

Title:	Really Random random number programs
Author:	Ian Johnson (August 2002)
Prerequisites:	A Linux based computer
Module:	UFS001C1
Awards:	B.Sc. CRTS / CSE / C&T
References:	Royce, T "C Programming", Macmillan, 1996 Kernighan, B & Ritchie, D, "The C Programming Language", Prentice-Hall 2 nd Ed. 1988

Introduction

This worksheet is intended to get you thinking about how to write programs. Learning the syntax of a programming language is the easy part! Designing well crafted solutions to problems is what programming is really about. Unfortunately, this cannot really be taught, you learn through experience, practice and exposure to other peoples code.

These tests become more difficult to program as you progress. The intention here is for you to practice your programming.

We are going to be using the `stdlib rand()` function and writing programs to assess its quality. The quality of random numbers has always been a source of concern. Nowadays, we generally use pseudo-random computer generated numbers and these are vital to fields such as neural networks, evolutionary computing & encryption. These numbers are not truly random, but rather a set of these numbers should display the property of randomness. The tests you are going to implement were suggested by Kendall & Babbington-Smith over a series of papers published in 1938/39. These tests obviously predate computer generated random numbers, but are ways of assessing the randomness of a set of numbers. Generally speaking, the `rand()` function is poor, and better (e.g. `random()`) functions are used. For more information see the manual page for `random()`.

The `stdlib rand()` function is generally regarded as of poor quality, however that is really irrelevant to this discussion. To obtain a description of the function you can type: `man -s3 rand`, (`man s3c rand` would be needed on Solaris). Section 3 of the manual contains details of the standard C library functions. You can lookup definitions of all functions using the manual.

The manual page is copied overleaf, (generated using `man2html`, my comments in the funny font).

RAND

The Name of the Page

NAME

rand, srand - random number generator. *The functions this page refers to.*

SYNOPSIS

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

The required C header file

```
int rand(void);
```

Function prototype(s) giving the return value type and argument (if any) type.

```
void srand(unsigned int seed);
```

DESCRIPTION

What the function does and how to use it

The rand() function returns a pseudo-random integer between 0 and RAND_MAX. The srand() function sets its argument as the seed for a new sequence of pseudo-random integers to be returned by rand(). These sequences are repeatable by calling srand() with the same seed value.

If no seed value is provided, the rand() function is automatically seeded with a value of 1.

RETURN VALUE

Explanation of return values

The rand() function returns a value between 0 and RAND_MAX. The srand() returns no value.

NOTES

Additional Info

The versions of rand() and srand() in the Linux C Library use the same random number generator as random() and srandom(), so the lower-order bits should be as random as the higher-order bits. However, on older rand() implementations, the lower-order bits are much less random than the higher-order bits.

So, on linux rand & random are the same

In Numerical Recipes in C: The Art of Scientific Computing (William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling; New York: Cambridge University Press, 1992 (2nd ed., p. 277)), the following comments are made:

"If you want to generate a random integer between 1 and 10, you should always do it by using high-order bits, as in

```
j=1+(int) (10.0*rand()/(RAND_MAX+1.0));
```

and never by anything resembling

```
j=1+(rand() % 10);
```

(which uses lower-order bits)."

A reference (link to online version from my web page if you're interested)

Random-number generation is a complex topic. The Numerical Recipes in C book (see reference above) provides an excellent discussion of practical random-number generation issues in Chapter 7 (Random Numbers).

For a more theoretical discussion which also covers many practical issues in depth, please see Chapter 3 (Random Numbers) in Donald E. Knuth's The Art of Computer Programming, volume 2 (Seminumerical Algorithms), 2nd ed.; Reading, Massachusetts: Addison-Wesley Publishing Company, 1981.

More references!

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

Standards this conforms to

SEE ALSO

random(3), srandom(3), initstate(3), setstate(3) *Related Pages*

We want to generate random numbers in the range 0-9 inclusive, however the random number generator generates numbers over the range 0 to RAND_MAX. RAND_MAX will be defined in stdlib.h. You should not replace it with a constant since its value may vary with implementation.

We need to rescale this value. The manual page has warned us of the correct way to do this. If we divide by RAND_MAX and multiple by 10, then we would generate values 0-10. In order to ensure that we generate numbers less than 10, we need to divide by (RAND_MAX+1). Just to complicate issues, we wish to perform the arithmetic using floating point, and then truncate the result to an integer, our code will therefore look something like this:

```
random_number = (int) (10.0*rand() / (RAND_MAX+1.0));
```

You are not expected to know enough of C to be able to implement all of the tests yet. You should however be able to work out the program logic necessary to implement them. Remember, the tests get harder as you progress.

You are expected to solve these over the next few weeks!

Get your lab tutor to initial your worksheet next to each example to prove you've completed it.

