PROG
TD#2:

2020-W40

Exercice 1 : Questions courtes *(extraits DS'19 ; ET'19)*

Q.1 : Pour chaque déclaration suivante, quel est le type de la variable déclarée ? Dans chaque cas, que pouvez-vous dire sur sa valeur après déclaration ?

1. `int a = 2;`
2. `double a;`
3. `uint16_t a[32];`
4. `uint16_t *a;`

Q.2 : Pour chacune des suites d'expressions suivantes, décrivez le résultat attendu et si celui-ci mène à une erreur ou un comportement non souhaité.

1. `int t[256]; t[256] = 0;`
2. `int *t; t = 64;`
3. `int *t; *t = 64;`
4. `int t[10][10]; t[0] = 4;`
5. `int t[10000][10000]; t[0][0] = 4;`

Q.3 : Quel problème pose l'extrait de code suivant ? Comment pouvez-vous y remédier ?

```
for (uint64_t i = 17; i >= 0; i--)
{
        ...
}
```

Q.4 : Quel problème pose l'extrait de code suivant ? Comment pouvez-vous y remédier ?

```
if (((x % n) > 3) && (n != 0))
{
        ...
}
```

Q.5 : Quel problème pose l'extrait de code suivant ? Comment pouvez-vous y remédier ?

```
/* calcule le produit de a et b mod 2**32 - 5 */
/* précond: 0 <= a,b < 2**32 - 5 */
uint32_t mul325(uint32_t a, uint32_t b)
{
    uint32_t r = (a*b) % 4294967291U; // 2**32 - 5 = 4294967291
    return r;
}
```

Exercice 2 : Un générateur pseudo-aléatoire

On souhaite implémenter un générateur « pseudo-aléatoire » de nombres de 32 bits. On suggère pour commencer d'utiliser le processus suivant : 1) on initialise un état de 4 entiers $R_0[0], \ldots, R_0[3]$ représentant chacun un élément de $\mathbb{Z}/2^{32} - 5\mathbb{Z}$; 2) chaque fois qu' il est nécessaire de générer un nombre aléatoire, on retourne la valeur courante contenue dans $R_i[3]$, puis modifie R_i en R_{i+1} de la façon suivante :

$$\begin{aligned} R_{i+1}[3] &= R_i[2] \; - 1714784390 \cdot R_i[3] \\ R_{i+1}[2] &= R_i[1] \; - 925613062 \cdot R_i[3] \\ R_{i+1}[1] &= R_i[0] \; - 4214021598 \cdot R_i[3] \\ R_{i+1}[0] &= -3813655611 \cdot R_i[3] \end{aligned}$$

Q.1 :

1. On suppose que le processus décrit ci-dessus est inversible (c-à-d qu'il est possible de calculer R_0 à partir de R_i pour n'importe quel i). Montrez alors que la suite générée est périodique, quelque soit la valeur de l'état initial R_0. (Bonus : comment pourriez-vous prouver que ce processus est bien inversible ?)
2. Quelle condition doit remplir R_0 afin d'éviter que la période soit égale à 1 ?
3. À supposer que cette condition est remplie, quelle est la période maximale que peut possiblement décrire la suite ?

Q.2 : Implémentez le processus décrit ci-dessus en stockant l'état R dans une variable globale et grâce au couple de fonctions suivant :

— `uint32_t genrandu32(void);`
— `void initstate(uint32_t s[4]);`

Q.3 : Comment pourriez-vous faire pour qu'une initialisation par défaut soit effectuée dans le cas où l'utilisateur n'en ferait pas ?

Q.4 : Que pouvez-vous dire sur la distribution des nombres renvoyés par ce processus ? Vous semble-t il générer une suite pseudo-aléatoire de « bonne qualité » ?

Indice : Vous devriez pouvoir identifier deux faiblesses : l'une concerne la structure sur laquelle les éléments de R sont définis, la seconde sur le type même de processus utilisé.

Q.5 : Supposez que vous disposez d'un générateur `uint64_t magicr(void)` renvoyant un aléa uniforme dans $[\![0, 2^{64} - 1]\!]$. La distribution de `magicr() % 2` est elle uniforme dans $[\![0, 1]\!]$? Quid de `magicr() % 3` dans $[\![0, 2]\!]$?