Chapitre 3

Entrée/Sorties en C++

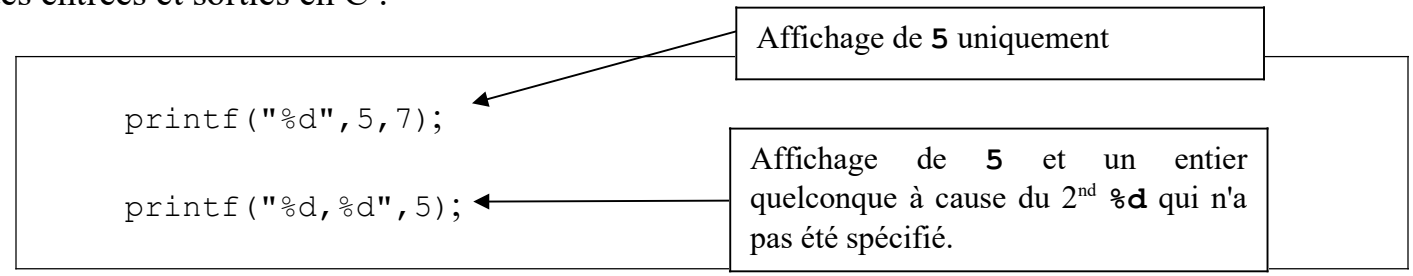
- Les entrées et sorties sont gérées dans C++ à travers des objets particuliers appelés **streams** ou bien **flots (ou flux)**.

- Pour utiliser ces objets, il faut inclure :

```
#include <iostream>
```

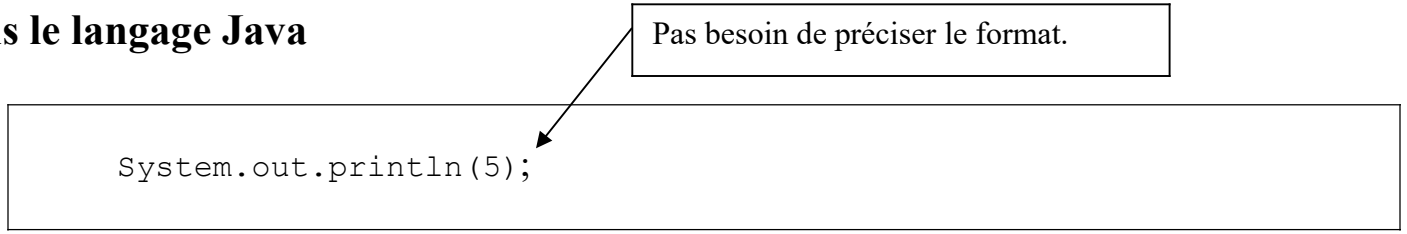
1. E/S dans le langage C

Un rappel des entrées et sorties en C :



« `printf` » est peu sécuritaire, car le programmeur doit spécifier le type.

2. E/S dans le langage Java



- C'est une bonne chose de ne pas être obligé de préciser le format de sortie.
- Java se charge d'afficher dans le format qui conviendrait le mieux.
- Un inconvénient à cette approche! Et si l'on voulait avoir une sortie formatée?
- C'est possible en Java mais assez complexe à faire.
- Dans la version 5 de Java, l'utilisation du formatage a été grandement facilitée.
- En effet, le formatage à la manière C a été introduit dans la version 5 de java.

3. E/S dans le langage C++

- <iostream> offre une interface orientée objet plus sécuritaire.
- Deux opérateurs sont surchargés de manière appropriée pour les flots :
    - l'opérateur d'insertion << (écriture)
    - l'opérateur d'extraction >> (lecture)
- On dit que ces deux opérateurs sont « surchargés » car ils font aussi autre chose.
- En réalité, ces opérateurs sont utilisés dans des opérations de manipulation de bits.
- Dans les opérations « 1 + 1 », « 1.5 + 2.5 », l'opérateur addition (« + ») fait l'addition de deux entiers et il est surchargé aussi pour faire l'addition de deux réels.
- Les flots prédéfinis sont :
    - `cout` associé à la sortie standard (équivalent à `stdout` dans le langage C),
    - `cerr` associé à la sortie erreur standard (équivalent à `stderr` dans le langage C),
    - `cin` associé à l'entrée standard (équivalent à `stdin` dans le langage C),

- Le bon format (bonne opération) est choisi en fonction du type des arguments,

- Les types écrits ou lus sont:

```
char short int long float double char*
```

```
#include <iostream>

int main() {
     int i;
     std::cout << "Entrer un entier: " << std::endl;
     std::cin >> i;
     std::cout << "Le carre de " << i \
               << " est: " << (i*i) << std::endl;
     return 0;
}
```

endl équivalent de \n en C.

\ pour signifier que l'instruction suit à la ligne suivante

Exemple.cpp

Si « `Exemple` » est le nom du programme exécutable de « `Exemple.cpp` », vous obtenez comme résultat :

```
prompt>Exemple
Entrer un entier:
20 (retour chariot obligatoire)
Le carre de 20 est: 400
prompt>
```

- On appelle « :: » **opérateur de résolution de portée**. Si l'on veut éviter son utilisation :

```
#include <iostream>

using namespace std;

int main() {
     int n;
     double x;

     // on peut écrire aussi le \n à l'intérieur de la chaîne à afficher
     cout << "Entrer la valeur de n, puis celle de x:\n";
     cin >> n >> x;

     // on incrémente n de 3 unités.
     n+=3;

     // à cause des règles de préséance des opérateurs, on pouvait s'abstenir d'écrire
     // (x*x) entre parenthèses, x*x suffirait.
     cout << n << (x*x) << endl;
     return 0;
}
```

On précise l'espace de nom d'où seront prises les différentes fonctions d'entrées et sorties

Exemple.cpp

Nous obtenons le résultat suivant :

```
prompt>Exemple
Entrer la valeur de n, puis celle de x:
10(espace obligatoire)12(retour chariot obligatoire)
13144
prompt>
```

- Il n'y a pas eu de séparation entre `13` et `144` car l'espace n'a pas été demandé. Pour l'avoir, il fallait écrire :

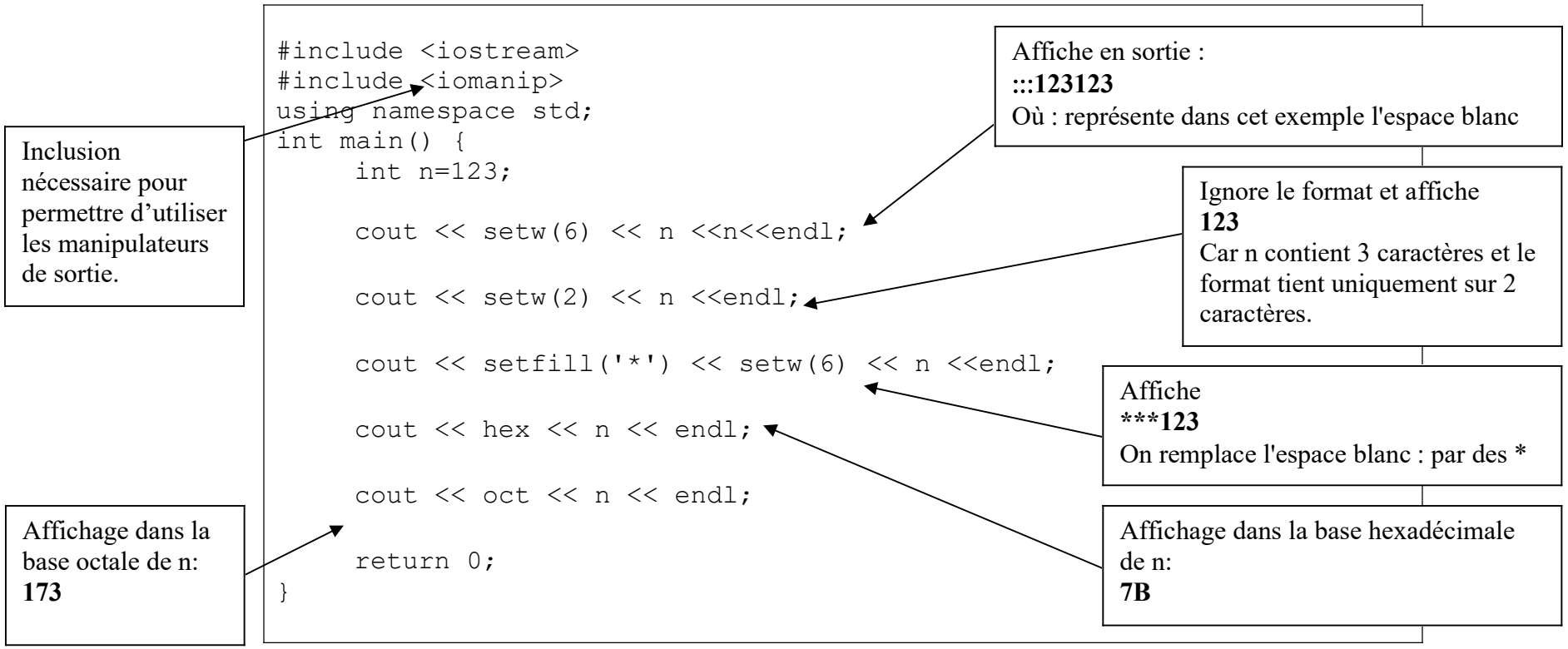
```
cout << n << ' ' << (x*x) << endl;
```

- On pouvait utiliser aussi « " » à la place de « ' ».

```
cout << n << " " << (x*x) << endl;
```

4. Mise en forme de la sortie

- Pour mettre en forme les sorties, on utilise des manipulateurs de sortie.
- Un manipulateur de sortie ne produit pas de sortie mais spécifie un paramètre de mise en page.
- Ce paramètre remplace le paramètre par défaut.



La même chose en utilisant « std::format ». Cette fonctionnalité a été introduite à partir de la norme « C++20 ».

```
#include <iostream>
#include <format> // C++20

int main() {
     int n = 123;

     // {:6} remplace setw(6).
     std::cout << std::format("{:6}{}\n", n, n);

     // {:2} remplace setw(2)
     std::cout << std::format("{:2}\n", n);

     // {:*>6} remplace setfill('*') et setw(6).
     // '>' spécifie alignement à droite (par défaut pour les nombres)
     std::cout << std::format("{:*>6}\n", n);

     // {:x} pour hex, {:o} pour octal
     std::cout << std::format("{:x}\n", n);
     std::cout << std::format("{:o}\n", n);

     return 0;
}
```

Le résultat obtenu par « std::format » peut-être préservé aussi en mémoire dans une variable du type chaîne de caractères.

La même chose en utilisant « std::print » et « std::println ». Sauf qu'ici, elles se contentent d'envoyer le résultat en sortie. Ces fonctionnalités ont été introduites à partir de la norme « C++23 ».

```
#include <iostream>
#include <print> // C++23

int main() {
     int n = 123;

     // println ajoute automatiquement une nouvelle ligne (\n)
     std::println("{:6}{}", n, n);
     std::println("{:2}", n);

     // même syntaxe que C++20
     std::println("{:*>6}", n);

     // Pour hex/octal, on peut utiliser '#' pour inclure le préfixe
     // (0x or 0)
     std::println("{:x}", n);  // affiche : 7b
     std::println("{:#x}", n); // affiche : 0x7b
     std::println("{:o}", n);  // affiche : 173

     return 0;
}
```

Attention sous Windows uniquement, cette fonctionnalité est toujours au niveau expérimental avec gcc. Il faut ajouter cette librairie au moment de faire l'édition de liens: « -lstdc++exp ».