EXAMPLE PROGRAM SET INTERSECTION

1. Requirements

To develop an Ada program that expands expressions of the form `(1+x)^n`, for any value of `n` ranging from 1 to 32, using Pascal's triangle to determine the appropriate coefficients. Recall that Pascal's triangle is of the form:

```
            1   1
          1   2   1
        1   3   3   1
      1   4   6   4   1
    1   5  10  10   5   1
  1   6  15  20  15   6   1
1   7  21  35  35  21   7   1
Etc.
```

where the start and end coefficient in each line are 1 (unity) and each of the other coefficients is the sum of the corresponding coefficient and the preceding coefficient in the line above. Examples:

```
                           (1 + x)  = 1 + x
        (1 + x)^2 = (1 + x)(1 + x)  = 1 + 2x + x^2
(1 + x)^3 = (1 + x)(1 + x)(1 + x)  = 1 + 3x + 3x^2 + x^3
      (1 + x)^4 = (1 + x)(1 = x)^3 = 1 + 4x + 6x^2 + 4x^3 + x^4
      (1 + x)^5 = (1 + x)(1 = x)^4 = 1 + 5x + 10x^2 + 10x^3 + 5x^4 + x^4
```

and so on.

2. Design

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

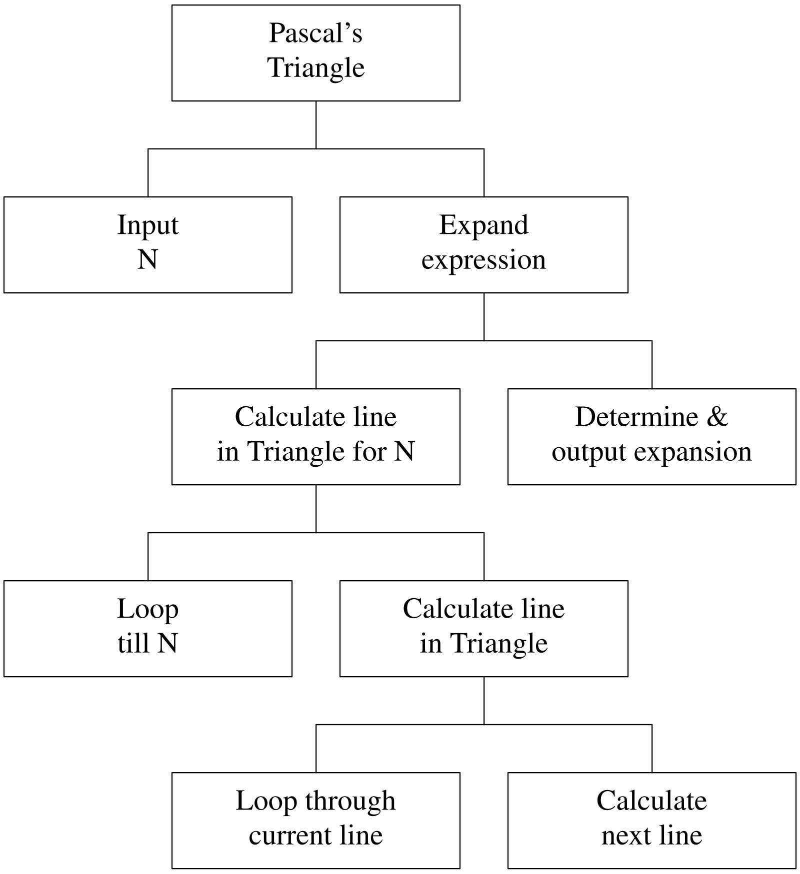


Table 1: *Top down analysis*

We will store the current line in the Triangle in an unconstrained array and use this to build the following line. The start line array will comprise two elements: `1` and `1`. We will use a recursive function to iterate through the levels in the triangle until level N is reached. Once we have established line N in the triangle we will use the resulting array of coefficients to expand the expression `(1-x)^N` for the appropriate value of `N`.

1) `PASCALS_TRIANGLE` (top level procedure): Input power N, check for power of 1 in which case direct output (`(1+x)`) Otherwise call `PASC_TRIANG`.

NAME	USAGE	TYPE	RANGE/VALUE
POWER_N	Local variable	INTEGER	1..32

2) `PASC_TRIANG` (Level 2 procedure): Initiate expansion. Calls to `LINE_IN_TRIANGLE` and `EXPANSION`. Include type definition for `LINE_ARRAY` and initialisation of start line (`{1, 1}`).

NAME	USAGE	TYPE	RANGE/VALUE
POWER	Formal parameter	INTEGER	Default
START_LINE	Local constant (array)	LINE_ARRAY	{1, 1}
COEFFICIENTS	Local variable (array)	LINE_ARRAY	n.a.

3) `EXPANSION` (Level 2 procedure): Using identified line in triangle output expansion of expressions `(1+x)^N`.

NAME	USAGE	TYPE	RANGE/VALUE
LINE_N	Formal parameter (array)	LINE_ARRAY	n.a.
POWER	Formal parameter	INTEGER	Default

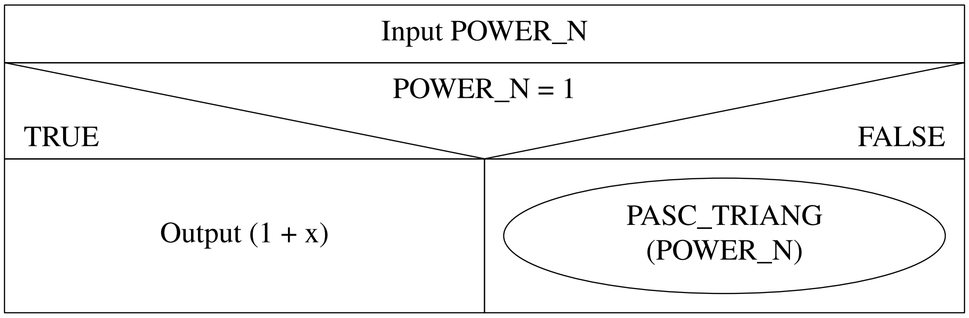
4) `LINE_IN_TRIANGLE` (level 2 function): Recure till appropriate line in triangle is arrived at. Call `NEXT_LINE_IN_TRIANGLE` and self.

NAME	USAGE	TYPE	RANGE
CURRENT_LINE	Formal parameter (array)	LINE_ARRAY	n.a.
LINE_LENGTH	Formal parameter	INTEGER	Default
LINE_NUMBER	Formal parameter	INTEGER	Default
NEXT_LINE	Local variable (array)	INTEGER	Default

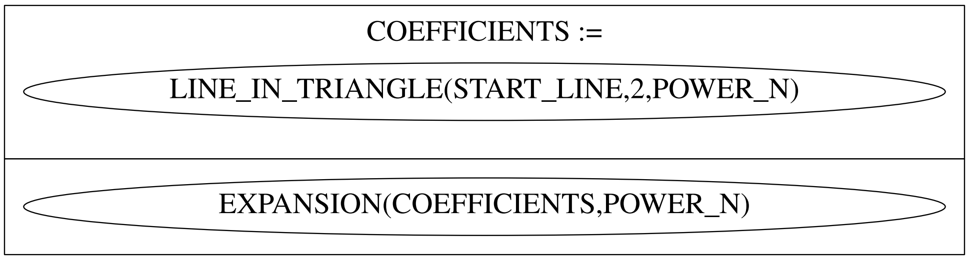
5) `NEXT_LINE_IN_TRIANGLE` (Level 4 function): Calculate next line in triangle form current line and return next line.

NAME	USAGE	TYPE	RANGE
OLD_LINE	Formal parameter (array)	LINE_ARRAY	n.a.
INDEX	Local variable	INTEGER	Default
NEW_LINE	Local variable	LINE_ARRAY	n.a.

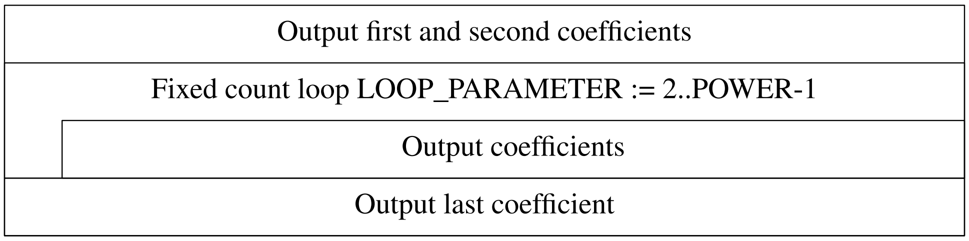
A complete Nassi-Shneiderman design for the above is given below:



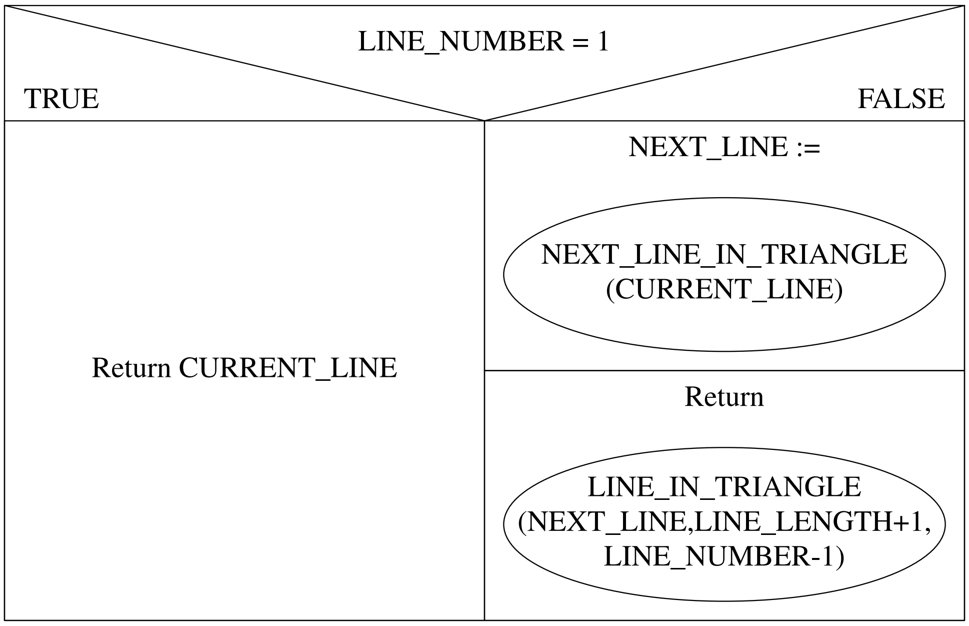
PASCALS_TRIANGLE
(Top Level Procedure)



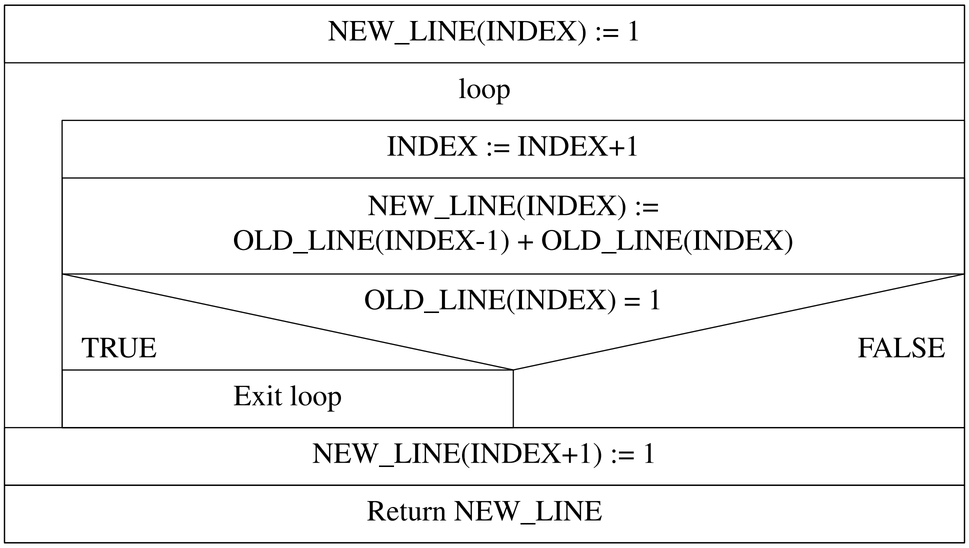
PASC_TRIANG
(Level 2 Procedure)



EXPANSION
(Level 2 Procedure)



LINE_IN_TRIANGLE
(Level 2 function)



NEXT_LINE_IN_TRIANGLE
(Level 3 function)

Table 2*: Nassi-Shneiderman chart*

A flow chart indicating the flow of control through this sequence of procedures/functions is given in Table 3.

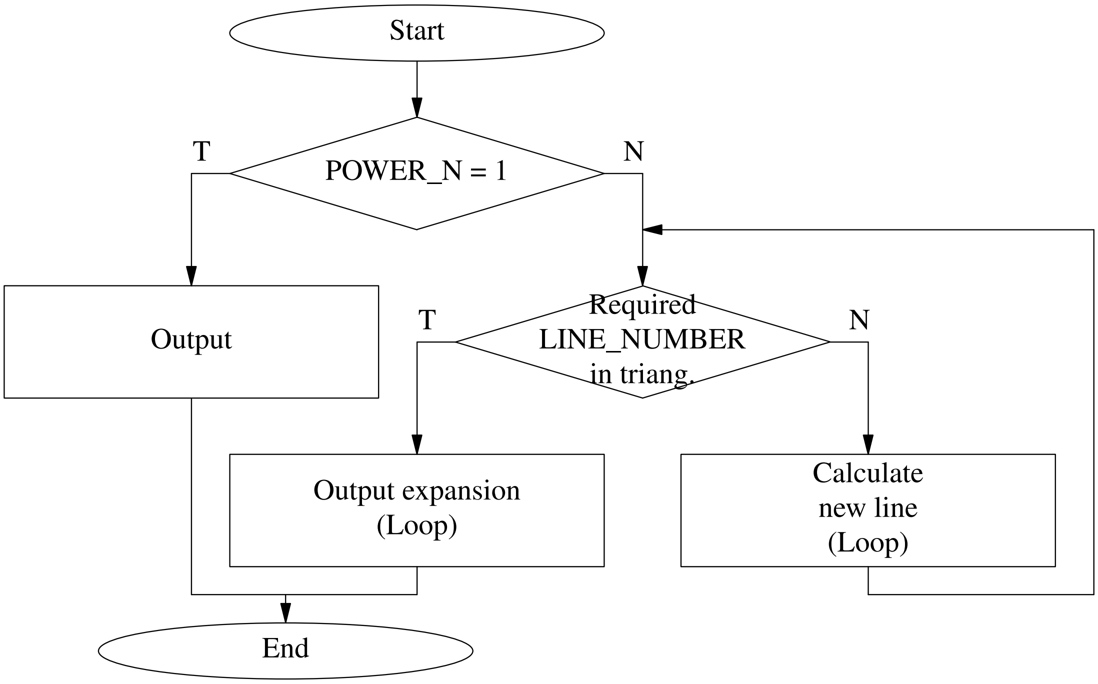


Table 4: *Control flow chart*

3. Implementation

The encoding of the above design is given in Table 4.

```
-- PASCALS TRIANGLE
-- 25 SEPTEMBER 1997
-- Frans Coenen
-- Dept Computer Science, University of Liverpool

with TEXT_IO;
use TEXT_IO;

procedure PASCALS_TRIANGLE is
    package INTEGER_INOUT is new INTEGER_IO(INTEGER);
    use INTEGER_INOUT;
    POWER_N : INTEGER range 1..32;

    ------------------------------------------------------------------------
    -- PASC. TRIANG. - Commence expansion
    procedure PASC_TRIANG(POWER_N : INTEGER) is
        type LINE_ARRAY is array (INTEGER range <>) of INTEGER;
        START_LINE   : constant LINE_ARRAY(1..2) := (1, 1);
        COEFFICIENTS : LINE_ARRAY(1..POWER_N+1);

        --------------------------------------------------------------------
        -- CALCULATE LINE IN TRIANGLE, recursive function to determine
        -- appropriate line in triangle
        function LINE_IN_TRIANGLE(CURRENT_LINE : LINE_ARRAY; LINE_LENGTH,
                        LINE_NUMBER : INTEGER) return LINE_ARRAY is
        NEXT_LINE : LINE_ARRAY(1..LINE_LENGTH+1);
            ----------------------------------------------------------------
            -- CALCULATE NEXT LINE IN TRIANGLE
            function NEXT_LINE_IN_TRIANGLE(OLD_LINE : LINE_ARRAY) return
                    LINE_ARRAY is
                INDEX    : INTEGER := 1;
                NEW_LINE : LINE_ARRAY(1..OLD_LINE'LENGTH+1) ;
            begin
                NEW_LINE(INDEX) := 1;
            -- Loop through previous line in triangle
                loop
                    INDEX := INDEX + 1;
                    NEW_LINE(INDEX) := OLD_LINE(INDEX-1) + OLD_LINE(INDEX);
                    exit when OLD_LINE(INDEX) = 1;
                end loop;
                NEW_LINE(INDEX+1) := 1;
                RETURN(NEW_LINE);
            end NEXT_LINE_IN_TRIANGLE ;
            ----------------------------------------------------------------
```

Table 4: *Implementation*

```
        begin
            -- Base case
            if LINE_NUMBER = 1 then
                RETURN(CURRENT_LINE);
            -- Recursive step
            else
                NEXT_LINE := NEXT_LINE_IN_TRIANGLE(CURRENT_LINE);
                RETURN(LINE_IN_TRIANGLE(NEXT_LINE, LINE_LENGTH+1,
                        LINE_NUMBER-1));
            end if;
        end LINE_IN_TRIANGLE;
        ------------------------------------------------------------------
        ------------------------------------------------------------------
        -- EXPANSION, output
        procedure EXPANSION(LINE_N: LINE_ARRAY; POWER: INTEGER) is
        begin
            PUT("1 + ");
            PUT(LINE_N(2), WIDTH=>1);
            PUT("x");
        -- Loop through coefficients in line in triangle
            for LOOP_PARAMETER in 2..POWER-1 loop
                PUT(" + ");
                PUT(LINE_N(LOOP_PARAMETER+1), WIDTH=>1);
                PUT("x^");
                PUT(LOOP_PARAMETER, WIDTH=>1);
            end loop;
            PUT(" + x^");
            PUT(POWER,WIDTH=>1);
            NEW_LINE;
        end EXPANSION;
        ------------------------------------------------------------------

    begin
        COEFFICIENTS := LINE_IN_TRIANGLE(START_LINE,2,POWER_N);
        EXPANSION(COEFFICIENTS,POWER_N);
    end PASC_TRIANG;
    ----------------------------------------------------------------------

begin
-- Get input
    PUT_LINE("Input power (integer 1..32): ");
    GET(POWER_N);
-- If power 1 direct output
    if POWER_N = 1 then
        PUT_LINE("1 + x");
-- Otherwise determine coefficients and then output expansion
    else
        PASC_TRIANG(POWER_N);
    end if;
end PASCALS_TRIANGLE;
```

Table 4 Continued

4. Testing

4.1. Black box testing

BVA, limit and arithmetic testing: Test input variable **POWER_N** using BVA analysis and limit testing. Test operation of arithmetic expression using (in this case) a positive sample value. A suitable set of test cases is given in Table 5 and results in Table 6.

TEST CASE POWER_N	EXPECTED RESULT OUTPUT
0	CONSTRAINT_ERROR
1	1 + x
2	1 + 2x + x^2
16	Expansion
31	Expansion
32	Expansion
33	CONSTRAINT_ERROR

Table 5: *BVA and Limit Test Cases*

```
kuban-167 $ pascals_triangle
Input power (integer 1..32):
0
Ada-runtime: Exception CONSTRAINT_ERROR raised in /tmp_mnt/
home/staff2/ra/frans/public_html/2CS21/week10/pascalsTriangle.ada
on line 88. Rangecheck error: 0 not in range 1..32.

kuban-168 $ pascals_triangle
Input power (integer 1..32):
1
1 + x

kuban-169 $ pascals_triangle
Input power (integer 1..32):
2
1 + 2x + x^2

kuban-170 $ pascals_triangle
Input power (integer 1..32):
16
1 + 16x + 120x^2 + 560x^3 + 1820x^4 + 4368x^5 + 8008x^6 + 11440x^7 +
12870x^8 + 11440x^9 + 8008x^10 + 4368x^11 + 1820x^12 + 560x^13 +
120x^14 + 16x^15 + x^16
```

Table 6: *Black box testing result*

```
kuban-171 $ pascals_triangle
Input power (integer 1..32):
31
1 + 31x + 465x^2 + 4495x^3 + 31465x^4 + 169911x^5 + 736281x^6 +
2629575x^7 + 7888725x^8 + 20160075x^9 + 44352165x^10 + 84672315x^11 +
141120525x^12 + 206253075x^13 + 265182525x^14 + 300540195x^15 +
300540195x^16 + 265182525x^17 + 206253075x^18 + 141120525x^19 +
84672315x^20 + 44352165x^21 + 20160075x^22 + 7888725x^23 + 2629575x^24 +
736281x^25 + 169911x^26 + 31465x^27 + 4495x^28 + 465x^29 + 31x^30 + x^31

kuban-172 $ pascals_triangle
Input power (integer 1..32):
32
1 + 32x + 496x^2 + 4960x^3 + 35960x^4 + 201376x^5 + 906192x^6 +
3365856x^7 + 10518300x^8 + 28048800x^9 + 64512240x^10 + 129024480x^11 +
225792840x^12 + 347373600x^13 + 471435600x^14 + 565722720x^15 +
601080390x^16 + 565722720x^17 + 471435600x^18 + 347373600x^19 +
225792840x^20 + 129024480x^21 + 64512240x^22 + 28048800x^23 +
10518300x^24 + 3365856x^25 + 906192x^26 + 201376x^27 + 35960x^28 +
4960x^29 + 496x^30 + 32x^31 + x^32

kuban-173 $ pascals_triangle
Input power (integer 1..32):
33
Ada-runtime: Exception CONSTRAINT_ERROR raised in
/tmp_mnt/home/staff2/ra/frans/public_html/2CS21/week10/pascalsTriangle.ada
on line 88. Rangecheck error: 33 not in range 1..32.
```

Table 6 continued

4.2. White box testing

Path Testing: We should test each path through the code. The control flow diagram presented in Table 4 indicates 3 paths through the code (if we include the recursion). An appropriate pair of test cases is given in Table 7, both of which have already been run as part of black box testing discussed above.

TEST CASE POWER_N	EXPECTED RESULT OUTPUT
1	1 + x
2	1 + 2x + x^2

Table 7: *Path Test Cases*

Loop Testing: We have three loops in the code (including the recursion). Loop testing theory dictates the following (where M is the maximum number of allowable passes):

1) Skip loop entirely
2) Only one pass through the loop
3) Two passes through the loop

4) M passes through the loop where M < N
5) N-1, N, N+1 passes through the loop

Test cases for the recursion, `NEXT_LINE_IN_TRIANGLE` loop and `EXPANSION` loops are given in Tables 8, 9 and 10 respectively. Results (where not included in Table 6) are given in Table 11. Note that with respect to the recursion and the `NEXT_LINE_IN_TRIANGLE` loop the loop is always exercised at least once.

TEST CASE POWER_N	DESCRIPTION	EXPECTED RESULT OUTPUT
2	Only one pass	1 + 2x + x^2
3	Two passes through the loop	1 + 3x + 3x^2 + x^3
10	M passes through the loop where M < N	Expansion
31	N-1, passes through the loop	Expansion
32	N passes through the loop	Expansion
33	N+1 passes through the loop	CONSTRAINT_ERROR

Table 8: *Loop Test Cases for Recursion*

TEST CASE POWER_N	DESCRIPTION	EXPECTED RESULT OUTPUT
2	Only one pass	1 + 2x + x^2
3	Two passes through the loop	1 + 3x + 3x^2 + x^3
10	M passes through the loop where M < N	Expansion
30	N-1, passes through the loop	Expansion
31	N passes through the loop	Expansion
32	N+1 passes through the loop	Expansion

Table 9: *Loop Test Cases for NEXT_LINE_IN_TRIANGLE function*

TEST CASE POWER_N	DESCRIPTION	EXPECTED RESULT OUTPUT
2	Skip loop entirely	1 + 2x + x^2
3	Only one pass	1 + 3x + 3x^2 + x^3
4	Two passes through the loop	1 + 4x + 6x^2 + 4x^3 + x^4
10	M passes through the loop where M < N	Expansion
30	N-1, passes through the loop	Expansion
31	N passes through the loop	Expansion
32	N+1 passes through the loop	Expansion

Table 10: *Loop Test Cases for EXPANSION procedure*

```
kuban-210 $ pascals_triangle
Input power (integer 1..32):
3
1 + 3x + 3x^2 + x^3

kuban-211 $ pascals_triangle
Input power (integer 1..32):
4
1 + 4x + 6x^2 + 4x^3 + x^4

kuban-212 $ pascals_triangle
Input power (integer 1..32):
10
1 + 10x + 45x^2 + 120x^3 + 210x^4 + 252x^5 + 210x^6 + 120x^7 + 45x^8 +
10x^9 + x^10

kuban-213 $ pascals_triangle
Input power (integer 1..32):
30
1 + 30x + 435x^2 + 4060x^3 + 27405x^4 + 142506x^5 + 593775x^6 +
2035800x^7 + 5852925x^8 + 14307150x^9 + 30045015x^10 + 54627300x^11 +
86493225x^12 + 119759850x^13 + 145422675x^14 + 155117520x^15 +
145422675x^16 + 119759850x^17 + 86493225x^18 + 54627300x^19 + 30045015x^20 +
14307150x^21 + 5852925x^22 + 2035800x^23 + 593775x^24 + 142506x^25 +
27405x^26  + 4060x^27 + 435x^28 + 30x^29 + x^30
```

Table 11: White box testing result

4.3. Data validation testing

Resyults of data validation testing are given in Table 12.

```
kuban-185 $ pascals_triangle
Input power (integer 1..32):
frans
Ada-runtime: Exception DATA_ERROR propagated out of main.

kuban-186 $ rans
ksh: rans:  not found

kuban-187 $ pascals_triangle
Input power (integer 1..32):
3.4
1 + 3x + 3x^2 + x^3

kuban-188 $ 4
ksh: 4:  not found
```

Table 12: Data valiadation testing result