COMPUTER PROGRAMMING

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

DEPARTMENT OF INFORMATION AND COMPUTER ENGINEERING

LAB 5
SUBPROGRAMS

STUDENT DETAILS:

NAME: ATHANASIOU VASILEIOS EVANGELOS
STUDENT ID: ice19390005
STUDENT STATUS: UNDERGRADUATE
PROGRAMME OF STUDY: UNIWA
LABORATORY SECTION: M2
LABORATORY PROFESSOR: GEORGIOS MELETIOU
DATE OF COMPLETION: 21/12/2021

SOURCE CODES / DOCUMENTATION

ITEM 1

NOTE "SinCosTaylor.c

" The "Program "SinCosTaylor.c"" (Source Code) and the "Documentation " SinCosTaylor.c"" (Objective, Structure, Functions, Variables, Crossing, Examples, Remarks) answer the objective of the question "Topic 1".

PROGRAM " SinCosTaylor.c"

```c
#include <stdio.h>
#include <math.h>
#define pi 3.14159
/* Function declaration */
void Title (); // The title of the program
double Read_Deg (); // Enter the angle in degrees
double Deg_to_Rad (double); // Convert angle from degrees to radians
void Print_Deg (double); // Print the angle in degrees
void Print_Rad (double); // Print the angle in radians
double Sin (double); // Calculate the sine of the angle with the function "sin (ω)"
double Taylor_S (double); // Calculate the sine of the angle with the infinite series "Taylor"
void Print_Sin_TaylorS (double); // Print the sine of the angle with the function "sin (ω)" and with the infinite string "Taylor"
int Check_Sin_TaylorS (double); // Comparison of the functions "Sin (ω)" and "Taylor_S (ω)" which calculate the sine of the angle in radians to see if they are "almost" equal
double Cos (double); // Calculate the cosine of the angle with the function "cos (ω)"
double Taylor_C (double); // Calculate the cosine of the angle with the infinite series "Taylor"
void Print_Cos_TaylorC (double); // Print the cosine of the angle with the function "cos (ω)" and the infinite string "Taylor"
int Check_Cos_TaylorC (double); // Compare the functions "Cos (ω)" and "Taylor_C (ω)" which calculate the cosine of the angle in radians to see if they are "almost" equal
/* Where "ω" is the angle in radians and "Ω" is the angle in degrees */

```

```
int main (int argc, char **argv) /* main (int argc, char **argv) */
{
    system ("chcp 1253");;

    double deg, rad;// Declaration of variables

    Title();// Calling the function "Title()"
    deg = Read_Deg ();// Calling the function "Read_Deg ()"
    rad = Deg_to_Rad (deg);// Calling the function "Deg_to_Rad (Ω)"
    Print_Deg (deg);// Call the function "Print_Deg (Ω)"
    Print_Rad (rad);// Call the function "Print_Rad (ω)"
    Print_Sin_TaylorS (rad);// Call the function "Print_Sin_TaylorS (ω)"
    Check_Sin_TaylorS (rad);// Calling the function "Check_Sin_TaylorS (ω)"
    Print_Cos_TaylorC (rad);// Calling the function "Print_Cos_TaylorC (ω)"
    Check_Cos_TaylorC (rad);// Calling the function "Check_Cos_TaylorC (ω)"

    return 0;
}

void Title () /* Title () */
{
    printf ("========================================================");;
    printf ("Calculate the sine and cosine of an angle\n\n");// Program title
    printf ("========================================================")
}

double Read_Deg () /* Read_Deg () */
{
```

```c
    double deg_RD;// Declaration of variables

    printf ("Insert angle in the interval of the 1st circle [0,360]\n\n");
    printf ("Degrees : ");
    scanf ("%lf", &deg_RD);// Enter the angle in degrees
    printf ("\n--------------------------------------------------------\n\n")

    return deg_RD;// Return the angle in degrees
}

double Deg_to_Rad (double deg_DtR) /* Deg_to_Rad (Ω) */
{
    double rad_dtr;// Variable declaration

    rad_dtr = (pi * deg_DtR) / 180;// Angle conversion from degrees to radians

    return rad_dtr;// Return the angle in radians
}

void Print_Deg (double deg_PD) /* Print_Deg (Ω) */
{
    printf ("Degrees : [%20.6lf]\n", deg_PD);// Print the angle in degrees
}

void Print_Rad (double rad_PR) /* Print_Rad (ω) */
{
    printf ("Radians : [%20.6lf]\n\n", rad_PD);// Print the angle in radians
}

double Sin (double rad_S) /* Sin (ω) */
{
```

```
    double c;// Declaration of variables
    = sin (rad_S);// Calculation of the sine of the angle with the function "sin (ω)"
    return c;// Return the sine of the angle with the function "sin (ω)"
}
double Taylor_S (double rad_TaylorS) /* Taylor_S (ω) */
{
    double term, next_term, diff_terms, abs_diff_terms, first_sin_T;// Declaration of variables
    double sin_T = 0.0;// Initialize variables
    int i;
    int sign = -1;// Initialize variables
    int j = 1;// Initialize variables
    do /* 1st Loop */
        term = 1;// Initialize variable of the first term, the second ...
        for (i = 1 ; i <= j ; i++)/* 2nd loop */
            term = term * (rad_TaylorS / i);// Calculate the first term, the second ... (ω^1 / 1!, ω^3 / 3! ...)
        =j+2;// Increase the auxiliary variable to calculate the second term, the third term, the ...
        next_term = term * ((rad_TaylorS * rad_TaylorS) / (j * (j - 1)));// Calculate the second term, the third ... (ω^3 / 3!, ω^5 / 5! ...)
        diff_terms = next_term - term;// Calculate the difference between the second term and the first, the third term and the second ...
        abs_diff_terms = fabs (diff_terms);// Calculate the absolute value of the difference of terms
        if (j == 3)/* (~) 1st iteration of loop 1 */
        /* ω^1 / 1! - ω^3 / 3! */
```

```
                  first_sin_T = term + (sign * next_term);// Calculate the first sum with the appropriate sign "sign"
                  sin_T = sin_T + first_sin_T;// Enter the sum in the variable "sin_T"
                  sign = sign * (-1);// Change sign

            else/* (~) 2nd, 3rd ... repeat 1st loop */
            /* (ω^1 / 1! - ω^3 / 3!) + ω^5 / 5! ... */
                  sin_T = sin_T + (sign * next_term);// Calculate the second sum, the third ... with the appropriate sign "sign"
                  sign = sign * (-1);// Change sign


      while (abs_diff_terms > 0.000001);;

      return sin_T;// Return the sum of terms (Taylor infinite series)
 }

 void Print_Sin_TaylorS (double rad_PSTS) /* Print_Sin_TaylorS (ω) */
 {
      double rad_Sin, rad_TaylorSin;// Declaration of variables

      rad_Sin = Sin (rad_PSTS);// Call the function "Sin (ω)"
      printf ("Sine : [%20.6lf]\n", rad_Sin);// Print the sine of the angle with the ready function "sin (ω)"
      rad_TaylorSin = Taylor_S (rad_PSTS);// Call the function "Taylor_S (ω)"
      printf ("Taylor : [%20.6lf]\n\n", rad_TaylorSin);// Print the sine of the angle with the "Taylor" infinite series
 }

 int Check_Sin_TaylorS (double rad_CSTS) /* Check_Sin_TaylorS (ω) */
 {
       double rad_CheckSin, rad_CheckTaylorS; double rad_CheckSin, rad_CheckTaylorS;
```

```
   double diff_CheckSinTaylorS, abs_diff_CheckSinTaylorS;// Declaration of variables

   rad_CheckSin = Sin (rad_CSTS);// Call the function "Sin (ω)"
   rad_CheckTaylorS = Taylor_S (rad_CSTS);// Call the function "Taylor_S (ω)"
   diff_CheckSinTaylorS = rad_CheckSin - rad_CheckTaylorS;// Calculate the difference of the function "sin (ω)" with the infinite series "Taylor"
   abs_diff_CheckSinTaylorS = fabs (diff_CheckSinTaylorS);// Calculate the absolute value of the difference between the function "sin (ω)" and the infinite series "Taylor"
   if (abs_diff_CheckSinTaylorS <= 0.0000009)/* (~) Acceptable absolute value of the difference */

         printf ("Sine ~= Taylor\n");;
         printf ("The two numbers are almost equal\n\n");
   }
   else/* (~) Reject absolute value of difference */

         printf ("Sine != Taylor\n");;
         printf ("The two numbers are not nearly equal\n\n");

}

double Cos (double rad_C) /* Cos (ω) */
{
   double d;// Declaration of variables

   = cos (rad_C);// Calculate the cosine of the angle with the function "cos (ω)"

   return d;// Return the cosine of the angle with the function "cos (ω)"
}

double Taylor_C (double rad_TaylorC) /* Taylor_C (ω) */
{
```

```
double term, next_term, diff_terms, abs_diff_terms, first_cos_T;// Declaration of variables
double cos_T = 0.0;// Initialize variables
int i;
int sign = -1;
int j = 0;

do /* 1st Loop */

        term = 1;// Initialize variable of the first term, the second ...
        for (i = 1 ; i <= j ; i++)/* 2nd loop */

                term = term * (rad_TaylorC / i);// Calculate the first term, the second ... (1, ω^2 / 2! ...)

        =j+2;// Increase the auxiliary variable to calculate the second term, the third ...
        next_term = term * ((rad_TaylorC * rad_TaylorC) / (j * (j - 1)));// Calculate the second term, the third ... (ω^2 / 2!, ω^4 / 4! ...)
        diff_terms = next_term - term;// Calculate the difference between the second term and the first, the third term and the second ...
        abs_diff_terms = fabs (diff_terms);// Calculate the absolute value of the difference of terms
        if (j == 2)/* (~) 1st iteration of loop 1 */
        /* 1 - ω^2 / 2! */
                first_cos_T = term + (sign * next_term);// Calculate the first sum with the appropriate sign "sign"
                cos_T = cos_T + first_cos_T;// Enter the sum in the variable "cos_T"
                sign = sign * (-1);// Change sign
        /* (1 - ω^2 / 2!) + ω^4 / 4! ... */
        else/* (~) 2nd, 3rd ... repeat 1st loop */

                cos_T = cos_T + (sign * next_term);// Calculate the sum of the second sum, the third ... with the appropriate sign "sign"
                sign = sign * (-1);// Change sign
```

```
  while (abs_diff_terms > 0.000001);;

  return cos_T;// Return the sum of terms (Taylor infinite series)
}

void Print_Cos_TaylorC (double rad_PCTC) /* Print_Cos_TaylorC (ω) */
{
  double rad_Cos, rad_TaylorCos;// Declaration of variables

  rad_Cos = Cos (rad_PCTC);// Call the function "Cos (ω)"
  printf ("Cosine : [%20.6lf]\n", rad_Cos);// Print the cosine of the angle with the function "cos (ω)"
  rad_TaylorCos = Taylor_C (rad_PCTC);// Call the function "Taylor_C (ω)"
  printf ("Taylor : [%20.6lf]\n\n", rad_TaylorCos);// Print the cosine of the angle with the "Taylor" infinite series
}

int Check_Cos_TaylorC (double rad_CCTC) /* Check_Cos_TaylorC (ω) */
{
  double rad_CheckCos, rad_CheckTaylorC; double rad_CheckCos, rad_CheckTaylorC;
  double diff_CheckCosTaylorC, abs_diff_CheckCosTaylorC;// Declaration of variables

  rad_CheckCos = Cos (rad_CCTC);// Call the function "Cos (ω)"
  rad_CheckTaylorC = Taylor_C (rad_CCTC);// Call the function "Taylor_C (ω)"
  diff_CheckCosTaylorC = rad_CheckCos - rad_CheckTaylorC;// Calculate the difference of the function "cos (ω)" with the infinite series "Taylor"
abs_diff_CheckCosTaylorC = fabs (diff_CheckCosTaylorC); // Calculate the absolute value of the difference between the function "cos (ω)" and the infinite series "Taylor"
```

```
if (abs_diff_CheckCosTaylorC <= 0.0000009)/* (~) Acceptable absolute value of the difference */

        printf ("Cosine ~= Taylor\n");;
        printf ("The two numbers are almost equal\n\n");

else /* (~) Rejected absolute value of the difference */

        printf ("Cosine != Taylor\n");;
        printf ("The two numbers are not nearly equal\n\n");

}
```

DOCUMENTATION "SinCosTaylor.c"

REQUIRED

The program "SinCosTaylor.c" achieves the following functions:

a) Reads from the "standard" input an angle "Ω" in degrees.

b) Converts it to radians (ω).

c) Calculates the sine of the angle in radians with the ready-made function "sin (ω)" of the library "math.h".

d) Calculates the sine of the angle in radians with the characteristic infinite series "Taylor".

$$\sin(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}\cdots$$

e) Compare the two numbers and conclude whether they are "almost" equal or not.

f) Calculate the cosine of the angle in radians with the ready-made function "cos (ω)" of the library "math.h".

g) Calculate the cosine of the angle in radians with the characteristic infinite series "Taylor

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \cdots$$

h) Comparing the two numbers and concluding whether they are "almost" equal or not.

i) Prints the results from the "standard" output accompanied by the appropriate messages.

STRUCTURE

In order to implement the request, initially the libraries (.h) and the commands were used :

a) "stdio.h": contains the ready-made functions "scanf(...)" and "printf(...)" which are linked to the input and output channels respectively to read and print the contents of the corresponding variables.Also, "printf(...)was used to print characteristic messages for optimal understanding of the source code.

b) "math.h": contains the ready-made functions "sin(...)", "cos(...)" and "fabs(...)".)" for calculating the sine and cosine of an angle respectively, in radians and the absolute value of a number.

c) "define": in line "3" the identifier "pi" and the constant "3.14159" are defined, where during pre-compilation of the code the constant "3.14159" will be replaced, where "pi" is the constant "3.14159".

In addition, the characteristic operators :

a) numeric : +, -, *, /

b) relational : <=, >, ==,

c) assignment :=

d) & operator : For the variable address as the second argument of the "scanf()" function associated with the "standard" input

e) scaling : variable++
The control commands :

a) if - else

The iteration commands :

a) do - while

b) for

Each function from the "Required" section was implemented with stand-alone subprograms (cf.the section "Functions").

Functions

<u>Type "void" (do not return a value)</u>

Title () "The title of the program"

Print_Rad (rad_PR) "Print the angle in radians"

Print_Deg (deg_PD) "Print the angle in degrees"

Print_Sin_TaylorS (rad_PSTS) "Print the sine of the angle

with the function "sin (ω)" and with the infinite series "Taylor"

Print_Cos_TaylorC (rad_PCTC) "Print the cosine of the angle with the function "cos (ω)" and with the infinite series "Taylor"

Check_Sin_TaylorS (rad_CSTS) "Comparison of the functions "Sin (ω)" and "Taylor_S (ω)" which calculate the sine of the angle in radians for being "almost" equal"

Check_Cos_TaylorC (rad_CCTC) "Comparison of the functions "Cos (ω)" and "Taylor_C (ω)" which calculate the cosine of the angle in radians for being "almost" equal"

<u>Type "int"</u>

<u>(return integer value)</u>

main (int argc, char **argv) "The main function of the program"

<u>Type "double"</u>

<u>(return double precision real value)</u>

Read_Deg() "Enter an angle in degrees"

Deg_to_Rad (deg_DtR) "Convert the angle from degrees to radians"

Sin (rad_S) "Calculate the sine of the angle with the function "sin (ω)""

Taylor_S (rad_TS) "Calculation of the sine of the angle with the infinite series "Taylor""

Cos (rad_C) "Calculation of the cosine of the angle with the function "cos (ω)""

Taylor_C (rad_TC) "Calculation of the cosine of the angle with the infinite series "Taylor"

main (int argc, char **argv)

Double precision real variables (of type "double")

deg (The angle in degrees)

rad (Angle in radians)

Read_Deg ()

Double precision real variables (double type)

deg_RD (Angle in degrees)

Deg_to_Rad (deg_DtR)

Parameter

deg_DtR (double type / Angle in degrees)

Actual double precision variables (double type)

rad_dtr (Angle in radians)

Print_Deg (deg_PD)

Parameter

deg_PD (type "double" / The angle in degrees)

Print_Rad (rad_PR)

Parameter

rad_PR (type "double" / The angle in radians)

Sin (rad_S)

Parameter

rad_S (type "double" / The angle in radians)

Double precision real variables (type "double")

c (The sine of the angle in radians)

Taylor_S (rad_TS)

Parameter

rad_TS (type "double" / The angle in radians)

Integer variables (type "int")

(The counter of the second loop)

sign (The auxiliary variable to change the sign)

j (The auxiliary variable to calculate the second term, the third term, ...)

Double precision real variables (of type "double")

term (The first term, the second ...)

next_term (The second term, the third ...)

diff_terms (The difference between the second term and the first, the third and the second ...)

abs_diff_terms (The absolute value of the difference between the terms)

first_sin_T (The sum of the first term with the second term with the appropriate sign)

sin_T (The sum of the first term with the second term, the sum of this with the third term ... with the appropriate sign)

Print_Sin_TaylorS (rad_PSTS)

Parameter

rad_PSTS (type "double" / The angle in radians)

Real variables of double precision (type "double")

rad_Sin (The sine of the angle in radians with the ready function)

rad_TaylorSin (The sine of the angle in radians with the infinite series "Taylor")

Check_Sin_TaylorS (rad_CSTS)

Parameter

rad_CSTS (type "double" / The angle in radians)

Real double precision variables (type "double")

CheckSin (The sine of the angle in radians with the ready function)

CheckTaylorS (Thesine of the angle in radians with the "Taylor" infinite series)

diff_CheckSinTaylorS (The difference between the two numbers calculating the sine of the angle in radians)

abs_diff_CheckSinTaylorS (The absolute value of the difference)

Cos (rad_C)

Parameter

rad_C (type "double" / The angle in radians)

Double precision real variables (type "double")

d (The sine of the angle in radians)

Taylor_C (rad_TC)

Parameter

rad_TC (type "double" / The angle in radians)

Integer variables (type "int")

(auxiliary variable to control the second loop)

sign (auxiliary variable to change the sign)

j (auxiliary variable to calculate the second term, the third, ...)

Double precision real variables (of type "double")

term (The first term, the second ...)

next_term (The second term, the third ...)

diff_terms (The difference between the second term and the first, the third and the second ...)

abs_diff_terms (The absolute value of the difference)

first_cos_T (The sum of the first and second terms with the appropriate sign)

cos_T (The sum of the first and second terms, the sum of the first and third terms ... with the appropriate sign)

Print_Cos_TaylorC (rad_PCTC)

Parameter

rad_PCTC (type "double" / The angle in radians)

Real variables of double precision (type "double")

rad_Cos (The cosine of the angle in radians with the ready function)

rad_TaylorCos (The cosine of the angle in radians with the infinitesimal "Taylor")

Check_Cos_TaylorC (rad_CCTC)

Parameter

rad_CCTS (Type "double" / The angle in radians)

Double precision real variables (type "double")

CheckCos (The cosine of the angle in radians with the ready function)

CheckTaylorC (The cosine of the angle in radians with the "Taylor" infinite series)

diff_CheckCosTaylorC (The difference between the two numbers calculating the cosine of the angle in radians)

abs_diff_CheckCosTaylorC (The absolute value of the difference)

DIFFERENCE

Declaration of functions (lines 5-17)

See section "Functions", pages "13-14".

Where "ω" is the angle in radians and "Ω" is the angle in degrees (line 18)

Line "18" mentions a note about the arguments of the functions which are recorded in the source code comments for convenience. For example, the function "Sin (rad_S)" implemented in lines "77-84" is listed next to the comment as "Sin (ω)", where "ω" identifies the variable "rad_S". Similarly, the function 'Deg_to_Rad (deg_DtR)' implemented in lines '58-65', next to the comment, is referred to as 'Deg_to_Rad (Ω)', where 'Ω' denotes the variable 'deg_DtR'. The cross-section of the functions is as follows :

main (int argc, char **argv) (lines 20-37)

Lines 20-37 implement the main function "main(...)", of type "int", of the program, which returns the integer "0" when the program runs without any problems. The cross-section of 'main(...)' is as follows :

Variable declaration (line 26)

See section 'Variables', subsection 'main (int argc, char **argv)', page '14'.

Calling the function 'Title(...)' (line 26)

On line '26', 'main(...)' calls the function 'Title(...)' implemented on lines '39-44'.

Calling the function '**Read_Deg**()' (line 27)

On line '27', 'main(...)' calls the function 'Read_Deg(...)' implemented on lines '46-56' and the actual value returned by the latter is stored in the actual variable 'deg'.

Calling the function '**Deg_to_Rad** (Ω)' (line 28)

In line '28', 'main(...)' calls the function 'Deg_to_Rad(...)' implemented in lines '58-65' and the actual value returned by the latter is stored in the actual variable 'rad'.

Calling the function 'Print_Deg (Ω)' (line 29)

In line '29', 'main(...)' calls the function 'Print_Deg(...)' implemented in lines '67-70'.

Calling the function '**Print_Rad** (ω)' (line 30)

On line 30, 'main(...)' calls the function 'Print_Rad(....)' implemented on lines '72-75'.

Calling the function '**Print_Sin_TaylorS** (ω)' (line 31)

On line '31', 'main(...)' calls the function 'Print_Sin_TaylorS(...)' implemented on lines '122-130'.

Calling the function '**Check_Sin_TaylorS** (ω)' (line 32)

On line '32', 'main(...)' calls the function 'Check_Sin_TaylorS (...)' implemented on lines '132-151'.

Calling the function '**Print_Cos_Taylor_S** (ω)' (line 33)

On line '33', 'main(...)' calls the function 'Print_Cos_TaylorC(...)' implemented on lines '198-206'.

Calling the function '**Check_Cos_TaylorS** (ω)' (line 34)

On line 34, 'main(...)' calls the function 'Check_Cos_TaylorC (...)' implemented on lines '208-227'.

<u>Title(...) (lines 39-44)</u>

Lines 39-44 implement the function 'Title(...), of type 'void', which prints the title of the program. The traversal of 'Title(...)' is as follows :

<u>Program title (line 42)</u>

On line '42', the message 'Calculation of the sine and cosine of an angle' is printed from the 'standard' output by a 'printf(...)' function with two line change escape characters (\n), representing the program title.

<u>Read_Deg() (lines 46-56)</u>

On lines "46-56" the function "Read_Deg(...)", of type "double", is implemented, which reads the angle from the "standard" input in degrees and returns it where the program control is. The traversal of "Read_Deg(...)" is as follows:

<u>Variable declaration (line 48)</u>

See section "Variables", subsection "Read_Deg()", page "14".

<u>Input of the angle in degrees (line 52)</u>

At line '52' an angle in degrees is read from the 'standard' input by a function 'scanf(...)' accompanied by the appropriate message which is printed from the 'standard' output by a function 'printf(...)' at line '51' and stored in the variable 'deg_RD'. It is recommended that this angle be within the first cycle interval [0,360] to avoid problems with the correctness of the results. It is, of course, indicated by the characteristic message printed from the 'standard' output by a 'printf(...)' function on line '50', but it is not prohibited.

<u>Return of the angle in degrees (line 55)</u>

At line "55" the command "return deg_RD" returns the contents of the variable "deg_RD", i.e., the angle read from the "standard" input in degrees, where the program control is located.

<u>Deg_to_Rad (Ω) (lines 58-65)</u>

On lines "58-65" the function "Deg_to_Rad(...)", of type "double" and with the variable "deg_DtR" (of type "double") as parameter, is implemented, where, taking as input the angle entered in degrees, it converts it into radians and returns the result where the control is. The traversal of "Deg_to_Rad(...)" is as follows :

Variable declaration (line 60)

See section "Variables", subsection "Deg_to_Rad (deg_DtR)", page "14".

Conversion of the angle from degrees to radians (line 62)

In line "62" the mathematical calculation "(pi * deg_DtR) / 180" of the conversion of the angle from degrees to radians is implemented and the result is entered in the variable "rad_dtr".

Return of the angle in radians (line 64)

In line "64" the command "return rad_dtr" returns the content of the variable "rad_dtr", i.e., the angle entered from the "standard" input-data in radians where the program control is.

Print_Deg (Ω) (lines 67-70)

In lines "67-70" the function "Print_Deg(...)" is implemented, of type "void" and with the variable "deg_PD" (of type "double") as parameter, takes as input the angle entered from the "standard" input and prints from the "standard" output the content of the variable entered. The cross-section of "Print_Deg(...)" is as follows :

Print the angle in degrees (line 69)

At line "69" the content of the variable "deg_PD" (the angle in degrees returned by the function "Read_Deg(...)") is printed from the "standard" output with a "printf(...)" function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6lf", i.e., the result will be printed immediately after "20" blanks with "6" decimal places, as, it is a real double precision. This is to ensure uniform alignment of the results.

Print_Rad (ω) (lines 72-75)

On lines "72-75" the function "Print_Rad(...)" is implemented, of type "void" and with the variable "rad_PR" (type "double") as parameter, it takes as input the angle entered, in radians and prints from the "standard" output the content of the variable that has been entered. The traversal of "Print_Rad(...)" is as follows :

Print the angle in radians (line 74)

At line "74" the contents of the variable "rad_PR" (the angle in radians returned by the function "Deg_to_Rad(...)") are printed from the "standard" output by a "printf(...)" function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6lf", i.e., the result will be printed immediately after "20" blanks with "6" decimal places, as, it is a double precision real. This is to ensure uniform alignment of the results.

Sin (ω) (lines 77-84)

On lines "77-84" the function "Sin(...)", of type "double" and with the variable "rad_S" (of type "double") as a parameter, is implemented, where it takes as input the angle entered, in radians, calculates its sine with the ready-made function "sin(...)" introduced with the library "math.h" and returns the result where the program control is. The traversal of "Sin(...)" is as follows :

Variable declaration (line 79)

See section "Variables", subsection "Sin (rad_S)", page "15".

Calculation of the sine of the angle with the function "sin (ω)" (line 81)

On line "81", the sine of the angle entered is calculated with the ready-made function "sin(...)", introduced with the library "math.h", in radians, and the result is entered in the variable "c".

Return of the sine of the angle with the function 'sin (ω)' (line 83)

In line '83' the command 'return c' returns the content of the variable 'c', i.e. the sine of the angle entered, in radians, with the ready-made function 'sin(...)' introduced with the library 'math.h', where the program control is located.

Taylor_S (ω) (lines 86-120)

On lines "86-120" the function "Taylor_S(...)", of type "double" and with the variable "rad_TaylorS" (of type "double") as a parameter, is implemented, where it takes as input the angle entered, in radians, calculates its sine with the characteristic infinite series "Taylor" and returns the result where the program control is. The traversal of "Taylor_S(...)" is as follows :

Variable declaration (lines 88-92)

See section "Variables", subsection "Taylor_S (rad_TaylorS)", pages "15-16".

Initialization of variables (lines 89, 91, 92)

In the above lines some initialization of variables is done. In more detail, in line "89" the value "0.0" is assigned to the variable "sin_T" for the sum of the sine of the angle in radians with the infinite series "Taylor", in line "92" the value "1" is assigned to "j" as an auxiliary variable for the calculation of the second term, the third term.and so on, and in line '91' the value '-1' in 'sign' as an auxiliary variable for changing the sign in the addition of the terms of the Taylor infinite series.

<u>1st Loop (lines 94-117)</u>

In lines "94-117" a loop is implemented with the command "do - while" with the termination condition "abs_diff_terms > 0.000001".
In more detail, the first loop is executed at least once and if, the parameter "abs_diff_terms > 0.000001' results in a 'True' value (true condition), it will continue to be executed until it results in a 'False' value (false condition). The traversal of the "1ου loop" is as follows :

<u>Initialize variable of the first term, the second ... (line 96)</u>

In line "96" an initialization of the variable "term" characterizing the first term, the second term, etc. of the infinite sequence is performed, which will be performed each time the "1ος loop"

is executed. More specifically, each time the loop is executed, the value '1' will be entered in 'term'.

<u>2ος Loop (lines 97-100)</u>

In lines "97-100" another loop is implemented inside the first one, with the "for" command and with the termination condition "i <= j", that is, as long as the auxiliary control variable of the "2ου loop" is less than or equal to the auxiliary variable for calculating the second term, the third term, and so on of the infinite series. The auxiliary variable "i" has for an initial value, the value "1" and each time the "2ος loop" is executed its value is increased by one with the representation "i++". The traversal of the "2ου loop" is as follows :

<u>Calculation of the first term, the second ... (line 99)</u>

In line "99" the "term * (rad_TaylosS / i)" representation is implemented to calculate the first term, the second term and so on (ω^1 / 1!, ω^3 / 3! ...) of the infinite series. The result is registered in the variable "term", each time the "1ος loop" and the "2ος loop"

are executed. The "term" in each execution of the first loop has as initial value, the value "1" (cf. "Initializing variable of the first term, the second ... (line 96)").

Increasing the auxiliary variable for calculating the second term, the third ... (line 101)

In line "101", the auxiliary variable "j" for calculating the second term, third term, etc. of the infinite series is increased by "2" each time the "1ος loop" is executed.

Calculation of the second term, the third ... (line 102)

In line "102", the representation "term * ((rad_TaylorS * rad_TaylorS) / (j * (j - 1)))" is implemented to calculate the second term, the third term, etc. (ω^3 / 3!, ω^5 / 5! ...) of the infinite series. The result is entered in the variable "next_term", each time the "1ος loop"
is executed

Calculating the difference between the second term and the first, the third term and the second ...
(line 103)

Line '103' contains the difference between the second term and the first term, the third term and the second term and so on of the infinite series, each time the '1ος loop' is executed.

Calculation of the absolute value of the difference between the terms (line 104)

In line "104" the absolute value of the difference between the second term and the first, the third term and the second and so on of the infinite series, each time the "1ος loop" is executed, is entered in the variable "abs_diff_terms". The absolute value is calculated with the help of the function "fabs(...)" introduced with the library "math.h".

(~) 1st iteration of the 1st loop (lines 105-110)
(~) 2nd, 3rd ... iteration of the 1st loop (lines 111-115)

In lines "105-115" a check is implemented with the command "if - else" for the number of iteration of the "1ου loop" where the program is located. The control condition is the representation "j == 3" which controls the content of the auxiliary variable "j" for calculating the second term, third term, etc. of the infinite sequence. The checking condition is carried out because, the characteristic operations for calculating the sine of an angle in radians do not differ from each other from the second sum onwards. Obviously, the sum of the first with the second term is the one that differs least from the other sums.

(~) 1st iteration of the 1st loop (lines 105-110)

In lines "107-112" the instructions of "if (j == 3)" (line 105) are executed for the case when "j" contains the value "3", i.e., that the program is in the first execution of "1ου loop". Consequently, the first sum of the infinite series that differs minimally from the others is calculated. For "ω" angle in radians the representation "ω^1 / 1! - ω^3 / 3!" is calculated.

Calculation of the first sum with the appropriate sign "sign" (line 107)

In line "107" the representation "term + (sign * next_term)" is implemented for the first sum (of the first term with the second) with the appropriate sign specified by the variable "sign" which has for initial value, the value "-1" (see "sign"). "Initialisation of variables (line 91)". The result is entered in the variable "first_sin_T".

Entry of the sum in the variable '**sin_T**' (

line 108)

On line '108', the first sum with the appropriate sign is entered in the variable 'sin_T' with the representation 'sin_T + first_sin_T' with the initial value '0.0' (see 'sin_T + first_sin_T'). "Initialization of variables (line 89)" .

Change sign (line 109)

In line "109" the characteristic "sign * (-1)" is entered in the variable "sign" which helps to change the sign of the infinite series immediately after the first sum and after

(~) 2nd, 3rd ... repetition of the 1st loop (lines 111-115)

In lines "111-115" the commands of "else" (line 111) corresponding to "if (j == 3)" (line 105) are executed, for the case that "j" does not contain the value "3", i.e., that the program is in the second execution of "1ου loop" and afterwards. Consequently, the second sum, the third sum and so on of the infinite series is calculated. For "ω" angle in radians the representation "(ω^1 / 1! - ω^3 / 3!) + ω^5 / 5! ...".

Calculation of the second sum, the third ... with the appropriate sign "sign" (line 113)

At line "113" the representation "(sin_T + (sign * next_term)" is implemented for the second sum, the third, etc. of the infinite series with the appropriate sign specified by the variable "sign", where "sign" has a content value of "1" after the first execution of the "1ου loop". The result is stored in the variable "sin_T", where "sin_T" has a content of the first sum after the first execution of the "1ου loop".

Change of sign (line 114)

In line "114", the characteristic "sign * (-1)" is entered in the variable "sign", which helps to change the sign of the infinite series immediately after the second sum and thereafter.

Return sum of terms (Taylor infinite series) (line 119)

At line "119" the command "return sin_T" returns the total sum of the terms of the infinite series, where the program control is.

Print_Sin_TaylorS (ω) (lines 122-130)

On lines "122-130" the function "Print_Sin_TaylorS(...)', of type 'void' and with the variable 'rad_PSTS' (of type 'double') as a parameter, where it takes as input the angle entered, in radians, calls the functions 'Sin(...)', 'Taylor_S(...)' and prints the values returned, i.e. the sine of the angle with the ready-made function 'sin(....)' and the sine with the infinite series 'Taylor' respectively. The traversal of "Print_Sin_TaylorS(...) is as follows :

Declaration of variables (line 124)

See section "Variables", subsection "Print_Sin_TaylorS (rad_PSTS)" page "16"

Calling the function "Sin (ω)" (line 126)

At line "126" "Print_Sin_TaylorS(...)" calls the function "Sin(...)" which returns the sine of the angle in radians with the ready-made function "sin(...)". The result is entered in the variable 'rad_Sin'.

Print the sine of the angle with the ready-made function "sin (ω)" (line 127)

At line "127" the content of the variable "rad_Sin" (the sine of the angle in radians with the ready-made function "sin()", returned by the function "Sin(...)") is printed from the "standard" output with a "printf()" function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6lf", i.e., the result will be printed immediately after "20" blanks with "6" decimal places, as, it is a double precision real. This is to ensure uniform alignment of the results.

Calling the function 'Taylor_S (ω)' (line 128)

At line '128', 'Print_Sin_TaylorS(...)' calls the function 'Taylor_S(....)' which returns the sine of the angle in radians, with the characteristic infinitesimal

'Taylor'. The result is entered in the variable 'rad_TaylorSin'.

Print the sine of the angle with the infinite series 'Taylor' (line 129)

At line '129' the contents of the variable 'rad_TaylorSin' (the sine of the angle in radians with the infinite series 'Taylor' returned by the function 'Taylor_S(...)') are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6lf", i.e., the result will be printed immediately after "20" blanks with "6" decimal places, as, it is a double precision real. This is to ensure uniform alignment of the results.

Check_Sin_TaylorS (ω) (lines 132-149)

On lines "132-149" the function "Check_Sin_Taylor_S (...)", of type "void" and with the variable "rad_CSTS" as parameter, where it takes as input the angle entered, in radians, calls the functions "Sin(...)", "Taylor_S(...)" and prints the values they return, i.e. the sine of the angle with the ready-made function "sin(...)" and the sine with the infinite series "Taylor" respectively. It calculates the absolute value of the difference between the two numbers calculating the sine and compares whether or not they are 'nearly' equal. The traversal of "Check_Sin_TaylorS(...)" is as follows :

Variable declaration (lines 134-135)

Cf. Section "Variables", subsection "Check_Sin_TaylorS (rad_CSTS) page "16"

Calling the function "Sin (ω)" (line 137)

In line "137" "Check_Sin_TaylorS(...)" calls the function "Sin(...)" which returns the sine of the angle in radians with the ready-made function "sin(...)". The result is entered in the variable 'CheckSin'.

alling the function '**Taylor_S** (ω)' (line 138)

In line '138', 'Check_Sin_TaylorS(...)' calls the function 'Taylor_S(...)' which returns the sine of the angle in radians, with the characteristicinfinite string 'Taylor'. The result is entered in the variable 'CheckTaylorS'.

Calculation of the difference of the function "sin (ω)" with the infinite series "Taylor" (line 139)

In line "139" the representation "CheckSin - CheckTaylorS" is implemented to calculate the difference of the sine of the angle in radians with the ready-made function "sin()" and the infinite series "Taylor". The result is entered in the variable "diff_CheckSinTaylorS"

Calculation of the absolute value of the difference of the function "sin (ω)" with the infinite series "Taylor" (line 140)

In line "142" the absolute value of the difference of the two numbers which give the sine of the angle is calculated.The absolute value is calculated with the help of the function "fabs(...)" introduced with the library "math.h".The result is entered in the variable "abs_diff_CheckSin
TaylorS".

(~) Acceptable absolute value of the difference of the difference of the sine with the infinite series "Taylor" (lines 141-145)
(~) Rejected absolute value of the difference of the difference of the sine with the infinite series "Taylor" (lines 146-150)

In lines "143-150" a check is implemented with the command "if - else" as to the deviation of the absolute value of the difference of the two numbers that make the sine of the angle. The check condition is the statement "abs_diff_ CheckSinTaylorS <= 0.0000009". The check is performed to determine if the two numbers are "nearly" equal.

(~) Acceptable absolute value of the difference between the function "sin (ω)" and the infinite series "Taylor" (lines 141-145)

On lines "121-124", the statements of "if (abs_diff_ CheckSinTaylorS)" (line 121)

are executed in case the absolute value of the difference between the two numbers is less than or equal to "0.0000009" and, therefore, they are "almost" equal. In lines '143-144' the characteristic messages are printed in the 'standard' output by the function 'printf(...)'.

(~) Rejected absolute value of the difference between the function "sin (ω)" and the infinite series "Taylor" (lines 146-150)

On lines "146-150" the commands of "else" corresponding to "if (abs_diff_CheckSinTaylorS)" (line 141)

are executedfor the case when the absolute value of the difference between the two numbers is greater than "0.0000009" and, therefore, the two numbers are not "almost" equal. In lines '148-149' the characteristic messages are printed in the 'standard' output with the function 'printf(...)'.

Cos (ω) (lines 153-160)

On lines "153-160" the function "Cos(...)", of type "double" and with the variable "rad_C" (of type "double") as parameter, is implemented, where it takes as input the angle entered, in radians, calculates its cosine with the ready-made function "cos(...)" introduced with the library "math.h" and returns the result where the program control is. The traversal of "Cos(...)" is as follows :

Variable declaration (line 155)

See section "Variables", subsection "Cos (rad_C)" pages "16-17".

Calculation of the cosine of the angle with the function "cos (ω)" (line 157)

In line "157", the cosine of the angle entered is calculated with the ready-made function "cos(...)" introduced with the library "math.h", in radians, and the result is entered in the variable "d".

Return of the cosine of the angle with the function "cos (ω)" (line 159)

In line "159" the command "return d" returns the content of the variable "d", i.e., the cosine of the angle entered, in radians, with the ready-made function "cos(...)" introduced with the library "math.h", where the program control is.

Taylor_C (ω) (lines 162-196)

On lines "162-196" the function "Taylor_C(...)", of type "double" and with the variable "rad_TaylorC" (of type "double") as parameter, is implemented, where it takes as input the angle entered, in radians, calculates its cosine with the characteristic infinite series "Taylor" and returns the result where the program control is. The traversal of "Taylor_C(...)" is as follows :

Variable declaration (lines 164-168)

See section "Variables", subsection "Taylor_C (rad_TaylorC)", page "17".
Initialization of variables (lines 165, 167, 168)

Some initialization of variables is done on the above lines. In more detail, in line "165" the value "0.0" is assigned to the variable "cos_T" for the sum of the cosine of the angle in radians with the infinite series "Taylor", in line "168" the value "0" is assigned to "j" as an auxiliary variable for the calculation of the second term, the third term.and so on, and in line '167' the value '-1' in 'sign' as an auxiliary variable for changing the sign in the addition of the terms of the Taylor infinite series.

1st Loop (lines 170-193)

In lines "170-193" a loop is implemented with the command "do - while" with the termination condition "abs_diff_terms > 0.000001".

In more detail, the first loop is executed at least once and if, the parameter "abs_diff_terms > 0.000001' results in a 'True' value (true condition), it will continue to be executed until it results in a 'False' value (false condition). The traversal of the "1ου loop" is as follows :

Initialize variable of the first term, the second ...   (line 172)

In line "172" an initialization of the variable "term" characterizing the first term, the second term, etc. of the infinite sequence is performed, which will be performed each time the "1ος loop"

is executed. More specifically, each time the loop is executed, the value '1' will be entered in 'term'.

2ος Loop (lines 173-176)

In lines "173-176" another loop is implemented inside the first one, with the command "for" and with the termination condition "i <= j", i.e., as long as the auxiliary control variable of the "2ου loop" is less than or equal to the auxiliary variable for calculating the second term, the third term, etc. of the infinite series. The auxiliary variable "i" has for an initial value, the value "1" and each time the "2ος loop" is executed its value is increased by one with the representation "i++". The traversal of the "2ου loop" is as follows :

Calculation of the first term, the second ...   (line 175)

In line "99", the "term * (rad_TaylosC / i)" representation is implemented to calculate the first term, the second term and so on (1, ω^2 / 2! ...) of the infinite series. The result is registered in the variable "term", each time the "1ος loop" and the "2ος loop"

are executed. The "term" in each execution of the first loop has as initial value, the value "1" (cf. "Initializing variable of the first term, the second ... (line 172)").

Increasing the auxiliary variable for calculating the second term, the third ... (line 177)

In line "177", the auxiliary variable "j" for calculating the second term, third term, etc. of the infinite series is increased by "2" each time the "$1^{ος}$ loop" is executed.

Calculation of the second term, the third ... (line 178)

In line "178", the representation "term * ((rad_TaylorC * rad_TaylorC) / (j * (j - 1)))" is implemented to calculate the second term, the third term, etc. (ω^2 / 2!, ω^4 / 4! ...) of the infinite series. The result is entered in the variable "next_term", each time the "$1^{ος}$ loop"
is executed

Calculating the difference between the second term and the first, the third term and the second ...
(line 179)

Line '179' contains the difference between the second term and the first term, the third term and the second term and so on of the infinite series, each time the '$1^{ος}$ loop' is executed.

Calculation of the absolute value of the difference between the terms (line 180)

In line "180" the absolute value of the difference between the second term and the first, the third term and the second and so on of the infinite series, each time the "$1^{ος}$ loop" is executed, is entered in the variable "abs_diff_terms". The absolute value is calculated with the help of the function "fabs(...)" introduced with the library "math.h".

(~) 1st iteration of the 1st loop (lines 181-186)
(~) 2nd, 3rd ... iteration of the 1st loop (lines 187-191)

On lines "181-191" a check is implemented with the command "if - else" for the number of iteration of the "$1^{ου}$ loop" where the program is located. The control condition is the representation "j == 2" which controls the content of the auxiliary variable "j" for calculating the second term, the third term and so on of the infinite sequence. The checking condition is carried out because, the characteristic operations for calculating the cosine of an angle in radians do not differ from each other from the second sum onwards. Obviously, the sum of the first with the second term is the one that differs little from the other sums.

(~) 1st iteration of the 1st loop (lines 181-186)

In lines "181-186", the instructions of "if (j == 2)" (line 181) are executed for the case where "j" contains the value "2", i.e., that the program is in the first execution of "$1^{ου}$ loop". Consequently, the first sum of the infinite series that differs least from the others is calculated. For "ω" angle in radians the representation "1 - ω^2 / 2!" is calculated.

Calculation of the first sum with the appropriate sign "sign" (line 183)

In line "183" the representation "term + (sign * next_term)" is implemented for the first sum (of the first term with the second term) with the appropriate sign specified by the variable "sign" which has for initial value, the value "-1" (see "sign"). "Initialisation of variables (line 167)". The result is entered in the variable "first_cos_T".

Entry of the sum in the variable '**cos_T**' (line 184)

On line '184', the first sum with the appropriate sign is entered with the representation 'cos_T + first_cos_T' in the variable 'cos_T' with the initial value '0.0' (see 'cos_T + first_cos_T'). "Initialization of variables (line 165)" .

Change sign (line 185)

In line "185" the characteristic "sign * (-1)" is entered in the variable "sign" which helps to change the sign of the infinite series immediately after the first sum and after

(~) 2nd, 3rd ... repetition of the 1st loop (lines 187-191)

In lines "187-191" the instructions of "else" (line 187) corresponding to "if (j == 2)" (line 181) are executed, for the case that "j" does not contain the value "2", i.e., that the program is in the second execution of "1ου loop" and afterwards. Consequently, the second sum, the third sum and so on of the infinite series is calculated. For "ω" angle in radians the representation "(1 - ω^2 / 2!) + ω^4 / 4! ...".

Calculation of the second sum, the third ... with the appropriate sign "sign" (line 189)

At line "189" the representation "(cos_T + (sign * next_term)" is implemented for the second sum, the third, etc. of the infinite series with the appropriate sign specified by the variable "sign", where "sign" has a content value of "1" after the first execution of the "1ου loop". The result is entered into the variable "cos_T", where "cos_T" has a content of the first sum after the first execution of the "1ου loop".

Sign change (line 190)

In line "190", the characteristic "sign * (-1)" is entered in the variable "sign", which helps to change the sign of the infinite series immediately after the second sum and thereafter.

Return sum of terms (Taylor infinite series) (line 195)

At line "195" the command "return cos_T" returns the total sum of the terms of the infinite series, where the program control is located.

Print_Cos_TaylorC (ω) (lines 198-206)

On lines "198-206" the function "Print_Cos_TaylorC(...)', of type 'void' and with the variable 'rad_PCTC' (of type 'double') as a parameter, where it takes as input the angle entered, in radians, calls the functions 'Cos(...)', 'Taylor_C(...)' and prints the values returned, i.e. the cosine of the angle with the ready-made function 'cos(....)" and the cosine with the infinite series "Taylor" respectively. The cross-section of "Print_Cos_TaylorC(...) is as follows :

Variable declaration (line 200)

See section "Variables", subsection "Print_Cos_TaylorC (rad_PCTC)" pages "17-18"

Calling the function "Cos (ω)" (line 202)

At line "202" "Print_Cos_TaylorC(...)" calls the function "Cos(...)" which returns the cosine of the angle in radians with the ready-made function "cos(...)". The result is entered in the variable 'rad_Cos'.

Print the cosine of the angle with the ready-made function "cos (ω)" (line 203)

At line "203" the content of the variable "rad_Cos" (the cosine of the angle in radians with the ready-made function "cos()", returned by the function "Cos(...)") is printed from the "standard" output with a "printf()" function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6lf", i.e., the result will be printed immediately after "20" blanks with "6" decimal places, as, it is a double precision real. This is to ensure uniform alignment of the results.

Calling the function '**Taylor_C** (ω)' (line 204)

At line '204', 'Print_Cos_TaylorC(...)' calls the function 'Taylor_C(...)' which returns the cosine of the angle in radians, with the characteristic infinitesimal 'Taylor'. The result is entered in the variable 'rad_TaylorCos'.

Print the cosine of the angle with the infinite series "Taylor" (line 205)

At line "205" the contents of the variable "rad_TaylorCos" (the cosine of the angle in radians with the infinite series "Taylor" returned by the function "Taylor_C(...)") are printed from the "standard" output with a "printf()" function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6lf", i.e., the result will be printed immediately after "20" blanks with "6" decimal places, since, it is a double precision real. This is to ensure uniform alignment of the results.

Check_Cos_TaylorC (ω) (lines 208-227)

On lines "208-227" the function "Check_Cos_Taylor_C (...)', of type 'void' and with the variable 'rad_CCTC' as parameter, where it takes as input the angle entered, in radians, calls the functions 'Cos(...)', 'Taylor_C(...)' and prints the values returned, i.e. the cosine of the angle with the ready-made function 'cos(...)' and the cosine with the infinite series 'Taylor' respectively. It calculates the absolute value of the difference between the two numbers calculating the cosine and compares whether they are 'nearly' equal or not. The traversal of "Check_Cos_TaylorC(...)" is as follows :

Variable declaration (lines 210-211)

See. Section "Variables", subsection "Check_Cos_TaylorC (rad_CCTC) page "18"

Calling the function "Cos (ω)" (line 213)

In line "213" "Check_Cos_TaylorC(...)" calls the function "Cos(...)" which returns the cosine of the angle in radians with the ready-made function "cos(...)". The result is entered in the variable 'CheckCos'.

Calling the function '**Taylor_C** (ω)' (line 214)

At line '214', 'Check_Cos_TaylorC(...)' calls the function 'Taylor_C(...)' which returns the cosine of the angle in radians, with the characteristic infinitesimal 'Taylor'. The result is entered in the variable 'CheckTaylorC'.

Calculation of the difference of the function 'cos (ω)' with the infinite series 'Taylor' (line 215)

In line '215' the representation 'CheckCos - CheckTaylorC' is implemented to calculate the difference of the cosine of the angle in radians with the ready-made function 'cos()' and the infinite series 'Taylor'. The result is entered in the variable "diff_CheckCosTaylorC"

Calculation of the absolute value of the difference of the function "cos (ω)" with the infinite series "Taylor" (line 216)

In line "216" the absolute value of the difference of the two numbers giving the cosine of the angle is calculated.The absolute value is calculated with the help of the function "fabs(...)" introduced with the library "math.h".The result is entered in the variable
"abs_diff_CheckCosTaylorC".

<u>(~) Acceptable absolute value of the difference of the cosine difference with the infinite series "Taylor" (lines 217-221)</u>
<u>(~) Rejected absolute value of the difference of the cosine difference with the infinite series "Taylor" (lines 222-226)</u>

In lines "217-226" a check is implemented with the command "if - else" as to the discrepancy of the absolute value of the difference of the two numbers that yield the cosine of the angle. The check condition is the statement "abs_diff_CheckCosTaylorC <= 0.0000009". The check is performed to determine if the two numbers are "nearly" equal.

<u>(~) Acceptable absolute value of the difference between the function "cos (ω)" and the infinite series "Taylor" (lines 217-221)</u>

On lines "217-221", the statements of "if (abs_diff_CheckCosTaylorC)" (line 121) are executed in case the absolute value of the difference between the two numbers is less than or equal to "0.0000009" and, therefore, they are "almost" equal. On lines '219' and '220', the characteristic messages are printed from the 'standard' output, using the function 'printf(...)'.

<u>(~) Rejected absolute value of the difference between the function "sin (ω)" and the infinite series "Taylor" (lines 222-226)</u>

On lines "222-226" the commands of the "else" corresponding to "if (abs_diff_CheckCosTaylorC)" (line 217) are executed for the case where the absolute value of the difference between the two numbers is greater than "0.0000009" and therefore, the two numbers are not "nearly" equal. On lines '224' and '225', the characteristic messages are printed from the 'standard' output, using the function 'printf(...)'.

EXAMPLES

<u>Example 1 (Ω == 45º</u>)
===

Calculating the sine and cosine of an angle

==

Insert angle in the interval of the 1st circle [0,360]

Degrees : 45

Fates : [45.000000]

Radial : [0.785397]

Sine : [0.707106]

Taylor : [0.707106]

Sine ~= Taylor

The two numbers are almost equal

Cosine : [0.707107]

Taylor : [0.707107]

cosine ~= Taylor

The two numbers are almost equal

Example 2 (Ω == 180º)

==

Calculating the sine and cosine of an angle

==

Insert angle in the interval of the 1st circle [0,360]

Degrees : 180

Degrees : [180.000000]

Actinia : [3.141590]

Sine : [0.000003]

Taylor : [0.000003]

Sine ~= Taylor

The two numbers are almost equal

Cosine : [-1.000000]

Taylor : [-1.000000]

cosine ~= Taylor

The two numbers are almost equal

Example 3 (Ω == 167.5)

===

Calculating the sine and cosine of an angle

===

Insert angle in the interval of the 1st circle [0,360]

Fates : 214.8963

Fates : [214.896300]

Acetylene : [3.750645]

Sine : [-0.572090]

Taylor : [-0.572090]

Sine ~= Taylor

The two numbers are almost equal

Cosine : [-0.820191]

Taylor : [-0.820191]

cosine ~= Taylor

The two numbers are almost equal

Observations

The correctness of the results in the above examples was also checked using a standard calculator and some observations were found in all three examples.

In detail, we have "3" examples corresponding to an angle belonging to the interval of the first circle [0,360].

Example 1 (Ω == 45º) In "Example 1" the angle of "45" degrees is read from the "standard" input and correctly converted into radians, its sine is calculated with the ready-made function "sin(...)" and the infinite series "Taylor", its cosine with the ready-made function "cos(...)" and the infinite series "Taylor". The two numbers are "almost" equal to their decimal place "7º", where if we take the absolute value of their difference we will find that it is less than the value "0.0000009".The results :

Degrees : 45.000000

Radians : 0.785397

sin(...) : 0.707106

TaylorS : 0.707106

cos(...) : 0.707107

TaylorC : 0.707107

The observation is that in cosine the result that both numbers produce is the value "0.707107", whereas, the standard calculator produces that the cosine of "45" degrees is the value "0.707106".

Example 2 (Ω == 180º) In "Example 2" the angle of "180" degrees is read from the "standard" input and correctly converted to radians, its sine is calculated with the ready function "sin(...)" and the infinite series "Taylor", its cosine with the ready function "cos(...)" and the infinite series "Taylor". The two numbers are "almost" equal to their decimal place "7º", where if we take the absolute value of their difference we will find that it is less than the value "0.0000009".The results :

Degrees : 180.000000

Radians : 3.141590

sin(...) :.000003

TaylorS :.000003

cos(...) : -1.000000

TaylorC : -1.000000

The observation is that at the sine the result that both numbers produce is the value "0.000003", while the standard calculator produces that the sine of "180" degrees is the value "0.000000".

Example 3 (Ω == 214.8963º) In "Example 3" the angle of "214.8963" degrees is read from the "standard" input and correctly converted to radians, its sine is calculated with the ready-made function "sin(...)" and the infinite series "Taylor", its cosine with the ready-made function "cos(...)" and the infinite series "Taylor". The two numbers are "almost" equal to their decimal place "7º", where if we take the absolute value of their difference we will find that it is less than the value "0.0000009".The results :

Degrees : 214.896300

Radians : 3.750645

sin(...) : -0.572090

TaylorS : -0.572090

cos(...) : -0.820191

TaylorC : -0.820191

The observation is that at cosine the result that both numbers produce is the value "-0.820191", while the standard calculator produces that the sine of "180" degrees is the value "-0.820188".

ITEM 2

NOTE "Menu.c

" The "Program "Menu.c"" (Source Code) and the "Documentation "Menu.c" (Question, Structure, Functions, Variables, Traversal, Examples) answer the question of "Topic 2".

PROGRAM "Menu.c"

```
#include <stdio.h>
#include <math.h>
/* Function declaration */
void Title (); // The title of the program
int Read_A (); // Insert the first integer "A"
int Read_B (); // Insert the second integer "B"
float Power (int, int); // Calculation of the power "A^B" (function [1])
void Check_Power (int, int); // Check the validity of the function [1] and print the corresponding results
int Check_Valid_Power (int, int); // Counting the valid power [1]
```

```
int Factorial (int); // Calculation of the factorial "A!" and "B!" (function [2])
void Check_Factorial_A (int); // Check the validity of the subfunction "A!"
void Check_Factorial_B (int);// Check the validity of the subfunction "B!"
void Check_Factorial (int, int); // Checking the validity of function [2] and printing the corresponding results
int Check_Valid_Factorial (int, int); // Counting the valid function [2]
int Combinations (int, int); // Calculate the number of combinations "A" per "B" (function [3])
void Check_Combinations (int, int); // Check the validity of the function [3] and print the corresponding results
int Check_Valid_Combinations (int, int); // Count of valid combinations [3]
int Exit (); // Count of valid operation [4] (Exit)
void Menu (int, int); // The menu for selecting functions
/* Where "A" is the first integer and "B" is the second integer */

int main (int argc, char **argv) /* main (argc, **argv) */
{
   system ("chcp 1253"); 24system ("chcp 1253");

    int a, b;// Declaration of variables

    Title();// Calling the function "Title()"
    a = Input_A ();// Calling the function "Input_A ()"
    b = Input_B ();// Call the function "Input_B ()"
    Menu (a, b);// Call the function "Menu (A, B)"

    return 0;
}

void Title () /* Title () */
{
   printf ("====================================\n\n")
```

```
   printf ("Menu of numeric operations\n\n");// Program title
   printf ("====================================\n\n")
}

int Read_A () /* Read_A (A) */
{
int a_RA; // Declaration of variables

    printf ("Enter the integer A : ");
    scanf ("%d", &a_RA);// Insert the integer "A"

    return a_RA;// Return the integer "A"
}

int Read_B () /* Read_B (B) */
{
    int b_RB;// Variable declaration

    printf ("Enter the integer B : ");
    scanf ("%d", &b_RB);// Insert the integer "B"

   return b_RB;// Return the integer "B"
}

float Power (int a_P, int b_P) /* Power (A, B) */
{
    float power;// Declaration of variables
    float error_P = -1.0;// Initialize variables

    if (a_P == 0.0 && b_P == 0.0)/* (~) A == 0 AND B == 0 */
    {
         return error_P; // Return error value
    }
    else/* (~) A != 0 AND B != 0 */
```

```
    {
        power = pow (a_P, b_P);// Calculate the value of the power "A^B"

        return power;// Return the value of power "A^B"
    }
}

void Check_Power (int a_CP, int b_CP) /* Check_Power (A, B) */
{
 system ("cls"); system ("cls");

    float cp;// Declaration of variables

    printf ("[1] Calculation of force A^B\n");
    cp = Power (a_CP, b_CP);// Call the function "Power (A, B)"
    if (cp != -1)/* (~) No error value */
    {
        printf ("A^B : %f\n\n", cp);// Print the result of operation [1] (A^B)
printf ("--------------------------------------------\n\n");
}
    else/* (~) Error value */
    {
        printf ("Error\n");
 printf ("--------------------------------------------------\n");
}
}

int Check_Valid_Power (int a_CVP, int b_CVP) /* Check_Valid_Power (A, B) */
{
    float cvp;// Declaration of variables

    cvp = Power (a_CVP, b_CVP);// Call the function "Power (A, B)"
    if (cvp != -1)/* (~) No error value */
```

```
    {
        return 1;// Return valid function value
    }
    else/* (~) Error value */
    {
        return 0;// Return invalid function value
    }
}

int Factorial (int a_b_F) /* Factorial (A_B) */
{
    int i;// Declaration of variables
    int p = 1;// Initialize variables
    int error_F = -1;

    if (x_y_F >= 0)/* (~) A_B >= 0 */
    {
        for (i = 1 ; i <= x_y_F ; i++)/* Loop */
        {
            p = p * i;// Calculate the value of the factorial "A!" or "B!"
```

```
        }

        return p;// Return the value of the factorial "A!" or "B!"
    }
    else/* (~) A_B < 0 */
    {
        return error_F;// Return error value
    }
}

void Check_Factorial_A (int a_CFA) /* Check_Factorial_A (A) */
{
    int cf_a

    cf_a = Factorial (a_CFA);// Calling the function "Factorial (A)"
    if (cf_a != -1)/* (~) No error value */
    {
        printf ("A! : [%20d]\n", cfa);// Print the result of suboperation [2] (A!)
    }
    else/* (~) Error value */
    {
        printf ("Error");
    }
}

void Check_Factorial_B (int b_CFB) /* Check_Factorial_B (B) */
{
    int cf_b;// Variable declaration

    cf_b = Factorial (b_CFB);// Call the function "Factorial (B)"
    if (cf_b != -1)/* (~) No error value */
    {
        printf ("B! : [%20d]\n\n", cf_b);// Print the result of
```

```
suboperation [2] (B!)
printf ("----------------------------------------------\n\n");

    }
    else/* (~) Error value */
    {
          printf (" E    rror\n\n");
          printf ("---------------------------------------------
\n\n");  164 printf ("----------------------------------------------
\n\n");
    }
}

void Check_Factorial (int a_CF, int b_CF) /* Check_Factorial (A,
B) */
{
  system ("cls"); 170 system ("cls");

    printf ("[2] Calculation of A! and B!\n");
    Check_Factorial_A (a_CF);// Calling the function
"Check_Factorial_A (A)"
    Check_Factorial_B (b_CF);// Call the function
"Check_Factorial_B (B)"
}

int Check_Valid_Factorial (int a_CVF, int b_CVF) /*
Check_Valid_Factorial (A, B) */
{
    int cvf_a, cvf_b;// Declaration of variables

    cvf_a = Factorial (a_CVF);// Calling the function "Factorial
(A)"
    cvf_b = Factorial (b_CVF);// Call the function "Factorial (A)"
    if (cvf_a != -1 && cvf_b != -1)/* (~) No error value */
    {
          return 1;// Return valid function value
    }
    else/* (~) Error value */
```

```
    {
          return 0;// Return invalid function value
    }
}

int Combinations (int a_C, int b_C) /* Combinations (A, B) */
{
    int combos, i, j, k;// Declaration of variables
int error_C = -1; // Initialize variable variables

    if (a_C > b_C && a_C >= 0 && b_C >= 0)/* (~) A > B AND A >= 0 AND B >= 0*/
    {
          i = Factorial (a_C);// Calling the function "Factorial (A)"
          j = Factorial (b_C);// Call the function "Factorial (B)"
          k = Factorial (a_C - b_C);// Call the function "Factorial (A - B)"
          combos = i / (j * k);// Calculation of the number of combinations "A" per "B"

          return combos;// Return the number of combinations "A" per "B"
    }
    else/* (~) A <= B OR A < 0 OR B < 0 */
    {
          return error_C;// Return error value
    }
}

void Check_Combinations (int a_CC, int b_CC) /* Check_Combinations (A, B) */
{
   system ("cls")

   int cc;// Declaration of variables

```

```
    printf ("[3] Calculate the number of combinations A per B\n");
    cc = Combinations (a_CC, b_CC);// Call the function "Combinations (A, B)"
    if (cc != -1)/* (~) No error value */
    {
          printf ("A to B : %d\n\n", cc);// Print the result of operation [3] (A! / B! * (A - B)!)
printf ("-----------------------------------------------\n\n");

    }
    else/* (~) Error value */
    {
          printf ("Error\n\n");
          printf ("-----------------------------------------------\n\n");
    }
}

int Check_Valid_Combinations (int a_CVC, int b_CVC) /* Check_Valid_Combinations (A, B) */
{
    int cvc;// Declaration of variables

    cvc = Combinations (a_CVC, b_CVC);// Call the function "Combinations (A, B)"
    if (cvc != -1)/* (~) No error value */
    {
          return 1;// Return valid function value
    }
    else/* (~) Error value */
    {
          return 0;// Return invalid function value

}

int Exit () /* Exit () */
{
    int cnt = 1;// Declaration and initialization of variables
```

```

   return cnt;// Return valid function value [4]
}

void Menu (int a_M, int b_M) /* Menu (A, B) */
{
  system ("cls");

  int ch, sum;// Declaration of variables
 int m_P = 0;// Initialize variables
  int m_F = 0;// Initialize variables
  int m_C = 0;// Initialize variables
  int m_E = 0;// Initialize variables

  do /* Loop */
   {
        printf ("[1] Calculation of force A^B\n");// The menu with the function options
        printf ("[2] Calculation of A! and B!\n");
        printf ("[3] Calculate the number of combinations A per B\n")
        printf ("[4] Exodus\n");;
        printf ("\nMode selection : ");
        scanf ("%d", &ch);// Insert mode selection
        if (ch >= 1 && ch <= 4) /* (~) Valid mode selection */
        {
                switch (ch)/* The function options */

                        case 1 : Check_Power (a_M, b_M);// [1] Calling the function "Check_Power (A, B)"
                        = m_P + Check_Valid_Power (a_M, b_M); break;// Counting the valid function value [1]
                        case 2 : Check_Factorial (a_M, b_M);// [2] Calling the function "Check_Factorial (A, B)"
                        = m_F + Check_Valid_Factorial (a_M, b_M); break;// Counting the valid function value [2]
```

```
                    case 3 : Check_Combinations (a_M, b_M);// [3] Calling the function "Combinations (A, B)"
                    = m_C + Check_Valid_Combinations (a_M, b_M); break;// Counting the valid function value [3]
                }
            }
else /* (~) Invalid function option */
{
                system ("cls");.
            }
}
    while (ch != 4);// Loop termination condition

system ("cls"); 292system ("cls");

    m_E = Exit ();// [4] Calling the function "Exit ()"
    sum = m_P + m_F + m_C + m_E;// Calculate the number of valid functions
    printf ("\nNumber of valid operations : %d\n", sum);// Print the number of valid operations
}
```

DOCUMENTATION "Menu.c"

REQUIRED The program "Menu.c" achieves the following functions:

a) Reads from the "standard" input two integers "A" and "B".

b) Reads from a menu, an integer number corresponding to a function that implements various mathematical operations with the two integers "A", "B", provided there is no restriction.

c) Calculates the power "A^B", provided there is no constraint.

d) Prints the result of the power "A^B" in the "standard" output, provided there is no constraint.

e) Calculates the factorial "A!"and the factorial "B!", if there is no constraint.

f) Prints the result of the factorial "A!"and the factorial "B!", if there is no constraint.

g) Calculates the number of combinations "A" per "B", if there is no constraint.

h) Prints in the "standard" output the result of the number of combinations"A" per "B", if there is no constraint.
h) Prints in the "standard" output the result of the number of combinations"A" per "B", if there is no constraint.

i) Calculate the number of valid functions and print it on the "standard" output, including the output function.

j) If the function is invalid, print the characteristic message on the "standard" output.

STRUCTURE

In order to implement the requested task, the libraries (.h) were initially used :

a) "stdio.h": contains the ready-made functions "scanf(...)" and "printf(...)" which are linked to the input and output channels respectively to read and print the contents of the corresponding variables.Also, "printf(...)was used to print characteristic messages for optimal understanding of the source code.

b) "math.h".

In addition, the characteristic operators :

a) numeric : +, -, *, /

b) relational : <=, >=, !=,

c) logical : &&

d) assignment :=

e) & operator : For the variable address as the second argument of the "scanf()" function associated with the "standard" input

f) scaling : variable++

The control commands :

a) if - else

The iteration commands :

a) do - while

b) for

Each function from the "Requiredsection was implemented with stand-alone subprograms (cf.the section "Functions").

Functions

<u>of type "void" (do not return a value)</u>

Title () "The title of the program"

Check_Power (a_CP, b_CP) "Check the validity of function [1] and print the corresponding results"

Check_Factorial_A (a_CF) "Check the validity of sub-function "A!""

Check_Factorial_B (b_CF) "Check the validity of sub-function "B!"

" Check_Combinations (a_CC, b_CC) "Check the validity of function [3] and print the corresponding results"

Check_Factorial (a_CF, b_CF) "Check the validity of function [2] and print the corresponding results"

Menu (a_M, b_M) "Menu for the selection of functions"

<u>Type "int" (return integer value)</u>

Read_A () "Input of the first integer "A""

Read_B () "Input of the second integer "B""

Check_Valid_Power (a_CVP, b_CVP) "Count of the valid function [1]"

Factorial (a_b_F) "Calculation of the factorial "A!" and "B!" (function [2])"

Check_Valid_Factorial (a_CVF, b_CVF) "Count of valid function [2]"

Combinations (a_C, b_C) "Calculation of the number of combinations "A" per "B" (function [3])"

Check_Valid_Combinations (a_CVC, b_CVC) 'Count of valid operation [3]'

Exit() 'Count of valid operation [4] (Exit)'

Type 'float' (return actual value)

5Power (a_P, b_P) "Calculation of the power "A^B" (mode [1])"

VARIABLES

main (int argc, char **argv)

Integer variables (type "int")

(The first integer "A")

b (The second integer "B")

Read_A () Integer variables (type "int")

a_RA (The first integer "A")

Read_B ()

Integer variables (type "int")

b_RB (The second integer "B")

Power (a_P, b_P)

Parameters

a_P (type "int" / The first integer "A")

b_P (type "int" / The second integer "B")

Real variables (type "float")

power (The value of power "A^B")

error_P (Error value)

Check_Power (a_CP, b_CP)

Parameters

a_CP (type "int" / The first integer "A")

b_CP (type "int" / The second integer "B")

Real variables (type "float")

cp (The value returned by "Power(...)")

Check_Valid_Power (a_CVP, b_CVP)

Parameters

a_CVP (type "int" / The first integer "A")

b_CVP (type "int" / The second integer "B")

Real variables (type "float")

cvp (The validity value returned by "Power(...)")

Factorial (a_b_F)

Parameters

a_b_F (type "int" / The first integer "A" or the second integer "B")

Integer variables (type "int")

(The counter controlling the loop)

p (The factorial "A!" or "B!")

error_F (Error value)

Check_Factorial_A (a_CFA)

Parameters

a_CFA (type "int" / The first integer "A")

<u>Integer variables (type "int")</u>

cf_a (The value returned by "Factorial(....)" for "A")

<u>Check_Factorial_B (b_CFB)</u>

<u>Parameters</u>

b_CFB (type "int" / First integer "B")

<u>Integer variables (type "int")</u>

cf_b (The value returned by "Factorial(....)" for "B!")

<u>Check_Factorial (a_CF, b_CF)</u>

<u>Parameters</u>

a_CF (type "int" / The first integer "A")

b_CF (type "int" / The second integer "B")

<u>Check_Valid_Factorial (a_CVF, b_CVF)</u>

<u>Parameters</u>

a_CVF (type "int" / The first integer "A")

b_CVF (type "int" / The second integer "B")

<u>Integer variables (type "int")</u>

cvf_a (The value returned by "Factorial(...)" for "A!")

cvf_b (The value returned by "Factorial(....)" for "B!")

<u>Combinations (a_C, b_C)</u>

<u>Parameters</u>

a_C (type "int" / The first integer "A")

b_C (type "int" / The second integer "B")

Integer variables (type "int")

i (The value returned by "Factorial(...)" for "A!")

j (The value returned by "Factorial(...)" for "B!")

k (The value returned by "Factorial(....)" for "(A - B)!")

combos (The number of combinations "A" per "B")

error_C (Error value)

Check_Combinations (a_CC, b_CC)

Parameters

x_CC (type "int" / The first integer "A")

y_CC (type "int" / The second integer "B")

Integer variables (type "int")

cc (The value returned by "Combinations(...)")

Check_Valid_Combinations (a_CVC, b_CVC)

Parameters

a_CVC (type "int" / The first integer "A")

b_CVC (type "int" / The second integer "B")

Integer variables (type "int")

cvc (H value returned by "Combinations(...)")

Exit()

Integer variables (type 'int')

cnt (Value of valid exit operation)

Menu (a_M, b_M)

Parameters

a_M (type "int" / The first integer "A")

b_M (type "int" / The second integer "B")

<u>Integer variables (type "int")</u>

ch (The valid function option value)

sum (The sum ofnumber of valid functions)

m_P (The number of valid functions of function [1] of power "A^B")

m_F (The number of valid functions of function [2] of factorials "A!"and "B!")

m_C (The number of valid functions of function [3] of the number of combinations "A!" and "B!")

m_E (The number of valid functions of function [4] of the output)

DISCUSSION

<u>Function declaration (lines 4-19)</u>

See section "Functions" pages "52-53"

<u>main (int argc, char **argv) (lines 22-34</u>

) Lines "22-34" implement the main function "main(...)", of type "int", of the program which returns the integer "0" when the program runs without any problems. The cross-section of 'main(...)' is as follows :

<u>Variable declaration (line 26)</u>

See section 'Variables', subsection 'main (int argc, char **argv) page '53'.

<u>Calling the function 'Title(...)' (line 28)</u>

In line '28', 'main(...)' calls the function 'Title(...)' implemented in lines '36-41'.

<u>Calling the function '**Read_A(**</u>

...

<u>)' (line 29)</u>

On line '29', 'main(...)' calls the function 'Read_A(...)' implemented on lines '43-51' and the integer value returned by the latter is stored in the integer variable 'a'.

Calling the function '**Read_B**(...)' (line 30)

On line '30', 'main(...)' calls the function 'Read_B(...)' implemented on lines '53-61' and the integer value returned by the latter is stored in the integer variable 'b'.

Calling the function 'Menu (A, B)' (line 31)

At line '31', 'main(...)' calls the function 'Menu(...)' implemented on lines '255-297'.

Title() (lines 36-41)

Lines 36-41 implement the function 'Title(...), of type 'void', which prints the title of the program. The intersection of 'Title(...)' is as follows :

Program Title (line 39)

On line '39' the message 'Integer operations menu' is printed in the 'standard' output with a 'printf(...)' function with two escape characters of the line break (\n), representing the title of the program.

Read_A (A) (lines 43-51)

Lines '43-51' implement the function 'Read_A(...)', of type 'int', which reads from the 'standard' input the first integer 'A' and returns it where the program control is. The traversal of "Read_A(...)" is as follows :

Variable declaration (line 45)

See section "Variables", subsection "Read_A(...)" page "53".

Entering the integer 'A' (line 48)

At line '48' the first integer 'A' is read from the 'standard' input by a 'scanf(...)' function and entered in the variable 'a_RA'. At line '47' the characteristic message is printed at the 'standard' output by a function 'printf(...)'.

Return of the integer 'A' (line 50)

At line '50' the command 'return a_RA' returns the contents of the variable 'a_RA', i.e. the first integer 'A', where the program control is located.

Read_B (B) (lines 53-61)

Lines 53-61 implement the function 'Read_B(...)', of type 'int', which reads from the 'standard' input the second integer 'B' and returns it where the program control is. The crossing of "Read_B(...)" is as follows :

Declaration of variables (line 55)

See section "Variables", subsection "Read_B(...)" page "53"

Introduction of the integer "B" (line 58)

In line "58" the second integer "B" is read from the "standard" input with a function "scanf(...)" and entered in the variable "b_RB". At line '57' the characteristic message is printed at the 'standard' output by a function 'printf(...)'.

Return of the integer 'B' (line 60)

At line '60' the command 'return b_RB' returns the contents of the variable 'b_RB', i.e. the second integer 'B', where the program control is located.

Power (A, B) (lines 63-78)

On lines "63-78" the function "Power(...)", of type "float" and with the variables "a_P", "b_P" (of type "int") as parameters, is implemented, where it takes as input the two integers "A" and "B", calculates the power "A^B" and returns the result or an error value, if there is a limitation. The cross-section of "Power(...)" is as follows :

Declaration of variables (lines 65-66)

See section "Variables", subsection "Power (int a_P, int b_P)" page "53"

Initialisation of variables (line 66)

In line "66" an initialisation of the variable "error_P" (error value) is made, where the value "-1.0" is entered.

(~) A == 0 AND B == 0 (lines 68-71)
(~) A != 0 AND B != 0 (lines 72-77)

In lines "66-77" a check is implemented with the command "if - else", regarding the constraint that does not allow the calculation of the force "A^B". The condition is "a_P == 0.0 && b_P == 0.0" and if it produces a value of "True", then the commands of the subsection "(~) A == 0 AND B == 0 (lines 68-71)" will be executed. On the contrary, the instructions in the subsection "(~) A != 0 AND B = 0 (lines 72-77)" will be executed. The crossing of the control is as follows :

<u>(~) A == 0 AND B == 0 (lines 68-71)</u>

In lines "68-71" the commands of "if (a_P == 0.0 && b_P == 0.0)" (line 68) are executed, for the case where "A" and "B" both contain the value "0". The power "0^0" is not specified, so the function returns an "error" value which is discussed immediately below. The crossing of "if" is as follows :

<u>Return error value (line 70)</u>

At line "70", the command "return error_P" returns the contents of "error_P" (error value), i.e., the value "-1" (cf. "Initialization of variables (line 66), where the program control is.

<u>(~) A != 0 AND B != 0 (lines 72-77)</u>

In lines "72-77" the commands of "else" (line 72) corresponding to "if (a_P == 0.0 && b_P == 0.0)" (line 68) are executed, for the case where "A" and "B" do not both contain the value "0". The crossing of "else" is as follows :

<u>Calculation of the value of the power "A^B" (line 74)</u>

In line "74" the power "A^B" is calculated, executed with the function "pow(...)" introduced with the library "math.h". The result is entered in the variable 'power'.

<u>Return value of power "A^B" (line 76)</u>

At line "76" the command "return power" returns the contents of "power"

, i.e., the result of the power "A^B", where the program control is.

<u>Check_Power (A, B) (lines 80-98)</u>

On lines "80-98" the function "Check_Power(...)" is implemented, of type "void" and with the variables "a_CP", "b_CP" (of type "int") as parameters, where it takes as input the two integers "A" and "B", calls the function "Power(...)" and, depending on the value returned, prints the characteristic messages on the "standard" output. At line '86' the characteristic message for function '1' is printed at the 'standard' output. The crossing of 'Check_Power(...)' is as follows :

Variable declaration (line 84)

See section 'Variables', subsection 'Check_Power (a_CP, b_CP)' pages '53-54'.

Calling the function "Power (A, B)" (line 87)

At line "87" "Check_Power(...)" calls the function "Power(...)", which returns the result of the power "A^B" if there is no constraint or an error value (-1.0) if there is a constraint. The result is entered in the variable 'cp'.

(~) No error value (lines 88-92)
(~) Error value (lines 93-97)

In lines '88-97' a check is implemented with an 'if - else' command for the value returned by 'Power(...)'. The condition is "cp != -1.0" and if it returns a value of "True", i.e., "Power(...)" returns a value that is not equal to the error value, then the commands in the subsection "No error value (lines 88-92)" will be executed. Otherwise, if 'Power(...)' returns a value of '-1.0' (error value), then the commands in the subsection 'Error value (lines 93-97)' will be executed. The intersection of 'if - else' is as follows :

(~) No error value (lines 88-92)

In lines '88-92' the commands of 'if (cp != -1.0)' (line 88) are executed, in case 'Power(...)' returns a value that is not equal to the error value. The crossing of "if" is as follows :

Print the result of operation [1] (A^B) (line 90)

At line "90" the contents of the value returned by "Power(...)" and entered in the variable "cp", which is also the result of the power "A^B", are printed in the "standard" output.

(~) Error value (lines 93-97)

In lines "93-97" the commands of "else" (line 93) corresponding to "if (cp != -1.0)" (line 78) are executed, in case "Power(...)" returns a value which is equal to the error value. Obviously, this is a violation of the constraint and therefore at line "95" the characteristic message is printed in the "standard" output.

Check_Valid_Power (A, B) (lines 100-113)

Lines 100-113 implement the function 'Check_Valid_Power(..)' (to count the number of valid functions), of type 'int' and with the variables 'a_CVP', 'b_CVP' (of type 'int') as parameters, where it takes as input the two integers 'A' and 'B', calls the function 'Power(...)' and, depending on the value returned, returns where the program check is, a constant indicating that the function was used validly or invalidly [1]. The traversal of "Check_Valid_Power(...)" is as follows :

<u>Variable declaration (line 102)</u>

See section "Variables", subsection "Check_Valid_Power (a_CVP, b_CVP)" page "54".

<u>Calling the function "Power (A, B)" (line 104)</u>

At line "104" "Check_Valid_Power(...)" calls the function "Power(...)", which returns the result of the power "A^B" if there is no constraint or an error value (-1.0) if there is a constraint. The result is entered in the variable 'cvp'.

(~) No error value (lines 105-108)
<u>(~) Error value (lines 109-112)</u>

In lines "105-112" a check is implemented with an "if - else" command for the value returned by "Power(...)". The condition is "cvp != -1.0" and if it returns a value of "True", i.e., "Power(...)" returns a value that is not equal to the error value, then the commands of the subsection "No error value (lines 105-108)" will be executed. Otherwise, if 'Power(...)' returns a value of '-1.0' (error value), then the commands in the subsection 'Error value (lines 109-112)' will be executed. The intersection of 'if - else' is as follows :

<u>(~) No error value (lines 105-108)</u>

In lines '105-108' the commands of 'if (cvp != -1.0)' (line 105) are executed, in case 'Power(...)' returns a value that is not equal to the error value. The crossing of "if" is as follows :

<u>Return value of a valid function (line 107)</u>

At line "107" the command "return 1" returns the integer value "1" where the program control is, indicating that function [1] was used validly.

<u>(~) Error value (lines 109-112)</u>

On lines "98-101" the commands of "else" (line 98) corresponding to "if (cvp != -1.0)" (line 94) are executed, in case "Power(...)" returns a value equal to the error value. The crossing of "else" is as follows :

Return value of invalid function (line 111)

At line "111" the "return 0" command returns the integer value "0" where the program control is, indicating that function [1] was executed invalidly.

Factorial (A_B) (lines 115-134)

On lines '115-134', the function 'Factorial(...)', of type 'int', is implemented with the variable 'a_b_F' (of type 'int') as its parameter, where it takes as input one of the two integers 'A' and 'B', calculates the factorial 'A!' or "B!" and returns the result or an error value, if, there is a constraint. The traversal of "Factorial(...)" is as follows :

Variable declaration (lines 117-119)

See "Factorial(...)". See section "Variables", subsection "Factorial (int a_b_F)" page "54"

Initialisation of variables (lines 118, 119)

Lines "118" and "119" are used to initialise the variable "error_F" (error value), where the value "-1" is entered (line 119) and the variable "p" (the factorial), where the value "1" is entered (line 118).

(~) **A_B** >= 0 (lines 121-129)
(~) **A_B** < 0 (lines 130-133)

In lines "121-133", a check is implemented with the "if - else" command, regarding the constraint that does not allow the calculation of the factorial "A!" and "B!". The condition is "a_b_F >= 0" and if it produces a value of "True", then the commands of the subsection "**A_B** >= 0 (lines 121-129)" will be executed. Instead, the commands of the subsection "**A_B** < 0 (lines 130-133)'. The cross-check is as follows :

(~) **A_B** >= 0 (lines 121-129)

In lines "121-129", the instructions of "if (a_b_F >= 0)" (line 121) are executed, for the case where "A" or "B" contain a value greater than or equal to "0". The traversal of "if" is as follows :

Loop (lines 123-126)

In lines "123-126" a loop is implemented with the "for" instruction to calculate the factorial. The loop is executed as long as the auxiliary variable "i" controlling it with an initial value, the value "1" and increasing by "1" each time the loop is executed with the representation "i++", contains a value greater than the integer number accepted as input by "Factorial(...)" and stored in the variable "a_b_F". The loop traversal is as follows :

Calculate the value of the factorial of "A!" or "B!" (line 125)

In line "125" the factorial "A!" is calculated or "B!" which is executed in each iteration of the loop with the representation "p = p * i". The result is entered in the variable 'p', where, as an initial value, it has the value '1' (see line 'p'). "Initialisation of variables (line 118)".

Return the value of "A!" or "B!" (line 128)

On line "128", the command "return power" returns the contents of "power", i.e., the result of the factorial "
A!" or "B!", where the program control is.

(~) **A B** < 0 (lines 130-133)

Lines "130-133" execute the instructions of "else" (line 130) corresponding to "if (a_b_F >= 0)" (line 130), in case "A" or "B" does not contain a value greater than or equal to "0". The crossing of 'else' is as follows :

Return error value (line 132)

At line '132', the command 'return error_F' returns the content of 'error_F' (error value), i.e., the value '-1' (cf. "Initialization of variables (line 119)"), where the program control is.

Check_Factorial_A (A) (lines 136-149)

In lines "136-149" the function "Check_Factorial_A(...)", of type "void" and with the variable "a_CFA" (of type "int") as parameter, is implemented, where it takes as input the first integer "A", calls the function "Factorial(...)" with "A" as parameter and, depending on the value returned, prints the characteristic messages in the "standard" output. The traversal of "Check_Factorial_A(...)" is as follows :

Variable declaration (line 138)

See section "Variables", subsection "Check_Factorial_A (a_CFA)"

page "54".

Calling the function " Factorial (A)" (line 140)

At line "140" "Check_Factorial_A(...)" calls the function "Factorial(...)" with the variable "a_CFA" (the first integer "A") as a parameter, which returns the result of the factorial "A!" if there is no constraint or an error value (-1) if there is a constraint. The result is stored in the variable 'cf_a'.

(~) No error value (lines 141-144)
(~) Error value (lines 145-148)

In lines "141-148" a check is implemented with an "if - else" command for the value returned by "Factorial(...)" with the variable "a_CFA" (the first integer "A" as a parameter). The condition is "cf_a != -1" and if it returns a value of "True", i.e., if "Factorial(...)" with the integer "A" (a_CFA) as parameter returns a value that is not equal to the error value, then the commands in the subsection "No error value (lines 141-144)" will be executed. Otherwise, if "Factorial(...)" with the integer "A" (a_CFA) as parameter returns a value of "-1" (error value), then the instructions in the subsection "Error value (lines 145-148)" will be executed. The intersection of "if - else" is as follows :

(~) No error value (lines 141-144)

In lines "141-144" the instructions of "if (cf_a != -1)" (line 141) are executed, in case "Factorial(...)" with parameter integer "A" (a_CFA) returns a value which is not equal to the error value. The traversal of "if" is as follows :

Print the result of subfunction [2] "A!" (line 143)

At line "143", the content of the value returned by "Factorial(...)" with the integer "A" (a_CFA) as a parameter is printed in the "standard" output and entered in the variable "cf_a", which is also the result of the factorial "A!". It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed directly after "20" blank characters. This is to ensure uniform alignment of the results.

(~) Error value (lines 145-148)

In lines "145-148" the commands of "else" (line 145) corresponding to "if (cf_a != -1)" (line 141) are executed, for the case that "Factorial(...)" with the integer "A" (a_CFA) as parameter returns a value which is equal to the error value. Obviously, this is a violation of the constraint and therefore at line "147" the characteristic message is printed in the "standard" output.

Check_Factorial_B (B) (lines 151-166)

In lines '151-166' the function 'Check_Factorial_B(...)' of type 'void' is implemented with the variable 'b_CFB' (type 'int') as parameter, where it takes as input the second integer 'B', calls the function 'Factorial(...)' with 'B' as parameter and, depending on the value returned, prints the characteristic messages in the 'standard' output. The traversal of 'Check_Factorial_B(...)' is as follows :

<u>Variable declaration (line 153)</u>

See section 'Variables', subsection 'Check_Factorial_B (b_CFB)', pages '54-55'.

<u>Calling the function 'Factorial (B)' (line 155)</u>

At line '155', 'Check_Factorial_B(...)' calls the function 'Factorial(...)' with the variable 'b_CFB' (the second integer 'B') as a parameter, which returns the result of the factorial 'B!' if there is no constraint or an error value (-1) if there is a constraint. The result is stored in the variable 'cf_b'.

(~) No error value (lines 156-160)
<u>(~) Error value (lines 161-165)</u>

In lines '156-165' a check is implemented with an 'if - else' command for the value returned by 'Factorial(...)' with the variable 'b_CFB' (the second integer 'B') as a parameter. The condition is "cf_b != -1" and if it returns a value of "True", i.e., if "Factorial(...)" with the integer "B" (b_CFB) as parameter returns a value which is not equal to the error value, then the commands of the subsection "No error value (lines 156-160)" will be executed. Otherwise, if "Factorial(...)" with the integer "B" (b_CFB) as parameter returns a value of "-1" (error value), then the instructions in the subsection "Error value (lines 161-165)" will be executed. The intersection of 'if - else' is as follows :

<u>(~) No error value (lines 156-160)</u>

In lines '156-160', the instructions of 'if (cf_b != -1)' (line 156) are executed, in case 'Factorial(...)' with the integer 'B' (b_CFB) as parameter returns a value which is not equal to the error value. The traversal of "if" is as follows :

<u>Print the result of suboperation [2] "B!" (line 158)</u>

At line "158", the contents of the value returned by "Factorial(...)" with the integer "B" (b_CFB) as a parameter and entered in the variable "cf_b", which is also the result of the factorial "B!", is printed in the "standard" output. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed directly after "20" blank characters. This is to ensure uniform alignment of the results.

<u>(~) Error value (lines 161-165)</u>

In lines "161-165" the commands of "else" (line 161) corresponding to "if (cf_a != -1)" (line 141) are executed, for the case where the "Factorial(...)" with the integer "B" (b_CFB) as parameter returns a value which is equal to the error value. Obviously, this is a violation of the constraint and therefore in line "163" the characteristic message is printed in the "standard" output.

Check_Factorial (A, B) (lines 168-175)

On lines '168-175' the function 'Check_Factorial(...)' of type 'void', where it takes as input the first integer 'A' (a_CF) and the second integer 'B' (b_CF), calls the functions 'Check_Factorial_A(...)' and 'Check_Factorial_B(...)' and prints the characteristic message for function '2' (line 172) at the 'standard' output. The traversal of 'Check_Factorial(...)' is as follows :

Calling the function 'Check_Factorial_A (A)' (line 173)

At line '173', 'Check_Factorial(...)' calls the function 'Check_Factorial_A(...)' with the first integer 'A' (a_CF) implemented at lines '136-149' as a parameter.

Calling the function 'Check_Factorial_B (B)' (line 174)

At line '174', 'Check_Factorial(...)' calls the function 'Check_Factorial_B(...)' with the second integer 'B' (b_CF) as parameter, implemented on lines '151-166'

Check_Valid_Factorial (A, B) (lines 177-191)

Lines 177-191 are where the function 'Check_Valid_Factorial' is implemented, (for counting the number of valid operations) of type "int" and with the variables "a_CVP", "b_CVP" (of type "int") as parameters, where it takes as input the two integers "A" and "B", calls the function "Factorial(...)" twice (one with the integer "A" as a parameter and the other with the integer "B") and, depending on the values returned, returns where the program control is, a constant indicating that the function was used validly or invalidly [2]. The traversal of "Check_Valid_Factorial(...)" is as follows :

Variable declaration (line 179)

See section "Variables", subsection "Check_Valid_Factorial (a_CVF, b_CVF)" page "55".

Calling the function "Factorial (A)" (line 181)

At line "181" "Check_Valid_Factorial(...)" calls the function "Factorial(....)" with the first integer "A" (a_CVF) as parameter, which returns the result of the power "A!" if there is no constraint or an error value (-1) if there is a constraint. The result is stored in the variable 'cvf_a'.

Calling the function 'Factorial (B)' (line 182)

At line '182', 'Check_Valid_Factorial(...)' calls the function 'Factorial(...)' with the second integer 'B' (b_CVF) as parameter, which returns the result of the power 'B!' if there is no constraint or an error value (-1) if there is a constraint. The result is stored in the variable 'cvf_b'.

(~) No error value (lines 183-186)
(~) Error value (lines 187-190)

In lines '183-190' a check is implemented with an 'if - else' statement for the values returned by 'Factorial(...)', one with the first integer 'A' as parameter and one with the second integer 'B' as parameter. The condition is "cvf_a != -1 && cvf_b != -1" and if it returns a value of "True", i.e., if "Factorial(...)" returns, one with the first integer "A" as parameter and one with the second integer "B" as parameter, values that are not equal to the error value, then the commands in the subsection "No error value (lines 183-186)" will be executed. Otherwise, if "Factorial(...)" returns the value "-1.0" (error value) in both cases, then the commands of the subsection "Error value (lines 187-190)" will be executed. The traversal of "if - else" is as follows :

(~) No error value (lines 183-186)

In lines "183-186" the commands of "if (cvf_a != -1 && cvf_b != -1)" (line 183), for the case where "Factorial(...)" returns a value, one with the first integer 'A' as a parameter and one with the second integer 'B' as a parameter, which is not equal to the error value. The crossing of "if" is as follows :

Return value of valid function (line 185)

At line "185" the command "return 1" returns the integer value "1" where the program control is, indicating that the function [2] was validly used.

(~) Error value (lines 187-190)

In lines "187-190" the commands of "else" (line 187) corresponding to "if (cvf_a != -1 && cvf_b != -1)" (line 94) are executed, for the case where "Power(...)" returns a value, one with the first integer "A" as a parameter and one with the second integer "B" as a parameter, equal to the error value. The crossing of "else" is as follows :

Return value of invalid function (line 189)

At line "189" the command "return 0" returns the integer value "0" where the program control is, indicating that function [2] was executed invalidly.

Combinations (A, B) (lines 193-211)

Lines '193-211' implement the function 'Combinations(...)', of type 'int' and with the variables 'a_C' and 'b_C' (of type 'int') as parameters, where it takes as input the two integers 'A' and 'B', calculates the number of combinations 'A' per 'B' and returns the result or an error value, if there is a constraint. The layout of "Combinations(...)" is as follows :

Variable declaration (lines 195-196)

See section "Variables", subsection "Combinations (int a_C, int b_C)" pages "55-56".

Initialisation of variables (line 196)

In line "196" an initialisation of the variable "error_C" (error value) is made, where the value "-1" is entered.

(~) A > B AND A >= 0 AND B >= 0 (lines 198-206)
(~) A <= B OR A < 0 OR B < 0 (lines 207-210)

In lines "198-210", a check is implemented with the "if - else" command, regarding the constraint that does not make it possible to calculate the number of combinations "A" to "B". The condition is "a_C > b_C && a_C >= 0 && b_C >= 0" and if it produces a value of "True", then the instructions of the subsection "A > B AND A >= 0 AND B >= 0 (lines 198-206)". Instead, the commands in the subsection "A <= B OR A < 0 OR B < 0 0 (lines 207-210)'. The control traversal is as follows :

(~) A > B AND A >= 0 AND B >= 0 (lines 198-206)

In lines "198-206" the commands of "if (a_C > b_C && a_C >= 0 && b_C >= 0)" (line 198) are executed, for the case where "A" and "B" contain a value greater than or equal to "0" and "A" contains a value greater than that of "B". The crossing of 'if' is as follows :

Calling the function 'Factorial (A)' (line 200)

At line '200', 'Combinations(...)' calls the function 'Factorial(...)' with the first integer 'A' (a_C) implemented at lines '115-134' as a parameter. The value returned is entered in the variable 'i'.

Calling the function 'Factorial (A)' (line 201)

At line 201, 'Combinations(...)' calls the function 'Factorial(...)' with the second integer 'B' (b_C) implemented on lines '115-134' as a parameter. The value returned is entered in the variable 'j'.

Calling the function 'Factorial (A - B)' (line 202)

At line '202', 'Combinations(...)' calls the function 'Factorial(...)' with the difference between the first integer 'A' (a_C) and the second integer 'B' (b_C), implemented on lines '115-134', as a parameter. The value returned is entered in the variable 'k'.

Calculation of the number of combinations 'A' per 'B' (line 203)

In line '203', the number of combinations 'A' to 'B' is calculated using the arithmetic expression 'I / (j * k)' and the result is entered in the variable 'combos'.

Return of the number of combinations 'A' per 'B' (line 205)

At line '205' the command 'return combos' returns the content of the variable 'combos' (the number of combinations 'A' per 'B'), where the program control is located.

(~) A <= B OR A < 0 OR B < 0 (lines 207-210)

Lines "207-210" execute the commands of "else" (line 207) corresponding to "if (a_C > b_C && a_C >= 0 && b_C >= 0)" (line 198), for the case where "A" or "B" contain a value less than "0" or "A" contains a value less than that of "B". The crossing of "if" is as follows:

Return error value (line 209)

At line "209", the command "return error_C" returns the content of the variable "error_C" (error value), i.e., the value "-1" (cf. "Initialization of variables (line 196)"), where the program control is located.

Check_Combinations (A, B) (lines 213-231)

On lines "213-231" the function "Check_Combinations" is implemented, of type "void" and with the variables "a_CC", "b_CC" (of type "int") as parameters, where it takes as input the two integers "A" and "B", calls the function "Combinations(...)" and depending on the value returned prints the characteristic messages on the "standard" output. On line '219' the characteristic message for function '3' is printed on the 'standard' output. The crossing of 'Check_Combinations(...)' is as follows :

Variable declaration (line 217)

See section 'Variables', subsection 'Check_Combinations (a_CC, b_CC)', page '56'.

Calling the function 'Combinations (A, B)' (line 220)

At line '220', 'Check_Combinations(...)' calls the function 'Combinations(...)', which returns the result of the number of combinations 'A' per 'B', if there is no restriction, or an error value (-1), if there is a restriction. The result is stored in the variable 'cc'.

(~) No error value (lines 221-225)
(~) Error value (lines 226-230)

In lines '221-225' a check is implemented with an 'if - else' command for the value returned by 'Combinations(...)'. The condition is "cc != -1" and if it returns a value of "True", i.e., "Combinations(...)" returns a value that is not equal to the error value, then the commands of the subsection "No error value (lines 221-225)" will be executed. Otherwise, if 'Combinations(...)' returns a value of '-1' (error value), then the commands in the subsection 'Error value (lines 226-230)' shall be executed. The intersection of 'if - else' is as follows :

(~) No error value (lines 221-225)

In lines '221-225' the commands of 'if (cc != -1)' (line 221) are executed, in case 'Combinations(...)' returns a value that is not equal to the error value. The traversal of "if" is as follows :

Print the result of operation [3] (A! / B! * (A - B)!)) (line 223)

In line "223", the contents of the value returned by "Combinations(...)" and entered in the variable "cc", which is the result of the "A! / B! * (A - B)!" (the number of combinations "A" per "B".

(~) Error value (lines 226-230)

In lines "226-230" the commands of "else" (line 226) corresponding to "if (cc != -1)" (line 221) are executed, in case "Combinations(...)" returns a value which is equal to the error value. Obviously, this is a violation of the constraint and therefore at line "228" the characteristic message is printed in the "standard" output.

Check_Valid_Combinations (A, B) (lines 233-246)

On lines '223-246' the stand-alone subroutine 'Check_Valid_Combinations(...)' is implemented to count the number of valid operations, of type 'int' and with the variables 'a_CVC', 'b_CVC' (of type 'int') as parameters, where it takes as input the two integers 'A' and 'B', calls the function 'Combinations(...)" and depending on the value returned, returns where the program control is, a constant indicating that the function was used validly or invalidly [3]. The traversal of "Check_Valid_Combinations(...)" is as follows:

Variable declaration (line 235)

See section "Variables", subsection "Check_Valid_Combinations (a_CVC, b_CVC)" page "56".

Calling the function 'Combinations (A, B)' (line 237)

At line '237', 'Check_Valid_Combinations(...)' calls the function 'Combinations(...)', which returns the result of the number of combinations 'A' per 'B', if there is no constraint, or an error value (-1), if there is a constraint. The result is stored in the variable 'cvc'.

(~) No error value (lines 238-241)
(~) Error value (lines 242-245)

In lines '238-245' a check is implemented with an 'if - else' command for the value returned by 'Combinations(...)'. The condition is "cvc != -1" and if it returns a value of "True", i.e., "Combinations(...)" returns a value that is not equal to the error value, then the commands of the subsection "No error value (lines 238-241)" will be executed. Otherwise, if 'Combinations(...)' returns a value of '-1' (error value), then the commands in the subsection 'Error value (lines 242-245)' will be executed. The intersection of 'if - else' is as follows :

(~) No error value (lines 238-241)

In lines '238-241' the commands of 'if (cvc != -1)' (line 238) are executed, in case 'Combinations(...)' returns a value that is not equal to the error value. The crossing of "if" is as follows :

Return value of a valid function (line 240)

At line "240" the command "return 1" returns the integer value "1" where the program control is, indicating that function [3] was validly executed.

(~) Error value (lines 242-245)

On lines "242-245" the commands of "else" (line 242) corresponding to "if (cvc

!= -1)" (line 238) are executed, in case "Combinations(...)" returns a value equal to the error value. The crossing of "else" is as follows :

Return value of invalid function (line 244)

At line "244" the "return 0" command returns the integer value "0" where the program control is, indicating that function [3] was executed invalidly.

Exit() (lines 226-230)

Lines '248-253' implement the function 'Exit(...)' (for counting the number of valid functions), of type 'int' (no parameters), where it returns the value '1' each time it is called in the program. The traversal of "Exit(...)" is as follows :

Declaration and initialization of variables (line 250)

Line "250" assigns the initial value "1" to the variable "cnt" (auxiliary variable for counting the valid function [4]).

Return value of valid function [4] (line 252)

At line "252", the command "return cnt" returns the content of "cnt", i.e., the integer value "1" (see "Declaration and initialization of variables (line 250)"), where the program control is, indicating that function [4] was validly executed.

Menu (A, B) (lines 255-297)

On lines "255-297" the function "Menu(...)" is implemented, of type "void" and with the variables "a_M" and "b_M" as parameters, accepting as input the two integers "A" and "B". The function repeatedly reads from the "standard" input an integer number representing one of the "4" functions, calls the corresponding function that performs the read function in detail and prints to the "standard" output the number of valid functions after the end of the iteration. The crossing of "Menu(...)" is as follows :

Variable declaration (lines 259-263)

See section "Variables", subsection "Menu (a_M, b_M)", pages "16-17"

Variable initialisation (lines 260, 261, 262, 263)

An initialisation of variables is performed on lines "260", "261", "262" and "263". In more detail, in line '260' the value '0' is assigned to the variable 'm_P' (the number of valid functions [1]), in line '261' the value '0' is assigned to the variable 'm_F' (the number of valid functions [2]), in line '262' the value '0' is assigned to the variable 'm_C' (the number of valid functions [3]) and in line

'263' the value '0' is assigned to the variable 'm_E' (the number of valid functions [4]).

Loop (lines 265-290)

On lines "265-290" a loop is executed with the "do - while" loopback instruction with the loop termination condition, the representation "ch != 4" (the number of operations read). The loop is executed at least once and then as long as this statement results in a value of "True" (as long as the user does not select the output function [4]).
The loop is traversed as follows :

The menu with function options (lines 267, 268, 269, 270)

On lines "267", "268", "269" and "270" the menu with the available functions that the user can select is printed on the "standard" output.

Entering a mode selection (line 272)

At line "272" the number representing a mode selection is read from the "standard" input with the function "scanf()", accompanied by the appropriate message printed at the "standard" output with the function "printf()" (line 271) each time the loop is executed.

(~) Valid mode selection (lines 273-283)
(~) Invalid mode selection (lines 285-288)

In lines "273-288" a check for the case of entering a valid or invalid mode selection is implemented with an "if-else" control instruction.
In detail, if the user enters a valid mode selection (ch >= 1 && ch <= 4), then the commands of the subsection "Valid mode selection (lines 273-283)" will be executed, otherwise if the user enters an invalid mode selection, then the commands of the subsection "Invalid mode selection (lines 285-288)" will be executed.

(~) Valid mode selection (lines 273-283)

In lines "273-283", the commands of "if ch >= 1 && ch <= 4)" (line 273) are executed, in case a valid mode selection is entered. The crossing of 'if' is as follows :

The function selections (lines 275-283)

On lines "275-283", the menu with the function selections is implemented with a "switch-case" control instruction, where the custom that executes the selected function (the content of the integer variable "ch" read from the "standard" input, each time the loop is executed) is called and if, executed without violating the constraints, a characteristic number is entered in a variable to calculate the valid selected functions each time the loop is executed

The traversal of the "switch-case" is as follows :

[1] Call of the function "Check_Power (A, B)" (line 277)

If the variable "ch" contains the value "1" (function [1] "A^B"), then, the call of the function "Check_Power(...)" is performed, where it takes as input the two integers "A" (a_M) and "B" (b_M) and is implemented on lines "80-98".

Count of the valid function value [1] (line 278)

Also, the function 'Check_Valid_Power' is called, where it takes as input the two integers 'A' (a_M) and 'B' (b_M) and is implemented on lines '100-113', and the number of valid functions [1] is calculated by the representation 'm_P + Check_Valid_Power (a_M, b_M)', each time the loop is executed. The result is stored in the variable "m_P", where it has as initial value, the value "0" (cf. "Initialisation of variables (line 260)". The 'break' instruction transfers the program flow to the subsequent instructions outside the body of the 'switch-case'.

[2] Call of the function "Check_Factorial (A, B)" (line 279)

If the variable "ch" contains the value "2" (function [2] "A!" and "B!"), then the function "Check_Factorial(...)" is called, where it takes as input the two integers "A" (a_M) and "B" (b_M) and is implemented on lines "168-175".

Counting the valid function value [2] (line 280)

Also, the call to the stand-alone subroutine "Check_Valid_Factorial(...)', where it takes as input the two integers 'A' (a_M) and 'B' (b_M) and is implemented on lines '177-191', and calculates with the representation 'm_F + Check_Valid_Factorial (a_M, b_M)' the number of valid functions [2], each time the loop is executed. The result is stored in the variable "m_F", where it has as initial value, the value "0" (see "Initialisation of variables (line 261)". The "break" command transfers the program flow to the subsequent commands outside the body of the "switch-case".

[3] Calling the function "Check_Combinations (A, B)" (line 281)

If the variable "ch" contains the value "3" (function [3] "A! / B! * (A - B)!)), then the function 'Check_Combinations(...)' is called, taking as input the two integers 'A' (a_M) and 'B' (b_M) and is implemented on lines '213-231'.

Valid function value count [3] (line 282)

Also, the call to the stand-alone subroutine "Check_Valid_Combinations(...)', which takes as input the two integers 'A' (a_M) and 'B' (b_M) and is implemented on lines '233-246', and calculates with the representation 'm_C + Check_Valid_Combinations (a_M, b_M)' the number of valid operations [3], each time the loop is executed. The result is stored in the variable "m_C", where it has as initial value, the value "0" (see Fig. "Initialisation of variables (line 262)". The "break" instruction transfers the program flow to the following instructions outside the body of the "switch-case".

(~) Invalid function selection (lines 285-288)

In lines "285-288" the instructions of "else" (line 285" corresponding to "if ch >= 1 && ch <= 4)" (line 273) are executed, for the case of introducing an invalid function selection. At line "287" the command "system ("cls");" clears the "standard" output for uniform menu mapping.

[4] Call of the function "Exit()" (line 294)

In case the variable "ch" contains the value "4" (function [4] Exit), then, the loop is stopped, because, the representation "ch != 4" results in the value "False", the call of the function "Exit(...)" is performed, where it does not accept parameters as input and is implemented on lines "248-253". The value it returns is stored in the variable 'm_E', where it has the value '0' as its initial value (see lines 248 and 248). "Initialisation of variables (line 263)".

Calculation of the number of valid operations (line 295)

At line '295', the number of valid operations is calculated by the arithmetic expression 'm_P + m_F + m_C + m_E' and the result is entered in the variable 'sum'.

Printing the number of valid operations (line 296)

At line '296', the contents of the variable 'sum' (the number of valid operations) are printed at the 'standard' output with a 'printf()' function, accompanied by the appropriate message.

EXAMPLES

Example 1 (A == 9 / B == 10 / ch == 1, 2, 3, 4)

```
============================================================

Arithmetic operations with integers menu

============================================================

Enter the integer A : 9
Enter the integer B : 10
************************************************************************************
[1] Calculation of force A^B
[2] Calculation of A! and B!
[3] Calculating the number of combinations A per B
[4] Exit

Mode selection : 1
************************************************************************************
[1] Calculation of force A^B

A^B : 3486784512.000000

----------------------------------------------------------------------------------------------------

[1] Calculation of force A^B
[2] Calculation of A! and B!
[3] Calculating the number of combinations A per B
[4] Exit
```

Mode selection : 2

****

[2] Calculation of A! and B!

A! : [362880]

B! : [3628800]

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 3

****

[3] Calculating the number of combinations A per B

Error

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 4

****

Number of valid functions : 3

Example 2 (A == 0 / B == 0 / ch == 1, 2, 3, 4)

==

Arithmetic operations with integers menu

==

Enter the integer A : 0

Enter the integer B : 0

****

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 1

****

[1] Calculation of force A^B

Error

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 2

****

[2] Calculation of A! and B!

A! : [1]

B! : [1]

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 3

****

[3] Calculation of the number of combinations A per B

Error

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Function selection : 4

****

Number of valid functions : 2

<u>Example 3 (A == 5 / B == -2 / ch == 1, 2, 3, 1, 4)</u>

==

Arithmetic operations with integers menu

==

Enter the integer A : 5

Enter the integer B : -2

****

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 1

****

[1] Calculation of force A^B

A^B : 0.040000

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 2

****

[2] Calculation of A! and B!

A! : [120]

Error

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 3

****

[3] Calculation of the number of combinations A per B

Error

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 1

****

[1] Calculation of force A^B

A^B : 0.040000

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 4

****

Number of valid modes : 3

Example 4 (A == -7 / B == -6 / ch == 2, 4)

==

Arithmetic operations with integers menu

==

Enter the integer A : -7

Enter the integer B : -6

****

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 2

****

[2] Calculation of A! and B!

Error

Error

----

[1] Calculation of force A^B

[2] Calculation of A! and B!

[3] Calculating the number of combinations A per B

[4] Exit

Mode selection : 4

****

Number of valid functions : 1

OBSERVATIONS

The correctness of the results in the above examples was also checked using a standard calculator and no difference was found.
The string "**" while-indicates the printed contents of the "standard" output, since, the command

"system ("cls");" clears the "standard" output each time it is executed. The functions in red are the invalid functions and are not counted in the number of valid functions.

In detail, we have three examples where each corresponds to specific conditions.

Example 1 (A == 9 / B == 10 / ch == 1, 2, 3, 4)

In "Example 1", the user enters first (A) the integer "9", second (B) the integer "10" and chooses the following operations :

[1] : A^B□ 9^10 □ 3486784512.000000

[2] : A!□ 9! □362880
B!10! □ 3628800

[3] : A! / B! (A - B)! □ 9! / 10! * (9 - 10)! □ 9! / 10! * (-1)! □Error
No factor with negative number specified

[4] : Exit

Number of valid operations : [1] + [2] + [4] == 3

Example 2 (A == 0 / B == 0 / ch == 1, 2, 3, 4)

In "Example 2", the user enters first (A) the integer "0", second (B) the integer "0" and selects the following operations:

[1] : A^B □ 0^0 □Error
Power "0^0 "is not specified

[2] : A!□ 0! □
B!□ 0! □ 1

[3] : A! / B! (A - B)! □ 0! / 0! * (0 - 0)! □ 0! / 0! * 0! □ 1 / 1 * 1 □Error
No combination with the same integer is specified e.g. 0 with 0

[4] : Output

Number of valid operations : [2] + [4] == 2

Example 3 (A == 5 / B == -2 / ch == 1, 2, 3, 1, 4)

In "Example 3", the user enters first (A) the integer "5", second (B) the integer "-2" and chooses the following operations :

[1] : A^B□ 5^(-2) □0.040000

[2] : A!□ 5! □120
B!□ (-2)! □Error
No factor with negative number specified

[3] : A! / B! (A - B)! □ 5! / (-2)! * (5 - (-2)! □Error
Factor with negative number not specified

[1] : A^B □ 5^(-2) □ 3486784512.000000

[4] : Output

Number of valid functions : [1] + [2] + [3] + [4] == 4

Example 4 (A == -7 / B == -6 / ch == 2, 4)

In "Example 4", the user enters first (A) the integer "-7", second (B) the integer "-6" and chooses the following operations :

[2] : A! □ (-7)! □Error
No factor with negative number is specified
B!□ (-6)! □Error
Factor with negative number not specified

[4] : Exit

Number of valid functions : [4] == 1

THEME 3

NOTE

The following program and its documentation answer the question of question "Topic 3".

PROGRAM

```
#include <stdio.h>

```

```
3 int MSum (int);
4
5 int main (int argc, char **argv)
6 {
7      system ("chcp 1253"); 7system ("chcp 1253");
8
9      int x;
10     int p;
11
12     printf ("====================================\n\n")
13     printf ("Recursive Function\n\n");;
14     printf ("====================================\n\n")
15     printf ("Enter integer : ");
16     scanf ("%d", &x);;
17     = MSum (x);.
18     printf ("------------------------------------\n\n");;
19     printf ("The integer : [%20d]\n", x);
20     printf ("Function result : [%20d]\n\n", p);;
21
22     return 0;
23 }
24
25 int MSum (int N)
26 {
27 if (N == 1)
28 return 1;
29 return N + MSum(N - 1);;
30 }
```

EXPLANATION

The above program reads into the main function "main()", with the function "scanf()", from the "standard" input an integer "x" (line 16) and calls the function "MSum()" with this number as a parameter (line 17). The function 'MSum()' is of type 'int', i.e. it returns an integer value on the line on which it is called and has as parameter the integer 'x' read from the 'standard' input. The number 'x' has been passed to the variable 'N' which is recognised by the function 'MSum()'. MSum' contains an 'if' control instruction (lines 27-28), which checks whether the value produced by 'N == 1' results in a value 'True' (other than '0'). If, the number read is '1' (N == 1), then MSum()' will return 'return 1' with the instruction 'return 1'. Otherwise, (N != 1) the program flow is transferred to the next command which is "return N + MSum(N - 1)". "MSum()" will return, with this command, the value of "N + MSum(N - 1)".

At line "29" of the program, the function "MSum()" calls itself and is therefore considered a recursive function. For example, suppose the number "5" is read from the "standard" input, in the main function "main()". Main() calls at line '17' the function 'MSum()' with the number '5' as a parameter. MSum()' checks at line 27 whether the parameter contains the value '1', detects that it does not, and so calls itself with the number '4' (N - 1) as a parameter.

The representation "N + MSum(N - 1)" with "N == 5" produces the result "5 + 4 = 9". The function is called recursively with the numbers '3' (9 + 3 = 12), '2' (12 + 2 = 14) and '1' as parameters. With parameter '1' it checks whether 'N == 1' (line 27) and returns with 'return 1' the value '1' added to the total result returned by 'MSum()' (14 + 1 = 15). Therefore, 'MSum()' with parameter '5' returns the value '15', i.e. the sum of the addition '1 + 2 + 3 + 3 + 4 + 5'. To summarise, 'MSum()' for 'N' integer parameter returns the sum of the addition 'N + (N - 1) + (N - 2)+...+ 1'.

ITEM 4

NOTE "Hanoi.c

" "Program "Hanoi.c"" (Source Code) and "Documentation "Hanoi.c" (Objective, Structure, Functions, Variables, Traversal, Examples) answer the objective of the question "Topic 2".

PROGRAM "Hanoi.c"

```
#include <stdio.h>

/* Function declaration */

void Title (); // The title of the program

int Read_num_Disks (); // Input the number of disks

void Print_num_Disks (int); // Print the number of disks
```

```
int Num_Min_Moves (int); // Calculate the minimum moves of the game
"Towers of Hanoi"

void Print_Num_Min_Moves (int); // Print the minimum moves of the
game "Towers of Hanoi"

void Move_Disks (int, char, char, char); // The path of the disks


/* Where N is the number of discs, A is post 1, B is post 2 and C is
post 3 */


int main (int argc, char **argv) /* main (int argc, char **argv) */

{

      system ("chcp 1253"); system ("chcp 1253");


      int n; // Variable declaration

      char pole_1, pole_2, pole_3;


      pole_1 = 'A'; // pole 1

      pole_2 = 'B'; // pole 2

      pole_3 = 'C'; // pole 3


      Title(); // Call the function "Title()"

      n = Read_num_Disks (); // Call the function "Read_num_Disks()"

      Print_num_Disks (n); // Call the function "Print_num_Disks (N)"

      Print_Num_Min_Moves (n); // Call the function
"Print_Num_Min_Moves (N)"

      Move_Disks (n, pole_1, pole_2, pole_3); // Call the function
"Move_Disks (N, A, B, C)"


      return 0;

}


void Title () /* Title () */

{

      printf
("================================================================
\n\n");

      printf ("Towers of Hanoi\n\n"); // Title of the program
```

```
	printf
("==============================================================
\n\n");

}


int Read_num_Disks () /* Read_num_Disks () */

{

	int n_RnD; // Variable declaration


	printf ("Enter number of disks : ");;

	scanf ("%d", &n_RnD); // Insert the number of disks

	printf ("\n-----------------------------------------------------
-------\n\n");


	return n_RnD; // Return the number of disks

}


void Print_num_Disks (int n_PnD) /* Print_num_Disks (N) */

{

	printf ("Number of disks : [%20d]\n", n_PnD); // Print the
number of disks

}


int Num_Min_Moves (int n_NMM) /* Num_Min_Moves (N) */

{

	int min_moves; // Variable declaration


	if (n_NMM == 0) /* (~) 0 disks */

	{

		return 0; // Return number of minimum moves

	}

	else /* (~) 0< disks */

	{

		min_moves = Num_Min_Moves (n_NMM - 1) + 1 + Num_Min_Moves
(n_NMM - 1); // Calculate the number of minimum moves
```

```
            return min_moves; // Return minimum number of moves
        }
}


void Print_Num_Min_Moves (int n_PNMM) /* Print_Num_Min_Moves */
{
        int print_min_moves; // Variable declaration


        print_min_moves = Num_Min_Moves (n_PNMM); // Call the function
"Num_Min_Moves (N)"
        printf ("Minimum moves : [%20d]\n\n", print_min_moves); //
Print the number of minimum moves
}


void Move_Disks (int n_MD, char pole_A, char pole_B, char pole_C) /*
Move_Disks (N, A, B, C) */
{
        if (n_MD == 1 && n_MD != 0) /* (~) 1 disk */
        {
                printf ("%c --> %c\n", pole_A, pole_C); // Route
        }
        else /* (~) 1< disks */
        {
                if (n_MD != 1 && n_MD != 0) /* (+) 1< trays */
                {
                        n_MD = n_MD - 1; // Subtract the number of disks
                        Move_Disks (n_MD, pole_A, pole_C, pole_B); //
Recursive call to the function "Move_Disks (N-1, A, C, B)"
                        printf ("%c --> %c\n", pole_A, pole_C); // Route
                        Move_Disks (n_MD, pole_B, pole_A, pole_C); //
Recursive call to the function "Move_Disks (N-1, B, A, C)"
                }
        }


}
```

DOCUMENTATION "Hanoi.c"

REQUIRED

The program "Hanoi.c" achieves the following functions:

a) Reads from "standard" the number of disks.

b) Prints to "standard" output the number of disks.

c) Calculate the number of minimum movements from pole "A" to pole "C".

d) Print the number of minimum movements from pole "A" to pole "C" on the "standard" output.

e) Execute the algorithm "Hanoi Towers" with the help of the recursive function.

f) Print in the "standard" output the algorithm "Hanoi Towers" with the help of the recursive function.

STRUCTURE

In order to implement the requested task, the libraries (.h) were initially used :

a) "stdio.h": contains the ready-made functions "scanf(...)" and "printf(...)" which are linked to the input and output channels respectively to read and print the contents of the corresponding variables. Also, "printf(...)" was used to print feature messages for optimal understanding of the source code.

In addition, the characteristic operators :

a) numeric : +, -

b) relational : ==, !=,

c) logical : &&

d) assignment :=

e) & operator : For the variable address as the second argument of the "scanf()" function associated with the "standard" input

The control statements :

a) if - else

Each function from the "Requiredsection was implemented with stand-alone subroutines (cf. section "Functions").

SYNOPSIS

See comments

lines "2-9"

VARIABLES

See lines "15-16

"(main) line "40" (Read_num_Disks)
line "56" (Num_Min_Moves)
line "72" (Print_Num_Min_Moves)

DISCUSSION

See lines "15-16" (main line "40" (Read_num_Disks)

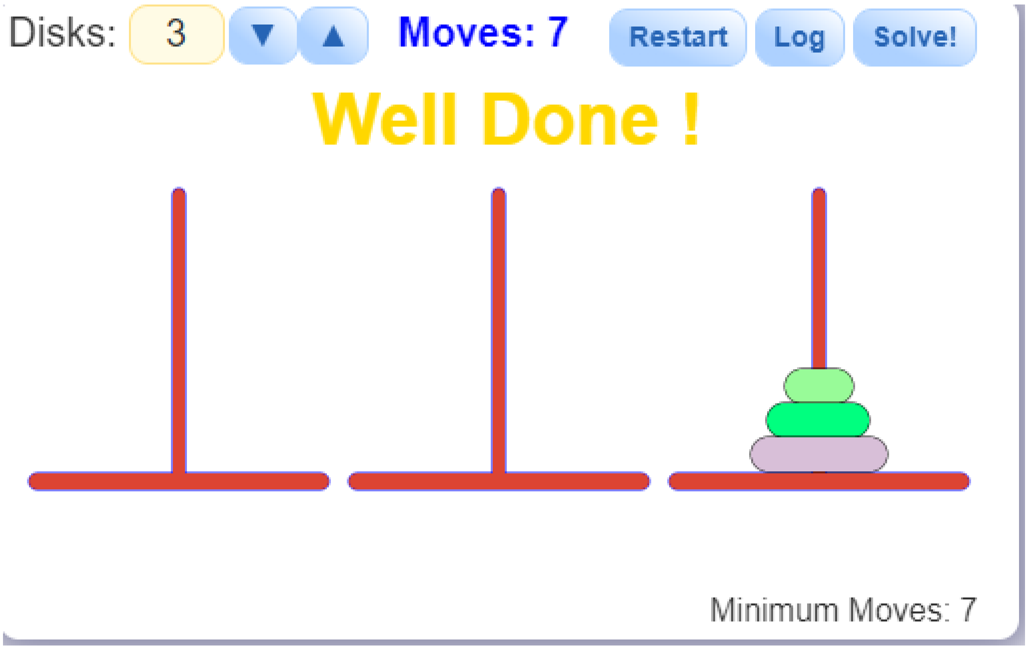
line "56" (Num_Min_Moves)

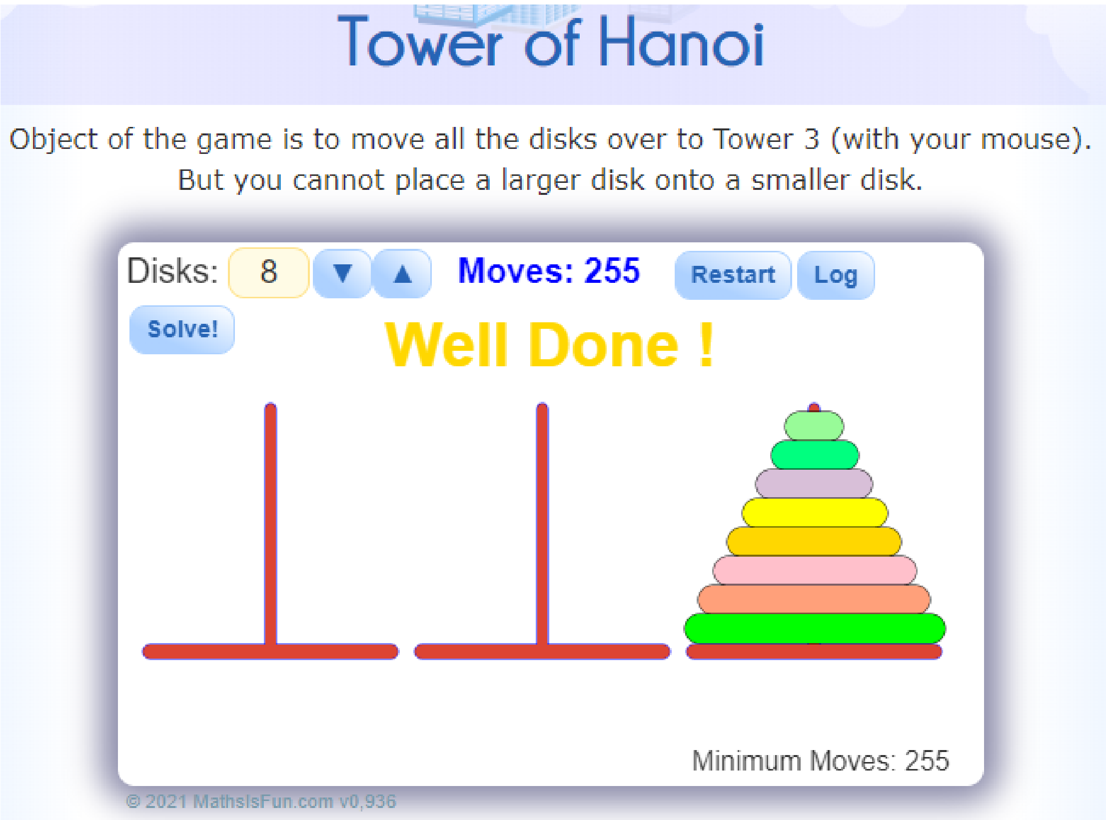
line**"72" (Print_Num_Min_Moves)**

Comments "PROGRAM "Hanoi.c"

EXAMPLES / OBSERVATIONS

```
Active code page: 1253
==============================================================

Towers of Hanoi

==============================================================

Enter the number of disks : 3

------------------------------------------------------------

Number of disks : [                   3]
Minimum moves   : [                   7]

A --> C
A --> B
C --> B
A --> C
B --> A
B --> C
A --> C
```





Tower of Hanoi
Object of the game is to move all the disks over to Tower 3 (with your mouse).
But you cannot place a larger disk onto a smaller disk.
Disks: 8
Moves: 255
Restart
Log
Solve!
Well Done !
Minimum Moves: 255
© 2021 MathsIsFun.com v0.936

Thank you for your attention.