Cours 1 : Introduction à la programmation C++

Rabii EL GHORFI

Module : Technique de programmation avancées

Département : Mathématiques, informatique et géomatique (MIG)

EHTP 2017-2018

Evaluation

- 1 note de suivi de TP et TD (25%)
- 1 examen (25%)
- 1 mini projet (50%)

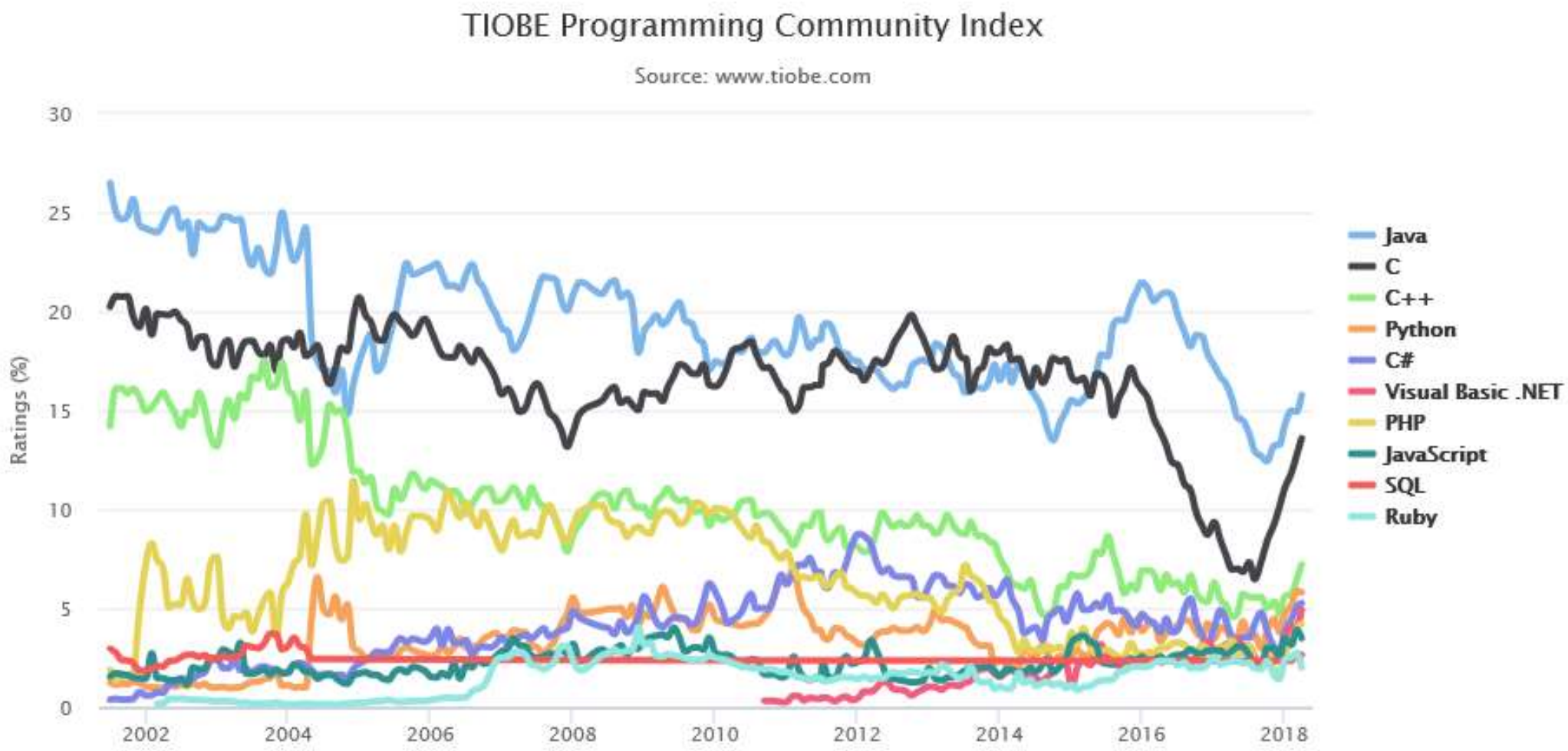
Récapitulatif de l'ensemble des cours

- Cours 1 : Introduction à la programmation C++
- Cours 2 : Les classes
- Cours 3 : L'héritage
- Cours 4 : Le polymorphisme
- Cours 5 : Les interfaces graphiques

Objectifs

- Apprendre un nouveau langage : C++
- Approfondir les connaissance en algorithmique
- Exploiter l'informatique pour résoudre des problèmes mathématiques
- Comprendre le modèle objet
- Exploiter et utiliser le modèle objet (classes, héritage, …)
- Apprendre à créer une interface graphique
- Réaliser une application de a à z

Popularité des langages de programmation



Evolution des langages de programmation

- 19__- L'assembleur
- 1952 - A-0
- 1954 - Mark I Autocode
- 1954 - FORTRAN
- 1959 - LISP
- 1971 - Pascal
- **1972 - C**
- 1972 - SmallTalk 72

- **1983 - C++** (C with classes)
- 1985 - QuickBASIC
- 1991 - Visual Basic
- 1994 - PHP
- 1995 - Java
- 1998 - C++ (ISO'98)
- 2000 - .Net, Flash
- 2002 - C#

Le langage C++

- Langage normalisé par l'ISO
- Défini dans les années 1980 (mais a évolué depuis : C++98, C++03, C++11, C++14, C++17)
- Amélioration de C qui facilite l'apprentissage pour quelqu'un qui connaît déjà C. Mais, faire du C en C++ n'est pas programmer en C++ !
- Doté d'une bibliothèque de classes et algorithmes
- Portable

C++ est presque partout :

Microsoft Windows, Office, Google Chrome, Edge, Mozilla Firefox, Oracle, Adobe Photoshop, MySQL, EA (FIFA), Ubisoft

Algorithme et programmation (1)

Définition : **Algorithme**

- Méthode pour résoudre un problème

Pour un problème donnée, il peut y avoir plusieurs algorithmes… ou aucun !

- Pour la plupart des problèmes intéressants, il n'existe pas aujourd'hui d'algorithme (Ex : Répartition des nombres premiers)
- Dans les problèmes qui restent, la grande majorité ont des algorithmes beaucoup trop durs pour être utilises ! (Ex : Jeu de Poker Vs Jeu d'échecs)
- On cherche des algorithmes simples, efficaces, élégants. . .

Définition : **Programmer**

- S'adresser à une machine = Ecrire des algorithmes

Algorithme et programmation (2)

Une fois l'algorithme trouvé, programmer en C++ comporte 3 phases:

- 1. Editer le programme avec votre éditeur favori
- 2. Compiler le programme
- 3. Exécuter le programme

  . . .
- 4. TESTER et DEBUGGER : retour au point 1 !

Ca peut durer assez longtemps...

Premiers code

L'inévitable hello world:

```
#include <iostream>
using namespace std;

int main() {
	cout << "hello world !" << endl;
	return 0;
}
```

Affiche Hello world! dans la console

Les types de bases

- Les entiers : Nombres entiers signés ou non (int)
- Les flottants : Nombres à virgule (float)
- Les chaines : Chaines de caractères (string)
- Les booléens : 2 valeurs possibles True or False (bool)
- Les tableaux : Liste d'un même type (int Tab[n])
- Les énumérations : Regroupement de plusieurs constantes (enum)
- Les structures : Regroupement de plusieurs types (struct)

Les entiers

Type	Taille en octets	Plage
short	2	[-32768, 32767]
int	4	[-2147483648, 2147483647]
long (32b)	4	[-2147483648, 2147483647]
long (64b)	8	[-9223372036854775808, 9223372036854775807]
long long	8	[-9223372036854775808, 9223372036854775807]
unsigned short	2	[0, 65535]
unsigned int	4	[0, 4294967295]
unsigned long (32b)	4	[0, 4294967295]
unsigned long (64b)	8	[0, 18446744073509551615]
unsigned long long	8	[0, 18446744073509551615]

Les flottants

Type	Taille en octets	Plage
float	4	$[-3.4*10^{-38}, 3.4*10^{38}]$
double	8	$[-1.7*10^{-308}$ à $1.7*10^{308}]$
long double	10	$[-3.4*10^{-4932}$ à $3.4*10^{4932}]$

Déclaration d'un entier :

```
int a = 89;
```

Déclaration d'un flottant :

```
float b = 76.43;
```

Les chaines de caractères

Type	Taille en octets	Plage
char	1	[-128, 127]

Pour stocker du texte 2 solutions :

Un tableau de char :

```
char mot[10] = "bonjour";
```

Le type string :

```
#include <string>
string mot = "bonjour";
```

Les booléens

Type	Taille en octets	Plage
bool	4	[-2147483648, 2147483647]

Déclaration d'un booléen :

```
bool var = true;
```

Déclaration d'un tableau de 2 booléen :

```
bool Tab[2] = { true, false };
cout << Tab[0] << endl;
```

Puisque, `Tab[0] = true` le résultat affiché est 1

Les énumérations

Syntaxe d'une énumération :

```
enum JourTravail { lundi, mardi, dimanche };
JourTravail jour = dimanche;
cout << jour << endl;
```

- La variable jour ne peut prendre que 3 valeurs
- Le résultat qui s'affiche est l'indice de la variable (= 2)

Autre exemple :

```
enum SalaireJourTravail { lundi = 200, mardi = 200, dimanche = 400 };
```

Les structures

Syntaxe d'une structure :

```
struct Point { double x; double y; };
Point p;
p.x = 2.0;
p.y = 3.0;
cout << p.x << endl;
```

- La variable point contient 2 variables de type double
- La copie d'une structure se fait facilement avec l'opérateur =

Les opérateurs

- Opérateurs arithmétiques

*, +, -, / (division entière et réelle), % (modulo)

- Opérateurs de comparaison

< (inférieur), <= (inférieur ou égal), == (égal), > (supérieur), >= (supérieur ou égal) et != (différent)

- Opérateurs booléens

&& (ET), || (OU), !(NON)

Par exemple : `(x < 12) && (z != 4)`

Les instructions (1)

- La condition IF

```
if (i == 5)
        { i = 0; }
```

- La boucle FOR

```
for (int i = 0; i < 20; i++)
        { cout << i << endl; }
```

- L'instruction BREAK

```
for (int i = 0; i < 20; i++)
        { cout << i << endl;
        if (i == 5) { break; } }
```

Les instructions (2)

- La boucle WHILE

```
while (i < 20)
        { cout << i << endl;
        i = i + 1; }
```

- La boucle DO WHILE

```
do      { cout << i << endl;
        i = i + 1; }
        while (i < 20);
```

Comportement quasi identique de WHILE et de DO WHILE

Les entrées sorties

- Afficher à l'écran

```
cout << expr1 << … << exprn;
```

- Lire au clavier

```
cin >> expr1 >> … >> exprn;
```

`cout` désigne le flot de sortie, il affiche du texte ou des variables notament grace à l'opérateur `<<`

`cin` désigne le flot d'entrée standard, on lui associe des variables grace à l'opérateur `>>` Les espaces, les tabulations et les entrées marquent le passage d'une variable à une autre

Evaluation des expressions

- `y = x++;  // y = x + 1`
- `y = ++x;  // y = x`
- `x % y`
- `if ( e1 && e2 ) { … }  // l'expression e2 n'est évaluée que si e1 est true`
- `if (e1 || e2 ) { … }  // l'expression e2 n'est évaluée que si e1 est false`

```
double i = 3, j = 2, k = 3.5;
int r;
r = (i / j) * k; // r = 5
```

```
int i = 3, j = 2, r;
double k = 3.5;
r = (i / j) * k; // r = 3
```

Remarque : Si une expression mélange plusieurs types, c'est le type le plus large qui est utilisé

Les pointeurs (1)

- Les pointeurs permettent d'accéder rapidement à des zones mémoires de façon linéaire
  L'inconvénient des pointeurs et qu'ils :
  - Nécessitent de maîtriser la taille de ses données
  - Peuvent endommager la mémoire de l'ordinateur

- Exemple :

```
int a = 25;  // Déclaration d'un entier a
int b = 10;  // Déclaration d'un entier b
int *p = &a; // Déclaration d'un pointeur sur a
*p = 7;      // Equivalent à a = 7
```

Adresse	Contenu
0x01	25 (a)
0x02	10 (b)
0x03	0x01 (p)

Les pointeurs (2)

- Deux codes source équivalents réalisant la somme des données avec et sans pointeurs :

```
int sum, tableau[256];
for (int i = 0; i<256; i++)
{ sum = sum + tableau[i]; }
```

```
int sum, tableau[256];
int *p;
p = tableau;  // p = &tableau[0]
for (int i = 0; i<256; i++)
      {sum = sum + *(p++);}
```

- Le code avec pointeurs présente un gain en performance

Les fonctions (1)

- Syntaxe d'une fonction

type fonction (type parametre1, …, type parametren) {

…

}

  - Si une fonction renvoie un résultat, il doit y avoir une instruction return
  - Si non le type de la fonction est void

- Exemple : La fonction max

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

Les fonctions (2)

- Exemple : La fonction permutation

```
void permutation(int &a, int &b) {
        int aux = b;
        b = a;
        a = aux; }

int main() {
        int x = 5, y = 10;
        permutation(x, y);
        cout << " x = " << x << endl ;
        return 0;}
```

Application (1)

- Le nombre e est considéré parmi les nombres les plus importants en mathématiques

- Euler (1707 – 1783) découvre une nouvelle façon d'exprimer e en fraction continue simple

- Les premiers chiffres de e sont :

e = 2.71828182845904523536028747135 27

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

$$e = \sum_{n=0}^{\infty}\frac{1}{n!} = \frac{1}{1}+\frac{1}{1}+\frac{1}{1\times 2}+\frac{1}{1\times 2\times 3}+\cdots$$

$$e = 2+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{4+\cfrac{1}{1+\cfrac{1}{1+\ddots}}}}}}}$$

$$e = [2; 1, 2, 1, 1, 4, 1, 1, 6, 1, \ldots, 1, 2n, 1, \ldots]$$

Application (2)

- Calculons de manière itérative les deux premières formules
- Calculons de manière récursive la fraction continue simple

Première formule : $$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n$$

```
#include <sstream>
int main() {
	double n = 100; // précision
	double formule1 = pow(1 + 1 / n, n); // (1 + 1/n)^n
	cout << "La valeur de e = " << setprecision(20) << formule1 << endl;
	return 0;
}
```

Application (3)

Deuxième formule : $$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1\times 2} + \frac{1}{1\times 2\times 3} + \cdots$$

```
int main() {
	double n = 100, formule2 = 0;
	for (int i = 1; i < n; i++) {
		double var = 1;
		for (double k = 1; k < i; k++)
			{var = var * 1/k;} // (1 * 1/2 * ... * 1/k)
		formule2 = formule2 + var;}
	cout << "La valeur de e = " << setprecision(20) << formule2 << endl;
	return 0;
}
```

Application (4)

Troisième formule :

$$e = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{1 + \ddots}}}}}}}$$

```
double f(double var, double i) {
if (i > 1) { i--;
return f(2 * i + 1 / (1 + 1 / (1 + 1 / var)), i); }
else { return 2 + 1 / (1 + 1 / var); }
}
int main() {
	double formule3 = f(1, 100); // Valeur de départ 1, Précision 100
	cout << "La valeur de e = " << setprecision(20) << formule3 << endl;
	return 0;
}
```