Programming I

Pseudo-Random Numbers

Dimitrios Michail

Dept. of Informatics and Telematics
Harokopio University of Athens

Pseudo-Random Numbers

A pseudo-random number generator is an algorithm which produces a sequence of numbers whose properties approximate the properties of sequences of random numbers.

The C language provides such a generator in its library.

`rand()` Function

- The function `rand()` is declared in the header file `stdlib.h` and provides us with the of generating pseudo-random numbers.
- The function returns an integer in the range $[0, RAND\_MAX]$ where `RAND_MAX` is a constant.
- Based on the spec of C the constant `RAND_MAX` must be at least 32767. Usually it has larger values. On the speaker's machine `RAND_MAX` has a value of 2147483647.

Fuction `rand()`

- If the returned numbers where truly random, each one of them would have a probability of appeareance equal to

$$\frac{1}{1 + \text{RAND\_MAX}}$$

- Since the generator is just an algorithm the probability is not the above.

- However, in order to tell the difference we need to use the generator too many times in a row.

Example

Function rand()

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

int main() {
    int i;
    for(i = 0; i < 10; i++) {
        printf("%d ", rand());
    }

    return 0;
}
```

The above program, when executed on the speaker's machine, prints:

```
1804289383 846930886 1681692777 1714636915
1957747793 424238335 719885386 1649760492
596516649 1189641421
```

Convert Range

Function `rand()`

In order to convert the range

- from 0 to `RAND_MAX`
- to 0 to N where $N < RAND\_MAX$

we use the modulo operator.

We divide the result of `rand()` with $N + 1$.

e.g.

```
int r = rand() % 100;
```

Producing Pseudo-Random Numbers from 0 to 99

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

int main() {
    int i;
    for(i = 0; i < 10; i++) {
        printf("%d ", rand()%100);
    }
    printf("\n");

    return 0;
}
```

The above program (again on the speaker's machine) prints:

```
83 86 77 15 93 35 86 92 49 21
```

Multiple Executions

Producing Pseudo-Random Numbers from 0 to 99

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

int main() {
    int i;
    for(i = 0; i < 10; i++) {
        printf("%d ", rand()%100);
    }
    printf("\n");

    return 0;
}
```

If we execute the above program twice we get:

```
83 86 77 15 93 35 86 92 49 21
83 86 77 15 93 35 86 92 49 21
```

How does the Generator Work?

- The generator starts from the previously generated number and performs certain transformations in order to produce the next number.
- The initial number from which the generator starts (but does not produce) is called the **seed** of the generator.
- The initial value of the seed is always 1, thus, we always get the same sequence of numbers.

Changing the Seed

Function `srand()`

Function `srand()` allows us to initialize the generator using a different seed (default is 1). It takes an `unsigned int` number as its single parameter.

After calling `srand()` the pseudo-random number generator begins from scratch and generates a new sequence of numbers.

Another Sequence

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

int main() {
    int i;

    srand(2);

    for(i = 0; i < 10; i++) {
        printf("%d ", rand()%100);
    }
    printf("\n");

    return 0;
}
```

When executed the above program prints:

```
90 19 88 75 61 98 64 77 45 27
```

Common Mistakes

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

int main() {
    int i;
    srand(2);
    for(i = 0; i < 10; i++)
        printf("%d ", rand()%100);
    printf("\n");

    srand(3); // No! Never initialize twice

    for(i = 0; i < 10; i++)
        printf("%d ", rand()%100);
    printf("\n");

    return 0;
}
```

Initializing the seed a second time breaks all the good properties of the generator.

Heads or Tails

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

#define HEADS 0
#define TAILS 1

int flip() {
    return rand()%2;
}

int main() {
    int i;
    for(i = 0; i < 20; i++) {
        printf("%c ", flip()==HEADS?'H':'T');
    }

    return 0;
}
```

In the speakers computer the above program prints:

H T H H H H T T H H T H T H H T T T T T

Seed and System Time

It is a common programming technique to initialize the pseudo-random number generator using the system time as a seed.

In order to read the time in C we can use the header file

```
#include <time.h>
```

which includes functions and data types in order to read system time.

Seed and System Time

It is a common programming technique to initialize the pseudo-random number generator using the system time as a seed.

In order to read the time in C we can use the header file

```
#include <time.h>
```

which includes functions and data types in order to read system time.

Make sure to initialize the random number generator **only once** in your programs, otherwise properties such as *period length* become broken.

Function `time()`

Function `time()` returns the time that has passed in seconds from a moment in time called the *Epoch* which is **00:00:00 UTC, January 1, 1970**.

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

int main() {
    printf("%ld", time(NULL));

    return 0;
}
```

The program above prints the time when the program was executed. When the speaker wrote the first version of theses slides the time was:

1259243300

Time and Seed

Heads or Tails

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

#define HEADS 0
#define TAILS 1

int flip() {
    return rand()%2;
}

int main() {
    int i;
    srand(time(NULL));
    for(i = 0; i < 20; i++) {
        printf("%c ", flip()==HEADS?'H':'T');
    }
    return 0;
}
```

It is now more difficult to get the same sequence time.

A Simple Generator

Linear Congruential Generator

In real-life it is highly unlikely to write a pseudo-random number generator.

Nevertheless, let us see a very simple method.

A Simple Generator

Linear Congruential Generator

```
#define a (1103515245)
#define c (12345)
#define m (1<<31)

static unsigned int seed = 1;

void srand(unsigned int s) {
    seed = s;
}

int rand() {
    seed = (a * seed + c) % m;
    return seed;
}
```

A Simple Generator

Linear Congruential Generator

The algorithm works fine only if a, c and m are chosen in a particular way!

The following wikipedia article `https://en.wikipedia.org/wiki/Linear_congruential_generator` discusses this generator technique in more detail.

True random number generator (TRNG)

Can we build a true random number generator? Yes, but we need to use hardware.

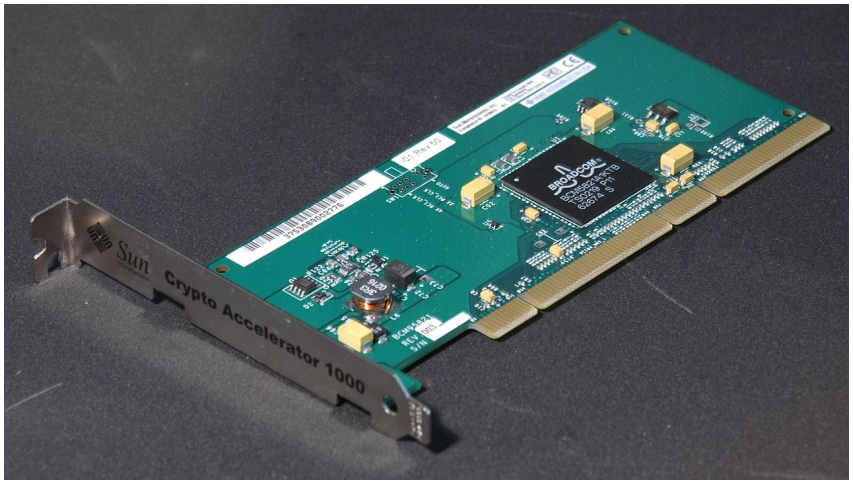

Image by Retro-Computing Society of Rhode Island

See `https://en.wikipedia.org/wiki/Hardware_random_number_generator`.

Material

- Chapter 5.9, C Programming, Deitel & Deitel, 3/e
- Donald Knuth, Art Of Computer Programming, Seminumerical Algorithms, Third Edition, Addison-Wesley, 1997.
- A fast pseudo-random number generator (Mersenne Twister). `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`
- `http://www.gnu.org/s/gsl/manual/html_node/Random-Number-Generation.html`