  }
```

```
1  // Définition de la fonction affichagePuissance
2  void affichagePuissance(double base, double exposant,
       double resultat){
3
4    cout << base << " à la puissance " << exposant
       << " vaut " << resultat << endl;
5  }
```

Fonctions et procédures : exemple en pratique

```
1  int main(){
2
3      double a = 2; double b = 4;
4      double resultat = 0;
5
6      // Enregistrement de la valeur retournée par la
       fonction puissance sur a, b dans resultat
7      resultat = puissance(a,b);
8      // --> a et b ne sont pas modifiés par la fonction
9
10     // Pas de variable de retour car
       affichagePuissance est une procédure
11     affichagePuissance(a,b,resultat);
12     return 0;
13 }
```

Fonctions et procédures

Pour plus de clarté, il est recommandé d'utiliser un marquage séparateur entre les fonctions et de commenter chaque en-tête pour décrire l'utilité et/ou le fonctionnement de la fonction.

```
1  /*************************************************/
2  /* La fonction addition calcule la somme de a et b */
3  /*************************************************/
4  double addition(double a, double b){
5
6    double somme;
7    somme = a + b ;
8    return (somme);
9  }
10 /*************************************************************/
11 /* affichageSomme affiche le resultat de la somme de a et b*/
12 /*************************************************************/
13 void affichageSomme(double a, double b, double somme){
14
15   cout << a << " + " << b << " = " << somme << endl;
16 }
```

Fonctions et procédures : passage d'un tableau en argument

Il est possible de fournir un tableau en paramètre à une fonction. Il faut pour cela faire suivre le nom de la variable par des crochets []. La fonction ne connait pas la taille du tableau : il faut un paramètre pour le préciser.

```
1  // Utiliser un tableau en argument
2   void AffichageTableau(double tab[], int Ntab){
3      for(int i = 0 ; i < Ntab ; i++){
4
5        cout << "tab[" << i << "] = " << tab[i] << endl;
6      }
7   }
```

ATTENTION : Dans ce cas, si les valeurs du tableau sont modifiées dans la fonction, **la modification s'appliquera aussi aux valeurs du tableau qui a été passé en paramètre**

NB : il s'agit en fait du même tableau, il n'a pas été copié.

Fonctions et procédures : passage d'un tableau en argument

Pour un tableau de dimension >1, il faut préciser la taille de chacune des dimensions.

```
1  void AffichageTableau2D(double tab[10][5])
2  {
3   for(int i = 0 ; i < 10 ; i++)
4    for(int j = 0 ; j < 5 ; j++)
5    {
6     cout << "tab[" << i << "][" << j << "] = "
       << tab[i][j] << endl;
7    }
8  }
```

Exercices (1/2)

1. Remplissage, affichage, tri et recherche dans un tableau
   - Écrire une fonction qui rempli un tableau quelconque de nombres aléatoires
   - Écrire une procédure qui affiche le tableau
   - Écrire une fonction qui recherche et rend l'élément minimum d'un tableau, ainsi que son indice (idem pour le maximum)
   - Écrire une fonction qui trie un tableau par sélection dans l'ordre croissant ou décroissant
   - Inclure tous ces éléments dans un programme qui va les utiliser pour remplir un tableau, l'afficher non trié, préciser l'élément minimum et l'élément maximum du tableau ainsi que leur indice et enfin le trier dans l'ordre choisi par l'utilisateur...

Exercices (2/2)

2. Calcul avec des vecteurs dans une base orthonormée :
   - Écrire une fonction prenant en paramètres 6 nombres représentant respectivement les 3 composantes de deux vecteurs de l'espace et qui calcule leur produit scalaire
   - Écrire une fonction prenant en paramètres 3 vecteurs de l'espace, qui calcule le produit vectoriel des deux premiers et place le résultat dans le troisième vecteur (Rem : utiliser des tableaux)
   - Appliquer ces fonctions aux vecteurs (10,5,5), (-2,2,2), (3,6,1) deux à deux et afficher les résultats
3. Calcul d'un déterminant :
   - demander à l'utilisateur d'encoder une matrice 3x3 ;
   - réaliser le calcul du déterminant à l'aide d'une fonction.