Introduction à la programmation
Travaux pratiques: séance 10
INFO0201-1

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

02/04/2014

Programme de la séance

- Rappels fonctions et procédures
  $\rightarrow$ séries d'instructions paramétrables et ré-utilisables

Programme de la séance

- Rappels fonctions et procédures
  $\rightarrow$ séries d'instructions paramétrables et ré-utilisables

- Fonctions récursives
  $\rightarrow$ fonctions faisant appel à elles-mêmes

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 : 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  }
```

Fonctions et procédures : la récursivité

Une fonction est dite **récursive** lorsqu'elle fait appel à elle-même dans sa définition.
Cette possibilité est particulièrement utile dans le cas de fonctions mathématiques définies par **récurrence**.
Exemples : la fonction factorielle, la suite de Fibonacci, ...

```
1 // Définit la fonction factorielle par récurrence
2 int factorielle(int n)
3 {
4   if(n == 1)
5     return 1;
6   else
7     return n*factorielle(n-1);
8 }
```

Comme dans le cas des boucles, il est important de veiller à ce que les appels successifs ne se répètent pas de manière infinie !

Exercices

1. Suite de Fibonacci :
   Les deux premiers éléments de la suite de Fibonacci sont 1,1.
   - demander à l'utilisateur quel terme de la suite il désire ;
   - écrire une fonction récursive qui permet de calculer l'élément i ;
   - afficher le terme demandé.

2. Recherche de zéro par dichotomie :
   Calculer le zéro de la fonction cosinus dans l'intervalle [0,2] par dichotomie en utilisant une fonction et des appels récursifs.