80 Orlin Avenue SE
Minneapolis MN 55414

May 10th, 1995

The Icon Analyst

The Icon Project
Dept. of Computer Science
Gould Simpson Building
The University of Arizona
Tucson, Arizona 85721

Dear Icon Analyst,
This letter is in response to your articles on the Icon Random Number Generator in numbers 28 and 29, and in particular, to your request for an explanation of the regularities shown in number 29 of the Analyst.

1. A Glitch.

When I first saw the “Rolling Your Own” section of the article in number 28 (p.6) I was struck by the fact that the procedure rand_int(i) has been given a scale factor of $1/(2^{31}-1)$.

When &random is equal to its maximum value of $2^{31}-1$ its scaled value will be 1.0 (provided floating point arithmetic is conducted to sufficient accuracy). Thus the value returned by rand_int(i) will be integer(i * 1.0) + 1 which is i + 1. Thus ?2 could evaluate to 3, for example. To test this hypothesis, I ran the following program.

```
procedure main()
    &random := 1276559117  #Seed that precedes 2^31 - 1
    writes(?2, "/")
    write(&random - (2^31 - 1))
end
```

The output of this program was <u>not</u> 3/0 as I expected, but 2/0. The zero means that &random was $2^{31}-1$ at the point where scaling occurred, and yet the overflow did not occur as predicted. Perhaps the scaling algorithm in icon was not the same as rand_int(i)? To test this hypothesis, I ran the following program.

```
link analyst   #the routines in "Rolling Your Own"
procedure main()
    &random := random := 1276559117
    writes(?2, "/")
    writes(&random - (2^31 - 1), ",")
    writes(rand_int(2), "/")
    writes(random - (2^31 - 1))
end
```

The output of this program was 2/0,3/0 as expected. Thus rand_int(2) can return 3. The following program detects the source of the trouble in the true icon random number generator.

```
    procedure main()
        &random := 1276559117
        write(?0)
    end
```

This program should write out 1.0 if scaling is occurring as stated, but in fact the output on my implementation is less than one——0.99999999672599

Examination of the icon source file rmacros.h shows that RandScale is defined to be 4.65661286e-10 with a comment that this is equal to $1/(2^{31}-1)$. However, this is not correct! It is more like $1/(2^{31}+6)$ The last digit should be an 8 at the accuracy given. Furthermore, in order that RandScale give 1.0 or more when multiplied by $2^{31}-1$, but not when multiplied by $2^{31}-2$, more decimal places need to be specified. On my implementation of icon, it is necessary to specify RandScale to eleven places for this to be true.

What the above shows, however, is that $1/(2^{31}-1)$ is not the correct scale factor anyway, as it can lead to ?i evaluating to i + 1, albeit with very small probability. A better scale factor is 2^{-31}. Changing the scale factor slightly does not make a great deal of difference for practical purposes. (N.B. The sequence of values that &random takes on remains unchanged.) Examination of the algorithm for computing ?i reveals why.

The algorithm computes ?i by placing equally spaced boundaries in the real interval [0,1) giving i sub-intervals that we can number in order from 1 to i. These sub-intervals are closed on the left and open on the right (because of the way that the icon integer function works). Then a new value of &random is computed, and scaled to the interval [0,1). The number of the sub-interval this new value falls into is the result of evaluating ?i.

Most of the time the scaled value of &random does not fall near an interval boundary, provided that i is much smaller than the number of different values that &random can take on. Thus a small change in the scale factor will not move the scaled value across a sub-interval boundary most of the time.

This intuitive argument can easily be quantified. Consider the case of changing RandScale from its present value in the icon source to an accurate representation of 2^{-31} (The decimal expansion of 2^{-31} is *exactly* $4.656612873077392578125\times10^{-10}$). For i < 32, the probability of choosing &random so that ?i would have a different value were RandScale changed is less than 10^{-7}.

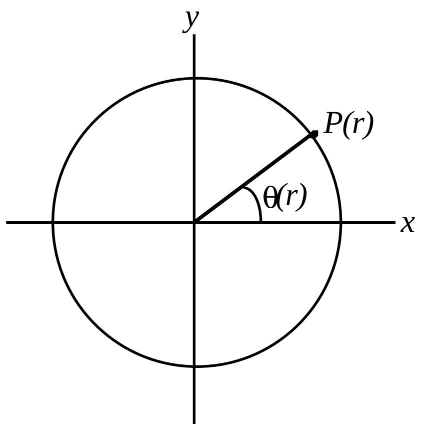
2. "Curiosity or Problem?"

Hereafter, we assume that the scale factor is exactly 2^{-31}. The modulus $m = 2^{31}$ so that this means the scale factor is exactly $1/m$. This is important for the following construction to work properly.

The random number algorithm is more intuitively stated by using a circle rather than the interval [0,1), because a circle reflects the modular arithmetic naturally in its geometry.

Informally, we regard the sequence of values taken on by &random as scaled into the interval [0,1), which is stretched in length by a factor of 2π, and bent nose to tail to become the unit circle. (Yet another reason why the value 1.0 should *not* be present.)

This transformation is properly done using complex exponentials. If r is a possible value of &random, then the corresponding point $P(r)$ on the unit circle in the complex plane is $\exp(2\pi ir/m)$. Here $\theta(r) = 2\pi r/m$ is the angle (in radians) illustrated in the following diagram.



This is the same as taking the unit circle, and placing 2^{31} equally spaced points around its circumference beginning on the positive x axis, corresponding in order to the values $0\ldots 2^{31}-1$ for &random.

Addition is naturally modulo $m = 2^{31}$, and corresponds to adding the angles of the two numbers, i.e. $\theta((r_1 + r_2) \bmod m) = \theta(r_1) + \theta(r_2)$.(This is also the same as multiplying the corresponding complex numbers i.e. $P((r_1 + r_2) \bmod m) = P(r_1) \cdot P(r_2)$.)

Now let's take a geometrical view of the "Curiosity or Problem?" from number 29 of the Icon Analyst, page 6. We know that the values of &random are iterated using the following relation.

$$r_{k+1} = \left(a\, r_k + c\right) \operatorname{mod} m$$
$$\text{where } \begin{cases} a = 1103515245 \\ c = 453816694 \end{cases}$$

Interpreting these values as angles as above (and converting to degrees), we have $\theta(a) \approx 185^\circ$ (and $\theta(c) \approx 76^\circ \approx 75^\circ$). This value interlocks with 360° in a coincidental way that we will show leads to the pattern in the first column of the output of the “Curiosity or Problem” program. (The conversion factor from integer values to angles is now $\left(360^\circ/m\right)$ rather than $\left(2\pi/m\right)$.)

Now to convert the recurrence relation to operations on angles, we apply the function θ to both sides, and use the rule for applying θ to a sum (given above).

$$\theta\left(r_{k+1}\right) = \theta\left(\left(a\, r_k + c\right) \operatorname{mod} m\right) = \theta\left(\left(a\, r_k\right) \operatorname{mod} m\right) + \theta(c)$$

It remains to simplify the term involving the application of θ to a product, modulo m. Since $\theta(x) = \left(360^\circ/m\right)x$ (modulo 360 degrees) we have $\theta\left(\left(a\, r_k\right) \operatorname{mod} m\right) = \theta(a) r_k$ (modulo 360 degrees). This just says that you add the angle $\theta(a)$ around the circle r_k times——it’s repeated addition for angles. So here’s the outcome.

$$\theta\left(r_{k+1}\right) = \theta(a) r_k + \theta(c) \approx r_k 185^\circ + 75^\circ$$

Now, what happens if we start with &random = r_0 = 0, 2, 4, ... as in the given program? Some of the importance of r_0 being even is now apparent. It means that $r_0 185^\circ$ is just a little over a multiple of 360°, i.e. $r_0 185^\circ$ is a “smallish” angle. In fact $r_0 185^\circ$ increases by about 10° each time r_0 is increased by 2. Thus $\theta\left(r_1\right)$ takes on roughly the values $75^\circ, 85^\circ, 95^\circ, 105^\circ, 115^\circ, \ldots$ Note that $\theta(c)$ is unimportant here: it just moves the regularity around so that it starts in a different place.

These numbers are somewhat similar: (“nearly” constant), so we might informally refer to them as having “approximate period 1”. Similarly, the numbers in column 8 of the output of the given program have “approximate period 4”.

The program computes ?20, so that the unit circle is partitioned like a dartboard into 20 equal segments of 18°, numbered (unlike a dart board) in order 1 through 20. Thus the first column of the output is “explained” by the observation that $\theta(a)$ is very close to 180°.

The above argument relied upon there being one application of a linear congruential function to a sequence of seed values with constant small differences. (The seed values themselves may be large.) After the first iteration, the new seed sequence is no longer of this form, and the argument cannot be repeated.

However, the composition of a linear congruential function with another linear congruential function is once again a linear congruential function. Thus, for example, column 8 of the output can be computed from the seeds r_0 = 0, 2, 4, 6, ... as follows: $r_8 = (a_8 r_0 + c_8) \bmod m$, for some constants a_8 and c_8. (This function is the linear congruential function corresponding to the usual linear congruential function composed with itself eight times.)

As the output in column 8 has “approximate period 4”, no doubt we will find that $\theta(a_8)$ is near to 45°, so that increasing r_0 by 2 moves the result through near to 90°; increasing r_0 by 2 four times moves the result approximately to the same value.

The relations (if any) between the columns in the output of the “Curiosity or Problem?” program remain unexplained at this point. We need a more global view to understand this feature.

3. Regularities in General.

Suppose we choose a general linear congruential random number generator, given by the recurrence $r_{k+1} = (a r_k + c) \bmod m$, where random numbers in the range 1 to n are extracted by the function $h(n, r_k) = \lfloor n r_k / m \rfloor + 1$, and $m = 2^j$. (i.e. until further notice, $a, c, m = 2^j$ are perfectly general.)

It is not difficult to see that the solution to this recurrence is $r_k = a^k r_0 + c_k$ where $c_k = \left(c \sum_{l=0}^{k-1} a^l \right)$ (all using modulo m arithmetic). Therefore, column k of the output of an analog of the “Curiosity or Problem” program using the above general random number generator would be determined from the initial seeds r_0 = 0, 1, 2, 3, ... by multiplying by $a^k \bmod m$, adding c_k and applying $h(n,\_)$.

So to understand column k in general, we need to know what possible values $\theta(a^k \bmod m)$ can take on. (Once again, as in the previous section, the value of $\theta(c_k)$ is unimportant, since it merely rotates any regularities.)

To understand $a^k \bmod m$, where $m = 2^j$, we need a little number theory. We will assume that $j \geq 3$, and that a satisfies the criterion $a \bmod 8 = 5$ (which is criterion 3 in number 28 of the Icon Analyst, page 5). We need the following fact about modulo 2^j arithmetic.

$$\boxed{\text{If } S \bmod 4 = 1 \text{ then } S = 5^l \bmod 2^j \text{ for some } 0 \leq l < 2^{j-2}.}$$

What this amounts to is that exactly half of the odd numbers ($S=1$, 5, 9, 13, 17, ..., 2^j-3), have a discrete logarithm l in the base 5. These logarithms will add modulo 2^{j-2}, when the corresponding numbers are multiplied modulo 2^j. Furthermore, $a \bmod 8=5$ implies that $a \bmod 4=1$, so that any valid choice of a will have such a logarithm.

To compute this logarithm, I constructed the following procedures. The procedure power(x, k, m) computes $x^k \bmod m$ by "repeated squaring". It is called by the procedure log5(S, j) that computes the unique value $0 \le l < 2^{j-2}$ such that $S=5^l \bmod 2^j$.

```
procedure power(x, k, m)
     if k = 0 then return 1
     if k % 2 = 0 then return power(x, k/2, m)^2 % m
                  else return x * power(x, k-1, m) % m
end


procedure log5(S, j)
     if S % 4 ~= 1 | j < 3 then fail

     L := 0
     m := 4
     every 1 to j - 2 do
      {
          m *:= 2
          if power(5, L, m) ~= S % m then L +:= m/8
       }
     return L
end
```

The procedure log5(S, j) computes the logarithm bit by bit. At each step, the modulus is doubled, and the old value of the logarithm is tested to see if it is correct with the new modulus. If so then the new uppermost bit of the logarithm must be a zero. If not then it must be a one, and an appropriate power of two is added on to correct the logarithm for the new modulus.

If $a \bmod 8=5$ then the bottom bit of the logarithm of a will be a one, i.e. the logarithm will be odd. Consequently, it will always have a multiplicative inverse modulo 2^{j-2}.

Let α be this inverse. We have $a=5^l \bmod 2^j$, where l is the logarithm, and is odd, and $1=l\alpha \bmod 2^{j-2}$. Raising both sides of $a=5^l \bmod 2^j$ to the power α gives $a^\alpha \bmod 2^j=5$, and since 5 can be expressed as a power of a, then any number S satisfying $S \bmod 4=1$

can be expressed as a power of a. In other words, a is a legitimate base for logarithms, just like 5.

Let S satisfy $S \bmod 4 = 1$ and let $\sigma = \log_5(S,j)$. Then $S = 5^\sigma \bmod 2^j$. But $a^\alpha \bmod 2^j = 5$, so $S = a^{\alpha\sigma} \bmod 2^j$. In other words, $\log_a(S,j) = \alpha \log_5(S,j) \bmod 2^{j-2}$ where $\alpha = (\log_5 a)^{-1} \bmod 2^{j-2}$.

A concrete calculation will illustrate this mechanics for $a = 1103515245$, the usual icon random number generator constant. A call of log5(1103515245, 31) returns 290333047, showing that $a = 1103515245$ is in fact $5^{290333047} \bmod 2^{31}$. A calculation with the extended Euclidean algorithm shows that $(290333047)^{-1} \bmod 2^{29} = 171903047$. This may easily be verified because $171903047 \times 290333047 \bmod 2^{29} = 1$. So, $5 = 1103515245^{171903047} \bmod 2^{31}$, and $\log_{1103515245}(S,31) = 171903047 \log_5(S,31) \bmod 2^{29}$.

What we've shown above is that any a satisfying $a \bmod 8 = 5$ will suffice as a base for logarithms (and we've given a means of calculating $\log_a(S,j)$ for any S satisfying $S \bmod 4 = 1$). *This means that $a^k \bmod 2^j$ may take on any value S satisfying $S \bmod 4 = 1$, simply by varying k.*

Specifically, we can choose a value S (satisfying $S \bmod 4 = 1$) that has $\theta(S)$ very close to a "coincidental" angle (e.g. 180°, 45°, or 60°). Then there always exists $k = \log_a(S,j)$ such that $a^k \bmod 2^j = S$. Thus, the k^{th} column of output of a "Curiosity or Problem?" type program would then (in principle) exhibit striking near periodicity of almost any kind desired.

Of course k could be rather large. But what we have established is that the kinds of columns output by the "Curiosity or Problem?" program are by no means untypical, and are not specific to the choice of constants made by the icon random number generator, but are inherent whenever the modulus is a power of 2.

As an example, let's choose $\theta(S) \approx 90^\circ$ with the standard icon random number generator. Then $S = 2^{29} + 1$ is very close, and satisfies $S \bmod 4 = 1$. A call of log5(2^29 + 1, 31) gives 402653184. From four paragraphs back, we have $\log_{1103515245}(S,31) = 171903047 \log_5(S,31) \bmod 2^{29}$, and substituting, we have $\log_{1103515245}(S,31) = 134217728$. Thus, in column 134217728 we should have approximate periodicity 4 (with minescule drift from exact periodicity), as in the following program.

```
procedure main()
    k := 134217728; i := 1000
    every &random := 0 to 19 do {
        every 1 to k - 1 do ?1
        write(?i)
     }
     end
```

The output from this program on my system is as follows. (It took an overnight run to collect these numbers.)

```
     125
     375
     625
     875
     126
     376
     626
     876
     126
     376
     626
     876
     126
     376
     626
     876
     126
     376
     626
     876
```

4. Conclusion.

We have shown that the regularities in the columns of output from the “Curiosity or Problem?” program are an instance of a general phenomenon of column regularity. There is an enormous amount of regularity in the output of the icon random number generator, so it’s not surprising that some of it wound up in the first few columns. One may choose any period, with smaller or larger amounts of “drift”, and by the procedure exemplified above, find a column exhibiting that behavior. Furthermore, this is inherent to all linear congruential random number generators using a modulus that is a power of two.

One possibility that you might wish to consider, is of having two random number generators in icon with a keyword variable as a switch. Then you could keep the original generator (the default), and provide a more sophisticated source if required. (Perhaps the sophisticated source could also be seeded by the clock at the point it was switched to? If this feature was not required, it could always be avoided by assigning to `&random`.)

Of course the problem of finding a “good” source of pseudo-random numbers is notorious for its apparent trade-off between “goodness” and computational effort. An aggressive definition of “good” is roughly that no polynomial-time algorithm can distinguish the source from a truly random source.

This statement can be made precise, and is the standard of randomness used in theoretical studies of cryptography. We do not need such a drastic definition, but it's worth looking at the consequences intuitively, to see that the idea of "high degree" (see below) is important.

The existence of such a source is an open question, and is equivalent to the existence of (a natural class of) one-way functions. In fact it is one these one way functions that would be used in place of the linear congruential function in such a pseudo-random source. An existence proof would imply $P \neq NP$ and much more.

One-way functions (if they exist) are functions of "high degree" (this has a proper definition, but take it intuitively), and consequently are much more painful to compute than a linear congruential function.

However, a "high degree" function (chosen carefully) is likely not to possess any simple (or "simple-ish") algebraic properties (unlike low degree functions), that give rise to patterns when iterated. And if there is a proof that iterating such a function eventually cycles through all the values in range, then it may well suffice for practical purposes provided that there is a reasonably efficient way to compute it.

If you are interested in installing such a random number generator, I have a candidate that is fairly easy to compute, for which I have some confidence of being able to prove some nice properties. A little testing may be in order too.

I hope I've communicated at the right level in this letter. If you want to use any of the ideas or material in this letter in an article in the Icon Analyst, then go ahead.

Sincerely,

Carl Sturtivant.
(carl@cs.umn.edu)