Hasard en C

Adapté du site developpez.com.

Les nombres aléatoires (ou plutôt *pseudo-aléatoires* car le hasard n'existe pas en informatique) ne sont pas au programme. La bibliothèque `stdlib` fournie deux fonctions `rand` et `srand` dont nous contentons. La seconde sert à initialiser une *graine* (un nombre utilisé par la suite dans la production de nombres pseudo-aléatoires). Elle n'est appelée qu'une fois par programme.

La seconde fonction utilise la graine pour calculer des nombres. Les nombres produits vivent entre 0 et `RAND_MAX` dont la valeur est définie dans `sdtlib`. Sur ma machine, ce maximum vaut $2^{31} - 1$. Affichez le sur la vôtre.

Par défaut, la graîne est de 1 (c'est en particulier ce qu'il se passe si on n'appelle pas `srand` explicitement). Compiler et exécuter le code suivant plusieurs fois :

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int my_rand (void);

int main (void){
   int i;

   for (i = 0; i<10; i++)   {
     printf ("%d\n", my_rand());
   }
   return (EXIT_SUCCESS);
}

int my_rand (void){
   return (rand ());
}
```

On observe que la séquence des nombres est toujours identique, ce qui n'est pas bien vu pour des nombres aléatoires. Cela vient de la graine : elle vaut toujours 1.

Remarque. Lorsqu'on est en phase de débuggage d'un projet nécessitant du hasard, il est utile d'obtenir toujours les mêmes valeurs par `rand`. On laisse donc la graine à une valeur connue.

En phase d'exploitation, on choisit une graine moins prévisible comme expliqué ci-dessous.

Utilisons une autre valeur : le temps courant. On l'obtient en incluant l'en-tête `<time.h>` et en appelant `time(NULL)`. L'usage du pointeur `NULL` signifie simplement qu'on ne stocke pas la valeur produite à une adresse particulière : on se contente juste de la calculer.

Changeons le code de `my_rand` :

```c
int my_rand (void){
   static int first = 0;

   if (first == 0)   {
      srand (time (NULL));
      first = 1;
   }
   return (rand ());
}
```

La variable `first` est *statique* : elle a une adresse fixe qui ne dépend pas du nombre d'appel à `myrand`. Au premier ou au 100ème appel, l'adresse de cette variable est la même.

Au premier appel, `first` vaut 0. On rentre alors dans la branche positive du `if`. La graine est initialisée avec la valeur courante du temps (nombre de secondes depuis le 1er janvier 1970 à priori) et n'est plus jamais modifiée puisque `first` est différent de 0.

Tous les appels à `rand` utilisent alors cette graine initiale.

Compiler. Relancer plusieurs fois le programme. On constate que deux lancement du programme effectués dans la même seconde donnent le même résultat : c'est normal, le temps courant (donc la graine) est le même. Mais deux appels séparés de plus d'une seconde donnent deux résultats différents. Nous nous en contentons.

Le problème maintenant c'est que les nombres produits sont dans l'intervalle entre 0 et `RAND_MAX`. Pour un entier n, on peut ramener le nombre aléatoire produit dans $[\![0, n-1]\!]$ en prenant le modulo selon n : `nb_produit % n`.

Malheureusement, la distribution ainsi obtenue n'est plus uniforme. certains nombres ont plus de chance d'être produits que d'autres. Avec $n = 10$ et 25 pour `RAND_MAX`, on obtient :

valeur calculée par `rand`	valeur après modulo
$[0, 10[$	$[0, 10[$
$[10, 20[$	$[0, 10[$
$[20, 25[$	$[0, 5[$

Les nombres entre 0 et 5 ont plus de chance d'être obtenus que les autres.

On se contente némamoins de ce calcul modulo.

Voici donc la fonction `int my_rand(int n)` qui renvoie un nombre entre 0 et $n-1$:

```c
int my_rand (int n){
   static int first = 0;

   if (first == 0)   {
      srand (time (NULL));
      first = 1;
   }
   return (rand () % n);
}
```