2019-2020

Programmation C++

1

DR. AMINA JARRAYA
JARRAYA.AMINA.FST@GMAIL.COM

Plan

- **Introduction à C++**
- **Du C à C++**
- **Programmation Orientée Objet en C++**
  - **Classes, Objets en C++**
  - **Encapsulation**
  - **Polymorphisme (surcharge et redéfinition)**
  - **Héritage simple et multiple**

Introduction à c++

- C++ est un langage orienté-objet. Il est issu des travaux de Bjarne Stroustrup dans les laboratoires d'AT&T Bell à la fin des années 70.
- C++ est étroitement apparenté au C, comme son nom l'indique. Il en reprend d'ailleurs l'ensemble des règles syntaxiques et constitue de fait une extension du C (d'où le terme C++, sémantiquement et lexicalement valide).
- C++ adopte le typage fort et rajoute de nouvelles bibliothèques

C++ = C + POO

Introduction à c++

- **Les inclusions**
  - La manière classique d'inclure les fichiers d'en-tête des bibliothèques du C était la suivante :

    #include <stdio.h>
  - En C++
    - Extension non nécessaire
    - Préfixer par le caractère c (il s'agit d'un fichier de la librairie C)

      #include <cstdio>
- **Les entrées/sorties**
  - En C, le programme pouvait communiquer avec l'extérieur sur trois canaux :
    - l'entrée standard : stdin (scanf, getchar(), gets(),...)
    - la sortie standard : stdout (printf, putchar(), puts(),...)
    - l'erreur standard : stderr
  - En C++, on retrouve cette notion des troix canaux sur lesquels vont travailler troix flux dédiés :
    - le flux en entrée : cin
    - le flux en sortie : cout
    - le flux d'erreur : cerr

Introduction à c++

- **Chaque flux est en fait un objet.**
  - La sortie d'objets sur les flux de sortie (cout et cerr) se fait avec l'opérateur <<.
  - La lecture d'objets sur le flux d'entrée (cin) repose sur l'opérateur >>.

- **Pour pouvoir utiliser les flux, il faut inclure la librairie iostream**

En c++	En C

#include <iostream>
Using namespace std;
 	
int main() {
cout<<“hello world!” << endl;
}
	#include <stdio.h>
	
	int main() {
	printf(“hello world! \n”);
	}

Introduction à c++

- **Un autre exemple…**

En c++

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

int main() {

int age;
char nom[64];
cin >> nom;
cin >> age;
cout << "Coucou " << nom
<< ", tu as " << age << "
ans" << endl;
}
```

En C

```
#include <stdio.h>

int main() {
int age;
char nom[64];

gets(nom);
Scanf("%d", &age);

Printf("coucou %s, tu as %d
ans", nom, age);
}
```

Introduction à c++

- Emplacement libre des déclarations
  - for (int i=0; i<MAX; i++) { ...}

En c++

```
int i=5;
int j=2;
for (int i=0; i<10; i++)
{
        j+=i; // on est sur le i local }
cout << i << endl; // i vaut 5 !
cout << j << endl; // j vaut 47 !
```

Introduction à c++

- Les nouveaux types : type bool
  - Le C ne possède pas de type booléen spécifique.
  - C++ introduit le type bool pour palier à cette carence. Il s'accompagne des mot-clés true et false
  - La conversion int-boolean respecte toujours le formalisme du C.

En c++

```
bool flag=true;
....
do {
....
if (....)
    flag=false; .
...
}
while (flag==true);
....
```

Introduction à c++

- Les nouveaux types : type référence
  - Les références sont des synonymes d'identificateurs. Elles permettent de manipuler une variable sous un autre nom que celui sous laquelle cette dernière a été déclarée.
  - Par exemple, si « id » est le nom d'une variable, il est possible de créer une référence « ref » de cette variable. Les deux identificateurs id et ref représentent alors la même variable, et celle-ci peut être accédée et modifiée à l'aide de ces deux identificateurs indistinctement.
  - Il est donc impossible de déclarer une référence sans l'initialiser (doit être lié à un identificateur de variable)
  - Syntaxe :

    type &référence = identificateur;

Introduction à c++

- Les nouveaux types : type référence

➔ Nouveau type de passage par paramètres dans les fonctions

- passage par valeur
- passage par adresse
- ***passage par référence***

// passage par valeur	// passage par pointeur	// passage par référence
void Fonction1(int a) {a = 2;}	void Fonction2(int * a) {*a = 2;}	void Fonction3(int & a) {a = 2;}

```
int x=0;    int y=0;    int z=0;
Fonction1(x); // la valeur de x ne change pas (passage par valeur)
Fonction2(&y); // la valeur de y passe à 2 (passage par pointeur)// on passe l'adresse de y à la fonction 2 qui réclame un pointeur en paramètre
Fonction3(z); // la valeur de z passe à 2 (passage par référence)
```

Introduction à c++

- Les pointeurs : rappel
  - si p est un pointeur vers un entier i, *p désigne la valeur de i.

```
int main()
{
int i = 3;
int *p;
p = &i;
Cout <<"*p ="  << *p <<endl;
}
```

  - Sortie du programme : ***p = 3**

Introduction à c++

- Lien entre les pointeurs et les références
  - une variable et ses différentes références ont la **même adresse**, puisqu'elles permettent d'accéder à un même objet.
  - Utiliser une référence pour manipuler un objet revient donc exactement au même que de manipuler un pointeur constant contenant l'adresse de cet objet.

    référence = pointeur constant contenant l'adresse de l'objet

  - Les références permettent simplement d'obtenir le même résultat que les pointeurs, mais avec une plus grande facilité d'écriture.

Introduction à c++

- Lien entre les pointeurs et les références

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

- Nous constatons que la référence ri peut être identifiée avec l'expression *pi, qui représente bel et bien la variable i.
- La référence ri encapsule la manipulation de l'adresse de la variable i et s'utilise comme l'expression *pi.
- La différence se trouve ici dans le fait que les références doivent être initialisées d'une part, et que l'on n'a pas à effectuer le déréférencement d'autre part.

Introduction à c++

- Lien entre les pointeurs et les références

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

cout << i << endl; cout << &i << endl; cout << ri << endl; cout << &ri << endl; cout << *pi << endl; cout << pi << endl; cout << &pi << endl;	1 0x28ff24 1 0x28ff24 1 0x28ff24 0x28ff20

Introduction à c++

- Les espaces de nommage
  - En C, pour éviter le conflit de noms, on était obligé de modifier les noms de chaque structure.
  - C++ propose un moyen simple de résoudre ce problème : les espaces de nommage. Ils permettent de définir une unité cohérente dans laquelle on regroupe les déclarations des différents objets (types, constantes, variables, fonctions). Les identificateurs présents dans l'espace de nommage possèdent alors une portée qui leur est spécifique.
  - On définit un espace de nommage par le mot-clé namespace

Introduction à c++

- Les espaces de nommage

```
namespace A
{
    typedef unsigned int B;
    ....
}
....
A::B i; // une variable de type B
```

```
#include <iostream>
int main()
{
std::cout << "Hello world !" << std::endl;
return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello world !" << endl;
return 0;
}
```

Introduction à c++

- L'allocation dynamique
  - En C, l'allocation/désallocation dynamique étaient gérées par les routines malloc/free (calloc, realloc, etc...).
  - Le mécanisme d'allocation dynamique a été complètement repensé dans le C++ de manière à apporter davantage de robustesse. Il repose sur deux nouveaux opérateurs :
    - **new** pour l'allocation : **new** type;
      - int *ipt=new int;
    - **delete** pour la désallocation : delete ipt;
- Pour les tableaux, elle est légèrement différente
  - allocation : int * tab=new int[10];
  - désallocation : delete[] tab;

Introduction à c++

- Chaine de caractères : la bibliothèque string.h
  - Pour l'utiliser : # include <string>
  - string t : définit t comme une variable…
  - string t = "bonjour" ;
  - t.size() représente la longueur de t
  - t[i] est le (i+1)-ème caractère de s (i = 0,1,. . . t.size()-1)
  - S, t deux chaînes ➔ s+t est une nouvelle chaîne correspondant à la concaténation de s et t.

```
#include  <iostream>
#include <string>
using namespace std;
int main () {
string s1, s2, s3;
cout << "Tapez une chaine : ";          cin >> s1;
cout << "Tapez une chaine : « ;         cin >> s2;
s3 = s1 + s2;
cout << "Voici la concatenation des 2 chaines :" << endl;
cout << s3 << endl;
return 0; }
```

Introduction à c++

- Tableaux dynamiques : la bibliothèque vector.h
  - Pour l'utiliser : # include <vector>
  - Structure générale : vector< type > Nom ;
  - vector< type > Nom1 = Nom2 ;
    - Les valeurs de Nom2 sont alors recopiées dans Nom1.
  - Nom.size() correspond à la taille de Nom.

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

int main() {
 // Déclaration d'un vecteur d'entiers de taille non connue
vector  <int> my_vector1;
// Ajout de trois elements
my_vector1.push_back(4);
my_vector1.push_back(2);
my_vector1.push_back(5);
// La méthode size précise le nombre d'entrée courante
for (size_t i = 0; i < my_vector1.size(); ++i)
      cout << i << " " << my_vector1[i] << endl;
Return 0;
}
```