EXAMPLE PROGRAM RANDOM NUMBER GENERATOR

1. Requirements

Develop and produce a program that generates a series of random numbers between 0 and 99 inclusive until the number 50 is generated. Use the equation:

$$N+1 = 3125 \times N \ mod \ 8192$$

where:

mod is the modulus operator.

N is the current term.

N+1 is the following term.

Note that to start the series we require an initial value for N which must be an odd number between 1 and 1891. Note also that the equation will produce terms ("random numbers") within the range 1 to 8191 therefore to produce random numbers within the range 0 to 99 inclusive we must multiply the result by 100/8191.

2. Design

A top down analysis of the problem is given in Table 1.

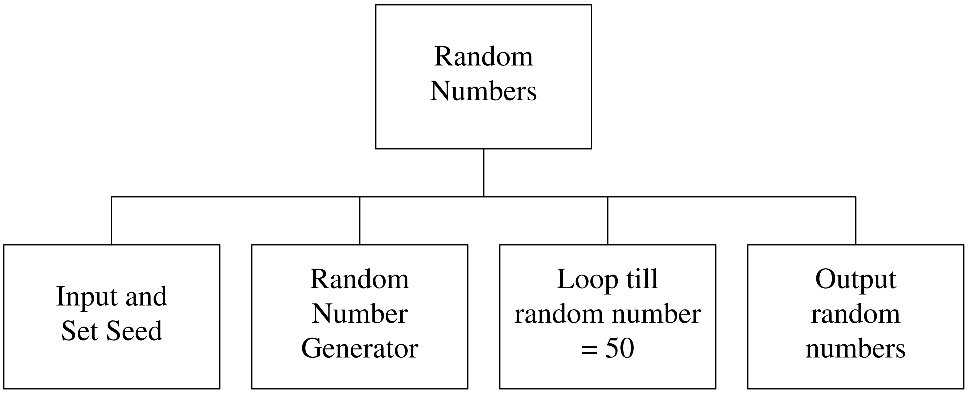

Table 1: *Top down analysis*

The operations given in Table 1 may be implemented using the following procedures/functions.

1) `RANDOM` (Top level procedure): Variable count loop till random number equivalent to 50 is arrived at, calls to `START_VALUE` and `RANDOM_NUMBER`.

Name	Description	Type	Range
M_CONSTANT	Global constant	INTEGER	8192
K_CONSTANT	Global constant	INTEGER	3125
RAND_NUM	Global variable	NATURAL	Default
TERM	Global variable	NATURAL	Default

2) `START_VALUE` (Level 2 procedure): Input seed, include error recovery loop.

Name	Description	Type	Range
SEED	local input variable	POSITIVE	1..M_CONSTANT-1;

3) `RANDOM_NUMBER` (Level 2 function): Generate and return random number.

Appropriate Nassi-Shneiderman charts are given below.

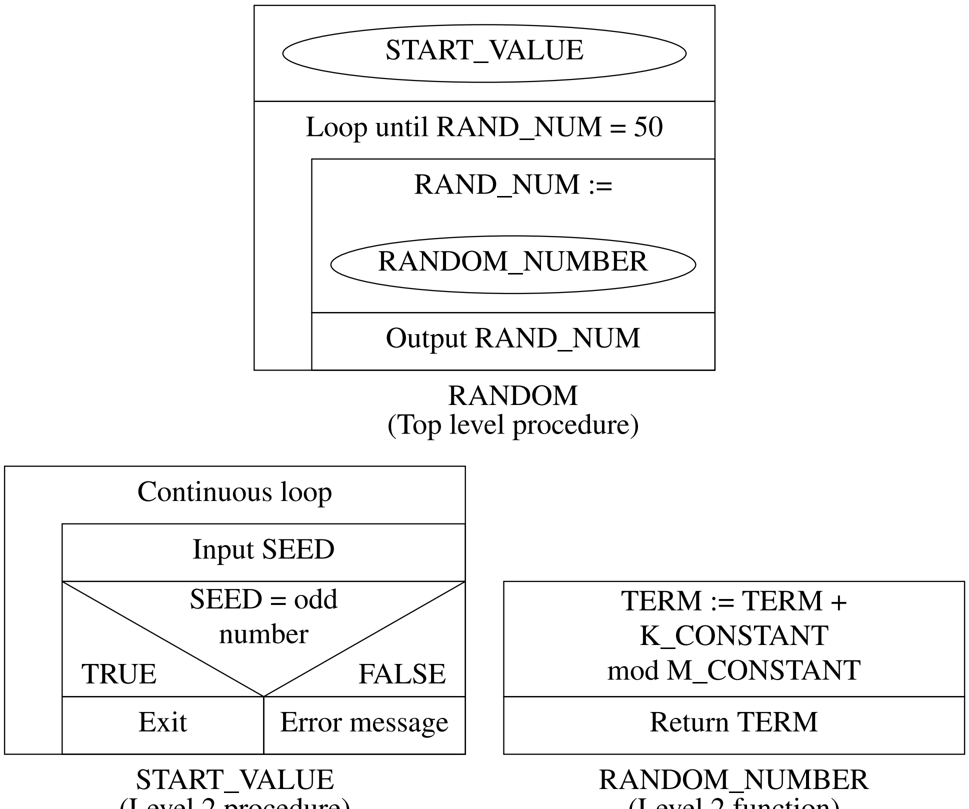

Table 2: *Nassi-Shneiderman chart*

3. Implementation

The encoding of the above design is given in Table 3.

```
-- RANDOM NUMBER GENERATOR
-- 16 September 1997
-- Frans Coenen
-- Dept Computer Science, University of Liverpool

with TEXT_IO;
use TEXT_IO;

procedure RANDOM is
    package INTEGER_INOUT is new INTEGER_IO(INTEGER);
    use INTEGER_INOUT;
    package FLOAT_INOUT is new FLOAT_IO(FLOAT);
    use FLOAT_INOUT;
    M_CONSTANT : constant := 8192;          -- 2**13
    K_CONSTANT : constant := 3125;          -- 5**5
    RAND_NUM   : NATURAL  := 0;
    TERM       : NATURAL;

    ------------------------------------------------------------
    -- START VALUE PROCEDURE
    procedure START_VALUE is
        SEED: POSITIVE range 1..M_CONSTANT-1;
    begin
        loop
    -- Output request to user and get SEED
            PUT("Enter seed (odd number within the range 1 to ");
            PUT(M_CONSTANT-1, WIDTH=>5);
            PUT_LINE("): ");
            GET(SEED);
    -- Check seed (seed must be an odd number
            if SEED rem 2 = 1 then
                TERM := SEED;
                exit;
            else
                PUT("Invalid seed: ");
                PUT(SEED);
                PUT_LINE(", try again");
                NEW_LINE;
            end if;
        end loop;
        NEW_LINE;
    end START_VALUE;

    ----------------------------------------------------------
    -- RANDOM NUMBER
    function RANDOM_NUMBER return NATURAL is
    begin
         TERM := TERM * K_CONSTANT mod M_CONSTANT;
         RETURN(TERM);
    end RANDOM_NUMBER;
    ----------------------------------------------------------
```

Table 3: *Implementation*

```
-- TOP LEVEL
begin
    START_VALUE;
    WHILE RAND_NUM /= 50 loop
        RAND_NUM := RANDOM_NUMBER*100/8191;
        PUT(RAND_NUM, WIDTH=>4);
    end loop;
    NEW_LINE;
end RANDOM;
```

Table 3 Continued

4. Testing

4.1. Black box testing

BVA testing: We should test the input range values for the variable `SEED` using a set of BVA test cases of the form given in Table 4. Results from running the test cases are given in Table 5.

TEST CASE SEED	EXPECTED RESULT OUTPUT
0	CONSTRAINT_ERROR
2	Even number
8190	Even number
8192	CONSTRAINT_ERROR

Table 4: BVA Test Cases

```
kuban-197 $ random
Enter seed (odd number within the range 1 to  8191):
0
Ada-runtime: Exception CONSTRAINT_ERROR raised in
/tmp_mnt/home/staff2/ra/frans/public_html/2CS21/week8/random.ada
on line 29. Rangecheck error: 0 not in range 1..8191.

kuban-199 $ random
Enter seed (odd number within the range 1 to  8191):
2
Invalid seed:           2, try again

Enter seed (odd number within the range 1 to  8191):
8190
Invalid seed:        8190, try again

Enter seed (odd number within the range 1 to  8191):
8192
Ada-runtime: Exception CONSTRAINT_ERROR raised in
/tmp_mnt/home/staff2/ra/frans/public_html/2CS21/week8/random.ada
on line 29. Rangecheck error: 8192 not in range 1..8191.
```

Table 5: Results of BVA Testing

Limit testing: A set of limit test cases is given in Table 6. Results from running the test cases are given in Table 7.

TEST CASE SEED	EXPECTED RESULT OUTPUT
1	Sequence of random numbers
8191	Sequence of random numbers

Table 6: Limit Test Cases

```
kuban-200 $ random
Enter seed (odd number within the range 1 to  8191):
1
38   9  29  69  12  73  56  66  78  41  48  42  84  62  55  46  81  86  29
27  18  63  17  37  46  44  72  25  14  77  42  42  74  13  78  36  73   3
29  59  64  96  46  59  95  42  79  88  17  91  77  95  79  93  91  30  32
98  70  43  25  57  65  84  10  18  26   3  34  33   3  51  50

kuban-201 $ random
Enter seed (odd number within the range 1 to  8191):
8191
61  90  70  30  87  26  43  33  21  58  51  57  15  37  44  53  18  13  70
72  81  36  82  62  53  55  27  74  85  22  57  57  25  86  21  63  26  96
70  41  35   3  53  40   4  57  20  11  82   8  22   4  20   6   8  69  67
 1  29  56  74  42  34  15  89  81  73  96  65  66  96  48  49  48  54  23
93  78  97  69  47   3  74  37  59  76  35  77  81  46  30  39  63  63  10
73  54  76  24  29   4  37  23  56  63  94  56   6  82  98  73  27  11  99
25  70  98  47  61  85  95  91  32  22  52  83  87  31  18  71  76  61  43
 7  50
```

Table 7: Results of Limit Testing

Arithmetic testing: Arithmetic testing dictates test cases using a negative, zero and positive sample values for the input variables. In this case there is only one input variable, a negative value is precluded by its type definition and we have already incorporated a zero sample value in the limit test cases given in Table 6. We therefore only need to define a single arithmetic test case involving a positive sample value (Table 8, result in Table 9).

TEST CASE SEED	EXPECTED RESULT OUTPUT
495	Sequence of random numbers

Table 8: Arithmetic Test Case

```
Enter seed (odd number within the range 1 to  8191):
495

82  98  73  27  11  99  25  70  98  47  61  85  95  91  32  22  52  83
87  31  18  71  76  61  43   7  50
```

Table 9: Results of Arithmetic Testing

4.2. White box testing.

Path Testing: We should test both paths through the `START_VALUE` procedure using an odd and an even value. However, this has already been taken into consideration by the proposed BVA test cases. The loop is a variable count loop which cannot be influenced by the user in a predictable manner. Therefore loop testing is not required in this case either.

4.3. Data Validation Testing

The output from a number of data validation tests is given in Table 10.

```
kuban-203 $ random
Enter seed (odd number within the range 1 to  8191):
34.6
Invalid seed:          34, try again

Enter seed (odd number within the range 1 to  8191):

Ada-runtime: Exception DATA_ERROR propagated out of main.

kuban-206 $ 6
ksh: 6:  not found

kuban-207 $ random
Enter seed (odd number within the range 1 to  8191):
frans

Ada-runtime: Exception DATA_ERROR propagated out of main.

kuban-208 $ rans
ksh: rans:  not found
```

Table 10: Results of Data Validation Test Cases