Introduction à la programmation
Travaux pratiques: séance 12
INFO0201-1

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

22/04/2014

Programme de la séance

- **Rappels sur les fonctions**

  $\rightarrow$ série d'instructions paramétrables et réutilisables

Programme de la séance

- **Rappels sur les fonctions**

  $\rightarrow$ série d'instructions paramétrables et réutilisables

- **Rappels lecture/écriture dans un fichier**

  $\rightarrow$ flux ifstream et ofstream

Programme de la séance

- **Rappels sur les fonctions**

  $\rightarrow$ série d'instructions paramétrables et réutilisables

- **Rappels lecture/écriture dans un fichier**

  $\rightarrow$ flux ifstream et ofstream

- **Exercices de synthèse sur ces deux points**

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 !

Écrire dans un fichier : exemple

```
1  #include <fstream>
2
3  int main()
4  {
5    int age = 19;
6    ofstream mon_fichier("data.txt", ios::trunc);
7    if(mon_fichier.is_open()) // fichier bien ouvert ?
8    {
9      // Le fichier est bien ouvert, on peut écrire
10     mon_fichier << "J'ai " << age << " ans." << endl;
11     mon_fichier.close(); // fermer le fichier
12   }
13   else
14   {
15     cout << "ERREUR: Impossible d'ouvrir le fichier";
16   }
17   return 0;
18 }
```

Lire dans un fichier : exemple

```
1  #include <fstream>
2  int main()
3  {
4    ifstream mon_fichier("data_input.txt"); //Ouverture du fichier
5    if(mon_fichier.is_open()) // fichier ouvert ?
6    {
7      while(mon_fichier.good()) // fichier lisible ?
8      {
9        int a = 0;
10       mon_fichier >> a;
11       cout << "Le fichier contient " << a << endl;
12     }
13     mon_fichier.close(); // fermer le fichier
14   }
15   else{
16     cout << "ERREUR: Impossible d'ouvrir le fichier" << endl;
17   }
18   return 0;
19 }
```

Exercices (1/2)

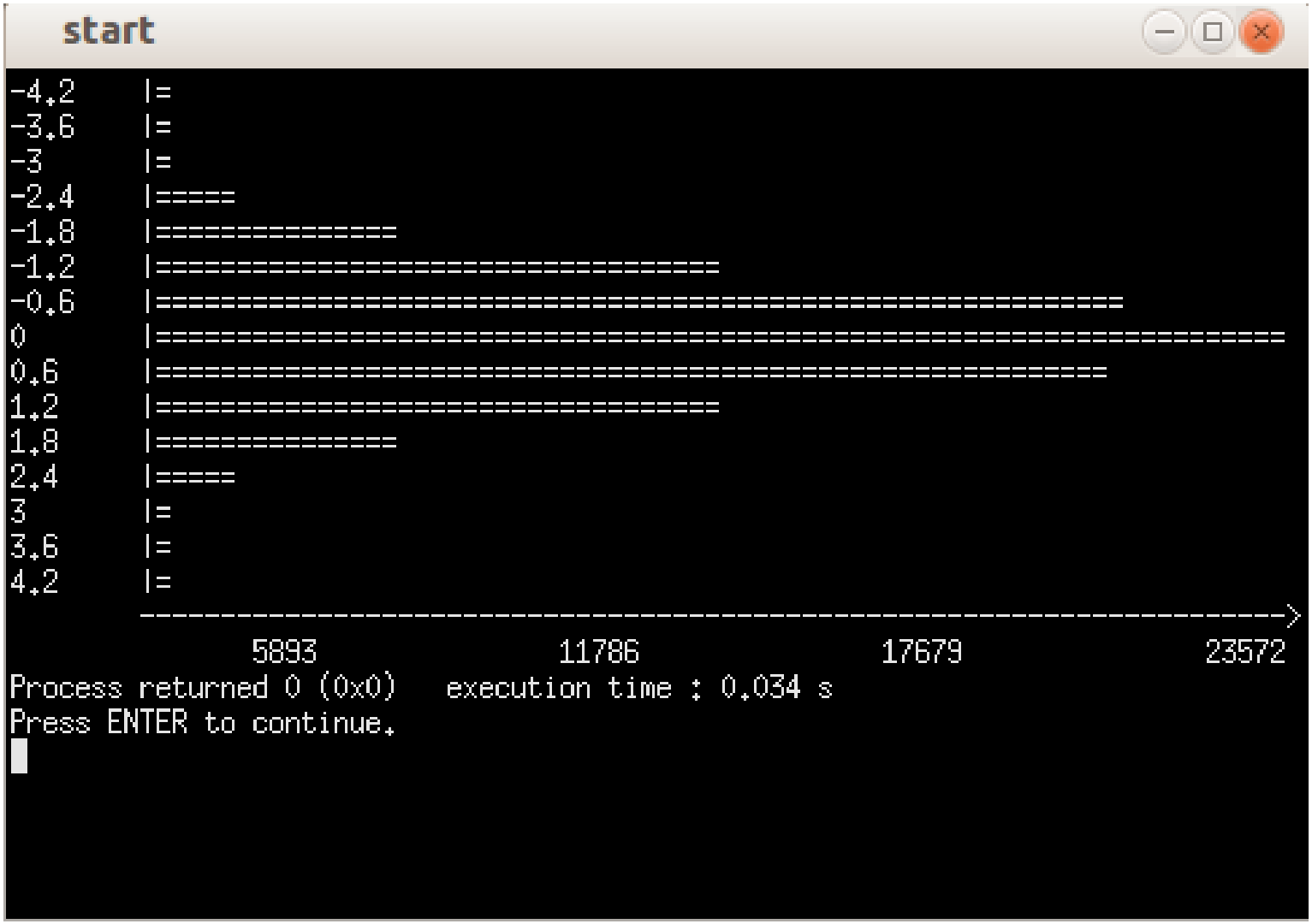
1. Crible d'Eratosthène

   Le crible d'Eratosthène est un procédé permettant de trouver les nombres premiers compris entre 2 et N. Pour ce faire, dans un tableau contenant tous les nombres de 2 à N, on supprime successivement tous les multiples d'un nombre entier. A la fin, il ne reste donc que les nombres qui ne sont multiples d'aucun autre entier. On commence ainsi par supprimer tous les multiples de 2. On supprime ensuite les multiples du plus petit entier restant suivant (3) et ainsi de suite. On s'arrête lorsqu'un entier $>= \sqrt{N}$ est atteint.

   - Ecrire une fonction qui prend comme paramètre un tableau de dimension N et, par la méthode du crible d'Eratosthène, ne laisse dans le tableau que les nombres premiers ;
   - Ecrire un programme qui calcule les nombres premiers compris entre 1 et 10 000 et les enregistre dans un fichier ;

Exercices (2/2)

2. Chargement de données et distribution : le fichier data.dat disponible à l'adresse `http://mate06.phys.ulg.ac.be/` contient des nombres.
   - Écrire une fonction qui répartit des données en classes. La fonction prend en paramètres le nombre de classes, un tableau avec la valeur centrale de chaque classe, la largeur des classes et un tableau dans lequel sera totalisé le nombre d'éléments de chaque classe ;
   - Appliquer cette fonction au fichier data.dat pour un ensemble de 15 classes de largeurs identiques couvrant l'intervalle [-4.5,4.5] ;
   - Écrire une fonction qui prend en paramètres un tableau contenant une distribution réparties en classes et la taille de celui-ci. La fonction affiche la distribution dans la console horizontalement (voir image) ;
     Remarque : la largeur des lignes dans la console est limitée à 80 caractères. Il est dès lors conseillé de ne pas dépasser 70 caractères de large pour l'affichage de la distribution...
   - Écrire la distribution dans un fichier distribution.txt
   - Tester également le programme avec le fichier data2.dat


start
-4,2 |=
-3,6 |=
-3 |=
-2,4 |=====
-1,8 |==========
-1,2 |==========
-0,6 |==========
0 |==========
0,6 |==========
1,2 |==========
1,8 |==========
2,4 |=====
3 |=
3,6 |=
4,2 |=
5893 11786 17679 23572
Process returned 0 (0x0) execution time : 0,034 s
Press ENTER to continue.