Les nombres pseudo-aléatoires

http://en.cppreference.com/w/cpp/numeric/random

Un nombre aléatoire est un nombre dont la valeur est choisit au hasard. Ils sont utilisé dans de nombreux contextes, par exemple dans un jeu vidéo (pour avoir un comportement non prédictible), en cryptographie (pour encoder des informations), dans les tests automatisés du code (pour vérifier le comportement d'un programme avec des valeurs indéterminées). Vous verrez dans différents projets de ce cours des cas d'utilisation des nombres aléatoires.

Qualité des nombres aléatoires

Tirer des nombres aléatoires peut sembler être une tâche facile. On peut par exemple prendre un dé de jeux (non pipé) à six faces et le lancer plusieurs fois pour obtenir une série de nombres aléatoires compris entre un et six. On peut également lancer une pièce de monnaie pour obtenir des nombres aléatoires valant zéro (Pile) ou un (Face).

La difficulté vient du fait qu'il est facile d'obtenir des suites de nombres dont on n'arrive pas à prédire la prochaine valeur, mais qui présente quand même des propriétés mathématiques prédictibles. Si on additionne par exemple la valeur de deux dés, on aura un nombre aléatoire compris entre 2 et 12. Mais la probabilité d'obtenir un 12 est bien inférieure (1 chance sur 36) à la probabilité d'obtenir un 7 (12 chances sur 36).

Quand vous jouez à un jeu de société, ce n'est pas très important que les nombres obtenus ne soient pas parfaitement aléatoire (et c'est de toute façon intégré comme composante du jeu). Mais lorsque les nombres aléatoires permettent de crypter des données importantes ou sécuriser une numéro de carte bancaire, la moindre propriété mathématique prédictible sur une série de nombres aléatoires peut permettre de

contourner ces sécurités.

Il ne donc pas simplement que les nombres semblent aléatoires. Il faut le prouver mathématiquement. (Et bien sûr, cela sort complètement du domaine de ce livre, ces sont des travaux de recherches très pointus).

Pour en revenir à la programmation, on peut se poser une question : comment générer des nombres parfaitement aléatoires sur un ordinateur ? Ce n'est pas si simple, puisqu'un ordinateur est prévu pour être parfaitement prédictible (la même suite d'instructions avec les mêmes données produira le même résultat).

Heureusement, il existe plusieurs approches possibles.

Nombres aléatoires en C++

La bibliothèque standard fournit plusieurs fonctionnalités pour générer des nombres aléatoires, en utilisant différents algorithmes. L'ensemble de ces fonctionnalités sont accessibles en utilisant le fichier d'en-tête `random` (“aléatoire” en anglais).

```
#include <random>
```

La documentation est accessible en ligne sur cppreference.comm: Pseudo-random number generation, n'hésitez pas à vous familiariser un peu avec elle avant de continuer ce chapitre.

Les fonctionnalités de génération de nombres aléatoires se décomposent en trois catégories :

- le générateur de nombres aléatoires non déterministe ;
- les générateurs de nombres aléatoires déterministes (nombres pseudo-aléatoires) ;
- les générateurs de distributions statistiques.

Si vous travaillez sur du code qui a besoin d'un générateur le plus aléatoire possible, il faudra vérifier le fonctionnement du générateur ou utiliser une bibliothèque dédiée à la génération de nombres aléatoires

pour la cryptographique (ces générateurs sont plus lents que les générateurs utilisés dans la bibliothèque standard, ce qui explique pourquoi ils ne sont pas utilisés par défaut).

Générateurs non déterministes

L'utilisation d'un générateur aléatoire est relativement simple, même si cela utilise une syntaxe que vous n'avez pas encore vu. Il faut créer une variable utilisant le générateur comme type, puis appeler cette variable comme une fonction. On parle d'"objet appelable" (*callable* en anglais). En pratique, la syntaxe est la suivante :

```
CallableType myVariable{};              // créé une variable
de type CallableType
auto result = myVariable(arg1, arg2);   // appel cette
variable comme une fonction
```

Le type `CallableType` représente n'importe quel type qui permet de créer un objet appelable. Comme n'importe quelle fonction, un objet appelable peut prendre des arguments en entrée et retourner une valeur.

Le générateur non déterministe est la classe `std::random_device`. Cet objet sera appelé sans utiliser d'argument, et retourne un entier aléatoire à chaque fois qu'il est appelé.

main.cpp

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

int main() {
    std::random_device rd{};            // création du
générateur

    std::cout << rd() << std::endl;     // génération d'un
nombre aléatoire
    std::cout << rd() << std::endl;     // génération d'un
nombre aléatoire
    std::cout << rd() << std::endl;     // génération d'un
nombre aléatoire
```

```
}
```

Ce code permet de générer trois nombres aléatoires, qui seront différents à chaque fois que vous relancez le programme. Par exemple, cela peut donner la série suivante :

```
3555021123
4089866385
3885819577
```

En pratique, l'implémentation des générateurs n'est pas définie dans la norme C++, chaque implémentation de la bibliothèque standard peut avoir un comportement différent. Il n'est donc pas possible de prévoir le comportement réel de ce générateur.
Par exemple, sur Coliru, à chaque fois que le programme est relancé, il va générer la même suite de nombres aléatoires. Le même code sur Ideone va produire des nombres aléatoires différents à chaque fois que le programme est lancé.

Le problème de ce générateur est qu'il n'est pas possible de prouver qu'il a un comportement parfaitement aléatoire, qu'on ne peut prédire aucune caractéristique mathématique sur la série de nombres générés. On préférera donc généralement les générateurs de nombres aléatoires déterministes.

Générateurs déterministes

Un générateur déterministe produit des nombres aléatoires selon un algorithme complexe donné. Ce type de générateur prend en argument une graine (*seed* en anglais), qui définie la série de nombres aléatoires à générer. Utiliser deux fois la même graine produit la même série de nombres.

graine - c'est quoi ? - utiliser la graine pour générer plusieurs fois la même série - choisir une graine aléatoire avec générateur non déterministe

algos - cf la doc - 3 algos de base - 9 générateurs, avec algo de base et valeurs (ie 1 algo de base avec n'importe quelle valeur ne produira pas forcement un générateur parfaitement aléatoire)

sont-il sur ? - pourquoi ne pas redéfinir la graine - peut on retrouver la graine si on une série de nombres ? Probabilité très faible - Peut on retrouver la suite si on connait le début ? Encore moins probable - importance des “bonnes” pratiques, pour éviter que les nombres aléatoires deviennent prédictibles (exemple de enigma)

Astuce : pour les tests, afficher la graine aléatoire, pour pouvoir reproduire.

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

int main() {
    std::default_random_engine engin0(0);
    std::cout << engin0() << std::endl;
    std::cout << engin0() << std::endl;
    std::cout << engin0() << std::endl;
    std::cout << std::endl;

    std::default_random_engine engin1(1000);
    std::cout << engin1() << std::endl;
    std::cout << engin1() << std::endl;
    std::cout << engin1() << std::endl;
    std::cout << std::endl;

    std::default_random_engine engin2(0);
    std::cout << engin2() << std::endl;
    std::cout << engin2() << std::endl;
    std::cout << engin2() << std::endl;
    std::cout << std::endl;
}
```

générateurs de distributions statistiques

Générateurs précédents : chaque nombre à même chance d'etre générer.

Mais souvent besoin de nombres aléatoires qui suivent une loi particulière. Par exemple, somme de deux dés 6 non equiprobable, mais suis loi particulière. Générateur distributions permettent de générer ces lois

HS sujet, voir cours de stat. Quelques lois importantes : - uniforme entre min et max : équiprobable, dans un range. Version int et real - binomiale : pile ou face - loi normale

En pratique

Le plus souvent, loi uniforme, générateur par défaut, graine aléatoire.

Travaux pratiques

Donner une série de nombres aléatoires, produit avec un algo (donné ou non), une graine comprise entre 0 et 100. Essayer de retrouver la graine/algo

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++