Programmation C++

ING2 – GSI

Pourquoi C++ ?

- Votre langage préféré ?

- https://www.tiobe.com/tiobe-index/
- https://www.codementor.io/learn-programming/beginner-programming-language-job-salary-community
- http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017
- http://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2017/

C++ versus C

- C est un sous-ensemble de C++ ?
- C++ :
  - multi-paradigme (procédural + orienté-objet)
  - programmation générique
  - type checking plus strict

C++ versus Java

	C++	Java
But	Efficacité d'exécution (performance)	Productivité du programmeur (portabilité)
Liberté	Faire confiance au programmeur	Imposer certaines contraintes
Paradigme de programmation	Procédurale et Orientée objet	Orientée objet
Gestion de mémoire	Manuellement, attention aux fuites mémoires	Garbage collection
Runtime	Compilé en code machine => exécuté par OS Buffer overflows, segmentation faults, …	Compilé en byte code puis interprété par JVM Exceptions
Performance	Compilation statique => code machine optimisé	Byte code mais progrès avec JIT

Plan

- 7 séances de 3 heures :
  - 0,5h cours
  - 2,5h de TP
- Contenu :
  1. Introduction au langage C++
  2. Classe : allocation dynamique, constructeur, destructeur
  3. Surcharge d'opérateurs
  4. Héritage & polymorphisme : classe abstraite, fonction virtuelle, redéfinition
  5. Gestion des flux I/O
  6. Templates + Librairie de templates STL
  7. QCM + TP noté

Références

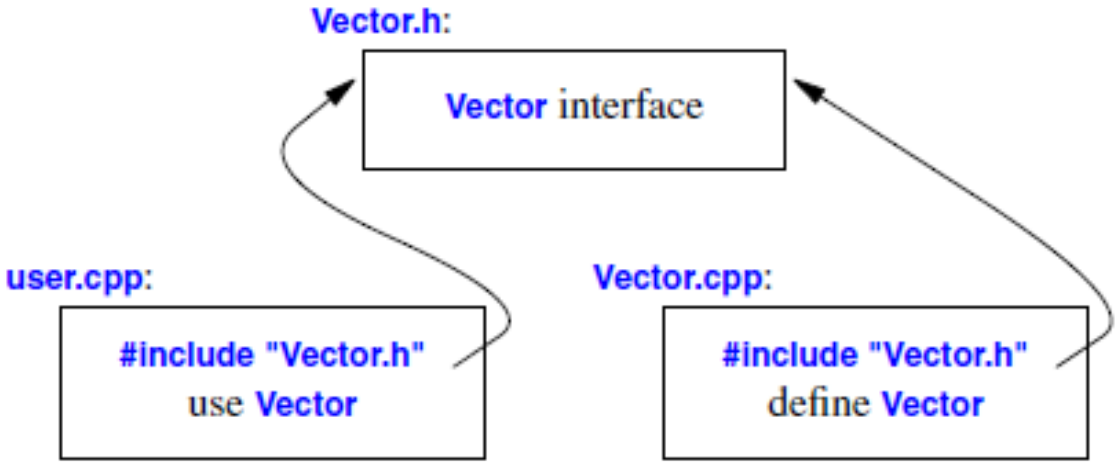
- **Bjarne Stroustrup, A Tour of C++, 2013**
- Scott Meyers, Effective C++
- …

Séance 1

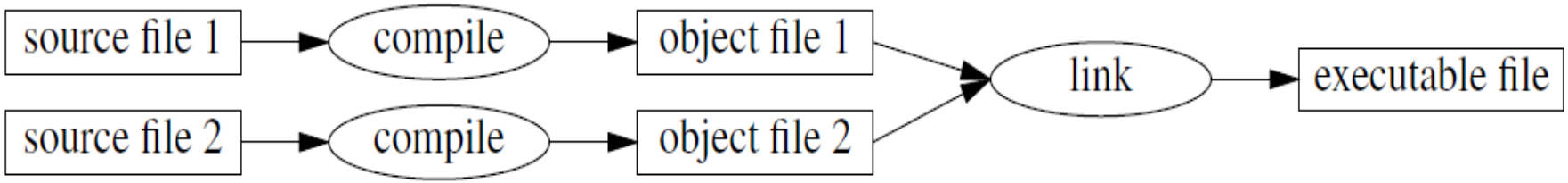
Introduction au C++

Un projet C++

- Fichier d'implémentation : file.cpp, file.cc, file.C
- Fichier de déclaration (header file) : file.h, file.hpp



Un projet C++



- Fichier d'objet : file.o
- Librairies :
  - file.lib
  - file.dll
- makefile
- Compilateurs : c++, g++, …

light.h

```
#ifndef LIGHT_H
#define LIGHT_H
class Light {
    private:
        bool on; // A light which may be on or off.
    public:
        Light (); // Makes a new light.
        void toggle (); // If light is on, turn it off, if off, turn it on
        bool isOn(); // Is the light on?
};
#endif
```

light.cpp

```
#include "light.h "

Light::Light () : on(false) {
}

void Light::toggle() {
    on = (!on);
}

bool Light::isOn() {
    return on;
}
```

Conventions de codage

- Objectifs :
  - Réutilisation de code, maintenance, portabilité
  - Optimisations
  - Éviction des erreurs
  - Sécurité
  - International Obfuscated Code Competition ☺
- Quelques règles :
  - 25 lignes 80 colonnes
  - Indentation
  - Accolades
  - Commentaires
  - Nom de variable : English, cohérent (numChars vs num_chars)
  - Nom de fonction : verbe, CheckForError() vs ErrorCheck(), dump_data_to_file() vs data_file()

Petit test

1. i
2. numberOfCharacters
3. do_this()
4. g()
5. number_of_chars
6. if
7. number of characters
8. get_number_of_characters()
9. is_char_limit()
10. charMax()
11. num1
12. 4nums
13. hxq
14. _num
15. num__chars
16. main
17. cout

Petit test

1. i
2. numberOfCharacters
3. do_this()
4. g()
5. number_of_chars
6. if
7. number of characters
8. get_number_of_characters()
9. is_char_limit()
10. charMax()
11. num1
12. 4nums
13. hxq
14. _num
15. num__chars
16. main
17. cout

Quelle version ?

```
int area_ftoi(float a, float b) { return (int) a * b / 2; }


int iTriangleArea(float fBase, float fHeight) {
    return (int) fBase * fHeight / 2;
}
```

Quelques spécificités du C++ (1)

1. **bool**
2. **inline**

```
inline int Max(int i, int j) {
        if (i>j)
                return i;
        else
                return j;
}
```

   - Pas de récursivité, pas de pointeur
   - Petite fonction, code rapide (boucle)
3. **const**
4. **volatile**

Quelques spécificités du C++ (2)

5. Initialisation {} :

```
double  d1 = 2.3;
double d2 {2.3};
string eisti {”POSE”};

int i = 7.2;
int i {7.2};
```

Quelques spécificités du C++ (2)

5. Initialisation **{ }** :

```
double  d1 = 2.3;
double d2 {2.3};
string eisti {”POSE”};

int i = 7.2;          // OK
int i {7.2};          // error
```

6. **auto**

```
auto ch = ‘x’;
auto z = sqrt(y);
```

Quelques spécificités du C++ (3)

7. **Référence vs pointeur**
   - Code plus clair, plus sûr

```
int i=0;
int & ri=i; // ri est un identificateur synonyme de i
ri=ri+1; // Manipulation de i via ri

int i=0;
int *pi=&i;
*pi=*pi+1; // Manipulation de i via pi
```

   - Passage de paramètre par référence : plus efficace

```
void test(int &i) {
    i = 2; // Modifie le paramètre passé en référence.
    return;
}
```

Petit test

- const int *pi[12];

- void (*pf)(int * const pi);

- const char *pc="Coucou !";
- char *pc="Coucou !";

Petit test

- const int *pi[12];

  pi est un tableau de 12 pointeurs, chaque pointeur pointe sur un entier constant

- void (*pf)(int * const pi);

  pf est un pointeur de fonction avec un seul paramètre pi qui est une constante de type pointeur d'entier, et cette fonction ne renvoie rien

- const char *pc="Coucou !";
- char *pc="Coucou !";

Macro

```
#define MAX(a,b) a>b?a:b

i = MAX(2,3)+5;
j = MAX(3,2)+5;
```

Macro

```
#define MAX(a,b) a>b?a:b

i = MAX(2,3)+5;
j = MAX(3,2)+5;

int i = 2>3?2:3+5;          // i = 8
int j = 3>2?3:2+5;          // j = 3
```

Macro

```
#define MAX(a,b) ((a)>(b)?(a):(b))
```

Mais

```
i = 2;
j = 3;
k = MAX(i++, j++);
```

Macro

```
#define MAX(a,b) ((a)>(b)?(a):(b))
```

Mais

```
i = 2;
j = 3;
k = MAX(i++, j++);  // side effect j = 5
```

Macro

- Ne pas utiliser des macros !

- Solutions :

```
inline max(int a, int b) { return a>b?a:b }
template<typename T> inline max(const T& a, const
   T& b) { return a>b?a:b }
std::max(3,4);
```

Namespace

```
#include <iostream>

int main(){
    std::cout << "Coucou ";
    using namespace std;
    cout << "les GSI !"   << " \n";
    return 0;
}
```

Namespace

```
#include <iostream>
namespace first {
    int first1;
    int x;
}
namespace second {
    int second1;
    int x;
}
namespace first {
    int first2;
}
int main(){
    //first1 = 1;
    first::first1 = 1;
    using namespace first;
    first1 = 1;
    x = 1;
    second::x = 1;
    using namespace second;
    //x = 1;
    first::x = 1;
    second::x = 1;
    first2 = 1;
    //cout << 'X';
    std::cout << 'X';
    using namespace std;
    cout << 'X';
    return 0;
}
```

Votre première classe C++ : ***Vector.h***

```
class Vector {
    private:
        double* elem;
        unsigned int sz;
    public:
        Vector(unsigned int s);
        ~Vector();
        double & operator[](unsigned int i);
        unsigned int size() const;
};
```

Votre première classe C++ : ***Vector.cpp***

```
#include "Vector.h" // get the interface

Vector::Vector(unsigned int s) :elem {new double[s]}, sz {s} { }

double & Vector::operator[](unsigned int i) {
    return elem[i];
}

unsigned int Vector::size() const {
    return sz;
}
// ...
```

Votre première classe C++ : ***main.cpp***

```
#include "Vector.h" // get Vector's interface

#include <cmath> // get the standard-library math function interface

double sqrt_sum(Vector & v) {
    double sum = 0;
    for (unsigned int i = 0; i < v.size(); ++i)
        sum += sqrt(v[i]);          // sum of square roots
    return sum;
}

// ...
```