Zufallszahlengeneratoren in F und C

F

Der F9x-Standard enthält neben anderen mathematischen Funktionen auch einen Zufallszahlengenerator, der gleichverteilte Pseudo-Zufallszahlen aus dem Intervall $[0, 1)$ liefert. Standardisiert (und portabel) ist allerdings nur der Aufruf; welcher Generator tatsächlich verwendet wird, hängt vom Compilerhersteller und von der Computerarchitektur ab.

Da bei einem Aufruf des Zufallszahlengenerators nicht nur ein Wert zurückgegeben, sondern auch der interne Status der Prozedur verändert werden muß (Nebeneffekt), ist der Generator nicht als Funktion, sondern als `subroutine` implementiert. Die Syntax lautet

```
call random_number(random)
```

Dabei ist *random* ein Skalar oder Array vom Typ `real`, der nach dem Aufruf die Zufallszahl bzw. einen ganzen Satz von Zufallszahlen enthält. Wiederholte Aufrufe liefern aufeinanderfolgende Zufallszahlen aus der Sequenz des Generators.

Der momentane Status des Generators ist in einem Vektor *seed* vom Typ `integer` gespeichert und kann mit

```
call random_seed(get=seed)
```

abgefragt und mit

```
call random_seed(put=seed)
```

gesetzt werden. Die (installationsabhängige) Länge *length* des Vektors *seed* kann man mit

```
call random_seed(size=length)
```

abfragen.

Die Möglichkeit, den Status des Generators auszulesen bzw. zu setzen, ist wichtig, wenn man eine lange, zusammenhängende Simulation in kürzere Teilabschnitte zerlegen will: Man liest am Ende jeder Teilrechnung den Status des Generators aus und initialisiert ihn damit am Beginn der nächsten. Will man (für Testzwecke) mehrere Male exakt dieselbe Folge von Zufallszahlen bekommen, so muß man zu Beginn jeweils den Generator auf denselben Ausgangszustand setzen.

In F von Imagine1 war ursprünglich der "Minimalgenerator"

$$i_{n+1} = 16807 \cdot i_n \mod 2^{31} - 1$$

implementiert, dessen Statusvektor *seed* aus einem einzigen Element, nämlich i_n, besteht. Derzeit (2008) baut F aber auf dem freien g95-Compiler auf und hat daher auch dessen Zufallszahlengenerator übernommen. Dieser ist zwar ein sogenannter Schieberegister-Generator, kann jedoch formal wieder als Rekursion

$$\mathbf{z}_{n+1} = \mathbf{A} \cdot \mathbf{z}_n$$

geschrieben werden, wobei $\mathbf{z}_n$ ein 96-stelliger Bitvektor ist, $\mathbf{A}$ eine binäre 96×96 Matrix und alle arithmetischen Operationen modulo 2 verstanden werden. Die höchsten Bits von $\mathbf{z}_n$ werden, nachdem sie zuvor nochmals mit einem Zufallsmuster des parallel laufenden Minimalgenerators "randomisiert" wurden, in eine `real`-Zahl der gewünschten Präzision im Einheitsintervall umgewandelt. Der Statusvektor für diesen Generator besteht aus vier Elementen.

C

In C gibt es eine ganze Reihe von Zufallszahlengeneratoren, die aber häufig nicht direkt Werte aus dem Intervall $[0, 1)$ liefern, sondern ganze Zahlen im Bereich $0, \ldots, m-1$. Ein halbwegs einfacher und ähnlich wie in F zu handhabender Generator ist `drand48`. Im Gegensatz zu F ist er als Funktion (vom Typ `double`) implementiert und kann daher entweder als Zuweisung

```
random=drand48();
```

oder direkt an Stelle eines `double`- oder `float`-Werts in numerischen Ausdrücken verwendet werden.

Der interne Status wird in einem 3-elementigen Vektor *seed* von vorzeichenlosen ganzen 16-Bit-Zahlen gespeichert und kann mit

```
unsigned short int dummy[3];
unsigned short int *seed;
...
seed=seed48(dummy);
```

ausgelesen und mit

```
unsigned short int seed[3];
...
seed48(seed);
```

gesetzt werden. Im ersten Fall ist *dummy* eine (eigentlich überflüssige, aber verlangte) Hilfsvariable vom selben Typ wie der Statusvektor *seed*, denn die Funktion `seed48` setzt sowohl den Statusvektor auf den neuen, als Argument übergebenen Wert und gibt gleichzeitig einen Pointer auf den alten Statusvektor zurück.

Der verwendete Generator ist

$$i_{n+1} = (25214903917 \cdot i_n + 11) \mod 2^{48}$$

wobei der "Funktionswert" $\xi_{n+1} = i_{n+1}/m$ allerdings nicht durch Division gebildet wird, sondern es werden die 48 Bits von i_{n+1} direkt als Mantisse der `double`-Zahl ξ_{n+1} verwendet. (Da in der üblichen IEEE-Darstellung 52 Bits für die Mantisse vorgesehen sind, werden die vier niedrigsten Bits von ξ_{n+1} nicht gesetzt.)