Introduction à la programmation
Travaux pratiques: séance 15
INFO0201-1

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

30/04/2014

Programme de la séance

- Type constant (variables)
  $\rightarrow$ variables dont la valeur ne peut être modifiée

Programme de la séance

- Type constant (variables)
  $\rightarrow$ variables dont la valeur ne peut être modifiée

- Mesurer le temps d'exécution d'un programme
  $\rightarrow$ Comparer la performance des algorithmes

Type constant

Comme leur nom l'indique, les valeurs des *variables* peuvent être modifiées. On peut cependant vouloir déclarer des variables particulières, dont la valeur est fixée à la première affectation et ne peut plus être modifiée. Il existe pour cela le type spécifique **const**. Une variable qui est déclarée de type **const** doit être initialisée immédiatement (lors de la déclaration) et sa valeur ne pourra être modifée dans le programme, sous peine de générer une erreur dès la compilation.

Type constant

Comme leur nom l'indique, les valeurs des *variables* peuvent être modifiées. On peut cependant vouloir déclarer des variables particulières, dont la valeur est fixée à la première affectation et ne peut plus être modifiée. Il existe pour cela le type spécifique **const**. Une variable qui est déclarée de type **const** doit être initialisée immédiatement (lors de la déclaration) et sa valeur ne pourra être modifée dans le programme, sous peine de générer une erreur dès la compilation.

```
1  int main()
2  {
3    const double PI = 3.14159265358979323846;
4    const int N = 1000;
5
6    N = 2000; // ERREUR !!!
7  }
```

Temps d'exécution d'un programme

Pour comparer les performances de différents algorithmes, il est nécessaire de pouvoir calculer le temps d'exécution d'une partie spécifique du programme.

Temps d'exécution d'un programme

Pour comparer les performances de différents algorithmes, il est nécessaire de pouvoir calculer le temps d'exécution d'une partie spécifique du programme.

On utilise pour cela la fonction **clock**, qui retourne le nombre de "battements d'horloge" depuis un moment proche du début de l'exécution du programme.

Temps d'exécution d'un programme

Pour comparer les performances de différents algorithmes, il est nécessaire de pouvoir calculer le temps d'exécution d'une partie spécifique du programme.

On utilise pour cela la fonction **clock**, qui retourne le nombre de "battements d'horloge" depuis un moment proche du début de l'exécution du programme.

Pour calculer un temps d'exécution, il suffit donc de calculer la différence entre le nombre de "battements d'horloge" à la fin de l'exécution d'une partie du programme et le nombre déjà écoulés au début.

Temps d'exécution d'un programme

Pour comparer les performances de différents algorithmes, il est nécessaire de pouvoir calculer le temps d'exécution d'une partie spécifique du programme.

On utilise pour cela la fonction **clock**, qui retourne le nombre de "battements d'horloge" depuis un moment proche du début de l'exécution du programme.

Pour calculer un temps d'exécution, il suffit donc de calculer la différence entre le nombre de "battements d'horloge" à la fin de l'exécution d'une partie du programme et le nombre déjà écoulés au début.

Enfin, pour avoir une durée qui puisse être exprimée en secondes, il suffit de diviser ce nombre de "battements d'horloge" par le nombre de battements par secondes qui vaut **CLOCKS_PER_SEC**

Temps d'exécution d'un programme

Pour utiliser tout cela, il faut inclure la librairie **ctime**

```
1 #include <ctime>
```

La fonction **clock()** retourne une variable de type **clock_t** qui est en fait un simple nombre entier.

```
1 double x = 0;
2 clock_t debut = clock();
3 for(int a=0; a < 1000000; a++)
4   for(int b=0; b < 1000000; b++
5     x = a*b*(b+1.)*(a+1.);
6 clock_t fin = clock();
7 cout << "temps = " << (fin-debut)*1000/CLOCKS_PER_SEC
    << " ms (" << fin-debut << " ticks)" << endl;
```

Temps d'exécution d'un programme

Pour utiliser tout cela, il faut inclure la librairie **ctime**

```
1 #include <ctime>
```

La fonction **clock()** retourne une variable de type **clock_t** qui est en fait un simple nombre entier.

```
1 double x = 0;
2 clock_t debut = clock();
3 for(int a=0; a < 1000000; a++)
4   for(int b=0; b < 1000000; b++
5     x = a*b*(b+1.)*(a+1.);
6 clock_t fin = clock();
7 cout << "temps = " << (fin-debut)*1000/CLOCKS_PER_SEC
    << " ms (" << fin-debut << " ticks)" << endl;
```

Temps d'exécution d'un programme

Pour utiliser tout cela, il faut inclure la librairie **ctime**

```
1 #include <ctime>
```

La fonction **clock()** retourne une variable de type **clock_t** qui est en fait un simple nombre entier.

```
1 double x = 0;
2 clock_t debut = clock();
3 for(int a=0; a < 1000000; a++)
4   for(int b=0; b < 1000000; b++
5     x = a*b*(b+1.)*(a+1.);
6 clock_t fin = clock();
7 cout << "temps = " << (fin-debut)*1000/CLOCKS_PER_SEC
    << " ms (" << fin-debut << " ticks)" << endl;
```

Exercices

1. Type constant
   - S'exercer rapidement à définir des constantes et à les impliquer dans des expressions...
   - Ex : aire du disque => Pi (TP1)
2. Temps d'exécution
   - Reprendre le programme du "Crible d'Eratosthène" (TP12) réalisé précédemment et s'arranger pour calculer et afficher le temps d'exécution de la fonction de calcul des nombres premiers pour des nombres premiers compris entre 1 et 100, 1 et 10 000 et 1 et 100 000.
3. Terminer le cryptage - décryptage...(TP14)