

NOTE AE: RANDOM NUMBER GENERATOR VBA ROUTINES

GENERATE DIEHARD TEST INPUT FILES

```
Sub BuildDiehardFile()  
Dim hn, hpos1, hout, hix, hiy, hiz As Long  
Dim xa, xb, xn As Double  
'Const reffh = 4294967296#  
'Const refl = 2147483647#  
  
hpos1 = 1&  
hn = 1&  
  
Rem uses Sobel's recommended starting seed  
hix = 5&  
hiy = 11&  
hiz = 7&  
  
Open "C:\newtrial.bin" For Binary As 2  
  
With Application.ActiveSheet  
Do  
    Call Generate_WH_FPRN(hix, hiy, hiz, xn) 'used for the equation set  
'Calculate 'optional, to ensure new RAND value in cell(1,1)  
'xn = .Cells(1, 1).Value 'cell(1,1) has the equation =RAND()  
    xa = reffh * xn  
    If xa > refl Then  
        xb = xa - refl  
    Else  
        xb = xa  
    End If  
    hout = CLng(xb) 'hout has the correct bit sequence of an unsigned long integer  
    Put #2, hpos1, hout  
    hpos1 = hpos1 + 4&  
    hn = hn + 1&  
    .Cells(1, 2) = hn 'visual track on progress  
    If hn > 3000000 Then 'for diehard-II, use 3000000, for Diehard-III, use  
67000000  
        Exit Do  
    End If  
Loop  
Close  
End With  
ActiveWorkbook.Save
```

End Sub

The fault here is that the file “C:\newtrial.bin” results in a catastrophically failure in Diehard-II. A detailed examination of the bit patterns shows that the fault appears to be in the VBA module.

Given the following first four bit patterns of hout, (going into the file)

```
0001 1001 1111 0100 0111 1110 0101 0101
0100 1100 0100 0100 0111 1011 0100 0100
0100 0101 1101 0110 1010 1111 1001 1110
0111 1110 1001 0010 0111 0110 1011 0001
```

The bit patterns in the file after closing and reopening are,

```
0111 1110 0101 0101 0000 0000 0000 0011
0111 1011 0100 0100 0000 0000 0000 0011
1010 1111 1001 1110 0000 0000 0000 0011
0111 0110 1001 0010 0000 0000 0000 0011
```

The Diehard-II reports shows a total failure in bits 16-32, indicating that the loss of the input bits is correct.

This is a fault in VBA, incorrectly building a binary file. The instruction “ Put #2, hpos1, hout” inserts the last 4 bytes of hout into the first 4 bytes of the address and puts in the sequence “0000 0000 0000 0011” to fill out the 8 byte number.

When running the above subroutine with stops, hout has the correct long integer value.

A Visual Basic routine was built to properly build a binary file from the respective random number algorithms. It was run in Visual Basic 6.

MARSAGLIA'S MWC256 RNG

Static Function MWC256()

‘Marsaglia's Multiply with carry 256 RNG, adapted to vb from C

Dim init As Integer

Dim xj, xa, xc, xcarry, ya, reg, upper, block As Double

Dim qv(1 To 256) As Double

Dim I As Integer

Dim t, TU, tl, tld, tldu, xb, xd, U2, u3, u4 As Double

Dim xhu, x1 As Double

If init = 0 Then

xj = 123456789#

xa = 69069#

xc = 12345#

xcarry = 362436#

ya = 809430660#

reg = 4294967296#

upper = 281474976710656#

```

block = 65536#

For I = 1 To 256
    xhu = xa * xj + xc
    x1 = xhu / reg
    xj = Fix(xhu - reg * Fix(x1))
    qv(I) = xl
Next I
init = 10
I = 0
End If
If init > 0 Then
    I = I + 1
    If I > 256 Then
        I = 1
    End If
    t = ya * qv(I) + xcarry      'Marsaglia's basic MWC long-long integer equation
    TU = Fix(t / reg)          'correct upper 32 bit number
    If t > upper Then
        tld = t - reg * TU      'defective lower 32 bit number
        tldu = Fix(tld / block)  'extract correct upper 16 bit portion of tld
        xb = ya - block * (Fix(ya / block))    'partial multiplicand of t
        xd = qv(I) - block * (Fix(qv(I) / block)) 'partial multiplicand of t
        U2 = xd * xb + xcarry    'total partial with correct lower 2 bytes
        u3 = Fix(U2 / block)     'incomplete upper 2 bytes
        u4 = U2 - block * u3     'extract correct lower 2 bytes from u2
        tl = tldu * block + u4   'correct lower 32 bits
    Else
        tl = t - reg * TU      'form when all 52 bits of mantissa are correct
        'Stop
    End If
End If
xcarry = TU
qv(I) = tl
MWC256 = tl / reg
End Function

```

The original version in C++ is much shorter and direct. The problem with VBA is that it does not have unsigned long integers, nor does it allow access to registers for shift operations. C++ also has other shorthand cuts that take some hard figuring on how to implement them in VBA. This makes implementation of a lot of good PRNGs written in C++ cumbersome when translated to VBA.