Jeu du plus ou moins et génération de nombres aléatoires

laurent.jospin.59@free.fr, `http://jospin.lstl.fr`

Lycée Saint-Louis, Paris

"Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin", John von Neumann

1 Jeu du plus ou moins

Afin de travailler sur la clareté du code, nous décomposerons le programme en fonctions correspondant à des tâches simples indépendantes.

1. Commencer un nouveau programme en important les modules standards et en préparant la fonction principale.

2. Ecrire une fonction `demande_un_nombre()` qui ne prend rien en argument, affiche un message invitant à saisir une proposition, lit un nombre sur l'entrée standard puis renvoie la valeur de ce nombre.

3. Ecrire une fonction `compare(proposition, secret)` qui affiche un message en fonction de la valeur relative de la proposition par rapport au secret.

4. En déduire une fonction `partie(secret)` qui prend argument le nombre à deviner, qui définit et initialise une variable locale pour compter le nombre de propositions nécessaires, ainsi qu'une variable proposition qu'on initialisera à -1 et qui répète les deux fonctions précédentes tant que la proposition n'est pas égale au secret et incrémente le compteur) chaque itération. La fonction `partie` renverra le nombre de propositions effectuées par le joueur.

5. Dans la fonction `main`, faire appel à la fonction `partie` puis afficher un message indiquant à l'utilisateur le nombre de propositions dont il a eu besoin pour trouver le secret 42.

2 Génération de hasard

Un ordinateur est une machine déterministe donc incapable de produire du hasard, pourtant indispensable à de nombreuses applications informatiques, en particulier pour les méthodes de Monte-Carlo, pour la simulation de phénomènes aléatoires et pour le domaine de l'intelligence artificielle. L'objectif de cette activité est de montrer deux méthodes simples de génération de nombre pseudo-aléatoires.

La plupart des générateurs de nombres pseudo-aléatoires utilisent une valeur numérique initiale, appelée graine, pour initialiser l'algorithme puis une fois la graine choisie, la suite de nombres est entièrement déterminée : avec la même graine, on obtient la même suite de nombres et cette suite de nombres est périodique à partir d'un certain rang.

Pour obtenir un comportement différent à chaque appel du programme, une méthode consiste à initialiser la graine avec une valeur qui peut être considérée comme aléatoire. Ainsi la suite de nombres pseudo-aléatoires est différente à chaque appel du programme. Cette valeur peut être issue de la mesure physique d'un phénomène admis aléatoire ou simplement l'horloge interne de l'ordinateur.

2.1 Méthode de von Neumann

6. Déclarer une variable globale `graine` initialisée avec une valeur quelconque à 10 chiffres (sans redondance particulière de chiffres). On utilisera le type `uint64_t` défini dans `stdint.h` dont l'entier maximal est de l'ordre de 10^{18}.

Pour les entiers de taille précise, les codes de formatage dépendent de l'architecture de la machine (les entiers 64 bits sur certaines architectures pourraient utiliser le code `lld` ou le code `ld` sur d'autres) mais sont définis par des constantes dans `inttypes.h` et peuvent être ainsi introduits par juxtaposition des dites chaînes avec la chaîne formatée. On trouve notamment `PRId64` et `PRIu64` pour des entiers signés ou non signés sur 64 bits.

```
1 printf("Mon entier non signé 64 bits vaut : %" PRIu64, monNombre)}
```

7. Ecrire une fonction `aleatoire` sans argument, qui élève la graine au carré, détermine les dix chiffres centraux (plus précisément du 6ème au 15ème chiffre en comptant à partir des unités), met à jour la graine avec cette nouvelle valeur (donc produit des effets de bord) et renvoie également cette valeur considérée comme le nombre aléatoire généré.

Pour déterminer les chiffres centraux, on pensera à utiliser les opérateurs de division euclidienne avec une puissance de 10 bien choisie.

8. En déduire, en faisant appel à la fonction `aleatoire` précédente, une fonction `flottant_aleatoire` qui renvoie un flottant aléatoire de $[0, 1[$.

En déduire de même une foncton `entier_aleatoire_entre_bornes(a, b)` qui renvoie un nombre entier compris (au sens large) entre les deux bornes passées en argument. Pour cette deuxième fonction, on utilisera d'abord l'opérateur modulo pour se ramener dans l'intervalle d'entiers $[\![0; b - a]\!]$.

2.2 Générateur congruentiel linéaire

9. Ecrire une nouvelle fonction `aleatoire2` qui multiplie la graine par un entier a, ajoute c, puis réduit le tout modulo m pour obtenir le nombre aléatoire et la nouvelle graine. On prendra : $m = 2^{31}, a = 1103515245, c = 12345$.

Ajouter une fonction `flottant_aleatoire2` qui renvoie un nombre flottant aléatoire dans $[0, 1[$.

10. En utilisant le générateur congruentiel linéaire, terminer le jeu du plus ou moins en ajoutant un tirage au hasard de la valeur à trouver. On pourra lire le maximum des valeurs à trouver comme argument en ligne de commande.

11. Pour que le nombre à deviner ne soit pas toujours le même, il faut initialiser la graîne avec une valeur différente à chaque appel. Utiliser le résultat de la fonction `time`, définie dans `time.h`, et appelée avec l'argument `NULL` pour initialiser la graine.

3 Méthode de Monte-Carlo

12. Ecrire une fonction `estime_pi` qui prend un argument un nombre de répétitions qui renvoie une estimation de π par la méthode de Monte-Carlo : pour chaque répétition, on tire deux flottants aléatoires de $[0, 1[$ qu'on assimile à des coordonnées dans le carré $[0; 1[^2$ puis on regarde si le point correspondant est dans le disque unitaire. On renvoie la proportion de points dans le disque multipliée par quatre. Comparer les résultats obtenus avec la fonction `flottant_aleatoire` et avec la fonction `flottant_aleatoire2`.

13. Ecrire une fonction `estime_integrale_positive` qui prend en argument un intervalle tel que sur cet intervalle la fonction `f` supposée définie dans le contexte est positive, un majorant de cette fonction sur cet intervalle et un nombre de repétitions et renvoie une estimation par la méthode de Monte-Carlo de l'intégrale de la fonction sur l'intervalle considéré.

> *"If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is. "*, John von Neumann

FIGURE 1 – John von Neumann (1903-1957) est notamment l'un des pionniers de l'informatique – le modèle d'architecture des ordinateurs modernes est par exemple appelé architecture de von Neumann – un grand contributeur à la théorie des automates avec en particulier l'introduction des automates cellulaires et le père de la théorie des jeux. *Source photo : Wikipédia*