Programmation C++

Introduction

ING2-GSI

CY Tech

2023-2024

Déroulement et évaluation

- 6 scéances de 3h
  - 1,5h de Cours & TD
  - 1,5h de TP
- Contenu :
  1. Introduction au langage C++
  2. Classe : allocation dynamique, constructeur, destructeur
  3. Surcharge d'opérateurs
  4. Héritage et polymorphisme : classe abstraite, fonction virtuelle, redéfinition
  5. Gestion des flux I/O
  6. Templates + Librairie de templates STL
- Évaluation : TP Noté (50%) + Examen (50%)

Introduction au C++

Evolution de la programmation

Programmation procédurale :
Suite de procédures destinées à traiter les données.
Les données et procédures sont totalement dissociées.

Programmation procédurale structurée :
Décomposition en fonctions simples
Organisation des données pour optimiser le code
Sépare les données et les fonctions

Programmation Orientée Objet (POO) :
Permet de traiter des applications très complexes.
Exploite des composants logiciels réutilisables.
Associe les données aux tâches qui les manipulent.

Evolution des langages de programmation

Langage machine : longue chaine de 0 et 1

Langage assembleur : utilisation de mnémonique (ADD, MOV,…) mais manipulation des adresses mémoires

Langage évolué interprété (script) : Syntaxe proche de la langue anglaise sous forme de ligne d'instruction
Exemple : Cobol, Fortran, Basic, Pascal, Java

Langage évolué compilé : Etape supplémentaire de compilation du code source en code objet
Exemple : Java, C, C++, C#

Introduction au 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
PARADIGME	Procédurale, Orientée objet	Orientée objet
BUT	Efficacité d'exécution (performance)	Productivité du programmeur (portabilité)
LIBERTÉ	Faire confiance au programmeur	Imposer certaines contraintes
GESTION de MÉMOIRE	Manuellement, attention aux fuites mémoires	Garbage collection
RUNTIME	Compilé en code machine, exécuté par l'OS : Buffer overflows, segmentation faults, ...	Compilé en byte code puis interprété par JVM ; Exceptions
PERFORMANCE	Compilation statique, code machine optimisé	Byte code avec JIT Mais progrès

Références

- **Bjarne Stroustrup**, A Tour of C++, 2013
- Scott Meyers, Effective C++
- http ://cplusplus.com/ cplusplus
- http ://google.github.io/styleguide/cppguide.html cppguide
- ...

Convertir un code C en C++

```
#include <stdio.h>

////////////////////////////////////////////////////////////////////////////
// Programme : Bonjour.c                                                  //
// Acces     : Public                                                     //
// But       : Bonjour CY'TECH!                                           //
//                                                                        //
// Arguments                                                              //
// - IN      : neant                                                      //
// - IN/OUT  : neant                                                      //
// - OUT     : neant                                                      //
// Retour    : neant                                                      //
//                                                                        //
// Historique                    Date        Version  par                 //
// Création Bonjour.c            30/08/2022  1.0.00   Alain BERTAILS     //
////////////////////////////////////////////////////////////////////////////

void main()
{ printf("Bonjour CY'TECH   !\n");
}
```

Convertir un code C en C++

```
#include <iostream>

////////////////////////////////////////////////////////////////////////////
// Programme : Bonjour.1.cpp                                              //
// Acces     : Public                                                     //
// But       : Bonjour CY'TECH!                                           //
//                                                                        //
// Arguments                                                              //
// - IN      : neant                                                      //
// - IN/OUT  : neant                                                      //
// - OUT     : neant                                                      //
// Retour    : EXIT_SUCCESS Execution OK                                  //
//             EXIT_FAILURE Execution KO                                  //
//                                                                        //
// Historique                      Date        Version  par               //
// Création Bonjour.1.cpp          25/12/2023  1.0.00   Alain BERTAILS    //
////////////////////////////////////////////////////////////////////////////

int main()
{     std::cout << "Bonjour CY'TECH !" << std::endl;
      return EXIT_SUCCESS;
}
```

Namespace

```
#include <iostream>
int main()
{    std::cout << "Bonjour CY'TECH !" << std::endl;
     return EXIT_SUCCESS;
}
```

Avec la définition du **namespace** :

```
using namespace std;

#include <iostream>
int main(){
     std::cout << "Bonjour CY'TECH !" << std::endl;
     return EXIT_SUCCESS;
}
```

Convertir un code C en C++

Convertir un code C en C++

```c
/* The following program computes
 *  the probability for dice possibilities */

#include <stdio.h>                  //librairies standards
#include <stdlib.h>
#include <time.h>

#define SIDES 6                             // preprocessor
#define R_SIDE (rand() % SIDES + 1)         // macro
```

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
```

Convertir un code C en C++

```
/* The following program computes
 *  the probability for dice possibilities */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SIDES 6
#define R_SIDE (rand() % SIDES + 1)

void main()
{
        ...
}
```

Convertir un code C en C++

```
srand(clock());
printf("\nEnter number oftrials: ");
scanf("%d",&trials);

for (i = 0;i < trials; ++i)
      outcomes[R_SIDE + R_SIDE]++;

printf("Probability\n");
for (i = 2;i <= n_dice*SIDES; ++i)
      printf("i=%d p=%.2f%%\n", i,
            (double)(100*outcomes[i]/ trials));
```

Convertir un code C en C++

```
/* The following program computes
 *  the probability for dice possibilities */

#include <iostream>                    //librairies standards
#include <ctime>
using namespace std;

const int sides = 6;
inline int r_sides() {return(rand() % sides + 1);}
```

Convertir un code C en C++

```
cout << "\nEnter number of trials : ";
int trials;
cin >> trials;

for (int i = 0; i < trials; ++i)
    outcomes[r_sides() + r_sides()]++;

cout << "Probability\n";
for (int   i = 2;      i <= n dice*sides; ++i) cout
    << "i = " << i << " p = "
    << static cast<double>(100*outcomes[i]/ trials)
    << endl;
```

Donc

- **const** ⇒ correction
- **inline** ⇒ performance
  - petites fonctions sans boucle, switch, récursivité
- **static_cast**<type> ⇒ safe cast
- **namespace** ⇒ encapsulation
  - ne pas abuser *using namespace*
- variable : déclaration et initialisation où il le faut ⇒ lisibilité
  - **for** (**int** i = 0; i < size; ++i)
  - https ://google.github.io/styleguide/cppguide.html#Local Variables

Local Variables

Swap en C

```
void swap(int *i, int *j)
{
   int tmp = *i;
   *i = *j;
   *j = tmp;
}
```

- utiliser pointeur pour passage par référence (vs passage par valeur)

Swap en C

```
void swap_double(double * i, double * j) {
    double tmp = *i;
    *i = *j;
    *j = tmp;
}

int main() {
    int m = 5, n = 10;
    double x = 5.3, y = 10.6;
    swap(&m, &n);
    swap_double(&x, &y);
}
```

Swap en C++

```
inline void swap(int & i, int & j) {
    int tmp = i;
    i = j;
    j = tmp;
}
```

- passage par référence : &

Swap en C++

```
inline void swap(int & i, int & j) {
    int tmp = i;
    i = j;
    j = tmp;
}

inline void swap(double & i, double & j) {
    double tmp = i;
    i = j;
    j = tmp;
}
```

- **overloading** !

Swap en C++

```
int main() {
    int m = 5, n = 10;
    double x =  5.3, y = 10.6;
    swap(m, n);
    swap(x, y);
}
```

- le compilateur va choisir la bonne fonction (*signature matching algorithm*) : nombre et type de paramètres !
  - faciliter la lisibilité du code (même nom même activité)
  - faciliter la réutilisation (un seul code avec la programmation générique)

Template avec C++

```
template <typename T>
inline void swap(T & i, T & j)
{
    T tmp = i; i = j;
    j = tmp;
}
```

- un seul code pour swap des ints, doubles, Pokemons, ... !

Exercice : convertir cette fonction en C++ avec template

```
double sum(double data[], int size) {
    double s = 0.0;
    int i;
    for (i = 0;i < size;++i)
    s += data[i];
    return s;
}
```

Réponse : convertir cette fonction en C++ avec template

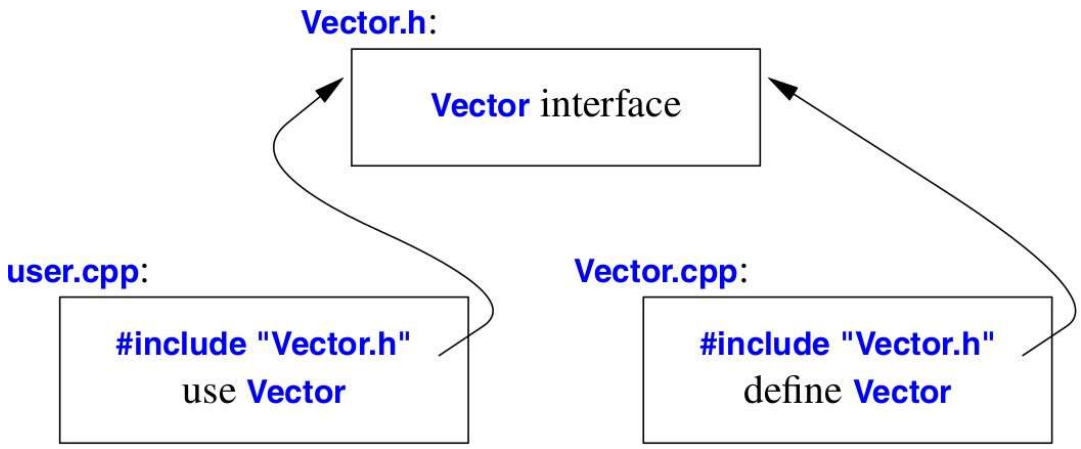
```
template <typename T>
T sum(const T data[], int size) {
    T s = 0;
    for (int   i = 0; < size; ++i)
                i s +=
        data[i];
}  return s;

int main() {
    int a[] = {1, 2, 3};
    double b[] = {2.1, 2.2};
    cout << sum(a, 3) << endl;
    cout << sum(b, 2) << endl;
}
```

Donc

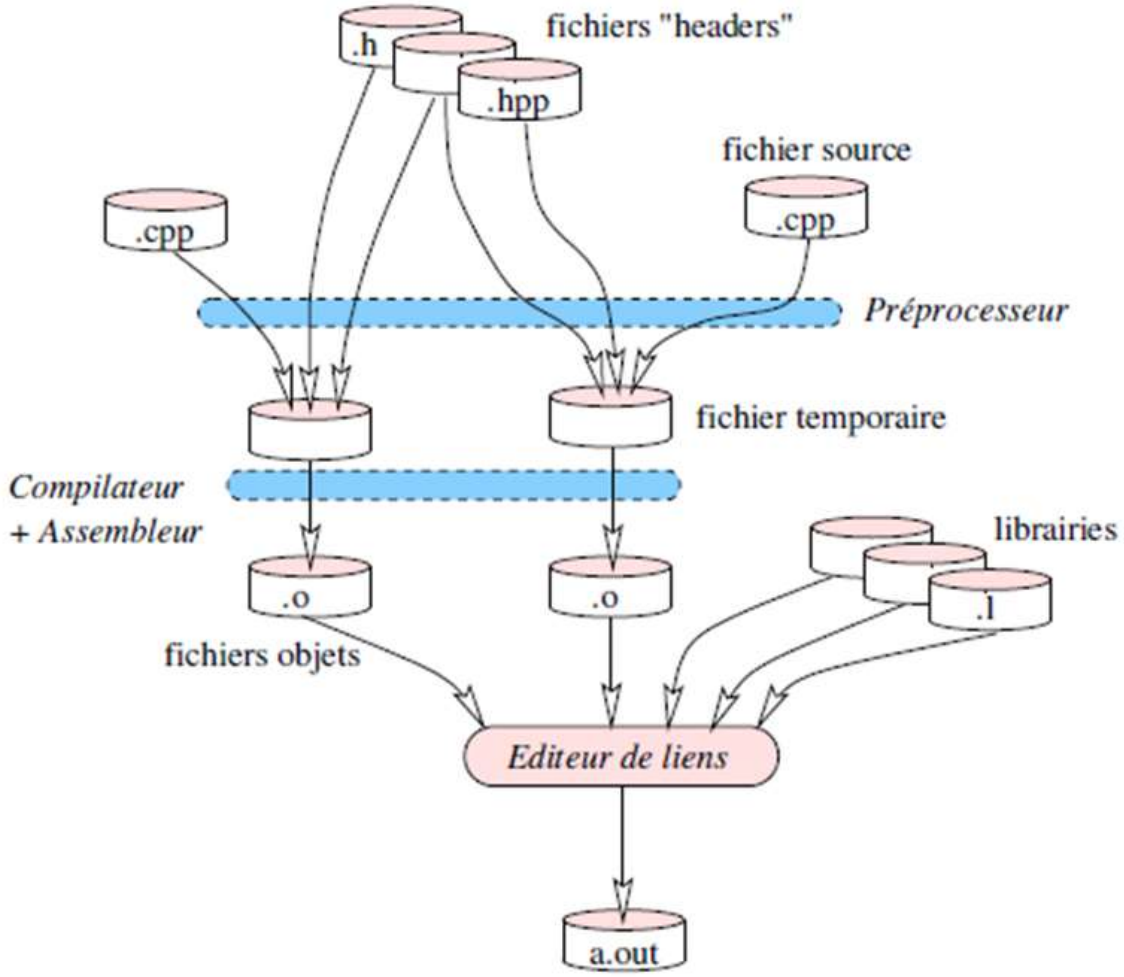
- **overload** ⇒ programmation orientée-objet
- **template** ⇒ programmation générique

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++


fichiers "headers"
.h
.hpp
fichier source
.cpp
.cpp
Préprocesseur
fichier temporaire
Compilateur
+ Assembleur
librairies
.o
.o
.l
fichiers objets
Editeur de liens
a.out

Un projet C++

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

```
g++ -Wall -std=c++14 -c file.cpp
g++ -o prog  file1.o file2.o
```

- Makefile

Votre première classe C++

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 off, turn it on
    bool isOn(); // is the light on? turn it on
};

#endif
```

Votre première classe C++

Light.cpp

```
#include "Light.hpp"

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

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

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

Votre deuxième classe C++

Vector.hpp

```
#ifndef __VECTOR_HPP_
#define __VECTOR_HPP_

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

#endif
```

Votre deuxième classe C++

Vector.cpp

```
#include "Vector.hpp"

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

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

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

Votre deuxième 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 (unsigned int i=0;i<v.size(); ++i)
      sum += sqrt(v[i]);
  return (sum);
}

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