EXAMPLE PROGRAM LINEAR EQUATION REPORT

1. Requirements

A linear equation with one unknown can be expressed as follows:

$$bx + c = 0$$

where `b` and `c` represent numeric quantities. Produce an Ada program which will solve equations of this form (i.e. find a value for `x` - the root of the equation. Assume `b` and `c` are integers and that the output is to presented accurate to four decimal places. Note that to find a value for x we must rearrange the above equation:

$$x = -c/b$$

Consequently b must not have a value of 0 (otherwise a "divide by zero error will occur").

2. Design

A top level analysis is presented in Table 1.

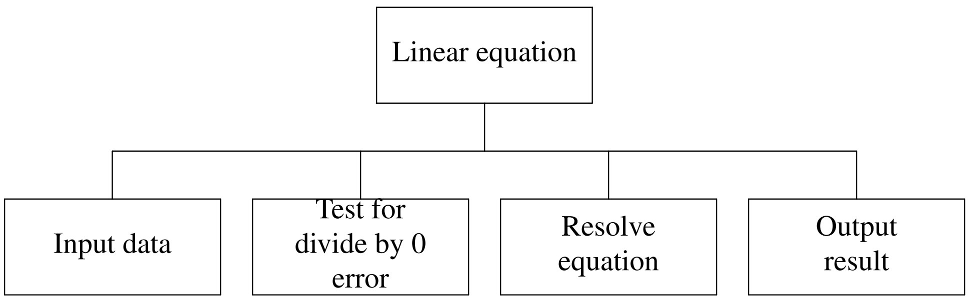

Table 1: *Top down analysis*

Identified operations can be encoded using a single procedure:

1) `LINEAR_EQUATION` (Top level procedure): Input data, test for "b = 0", resolve equation and output result.

Name	Description	Type	Range
B_VALUE	Global input variable	INTEGER	Default
C_VALUE	Global input variable	INTEGER	Default
X_VALUE	Global variable	FLOAT	Default
C_FLOAT	Global variable	FLOAT	Default
B_FLOAT	Global variable	FLOAT	Default

A Nassi-Shneiderman design for this procedure is presented in Table 2 and a flow chart in Table 3.

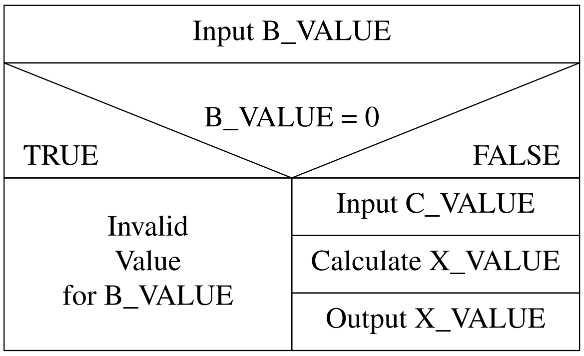

Table 2: *Nassi-Shneiderman Chart*

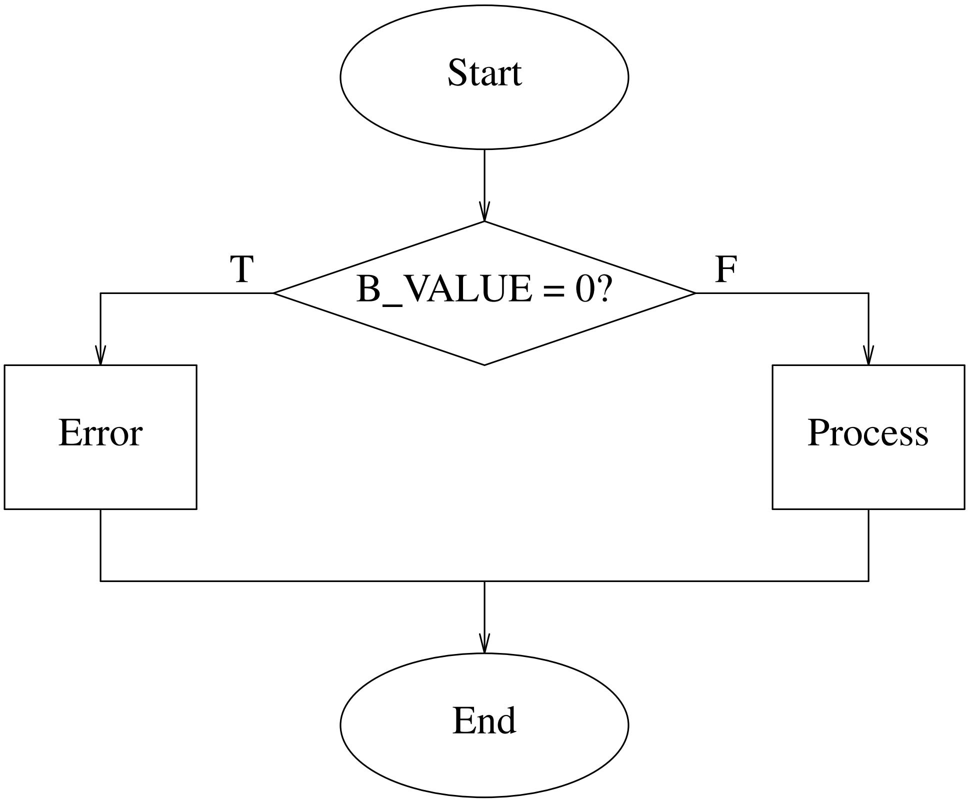

Table 3: *Flow chart*

3. Implementation

Implementation is as shown in Table 4:

```
-- LINEAR EQUATION
-- 6 August 1997
-- Frans Coenen
-- Dept Computer Science, University of Liverpool

with TEXT_IO;
use TEXT_IO;

procedure LINEAR_EQUATION is
        package INTEGER_INOUT is new INTEGER_IO(INTEGER);
        use INTEGER_INOUT;
        package FLOAT_INOUT is new FLOAT_IO(FLOAT);
        use FLOAT_INOUT;
        B_VALUE, C_VALUE          : INTEGER;
        B_FLOAT, C_FLOAT, X_VALUE : FLOAT;
begin
-- Read value for B_VALUE
        PUT_LINE("Input value for B: ");
        GET(B_VALUE);
-- Check B_VALUE = 0, if so output error message
        if B_VALUE = 0 then
                PUT_LINE("Invalid value for B_VALUE (B_VALUE = 0)");
-- Otherwise continue, obtain value for C_VALUE, convert B_VALUE and
-- C_VALUE to type FLOAT, and then  calculate and output X_VALUE
        else
                PUT_LINE("Input value for C: ");
                GET(C_VALUE);
                B_FLOAT := FLOAT(B_VALUE);
                C_FLOAT := FLOAT(C_VALUE);
                X_VALUE := (-1.0*C_VALUE)/B_VALUE;
                PUT("X is ");
                PUT(X_VALUE, FORE=>3, AFT=>4, EXP=>0);
                NEW_LINE;
        end if;
end LINEAR_EQUATION;
```

Table 4: *Implementation*

4. Testing

4.1. Black box testing

BVA testing: Input values have a notional range associated with them dictated by the machine dependent lowest and highest numbers available in the default integer range. An appropriate pair of BVA test cases is therefore given in Table 5.

TEST CASE		EXPECTED RESULT
B_VALUE	C_VALUE	X_VALUE
-2147483647	-2147483647	-1
2147483646	2147483646	-1

Table 5: *BVA test cases*

Limit testing: We should also test the effect of all possible combinations of input at the default limits of the integer range. A suitable set of test cases is given in Table 6.

TEST CASE		EXPECTED RESULT
B_VALUE	C_VALUE	X_VALUE
2147483647	2147483647	-1.0000
2147483647	-2147483648	1.0000
-2147483648	2147483647	1.0000
-2147483648	-2147483648	-1.0000

Table 6: *Limit test cases*

Arithmetic testing: Arithmetic testing involves test cases using negative, zero and positive sample input values. We have tested combinations of positive and negative input values, therefore we only need to test values which involve a zero value (Table 7).

TEST CASE		EXPECTED RESULT
B_VALUE	C_VALUE	X_VALUE
-10	0	0.0000
0	*	Invalid value
10	0	0.0000

Table 7: *Arithmetic test cases*

The resulting output when the above black box test cases are run is given in Table 8.

```
kuban-162 $ linear_equation
Input value for B:
-2147483647
Input value for C:
-2147483647
X is  -1.0000

kuban-163 $ linear_equation
Input value for B:
2147483646
Input value for C:
2147483646
X is  -1.0000

kuban-164 $ linear_equation
Input value for B:
2147483647
Input value for C:
2147483647
X is  -1.0000
```

Table 8: *Black box testing*

```
kuban-165 $ linear_equation
Input value for B:
2147483647
Input value for C:
-2147483648
X is   1.0000

kuban-166 $ linear_equation
Input value for B:
-2147483648
Input value for C:
2147483647
X is   1.0000

kuban-167 $ linear_equation
Input value for B:
-2147483648
Input value for C:
-2147483648
X is  -1.0000

kuban-168 $ linear_equation
Input value for B:
-10
Input value for C:
0
X is   0.0000

kuban-169 $ linear_equation
Input value for B:
0
Invalid value for B_VALUE (B_VALUE = 0)

kuban-170 $ linear_equation
Input value for B:
10
Input value for C:
0
X is   0.0000
```

Table 8 continued

4.2. White box testing

Path testing: The DFD given in Table 3 indicates two paths through the program. We should devise test cases to exercise both paths. Two appropriate test cases are given in the Table 9, The first is a duplicate of a black box test case, thus we only need to run the second (Table 10).

TEST CASE		EXPECTED RESULT
B_VALUE	C_VALUE	X_VALUE
0	*	Invalid value
100	400	4

Table 9: *Path test cases*

```
Input value for B:
100
Input value for C:
400
X is  -4.0000
```

Table 10: *White box testing*

4.3. Data validation testing

Table 10 shows the results produced when a number of data validation tests are run.

```
kuban-206 $ linear_equation
Input value for B:
1.2
Input value for C:
Ada-runtime: Exception DATA_ERROR propagated out of main.

kuban-207 $ 2
ksh: 2:  not found

kuban-208 $ linear_equation
Input value for B:
4
Input value for C:
frans
Ada-runtime: Exception DATA_ERROR propagated out of main.

kuban-209 $ rans
ksh: rans:  not found
```

Table 10: *Data validation test case output*