Programmation C++

Introduction

ING2-GSI

CY Tech

2020-2021

Pourquoi le C++ ?

- Votre langage préféré ?

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

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	Procédurale et Orientée objet	Orientée objet
Gestion de mémoire	Manuellement, attention aux fuites mémoires	Garbage collection
Runtime	Compilé en code machine et 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

Déroulement et évaluation

- 8 séances de CM/TP de 2h
  - 1 - Introduction au langage C++
  - 2,3 - Classe : allocation dynamique, constructeur, destructeur
  - 4 - Surcharge d'opérateurs
  - 5,6 - Héritage et polymorphisme : classe abstraite, fonction virtuelle, redéfinition
  - 7 - Gestion des flux I/O
  - 8 - Templates + Librairie de templates STL
- 9e séance : TP Noté de 2h (70% de la note)
- Semaine examen : QCM d'1h (30% de la note)

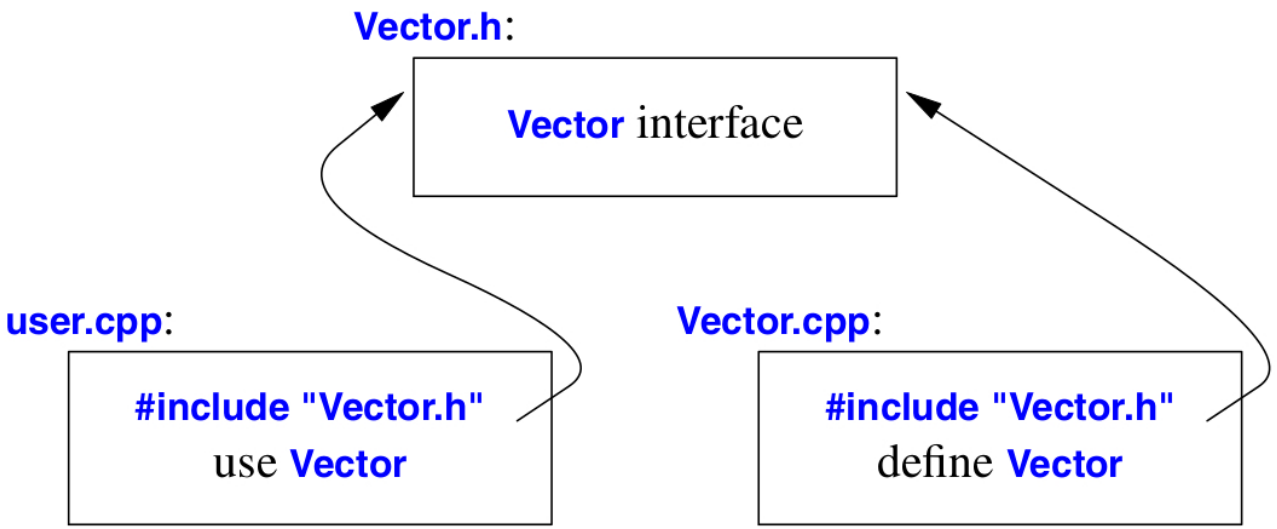
Références

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

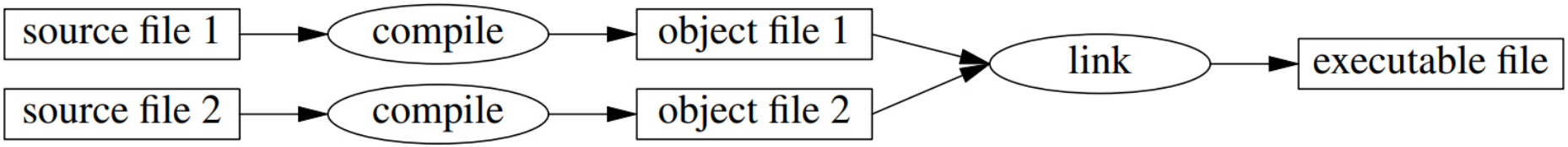
Introduction au C++

Un projet C++

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



Un projet C++



- Fichier d'objet : file.o
- Librairies : file.lib, file.dll ...
- Compilateurs : c++, g++, ... ▸ Liste partielle de compilateurs

```
g++ -Wall -std=c++11 -c file.cpp
g++ -o prog file1.o file2.o
```

- Makefile

Exemple : Hello World !

hello_world.cpp

```cpp
/* Inclusion des entetes de librairies */
#include <iostream>

int main () {
    // Affichage d'un message
    std::cout << "Hello World!" ;

    return (0);
}
```

Exemple de classe

Light.hpp

```
#ifndef __LIGHT_HPP_
#define __LIGHT_HPP_

class Light {
private :
    bool on; // A light witch 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
```

Exemple de classe

Light.cpp

```
#include "Light.hpp"

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
- 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()

Quelques spécificités du C++

Entrées et sorties standards

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

int main() {
    int age;
    char nom[] = "Bob";
    cout << "Quel est ton age ? ";
    cin >> age;
    cout << "Salut " << nom << ". Tu as " <<
        age << " ans." << endl;
    return (0);
}
```

Quelques spécificités du C++

Types, variables

- Même types qu'en C :
  - `int, char, float, double, ...`
- Existance du booléen : `bool`
  - *true, false*
- Initialisation :

```
int i = 5;
double d {3.14};
int a {42.7}; // Erreur
```

- Type **auto** :

```
auto ch = 'x';
auto z = sqrt(y);
```

Quelques spécificités du C++

Qualificateurs

- `const` : rend impossible la modification de la variable
  - évite de modifier des données par erreur, utile pour l'optimisation

```
const int cst1 = 42;
const char * fonction1(){
    return "Du texte";
}
void fonction2(const int* tab) {...}
void methodeClasse(...) const {...}
```

- `volatile` : pas d'optimisation de code par le compilateur

Quelques spécificités du C++

Macro vs. Inline

- Macro : problème de parenthèses

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

i = MAX(2,3)+5;   // i = 8
j = MAX(3,2)+5;   // j = 3    Erreur

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

Quelques spécificités du C++

Macro vs. Inline

- Macro : problème d'effet de bord

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

i = 2;
j = 3;

k = MAX(i++,j++); // j = 5
```

Quelques spécificités du C++

Macro vs. Inline

- Inline : propose au compilateur de ne pas instancier cette fonction
  - compilateur remplace l'appel de la fonction par le code correspondant
  - optimisation du code
  - pas de récursivité, pas de pointeur

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

Quelques spécificités du C++

Pointeur vs. Référence

- Pointeur

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

- Référence

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

- Passage de paramètres par référence : plus efficace

```
void test (int & i) {
    i = 2; //modifie le parametre
}
```

Quelques spécificités du C++

Namespace

- Exprimer que certaines déclarations vont ensemble
- Ne pas entrer en conflit avec d'autres noms

```
namespace toto {
    int x;
    double tmp;
}
namespace tata {
    int x;
}
int main () {
    toto::x = 5;
    using namespace toto;
    tmp = 3.14;
    tata::x = 42;
    std::cout << x << " " << tata::x ;
}
```

Votre première classe C++

Vector.hpp

```
#ifndef __VECTOR_HPP_
#define __VECTOR_HPP_

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

#endif
```

Votre première classe C++

Vector.cpp

```
#include "Vector.hpp"

Vector::Vector(int s) {
    sz = s;
    elements = new double[s];
    //TODO : initialisation des cases
}

int Vector::size() const{
    return (sz);
}

double& Vector::operator[](int i){
    //TODO :  test sur l'index
    return (elements[i]);
}
```

Votre première classe C++

main.cpp

```
#include "Vector.hpp"
#include <cmath>  // ou #include <math.h>
using namespace std;

double sum_sqrt (Vector & v) {
    double sum{0};
    for (int i=0; i<v.size(); ++i) {
        sum += sqrt(v[i]);
    }
    return (sum);
}

int main() {
    ...
}
```