INFO-F105 – Langages de programmation 1
De Python à C++

1 Typage

L'une des principales différences de C++ par rapport à Python est son typage statique. Cela signifie que les variables ont un type défini à la compilation.

Concrètement, le système de typage de C++ a plusieurs implications :

- Il faut définir le type des variables dans le code, y compris les paramètres de fonctions.
- Il faut définir le type retour des fonctions.
- Les variables ne peuvent pas changer de type.

1.1 Booléens

```python
# Python
var = True
```

```cpp
// C++
   bool var = true;
// ----
// type
```

1.2 Nombres entiers

Les types entiers traditionnellement utilisés en C++ sont repris ci-dessous. Leur taille en bits est donnée à titre indicatif et pourrait être différente en fonction de l'architecture.

Signé	Non signé	#bits
`char`	`unsigned char`	8
`short`	`unsigned short`	16
`int`	`unsigned int`	32
`long`	`unsigned long`	64

```cpp
unsigned int n = 12345;
```

La taille réelle d'un type peut être connue grâce à l'opérateur `sizeof`.

```cpp
std::cout << sizeof(int) << std::endl;
```

Représentation binaire

Les entiers sont encodés de la même façon qu'au cours de fonctionnement des ordinateurs.

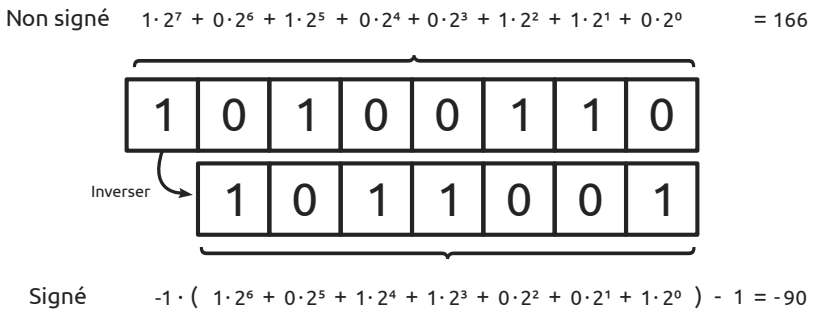

Figure 1: Encodage d'un entier sur 8 bits

Leur taille étant limitée, il n'est possible de représenter qu'un nombre limité d'entiers.

- `unsigned char` $\in [0, 2^8 - 1] = [0, 255] = [00000000_2, 11111111_2]$
- `char` $\in [-2^{8-1}, 2^{8-1} - 1] = [-128, 127] = [10000000_2, 011111111_2]$

Une fois le maximum atteint, on retourne au minimum puisque tout se fait modulo 2^n. En d'autres termes, toute la partie qui dépasse la taille du type est simplement ignorée.

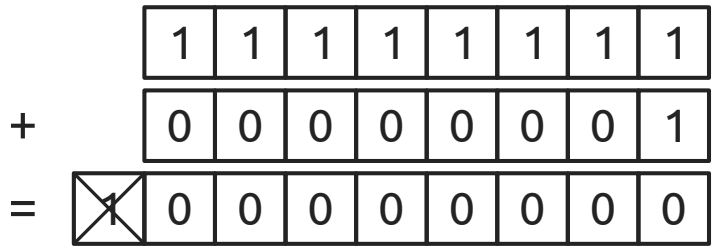

Figure 2: Addition modulo 2^8 de `unsigned char`

Caractères

Une particularité des entiers en C++ est qu'ils permettent de représenter des *caractères* en les entourant par des guillemets simples. Les 256 premiers caractères se codent sur 8 bits et forment l'alphabet ASCII[1].

```cpp
unsigned char c = 'A';
```

Les fonctions `ord()` et `chr()` en Python permettent de convertir un caractère en son encodage numérique et inversément.

```python
bin(ord('A'))   # '0b1000001'
chr(0b1000001)  # 'A'
```

Il est donc possible de faire de l'arithmétique sur le code binaire de ces caractères.

```cpp
unsigned char c = 'A' + 1;  // 'B'
```

1.3 Nombres flottants

Il existe trois types de nombres flottants en C++, dont la taille dépend de l'architecture.

Type	#bits
`float`	32
`double`	64
`long double`	128

```
double n = 3.14;
```

Représentation binaire

Les flottants sont *a priori* encodés selon la norme IEEE-754 (cf. INFO-F102).

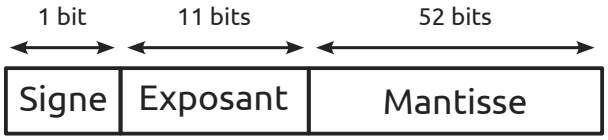

Figure 3: Encodage d'un flottant en double précision (64 bits)

Leur taille étant limitée, il n'est possible d'en représenter qu'un nombre limité.

2 Fonction `main`

Un programme C++ commence toujours par une fonction `main` qui contient le code à exécuter lors du lancement du programme. La valeur retour de cette fonction indique si le programme s'est exécuté correctement (par défaut 0 = succès).

```
int main() {
  // ...
  return 0;
}
```

3 Variables

La "vie" d'une variable, une fonction ou une classe est constituée de différente étapes :

- Déclaration : le fait de dire qu'une variable, fonction ou classe existe.
- Définition : le fait de décrire l'intérieur d'une fonction ou d'une classe.
- Assignation : le fait de donner une valeur à une variable.
- Instanciation : le fait de créer un objet à partir d'une classe ou structure.
- Destruction : le fait de supprimer une variable à la fin de sa portée (cf section 5.4).

Dans le cas d'une variable, on parle surtout de déclaration et d'assignation :

```
{
  int a;       // Déclaration
  int b = 5;   // Déclaration + assignation
  b = 2;       // Assignation
}              // Destruction
```

Dans le cas d'une fonction, structure ou classe, on parle de déclaration et de définition :

```
// Déclaration
int add(int, int);
```

```
// Définition
int add(int a, int b) {
  return a + b;
}
```

```
// Déclaration
struct Point;
```

```
// Définition
struct Point {
  int x;
  int y;
};
```

3.1 Constantes

Une *constante* est une sorte de variable dont la valeur ne peut pas changer.

```
const unsigned int MAX = 10;
```

Par convention, on nomme toujours les constantes globales en lettres capitales.

4 Opérations

La plupart des opérateurs sont les mêmes en C++ qu'en Python.

Arithmétique	*Bitwise*	Comparaison
+	&	==
-	|	!=
*	^	<
/	<<	>
%	>>	<=
		>=

```
5.0 / 2.0 == 2.5
5 / 2 == 2
0b11001011 << 2 == 0b00101100
```

Notons que ces opérateurs renvoient un résultat de même type que leurs opérandes, en tronquant les éventuelles parties qui dépassent.

En C++, la fonction `pow()` définie dans l'en-tête `<cmath>` remplace l'opérateur `**` de Python.

```
pow(2, 3) == 8
```

L'opérateur `~` (NOT) inverse les bits d'une valeur.

```
~0b11001011 == 0b00110100
```

Comme en Python, tous les opérateurs binaires (à 2 opérandes) ont une variante d'assignation.

```
a += b   // a = a + b
a *= b   // a = a * b
a |= b   // a = a | b
a <<= b  // a = a << b
```

Certains opérateurs unaires modifient également la variable.

Opération	Équivalent	Valeur retour
`i++`	`i = i + 1`	`i`
`i--`	`i = i - 1`	`i`
`++i`	`i = i + 1`	`i + 1`
`--i`	`i = i - 1`	`i - 1`

4.1 Priorité & associativité

En mathématiques, on parle de PEMDAS pour définir l'ordre des opérations :

1. Parenthèses
2. Exposants
3. Multiplications et divisions (dans le sens de la lecture)
4. Additions et soustractions (dans le sens de la lecture)

Ces règles s'appliquent aussi en C++ (et en Python), mais on y ajoute les autres opérateurs.

Dans le doute, utilisez des parenthèses pour donner la priorité à certaines opérations.

```
(a && b) || (c && d)
```

L'*associativité* s'applique lorsque vous utilisez plusieurs opérateurs de même priorité.

```
(a <= b <= c) == ((a <= b) <= c)
```

4.2 Expressions

Une *expression* est toute chose qui a une valeur. Une expression peut se décomposer en sous expressions comme dans la figure 4. Cela peut paraitre évident, mais il est important de garder ça en tête lorsqu'on écrit du code.

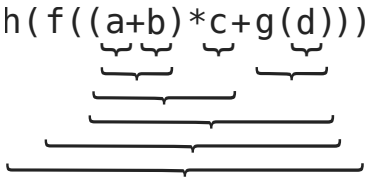

Figure 4: Décomposition d'une expression

5 Blocs

En Python, les blocs de code sont définis par l'indentation. Ce n'est pas le cas de C++ qui utilise des accolades `{ }`. Il reste cependent essentiel de bien indenter son code pour assurer sa lisibilité.

5.1 Conditions

```python
# Python
if a <= b <= c:
  # ...
elif b <= a <= c:
  # ...
else:
  # ...
```

```cpp
// C++
if (a <= b && b <= c) {
  // ...
} else if (b <= a && a <= c) {
  // ...
} else {
  // ...
}
```

Une différence à noter est le changement d'écriture des conditions entre les codes Python et C++. C'est parce que `(a <= b)` se traduit en `0` ou `1` à cause de l'associativité et de la conversion implicite des booléens en entiers.

5.2 Boucles

```python
# Python
while condition:
  # ...


for i in range(10):
  # ...
```

```cpp
// C++
while (condition) {
  // ...
}

for (unsigned i = 0; i < 10; ++i) {
  // --------------  ------  ---
  // ^-> initialisation
  //                 ^-> condition
  //                        ^-> incrément
  // ...
}
```

5.3 Fonctions

```python
# Python
def add(a, b) {
  return a + b
}
```

```cpp
// C++
int add(int a, int b) {
  return a + b;
}
```

5.4 Scope

Une variable déclarée à l'intérieur d'un bloc est inaccessible depuis l'extérieur.

```cpp
{
  int a = 5;
  // a est accessible
}
// a n'est pas accessible
```

Shadowing

Une variable déclarée à l'intérieur d'un bloc masque celle du même nom déclarée à l'extérieur.

```cpp
int a = 2;
// a == 2
{
  // a == 2
  int a = 5;
  // a == 5
}
// a == 2
```

6 Input / Output

Le code ci-dessous demande à l'utilisateur d'entrer un texte et l'affiche ensuite.

```python
# Python
text = input()
print(text)
```

```cpp
// C++
#include <iostream>

int main() {
  std::string text;
  std::cin >> text;
  std::cout << text << std::endl;
  return 0;
}
```

La directive `#include` dans le code C++ ci-dessus a le même rôle que `import` en Python. Il est nécessaire d'inclure l'en-tête `<iostream>` pour utiliser `std::cin` et `std::cout`.

Cette syntaxe peut sembler lourde, mais chaque élément a son importance :

- `std::` est un préfixe indiquant le *namespace* dans lequel est défini une variable/fonction.
- `string` est le type de la variable `text`, que nous reverrons dans une prochaine session.
- `cin` et `cout` sont des *objets* qui permettent de manipuler `stdin` [1] et `stdout` [2].
- `endl` vaut `\n` et est nécessaire car `cout` affiche le texte ligne par ligne [3].

References

[1] "Ascii(7) - linux man page." Available: https://linux.die.net/man/7/ascii

Exemples

Code 1 Affiche le n-ième nombre de Fibonacci

```cpp
#include <iostream>

const unsigned N = 8;

unsigned fibonacci(unsigned n) {
  if (n <= 1) {
    return n;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}

int main() {
  std::cout << fibonacci(N) << std::endl;
  return 0;
}
```

[1] L'entrée standard (ce qui est entré dans la console)
[2] La sortie standard (ce qui s'affiche dans la console)
[3] On peut aussi utiliser `std::flush`, qui n'insère pas de saut de ligne

Code 2 Affiche 10 nombres premiers générés aléatoirement

```cpp
#include <iostream>
#include <random>

const unsigned N = 10;
const unsigned MIN = 0;
const unsigned MAX = 10000;

bool is_prime(unsigned n) {
  for (unsigned i = 2; i < n; ++i) {
    if (n % i == 0) {
      return false;
    }
  }
  return true;
}

int main() {
  unsigned n = 0;

  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<unsigned> distr(MIN, MAX);

  while (n != N) {
    unsigned random_integer = distr(gen);
    if (is_prime(random_integer)) {
      std::cout << random_integer << ' ';
      ++n;
    }
  }
  std::cout << std::endl;

  return 0;
}
```