Introduction à C++

Frank Singhoff

Bureau C-203

Université de Brest, France

Laboratoire Lab-STICC UMR CNRS 6285

singhoff@univ-brest.fr

Remerciements

Merci à Éric Beaudry pour la mise à disposition de son support et de ses exemples, dont ces quelques transparents ont largement bénéficiés.

Introduction

- Extension orienté objet du langage C.

- Développé par Bjarne Stroustrup[STR 13] dans les années 1980.
- Normalisé par ISO/IEC depuis 1998 (ISO/IEC 14882:1998). Dernière version du standard en 2023.
- Language complexe à maitriser :
  - Héritage multiples vs héritage simple/interface
  - Passage d'arguments
  - Allocation/désallocation => constructeurs/destructeurs
  - Surcharge des opérateurs. Méthodes virtuelles.
  - A probablement motivé certains choix dans Java

Introduction

- Ceci n’est pas un cours de C++

- Non exhaustif, langage bien plus riche et complexe
- Focale sur l’essentiel pour CORBA
- Ne pas revenir sur C

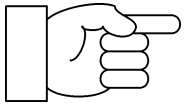
Sommaire

1. Types de base, flot de contrôle
2. Entrées/sorties
3. Passage d’argument
4. Classes et objets

Types de base, flot de contrôle

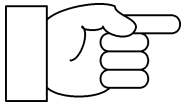
- void, bool, char, short int, int, long, float, double, unsigned
- Conditionnelle (if), boucles (for, while)

Entrées/sorties



Flux standards avec la bibliothèque $iostream$ qui définit

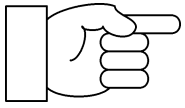
- $std :: cin$, flux d'entrée depuis $stdin$
- $std :: cout$, flux de sortie vers $stdout$
- $std :: cerr$, flux de sortie vers $stderr$



Fichiers avec la bibliothèque $fstream$ qui définit

- $ifstream$, fichier en lecture
- $ofstream$, fichier en écriture

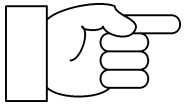
Entrées/sorties



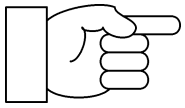
Exemple 1 :

```
#include <iostream>

using namespace std;

int main (int argc, char ** argv) {
     cout << "hello" << endl;
}
```

Entrées/sorties



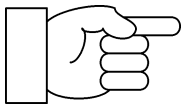
Opérateur de portée => `::`



Exemple 1 bis :

```
#include <iostream>

int main (int argc, char ** argv) {
    std::cout << "hello" << std::endl;
}
```

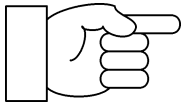
Entrées/sorties



Exemple 2 :

```
#include <iostream>

using namespace std;

int main (int argc, char ** argv) {
    int a, b;
    cout << "Entrez deux nombres:" << endl;
    cin >> a >> b;
    int somme = a + b;
    cout << "La somme est " << somme << endl;
}
```

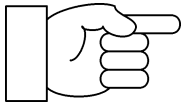
Entrées/sorties



Exemple 3 :

```
#include <iostream>
#include <fstream>

using namespace std;

int main (int argc, char ** argv) {
   int a, b;
   ifstream fin("nombres.txt");
   cout << "Lire deux nombres depuis nombres.txt" << endl;
   fin >> a >> b;
   int somme = a + b;
   ofstream fout("somme.txt");
   cout << "Ecrire resultat dans somme.txt" << endl;
   fout << somme << endl;
}
```

Passage d'argument



Règles de passage des arguments lors d'un appel de méthode/fonction:

- Par défaut, comme C : passage par copie.
- Quand le paramètre est un pointeur: opérateurs pour indiquer un pointeur ou l'objet pointé (opérateur *), son addresse (opérateur &).
- Notion de référence: modification de l'argument dans la fonction. Opérateur &
- Paramètre $const$: modification du paramètre interdite à la compilation dans la fonction/méthode

Passage d'argument

- Example 4 :

```
void test(int a, int* b, int& c, const int d, int*& e) {
   a=11; // modification dans la pile d'execution
   *b=13; // change la valeur
   c=14; // change la valeur
   //d=15; // compile pas car const
   e=b; // change la valeur du pointeur
}

int main (int argc, char ** argv) {
   int v1=1, v2=2, v3=3, v4=4, *p5=&v1;
   test(v1, &v2, v3, v4, p5);
   cout<<v1<<" "<<v2<<" "<<v3<<" "<<v4<<" "<<*p5<<" "<<endl;
   // affiche : 1 13 14 4 13
   return 0;
}
```

Classes et objets

- Classe: membres données, membres méthodes (dont constructeurs et destructeurs). Encapsulation.
- Membres $public$, $protected$, $private$.
- Exemple 5 :

```
class point
{
public:
  double x,y;
  point  milieu(const point &P);
};
```

Classes et objets

```
#include "p1.hh"

point point::milieu(const point &P) {
  point M;
  M.x = (P.x+x) / 2.0;
  M.y = (P.y+y) / 2.0;
  return M;
}
```

Classes et objets

```
#include <iostream>
#include "p1.hh"

using namespace std;

int main()
{
  point A, B, C;
  C.x=10; C.y=10;

  cout << "Donner x : "; cin >> A.x;
  cout << "donner y : "; cin >> A.y;
  B=A.milieu(C);
  cout<<B.x<<endl;
  cout<<B.y<<endl;
  cout << endl;
}
```

Classes et objets

- Une classe peut surcharger divers opérateurs
- Par exemple, les opérateurs arithmétiques (+,-,/, *), mais pas uniquement => opérateurs << et >>
- Exemple 6 :

```
class point
{
public:
  double x,y;
  point  milieu(const point &P);
  void operator<<(istream& in);
  void operator>>(ostream& out);
};
```

Classes et objets

```
void point::operator<<(istream& in) {
in >> x;
in >> y;
}

void point::operator>>(ostream& out) {
out << x;
out << y;
}

int main() {
  point A, B, C;
  C.x=10; C.y=10;

  cout << "Tapez x puis y : "<<endl;
  A << cin;
  B=A.milieu(C);
  B >> cout;
  cout << endl;
}
```

Classes et objets

- Constructeurs : allocation, initialisation des membres.
- Destructeurs : teminaison de l'objet, i.e. libération de la mémoire.
- Allocation dynamique : new
- Appel d'un destructeur : delete
- Exemple 7 :

```
class point {
public:
  point();
  point(double x_, double y_);
  ~point();

  double x, y;
  point  milieu(const point &P);
};
```

Classes et objets

```
point::point() {
  this->x=0;
  this->y=0;
}

point::point(double x_, double y_) {
  this->x=x_;
  this->y=y_;
}

point::~point() {
  cout<<"Appel du destructeur: "<<this->x<<" "<<this->y<<endl;
}
```

Classes et objets

```
int main() {
  point *A = new point ();
  point *C = new point (10,10);
  point B;

  cout << "SAISIE DU POINT A" << endl;
  cout << "Tapez x puis y : "<<endl;
  *A << cin;
  B=A->milieu(*C);
  B >> cout;
  cout << endl;
}
```

Classes et objets

- Héritage simple, héritage multiple.
- Héritage public, protected, private.
- Redéfinition de méthode et méthodes virtuelles
- Classes abstraites et méthodes pures.

Classes et objets

- Example 8 :

```
class Polygone {
public:
  Polygone() {nb_sommets=0;};
  Polygone(int n) {nb_sommets=n;};
  virtual int aire() = 0;
  virtual void affiche();
private:
  int nb_sommets;
};

void Polygone::affiche() {
   cout<<"Polygone::nb_sommets = "<<nb_sommets<<endl;
}
```

Classes et objets

```
class Carre : public Polygone {
public:
  Carre() {dimension=0;};
  Carre(int d) {dimension=d;};
  virtual int aire();
  virtual void affiche();
private:
  int dimension;
};
```

Classes et objets

```
class Rectangle : public Polygone {
public:
  Rectangle()
    {hauteur=0; largeur=0;};
  Rectangle(int h, int l)
    {hauteur=h; largeur=l;};
  virtual int aire();
  virtual void affiche();
private:
  int hauteur, largeur;
};
```

Classes et objets

```
void Carre::affiche() {
  cout<<"Carre::dimension = "<<dimension<<endl;
}
void Rectangle::affiche() {
  cout<<"Rectangle::hauteur = "<<hauteur<<endl;
  cout<<"Rectangle::largeur = "<<largeur<<endl;
}
int Carre::aire() {
   return dimension*dimension;
}
int Rectangle::aire() {
   return hauteur*largeur;
}

int main (int argc, char ** argv) {
  Rectangle r1(10,20);
  Carre c1(10);
  //Polygone  p1; ne compile pas
  r1.affiche();
  c1.affiche();
}
```

Classes et objets

- Le type d'héritage :
  - Spécifier différents niveaux s'encapsulation.
  - Contraindre casts et redéfinitions de méthode.
- Par défaut: héritage $private$.
- Membres $private$ de la classe mère inaccessibles pour la classe file.
- Membres $protected$ de la classe mère deviennent soit $protected$, soit $private$.

Classes et objets

	Héritage public	Héritage protected	Héritage private
Membre public	public	protected	private
Membre protected	protected	protected	private
Membre private	private	private	private

Références

[STR 13] Bjarne Stroustrup. « The C++ Programming Language », 2013.