Exemplier – Bases de la programmation en C++

Éditeur

Comment résoudre un exercice avec notre plugin VSCode :

- Télécharge la description et les exemples depuis le grader.
- Installe VSCode et notre plugin SOICode.
- "New Empty SOI Task" (dans le menu latéral ou CTRL+MAJ+P).
- Programme ta solution.
- Ajoute des données de test : Clic droit→"IORun Add IO".
- Exécute ton programme en entrant CTRL+Enter.

Modèle basique

```cpp
#include <iostream>  // possible de remplacer toutes
#include <algorithm> // les inclusions par une seule:
#include <vector>    // #include <bits/stdc++.h>
#include <numeric>   // (quand pas sur mac)
#include <tuple>
#include <utility>
// utiliser la bibliothèque standard sans "std::"
using namespace std;

#define int int64_t // utiliser des entiers 64-bits

// fonction principale, toujours exécutée en premier
// mets ton code à l'intérieur
signed main() {
        // optimisation de l'entrée/sortie
        ios::sync_with_stdio(false);
        cin.tie(0);
        // insère ton code ici...
}
```

Instructions

Les instructions finissent toujours par un point-virgule et sont exécutées de haut en bas. Voici quelques exemples d'instructions utiles :

- Déclarer la variable x du type 't' : `t x;`
- Tu peux déclarer plusieurs variables en même temps et leur assigner directement une valeur : `int a=10, b;`
- Assigner une valeur à une variable déjà déclarée : `a = 15;`
- Tu ne peux pas utiliser le même nom pour deux variables différentes (plus à ce sujet dans un cours ultérieur).

Quelques types

- **`int`** : entiers signés (peuvent être négatifs). Utilise `#define int int64_t` pour utiliser des entiers 64-bits ou bien utilise le type **`long long`**.
- **`unsigned long long`** : Entiers positifs 64-bits (pour quand **`int64_t`** ou **`long long`** ne suffisent pas).
- **`double`** : Nombres à virgule.
- **`bool`** : Valeurs booléennes de vérité : `true` (vrai) ou `false` (faux).
- **`char`** : Caractères. Utilise des guillemets simples, par exemple **`char`** `c = 'a';`
- `string` : Chaînes de caractères. Utilise des guillemets doubles, par exemple `string s = "Hello, world";`

Entrée et sortie

Récupère l'entrée grâce à `cin` :

```cpp
int a, b;
cin >> a >> b;
```

Utilise `cout` pour imprimer dans la sortie. Imprimer `'\n'` effectue un retour à la ligne.

```cpp
int a = 1, b = 2;
cout << "a+b: " << a+b << '\n'; // Imprime "a+b: 3".
```

Opérations sur les nombres entiers

Opérations standards :

- `a+b` : addition.
- `a-b` : soustraction.
- `a*b` : multiplication.
- `a/b` : division entière (p. ex. `5/3=1`).
- `a%b` : reste de la division entière de a par b (p. ex. `5%3=2`). Aussi appelé "modulo".

Fais attention à l'ordre des opérations. Les multiplications, divisions et modulos sont calculés avant les additions et soustractions.
Les opérations de même priorité sont exécutées de gauche à droite.
Utilise des parenthèses pour forcer un autre ordre d'opérations : `(a+b)*c-d/(e%(f*a))`.

Commentaires

Rends ton code plus lisible (aussi quand tu es le seul à le lire) en insérant des commentaires commençant par "//" et ignorés par le compilateur :

```cpp
cout << (h2-h1)/(x2-x1); // imprime la pente
```

Expressions conditionnelles

Tu peux utiliser des expressions conditionnelles pour contrôler ce que ton programme fait selon la vérité d'une certaine expression.
Pour faire cela, utilise la structure **`if`**.
Crée un bloc de code entre accolades `{ ... }` pour déterminer quelles instructions sont exécutées quand la condition est remplie.

```cpp
if(condition) {
        // exécuté si la condition est remplie
        ...
}
```

Utilise **`else`** pour spécifier que faire quand la condition n'est pas remplie :

```cpp
if(condition) {
        // exécuté si la condition est remplie
        ...
} else {
        // exécuté si la condition n'est pas remplie
        ...
}
```

Les conditions devraient être des expressions renvoyant une valeur booléenne (`true` ou `false`). Voici quelques expressions utiles que tu peux utiliser pour tester des conditions sur des nombres a et b :

- `a==b` est `true` si a et b sont égaux et `false` sinon.
- `a!=b` est `true` si a et b ne sont pas égaux et `false` sinon.
- `a<b` est `true` si a est plus petit que b et `false` sinon.
- `a>b` est `true` si a est plus grand que b et `false` sinon.
- `a<=b` est `true` si a est plus petit ou égal à b et `false` sinon.
- `a>=b` est `true` si a est plus grand ou égal à b et `false` sinon.

Tu peux combiner des expressions booléennes en utilisant des opérateurs logiques :

- `!a` est `true` si a est `false` et `false` sinon.
- `a&&b` est `true` si à la fois a et b sont `true` et `false` sinon.
- `a||b` est `true` si a ou b est `true` et est `false` si les deux sont `false`.

Ordre des opérations : `!` > opérateurs numériques > `==`, `!=` > `||` > `&&`.
Tu peux imbriquer des conditionnels :

```cpp
if(x==2) {
        ...
        if(!a-3==5) {
                ...
        }
        ...
}
```

Aller plus loin

Pour apprendre comment faire plus de choses en C++, suis les prochaines leçons attentivement, pose-nous des questions et n'hésite jamais à consulter des sites de référence comme https ://fr.cppreference.com/w.