COMPUTER PROGRAMMING

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

DEPARTMENT OF COMPUTER AND INFORMATION TECHNOLOGY AND COMPUTER ENGINEERING

PROJECT 4
LOOPS

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: 7/12/2021

COMPUTER PROGRAMMING

CONTENTS

ANSWERS TO THE THEORY QUESTIONS

ITEM 1

a) What are iterative structures (loops)?

Iterative structures are instructions that execute a body of instructions repeatedly for a specified number of times. The number of times is determined by the value of an instance used as a loop condition and, as long as the value of the instance produced is not equal to "0" (value "True"), then the loop will be executed until the value of the instance produced is "0" (value "False").

b) Which iterative structures do you know in the "C" language?

The iterative structures of the "C" programming language are the following:

a) while :

Function

Executes a body of instructions for as many times as the value of the representation
results in a 'True' value (other than '0').

Syntax

while (statement)
{

Commands

}

Example :

```
x = -5;
      while (x < 0)
      {
          printf ("Negative number");;
          x++;
```

```
}
```

Example explanation

First, the value "-5" is entered in the variable "x". The "while" instruction executes the instruction body as long as the variable "x" contains a value less than "0", i.e., as long as the representation "x < 0" results in a value of "True" (other than "0"). The instruction body contains a function "printf()" that prints the message "Negative number" to indicate that "x" contains a negative value and the arithmetic operation "x++" that increases the contents of the variable "x" by "1". Therefore, the loop will be executed "5" times as long as the variable "x" contains the values "-5", "-4", "-3", "-2" and "-1".

b) do - while :

Function

Executes a body of instructions, at least, once and for as many times as the value of the representation results in a "True" value (other than "0").

Syntax

do
{

Commands

}
while (instance);

Example

```
x = 5;
do
{
    printf ("Number : 5");;
}
while (x != 5);;
```

Example explanation

First, the value "5" is entered in the variable "x". The command "do - while" will execute the instruction body once and then as long as the variable "x" does not contain the value "5", i.e., as long as "x != 5" results in a value of "True"

(other than "0"). The instruction body contains a function "printf()" that prints the message "Number : 5" to indicate that "x" contains the value "5". Therefore, the loop will be executed "1" time regardless of the content of the variable "x".

c) for :

Function

Executes a body of instructions for as many times as the value of the results in a 'True' value (other than '0').

Syntax

for (instance1 ; instance2 ; instance3)
{

Commands

}

Example

```
for (i = 0 ; i < 3 ; i++)
   {
      printf ("Hello world");;
}
```

Example explanation

Initially, the value "0" is entered in the variable "i". The command "for" will execute the command body as long as the variable "i" contains a value less than "3", i.e., as long as "i < 3" results in a value of "True" (other than "0"). The command body contains a function "printf()" that prints the message "Hello world" Then the 'i++' statement is executed which increments the value of the variable 'i' by '1'. Therefore, the loop will be executed "3" times as long as the variable "i" contains the values "0", "1" and "2".

c) How are they related to each other?

The iterative structures "while", "do - while" and "for" have a common function, to execute a body of instructions repeatedly for a certain number of times. More specifically, the loop is executed for as long as the value of the argument, which the iteration instructions check to determine the number of times the loop is executed, results in a value of "True" (other than "0") with the difference being made by "do - while" which will first execute the instruction

body once and then check the value of the argument. Otherwise, the loop is interrupted.

ITEM 2

a) What do you know about the "for" iteration structure?

The "for" iteration structure executes a body of instructions repeatedly for a specified number of times (see Topic 1b, page "6", "c) for") and is written as follows:

for (instance1 ; instance2 ; instance3)
{
Instructions

}

More specifically, the loop is executed as long as "instance2" results in a "True"
value (a value other than "0"). The 'instance2' is usually a representation with logical or relational operators, so that when the instruction is executed the value that 'instance2' can produce is 'False' (value '0'), otherwise the loop will run and never stop (infinite loop). The "representation1" is usually an initial auxiliary variable (counter) entry used to allow the user to control the loop being executed, i.e., to define the number of times the loop is to be executed. Finally, "representation3" is a numeric representation that modifies the content of the auxiliary variable defined as "counter" in the "representation1" entry in order to achieve the smooth operation of the loop and produce the desired result. The three representations are separated by the character of the Greek question mark "?".
The "for" is for programmers the most "convenient" (not in all cases) iteration command for the reasons that it accepts three arguments with three or more representations, saving space and making the code more readable. The following examples repeatedly execute a "printf()" function that prints the message "Positive" indicating that the content of the variable "x" is a positive number. The first example is implemented with the "while" iteration instruction , while the second is implemented with "for". The value of the variable "x" is changed by the representation "x- -", so the loop is executed "1" time for "x = 1".

Example 1

```
x = 1

 while (x > 0)
```

```
{
  printf ("Positive");
  x-- }
```

Example 2

```
for (x = 1 ; x > 0 ; x--)
{
  printf ("Positive");
}
```

b) What do you know about the "break" command?

The "break" command interrupts the program from the line it is on to the end of a command line. More specifically, in a loop, "break" effectively cancels the loop at the point where it is located, regardless of what value is produced by the instance controlling it within it, and transfers the program to the instructions outside it.

For example :

```
for (x = 0 ; x < 100 ; x++)
  {
        printf ("Less than 100"); break;
        printf ("Positive");
  }
  printf ("Price 0!");;
```

In the example, "for" for "x = 0" checks and finds that "x < 100" results in a "True" value (other than 0), so the loop will be executed. "printf()" inside the loop prints the message "Less than 100" from the "standard" output and "break" cancels the loop which would continue until the variable "x" contained the value "100" and the program goes to "printf()" which is outside the loop and prints the message "Value 0!". In summary, the loop will only execute "1" time and the "printf("Positive");" command will never be executed.

c) What do you know about the "continue" command?

The "continue" command stops the loop from where it is and resets the program from the beginning of the loop.

More specifically, the loop commands below "continue" are not executed, as the program is moved to the beginning of the loop and is executed from the beginning to the line where this command is located as many times as the loop expression results in a "True" value (other than "0
"). For example :

```
for (x = -3 ; x < 0 ; x++)
{

   printf ("Negative"); continue;
   printf ("Positive");

}
   printf ("Price 0!");;
```

In the example, "for" for "x = -3" checks and finds that the statement "x < 0" results in a "True" value (other than "0"), so the loop will be executed. "printf()" inside the loop prints the message "Negative" from the "standard" output, and "continue" resets the loop from the beginning and then executes for "x = -2" and for "x = -1". The "printf("Positive");" command below "continue" will never be executed. To summarize, the loop will execute "3" times and print the message "Negative" from the "printf()" of the loop and when it finishes the "printf ("Value 0!");" command located outside the loop will be executed, where it prints the message "Value 0!" indicating that the variable "x" contains the value "0".

SOURCE CODES / DOCUMENTATION

ITEM 3A

NOTE "MathsWithIntegersInLoopA.c"

The "Program "MathsWithIntegersInLoopA.c"" (Source Code) and the "Documentation "MathsWithIntegersInLoopA.c"" (Question, Structure, Variables, Traversal, Examples, Observations) answer the question of "Topic 3A". More specifically, in the case where the user enters as many integers as they want and the loop stops until the user enters all the numbers they have declared to enter.

PROGRAM "MathsWithIntegersInLoopA.c"

```
1 #include <stdio.h>

2

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

4
```

```
5 {

6system  ("chcp 1253");;
7
8  int N, n; // Declaration of variables
9  int i = 0; // Initialize variables
10 int pow;
11 int evens = 0; // Initialize variables
12 int neg = 0; // Initialize variables
13 int prod = 1; // Initialize variables
14 int pos = 0; // Initialize variables
15 int sum = 0;      // Initialize variables
16 float avg;
17
18 printf
("=================================================\n\n");
19 printf ("Mathematical operations with integers\n\n"); // Program
title
20 printf
("=================================================\n\n");
21 printf ("Enter N integers : ");
22 scanf ("%d", &N); // Insert the number of integers
23 printf   ("\n-------------------------------------------------
\n\n");
24 printf ("The number of integers : [%20d]\n\n", N); // Print the
number of integers
25 if (N > 0) /* (~) One or more integers */
26 {

27       while (i < N) /* Loop */
28       {
29             printf ("-----------------------------------------
--------\n\n");
30             printf ("Enter integer : ");;
31             scanf ("%d", &n); // Insert the integers
32             printf ("\n-----------------------------------------
----------\n\n");
33             printf ("The integer : [%20d]\n\n", n); // Print the
integer numbers
34             if (n % 2 != 0) /* (!) Insert odd number */
35             {
36                   pow = n * n; // Calculate the square of the
odd
37                   printf ("Square of odd : [%20d]\n\n", pow);
// Print the square of the odd
38                   printf ("Redundant number\n");;

39             }
40             else /* (!) Insert even number */
41             {
42                   evens++;  // Calculating the number of even
numbers
43                   printf ("\nEven number\n");;

44             }
45             if (n >= 0) /* (+) Insert positive number */
46             {
```

```
47                      pos++; // Calculating the number of positive
numbers
48                      sum = sum + n; // Calculate the sum of positive
numbers
49                      avg = (float) sum / pos; // Calculate the
average of the positive numbers
50              }
51              else /* (+) Insert negative number */
52              {
53                      neg++; // Calculate the number of negative
numbers
54                      prod = prod * n; // Calculate the product of
negative numbers
55              }
56              i++;
57      }

59      if (neg != 0)       /* (-) One or more negative numbers */
60      {

61              printf ("-----------------------------------------
--------\n\n");
62              printf ("Number of even numbers : [%20d]\n", evens);
// Print the number of even numbers
63              printf ("Average of positive numbers : [%20.6f]\n",
avg); // Print the average of positive numbers
64              printf ("Product of negative numbers : [%20d]\n",
prod); // Print the product of negative numbers
65      }
66      else /* (-) No negative number */
67      {
68              printf ("-----------------------------------------
--------\n\n");
69              prod--; // Assign value "0" to variable "prod"
70              printf ("Number of even numbers : [%20d]\n", evens);
// Print the number of even numbers
71              printf ("Average of positive numbers : [%20.6f]\n",
avg); // Print the average of positive numbers
72              printf ("Product of negative numbers : [%20d]\n",
prod); // Print the product of negative numbers
73      }
   74}
75 else /* (~) No integer */
   76{
77      printf ("-----------------------------------------------
--\n\n");
78      prod--; // Assign value "0" to variable "prod"
79      printf ("Number of even numbers : [%20d]\n", evens); //
Print the number of even numbers
80      printf ("Average of positive numbers : [%20.6f]\n", avg);
// Print the average of positive numbers
81      printf ("Product of negative numbers : [%20d]\n", prod);
// Print the product of negative numbers
82 }
83
84 return 0;
```

```
85 }
```

DEFINITION "MathsWithIntegersInLoopA.c"

REQUESTED

The program "MathsWithIntegersInLoopA.c" achieves the following operations:

a) Reads from the "standard" input "N" integers.

b) Checks the number of integers given by the user.

c) Calculates the square of the odd numbers.

(d) Calculates the number of even numbers.

(e) Calculates the average of positive numbers.

(f) Calculates the product of negative numbers.

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

MODEL

In order to implement the requirement, the libraries :

a) "stdio.h" : Contains the ready-made functions "scanf()" and "printf()" associated with the input and output channels respectively for reading and printing contents of the respective variables. Also, "printf()" was used to print characteristic messages for optimal understanding of the source code.

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++

(f) decrease : variable--.

The control commands :

(a) if - else

The iteration commands :

a) while

VARIABLES

Integer variables (of type "int")

N (The number of integers)

n (Each integer number inserted into the loop)

i (Counter to control the loop)

pow (The square of the odd numbers)

evens (The number of even numbers)

pos (The number of positive numbers)

sum (The sum of positive numbers)

neg (The number of negative numbers)

prod (The product of the negative numbers)

Real variables (of type "float")

avg (The average of the positive numbers)

DISCUSSION

Declaration of variables (lines 8-16)

The program starts with the declaration of the variables in the c lines "8-16" (for details see subsection "Variables").

Initialisation of variables (lines 9, 11, 12, 13, 14, 15)

Some initialisation of variables is done in the above lines.
In more detail, line "9" is entered the value "0" in "i" for the counter controlling the loop, line "11" the value "0" in "evens" for the number of even numbers, line "12" the value "0" in "neg" for the number of negative numbers, line "13" the value "1" in "prod" for the product of negative numbers, line "14" the value "0" in "pos" for the number of positive numbers and in line "15" the value "0" in "sum" for the sum of positive numbers.

Program title (line 19)

On line '19' with a 'printf()' conjunction, the title of the program (Mathematical operations with integers) and two escape characters of this line change (\n) are printed from the 'standard' output.

Enter the number of integers (line 22)

On line '22', the function 'scanf()' reads from the 'standard' input the number of integers to be inserted. The function 'printf()' prints from the 'standard' output the corresponding message (line 21).

Print the number of integers (line 24)

On line '24' the contents of the variable 'N' (the number of integers to be entered) are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

(~) One or more integers (lines 25-75)
(~) No integer (lines 75-82)

In lines '25-82' a check for the number of integers entered by the user is implemented by an 'if-else' control command. In particular, if the user enters one or more integers, then the commands of the subsection 'One or more integers' on lines '25-75' will be executed, otherwise if the user does not enter any integer, then the commands of the subsection 'No integer' on lines '75-82' will be executed.

(~) One or more integers (lines 25-75)

On lines '25-75', the 'if' commands of line '25' are executed for the case where the statement 'N > 0' results in a value of 'True' (the user enters at least one integer).

Loop (lines 27-57)

In lines '27-57' a 'while' loop is executed with a repeat command 'while'. loop to input the integer numbers from the "standard" input and calculate the desired results (square of odd, number of even, average of positive and product of negative numbers).
The loop is executed as long as the value of the representation "i < N" results in a value of "True". In other words, the loop is executed until the user has entered all of the integers that the user chose to enter. The variable "i" used as a helper function to control the loop has as its initial value the value "0" entered in the variable declaration (for details see subsections "Variable declaration (lines 8-16)", "Variable initialization (lines 9)") and in the last command of the loop (line "56") its content is increased by "1" with the representation "i++".

Insertion of integers (line 31)

At line '31' the integer number entered by the user is read from the 'standard' input by the function 'scanf()', accompanied by the appropriate message printed from the 'standard' output by the function 'printf()' (line 30) each time the loop is executed.

Printing of integers (line 32)

Line '32' prints from the 'standard' output the contents of the variable 'n' (the integer number entered by the user each time the loop is executed) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

(!) Insert odd number (lines 34-39)
(!) Insert even number (lines 40-44)

In lines '34-44' a check is implemented by an 'if-else' control for the case of an odd or even number.
In particular, if the user enters an odd number, then the commands of the subsection 'Enter odd number' on lines '34-39'
will be executed, otherwise if the
user enters an even number, then the commands of the subsection 'Enter even number' on lines '40-44' will be executed.

(!) Insert odd number (lines 34-39)

Lines '34-39' execute the 'if' statements of line '34' in case the statement 'n % 2 != 0' results in a 'True' value (the user enters an odd number).

Calculation of the square of the odd (line 36)

In line '36', the power of the given odd number squared is calculated using the arithmetic expression 'n * n' and the result is entered in the variable 'pow'.

Printing the square of the redundant (line 37)

Line "37" prints from the "standard" output the contents of the variable "pow" (the power of the given odd number squared) with a "printf()" function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

(!) Insert even number (lines 40-44)

Lines '40-44' contain the 'else' commands (line 40) corresponding to the 'if' of line '34' for the case where the representation in 'n % 2 != 0' results in a 'False' value (the user enters an even number).

Calculation of the number of even numbers (line 42)

In line '42', the number of even numbers entered by the user is calculated using the arithmetic expression 'evens++' (evens = evens + 1) and the result is entered in the variable 'evens', which has as initial value, the value '0' (see paragraph '0'). "Initialisation of variables (line 11)".

(+) Enter a positive number (lines 45-50)
(+) Insert negative number (lines 51-55)

In lines '45-55' a check is implemented by an 'if-else' control for the case of insertion of a positive or negative number.
In particular, if the user enters a positive number, then the commands of the subsection 'Entering a positive number' on lines '45-50' will be executed, otherwise if the user enters a negative number, then the commands of the subsection 'Entering a negative number' on lines '51-55' will be executed.

(+) Enter a positive number (lines 45-50)

Lines '45-50' are used to execute the 'if' commands of line '45

for the case where the statement "n >= 0" results in a value of "True" (the user enters a positive number).

Calculation of the number of positive numbers (line 47)

In line "47", the number of positive numbers entered by the user is calculated using the arithmetic representation "pos++" (pos = pos + 1) and the result is entered in the variable "pos" which has the value "0" as initial value (see "pos + 1"). "Initialization of variables (line 14)".

Calculation of the sum of positive numbers (line 48)

In line "48" the sum of the given positive numbers is calculated with the arithmetic representation "sum + n" and the result is entered in the variable "sum" which has as initial value, the value "0" (cf. "Initialisation of variables (line 15)".

Calculation of the average of positive numbers (line 49)

In line '49', the average of the positive numbers
is calculated using the arithmetic expression '(float) sum / pos' and the result is entered in the variable 'avg'. Since integers are read from the 'standard' input and the variable 'avg', which calculates the average of the positive numbers, is real (float), the integer variables 'sum' and 'pos' are converted to real only for the arithmetic representation of line '49'.

(+) Insert negative number (lines 51-55)

Lines '51-55' contain the 'else' commands (line 51) corresponding to the 'if' of line '45', for the case where the statement 'n >= 0' results in the value 'False' (the user enters a negative number).

Calculation of the number of negative numbers (line 53)

In line '53', the number of negative numbers entered by the user is calculated using the arithmetic expression 'neg++' (neg = neg + 1) and the result is entered in the variable 'neg', which has as initial value, the value '0' (see paragraph 53). "Initialisation of variables (line 12)".

Calculation of the product of negative numbers (line 54)

In line "54" the sum of the given negative numbers is calculated with the arithmetic representation "prod * n" and the result is entered in the variable "prod" which has as initial value, the value "1" (cf. "Initialisation of variables (line 13)".

(-) One or more negative numbers (lines 59-65)
(-) No negative numbers (lines 66-73)

In lines "59-73" a check is implemented with an "if-else" control command for the case of negative numbers to better reflect the desired results.
In particular, if the user enters at least one negative number, then the commands of the subsection 'One or more negative numbers' in lines '59-65' will be executed, otherwise if the
user does not enter a negative number, then the commands of the subsection 'No negative number' in lines '66-73' will be executed.

(-) One or more negative numbers (lines 59-65)

Lines '59-65' execute the 'if' statements of line '59' in case the 'neg != 0' statement results in a 'True' value (the user enters at least one negative number).

Printing the number of even numbers (line 62)

Line '62' prints from the 'standard' output the contents of the variable 'evens' (the number of even numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

Printing the average of positive numbers (line 63)

On line '63', the contents of the variable 'avg' (the average of positive numbers)
are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6f", 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.

Printing the product of negative numbers (line 64)

Line '64' prints from the 'standard' output the contents of the variable 'prod' (the product of negative numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniformity.

a rough alignment of the results.

(-) No negative number (lines 66-73)

Lines 66-73 contain the commands of the 'else' (line 66) corresponding to the 'if' of line 59, for the case where the statement 'n >= 0' results in the value 'False' (the user does not enter a negative number).

Assignment of the value '0' to the variable 'prod' (line 69)

In line "69", the content of the variable "prod" is reduced by "1" with the representation "prod- -", where it originally has the value "1" from the variable declaration (for details see subsection "Initialisation of variables (line 13)"), in order to reflect the result "Product of negatives : 0" as soon as there are no negatives and not falsely the result "Product of negatives : 1".

Print the number of even numbers (line 70)

Line '70' prints from the 'standard' output the contents of the variable 'evens' (the number of even numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

Printing the average of positive numbers (line 71)

On line '71', the contents of the variable 'avg' (the average of positive numbers)
are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6f", 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.

Printing the product of negative numbers (line 72)

Line '72' prints from the 'standard' output the contents of the variable 'prod' (the product of the negative numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is intended to make it uniform.

form alignment of the results.

(~) No integer (lines 75-82)

Lines 75-82 contain the instructions of the 'else' (line 75) corresponding to the 'if' of line 25, for the case where the representation 'N > 0' results in the value 'False' (the user does not enter any integer or incorrectly gives a negative value to the number of integers).

Assigning a value of '0' to the variable 'prod' (line 78)

In line "78", the content of the variable "prod" is reduced by "1" with the representation "prod- -", where it originally has the value "1" from the variable declaration (for details see subsection "Initialisation of variables (line 13)"), in order to reflect the result "Product of negatives : 0" since there are no negatives and not falsely the result "Product of negatives : 1".

Print the number of even numbers (line 79)

On line '70', the contents of the variable 'evens' (the number of even numbers)
are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

Printing the average of positive numbers (line 80)

On line '71', the contents of the variable 'avg' (the average of positive numbers)
are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6f", 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.

Printing the product of negative numbers (line 81)

On line '72', the contents of the variable 'prod' (the product of negative numbers)
are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is '%20d', i.e., the result will be printed immediately.

perhaps after "20" blank characters. This is to ensure uniform alignment of the results.

EXAMPLES

Example 1 (N > 0)

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

Mathematical operations with integers

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

Enter N integers : 6

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

The number of integers : [ 6]

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

Enter integer : 8

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

The integer : [ 8]


Even number
------------------------------------------------------------------------------

Enter integer : -11

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

The integer : [ -11]

Square of redundancy : [ 121]

Redundant number
------------------------------------------------------------------------------

Enter integer : 0
```

The integer : [0]

Even number

Enter integer : 57

The integer : [57]

Square odd : [3249]

Redundant number

Enter integer : -12

The integer : [-12]

Even number

Enter integer : 5

The integer : [5]

Square of redundancy : [25]

Redundant number

Number of even : [3]
Average positive : [17.500000]
Product of negatives : [132]

Example 2 (N < 0 or N == 0)

==========

Mathematical operations with integers

==========

Enter N integers : 0

The number of integers : [0]

Number of even : [0]

Average positive : [0.000000]

Product of negatives : [0]

OBSERVATIONS

The correctness of the results in the above examples was also checked (with one small observation) using a standard calculator and no difference was found.
In detail, we have two examples where each corresponds to special conditions found within the corresponding command "if (N > 0)" (line 22) and of "else" (line 71) :

Example 1 (N > 0)

Represents the case where the user enters one or more integers.
In detail, the user enters '6' integers of which '3' are even numbers (8, 0, -12) and the other '3' are odd numbers (-11, 57, 5). Of these, "4" are positive (0, 5, 8, 57) and "2" are negative (-11, -12).

The relevant observation is that in this program the number "0" is considered to be even and positive. The results :

Square of odd "-11" : 121 {(-11)^2}
Square of odd "57" : 3249 (57^2)
Square of odd "5" : 25 (5^2)
Number of harts : 3 (8, 0, -12)
Average positive : 17.500000 {(0 + 5 + 8 + 57) / 4}
Product of negatives : 132 (-11 * -12)

Example 2 (N < 0 or N == 0)

Illustrates the case where the user does not enter any integer or accidentally enters a negative value in the number of integers he wants to enter.
Clearly, there are no even, odd, positive, negative numbers. The results :

Number of arities : 0
Average of positives : 0.000000
Product of negatives : 0

ITEM 3B

REFERENCE "MathsWithIntegersInLoopB.c"

The "MathsWithIntegersInLoopB.c" (Source Code) and the "MathsWithIntegersInLoopB.c" documentation (Objective, Structure, Variables, Crosswalk, Examples, Examples, Remarks) answer the question of the question "Topic 3B". More specifically, in the case where the user enters as many integers as they want and the loop stops until the user enters the value "0".

PROGRAM "MathsWithIntegersInLoopB.c"

```
1 #include <stdio.h>
2
3 int main (int argc, int **argv)
4
5 {
6  system ("chcp 1253"); system ("chcp 1253");
7
```

```
8  int n; // Variable declaration

9  int pow;
10 int evens = 0; // Initialize variables
11 int neg = 0; // Initialize variables
12 int prod = 1; // Initialize variables
13 int pos = 0; // Initialize variables
14 int sum = 0; // Initialize variables
15 float avg;
16
17 printf
("=================================================\n\n");
18 printf ("Mathematical operations with integers\n\n"); // Program
title
19 printf
("=================================================\n\n");
   20do /* Loop */
   21{
22       printf ("-----------------------------------------------
--\n\n");
23       printf ("Enter integer : ");;
24       scanf ("%d", &n); // Insert the integers
25       printf ("\n-------------------------------------------
----\n\n");
26       printf ("The integer : [%20d]\n\n", n); // Print the
integer numbers
27       if (n != 0) /* (~) Insert value other than "0" */
28       {
29             if (n % 2 != 0) /* (!) Insert odd number */
30             {
31                   pow = n * n; // Calculate the square of the
odd
32                   printf ("Square of odd : [%20d]\n\n", pow); //
Print the square of the odd
33                   printf ("\nExact number\n");;
34             }
35             else /* (!) Insert even number */
36             {
```

```
37                    evens++; // Calculating the number of even
numbers
38                    printf ("\nEven number\n");;
39              }
40              if (n > 0) /* (+) Insert positive number */
41              {
42                    pos++;      //  Calculate  the  number  of
positive numbers
43                    sum = sum + n; // Calculate the sum of positive
numbers
44                    avg  =  (float)  sum  /  pos;  // Calculate  the
average of the positive numbers
45              }
46              else /* (+) Insert negative number */
47              {
48                    neg++;      //  Calculate  the  number  of
negative numbers
49                    prod = prod * n; // Calculate the product of
negative numbers
50              }
51        }
52        else /* (~) Enter the value "0" */
53        {
54              printf ("\n~End of loop~\n");
55         }
   56}
57 while (n != 0);;
58
59 if (neg != 0) /* (-) One or more negative numbers */
   60{

61        printf ("-----------------------------------------------
--\n\n");
62        printf ("Number of even numbers : [%20d]\n", evens); //
Print the number of even numbers
63        printf ("Average of positive numbers : [%20.6f]\n", avg);
// Print the average of positive numbers
64        printf ("Product of negative numbers : [%20d]\n", prod);
// Print the product of negative numbers
```

```
   65}
66 else /* (-) No negative number */
   67{
68       printf ("-----------------------------------------------
--\n\n");
69       prod--; // Assign value "0" to variable "prod"
70       printf ("Number of even numbers : [%20d]\n", evens); //
Print the number of even numbers
71       printf ("Average of positive numbers : [%20.6f]\n", avg);
// Print the average of positive numbers
72       printf ("Product of negative numbers : [%20d]\n", prod);
// Print the product of negative numbers
   73}
74
75 return 0;
76 }
```

DOCUMENTATION "MathsWithIntegersInLoopB.c"

REQUESTED

The program "MathsWithIntegersInLoopB.c" achieves the following operations:

a) Reads from the "standard" input "N" integers until it reads the number "0".

b) Calculates the square of the odd numbers.

c) Calculates the number of even numbers.

(d) Calculates the average of positive numbers.

(e) Calculate the product of negative numbers.

(f) Prints the desired results from the "standard" output accompanied by the appropriate messages.

MODEL

In order to implement the requirement, the libraries :

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 contents of the respective variables. Also, "printf()" was used to print characteristic messages for optimal understanding of the source code.

In addition, the characteristic operators :

a) numeric : +, *, %

(b) relative : !=, >, <, ==

c) assignment : =

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

(e) scaling : variable++

(f) decrease : variable--.
The control commands :

(a) if - else

The iteration commands :

(a) do - while

VARIABLES

Integer variables (of type "int")

n (Each integer number inserted in the loop)

pow (The square of the odd numbers)

evens (The number of even numbers)

pos (The number of positive numbers)

sum (The sum of positive numbers)

neg (The number of negative numbers)

prod (The product of the negative numbers)

Real variables (of type "float")

avg (The average of the positive numbers)

DISCUSSION

Declaration of variables (lines 8-15)

The program starts with the declaration of variables on lines "15" (see subsection "Variables" for details).

Initialisation of variables (lines 10, 11, 12, 13, 14)

Some initialisation of variables is done in the above lines. In more detail, in line "10" the value "0" in "evens" for the number of even numbers, in line "11" the value "0" in "neg" for the number of negative numbers, in line '12', the value '1' in 'prod' for the product of the negative numbers, in line '13', the value '0' in 'pos' for the number of positive numbers and in line '14', the value '0' in 'sum' for the sum of the positive numbers.

Program title (line 18)

On line '18', a 'printf()' function prints from the 'standard' output the title of the program (mathematical operations with integers) and two escape characters for this line change (\n).

Loop (lines 20-57)

On lines '20-57', a loop is executed with a 'do - while' instruction to insert the integers from the 'standard' input and to calculate the desired results (square of an odd number, number of even numbers, average of positive and product of negative numbers).
The loop will be executed at least once, regardless of what value the controlling argument produces, and is executed as long as the value of the argument "n != 0" produces a result of a value of "True". In other words, the loop is executed until the user enters the number "0". The control of the representation controlling the loop is performed at line "57"

Input of integers (line 24)

At line '24' the integer number entered by the user is read from the 'standard' input by the function 'scanf()', accompanied by the appropriate message printed from the 'standard' output by the function 'printf()' (line 23) each time the loop is executed.

Printing of integers (line 26)

At line "26" the content of the variable "n" (the integer number entered by the user each time the loop is executed) is printed from the "standard" output with a "printf()" function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

(~) Enter a value other than '0' (lines 27-51)
(~) Enter the value '0' (lines 52-55)

In lines '27-55' a check is implemented with an 'if-else' control instruction for the case where the number '0' is inserted which terminates the loop.
In particular, if the user does not enter the number '0', then the commands of the subsection 'Enter a value other than '0" in lines '27-51' will
be executed, otherwise if the
user enters the number '0', then the commands of the subsection 'Enter the value '0" in lines '52-55' will be executed.

(~) Enter a value other than '0' (lines 27-51)

Lines '27-51' execute the 'if' commands of line '27' (n != 0)' (line 23) in case the representation 'n != 0' results in a value of 'True' (the user does not enter the number '0
').
(!) Enter an odd number (lines 29-34)
(!) Enter an even number (lines 35-39)

In lines '29-39' a check for the case of an odd or even number is implemented by an 'if-else' control command.
In particular, if the user enters an odd number, then the commands of the subsection 'Enter odd number' on lines '29-34'
will be executed, otherwise if the
user enters an even number, then the commands of the subsection 'Enter even number' on lines '35-39' will be executed.

(!) Insert odd number (lines 29-34)

Lines '29-34' are used to execute the 'if' commands of line '29' in case the statement 'n % 2 != 0' results in a 'True' value (the user enters an odd number).

Calculation of the square of the odd number (line 31)

In line '31', the arithmetic expression 'n * n' is used to calculate

raising the power of the given odd number to the square and the result is entered in the variable "pow".

Printing the square of the redundant (line 32)

At line "32" the content of the variable "pow" (the power of the given odd number squared) is printed from the "standard" output with a "printf()" function accompanied by the appropriate message, each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

(!) Insert even number (lines 35-39)

Lines '35-39' contain the 'else' commands (line 35) corresponding to the 'if' of line '29' for the case where the representation in 'n % 2 != 0' results in a 'False' value (the user enters an even number).

Calculation of the number of even numbers (line 37)

In line '37', the number of even numbers entered by the user is calculated using the arithmetic expression 'evens++' (evens = evens + 1) and the result is entered in the variable 'evens', which has as initial value, the value '0' (see paragraph '0'). "Initialisation of variables (line 10)".

(+) Enter a positive number (lines 40-45)
(+) Insert negative number (lines 46-50)

In lines '40-50' a check for the case of insertion of a positive or negative number is implemented by an 'if-else' control command.
In particular, if the user enters a positive number, then the commands of the subsection 'Entering a positive number' on lines '40-45'
will be executed, otherwise if the
user enters a negative number, then the commands of the subsection 'Entering a negative number' on lines '46-50' will be executed.

(+) Enter a positive number (lines 40-45)

Lines '40-45' are used to execute the 'if' commands of line '40' in the case where the statement 'n > 0' results in a 'True' value (the user enters a positive number).

Calculation of the number of positive numbers (line 42)

In line "42" is calculated with the arithmetic representation "pos++"

(pos = pos + 1) the number of positive numbers entered by the user and the result is entered in the variable "pos" which has as initial value, the value "0" (see. "Initialization of variables (line 13)".

Calculation of the sum of positive numbers (line 43)

In line "43" the sum of the given positive numbers is calculated with the arithmetic representation "sum + n" and the result is entered in the variable "sum" which has the value "0" as initial value (see "sum + n"). "Initialisation of variables (line 14)".

Calculation of the average of positive numbers (line 44)

In line '44', the average of the positive numbers is calculated using the arithmetic expression '(float) sum / pos' and the result is entered in the variable 'avg'. Since integers are read from the 'standard' input and the variable 'avg', which calculates the average of the positive numbers, is real (float), the integer variables 'sum' and 'pos' are converted to real only for the arithmetic representation of line '44'.

(+) Insert negative number (lines 46-50)

Lines '46-50' contain the 'else' commands (line 46) corresponding to the 'if' of line '40', in the case where 'n > 0' results in the value 'False' (the user enters a negative number).

Calculation of the number of negative numbers (line 48)

In line '48', the number of negative numbers entered by the user is calculated using the arithmetic expression 'neg++' (neg = neg + 1) and the result is entered in the variable 'neg', which has as initial value, the value '0' (see paragraph '48'). "Initialisation of variables (line 11)".

Calculation of the product of negative numbers (line 49)

Στη γραμμή «49» υπολογίζεται με την αριθμητική παράσταση «prod * n» το άθροισμα των δοθέντων αρνητικών αριθμών και το αποτέλεσμα καταχωρείται στη μεταβλητή «prod» που έχει για αρχική τιμή, την τιμή «1» (βλ. "Initialization of variables (line 12)".

(~) Enter the value "0" (lines 52-55)

Lines "52-55" execute the "else" commands (line 52) which

corresponds to the 'if' of line '27' for the case where the statement 'n != 0' results in the value 'False' (the user enters the number '0
'). More specifically, it checks whether the user has entered the value '0' in the variable 'n' and whether the statement 'n != 0" results in the value "False", then the message "~End of loop~" is printed from the "standard" output with a "printf()" function with two characters of line change escape (\n) on line "54" indicating the end of the loop, as the user has entered the terminal value "0".

(-) One or more negative numbers (lines 59-65)
(-) No negative numbers (lines 66-73)

In lines "59-73" a check is implemented with an "if-else" control command for the case of negative numbers to better reflect the desired results.
In particular, if the user enters at least one negative number, then the commands of the subsection 'One or more negative numbers' in lines '59-65' will be executed, otherwise if the
user does not enter a negative number, then the commands of the subsection 'No negative number' in lines '66-73' will be executed.

(-) One or more negative numbers (lines 59-65)

Lines '59-65' execute the 'if' statements of line '59' in case the 'neg != 0' statement results in a 'True' value (the user enters at least one negative number).

Printing the number of even numbers (line 62)

Line '62' prints from the 'standard' output the contents of the variable 'evens' (the number of even numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

Printing the average of positive numbers (line 63)

On line '63', the contents of the variable 'avg' (the average of positive numbers)
are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6f", 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 that the results are evenly stacked.

Printing the product of negative numbers (line 64)

Line '64' prints from the 'standard' output the contents of the variable 'prod' (the product of the negative numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

(-) No negative number (lines 66-73)

Lines 66-73 contain the 'else' (line 66) corresponding to the 'if' of line 59, for the case where the statement 'n >= 0' results in the value 'False' (the user does not enter a negative number).

Assignment of the value '0' to the variable 'prod' (line 69)

In line "69", the content of the variable "prod" is reduced by "1" with the representation "prod- -", where it originally has the value "1" from the variable declaration (for details see subsection "Initialisation of variables (line 13)"), in order to reflect the result "Product of negatives : 0" as soon as there are no negatives and not falsely the result "Product of negatives : 1".

Print the number of even numbers (line 70)

Line '70' prints from the 'standard' output the contents of the variable 'evens' (the number of even numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

Printing the average of positive numbers (line 71)

On line '71', the contents of the variable 'avg' (the average of positive numbers)
are printed from the 'standard' output with a 'printf()' function accompanied by the appropriate message. It is worth noting that the formatting alphanumeric is "%20.6f", 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 that the results are evenly stacked.

Printing the product of negative numbers (line 72)

Line '72' prints from the 'standard' output the contents of the variable 'prod' (the product of the negative numbers) with a 'printf()' function accompanied by the appropriate message each time the loop is executed. It is worth noting that the formatting alphanumeric is "%20d", i.e., the result will be printed immediately after "20" blank characters. This is to ensure uniform alignment of the results.

EXAMPLES

Example 1

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


Mathematical operations with integers

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


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

Enter integer : 2

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

The integer : [ 2]


Even number
------------------------------------------------------------------------------------------

Enter integer : -3
```

The integer : [-3]

Square of redundancy : [9]

Redundant number

Enter integer : -4

The integer : [-4]

Even number

Enter integer : 11

The integer : [11]

Square of redundancy : [121]

Redundant number

Enter integer : -65

The integer : [-65]

Square of redundancy : [4225]

Redundant number

Enter integer : 72

The integer : [72]

Even number

Enter integer : 0

The integer : [0]

~End of loop~

Number of even : [3]

Average positive : [28.333334]

Negative product : [-780]

Example 2

==========

Mathematical operations with integers

==========

Enter integer : 0

The integer : [0]

~End of loop~

Number of even : [0]

Average positive : [0.000000]

Product of negatives : [0]

OBSERVATIONS

The correctness of the results in the above examples was also checked using a standard calculator and no difference was found except for some less significant digits in a result with a decimal part (see "Example 1"), which does not greatly affect the correctness of the results.
In detail, we have two examples where the user enters as many integers as desired.

Example 1

In this example, the user enters the numbers (2, -3, -4, 11, -65, 72). Of these, "3" are even (2, -4, 72) and the remaining "3" are odd (-3, 11, -65). The "3" are positive (2, 11, 72) and the remaining "3" are negative (-3, -4, -65).
The user enters at the end and the number "0" which marks the end of the loop, clearly the end of the input of integer numbers from the "standard" input.

Square of odd "-3" : 9 {(-3)^2}
Square of odd "11" : 121 (11^2)
Square of odd "-65" : 4225 {(-65^2)
Number of arcs : 3 (2, -4, 72)
Average of positives : 28.333334 {(2, 11, 72) / 3} /* (28.333333) */
Product of negatives : -780 (-3 * -4 * -65)

Example 2

In this example, the user enters only the number "0" which marks the end of the loop, clearly the end of entering integer numbers from the "standard" input.
Therefore, there are no even, odd, positive, negative numbers. Clearly, the results are :

Number of even numbers : 0
Average positive : 0.000000
Product of negatives : 0

ITEM 4

NOTE "Stars.c"

The "Stars.c Program" (Source Code) and the "Documentation "Stars.c"" (Objective, Structure, Variables, Crossing, Examples,

Comments) answer the question of the question "Theme 4".

PROGRAM "Stars.c"

```
1 #include <stdio.h>

2

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

4

5 {

6system  ("chcp 1253");;

7

8int     lines; // Variable declaration

   9int i, j, k;

10

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

12printf       ("Shapes with asterisks\n\n"); // Program title

   13printf
("================================================\n\n");

14printf       ("Enter number of lines : ");;

15scanf  ("%d", &lines); // Insert the number of lines

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

17for    (i = 1 ; i <= lines ; i++) /* 1st figure */

   18{

19       for (j = 1 ; j <= i ; j++)

20       {

21             printf ("*");;

22       }

23       printf ("\n");;

   24}

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

26for    (i = 0 ; i < lines ; i++) /* 2nd figure */

   27{

28       for (j = lines - 1 ; j > i ; j--)
```

```
29        {
30              printf (" ");;
31        }
32        for (k = 0 ; k <= j ; k++)
33        {
34              printf ("*");;
35        }
36        printf ("\n");;
   37}
38        printf ("\n-----------------------------------------------
----\n\n");
39for      (i = 0 ; i < lines ; i++) /* 3rd figure */
   40{
41        for (j = lines - 1 ; j > i ; j--)
42        {
43              printf (" ");;
44        }
45        for (k = 0 ; k <= j * 2 ; k++)
46        {
47              printf ("*");;
48        }
49        printf ("\n");;
   50}
51 printf    ("\n-------------------------------------------------
\n\n");
52for      (i = 0 ; i < lines ; i++) /* 4th figure */
   53{
54        for (j = 0 ; j < lines ; j++)
55        {
56              if (i == 0 || j == 0 || i == lines - 1 || j == lines
- 1) /* (~) Point within the perimeter of the figure */
57              {
58                    printf ("*");;
59              }
60              else /* (~) Point outside the perimeter of the shape
*/
```

```
61              {
62                      if  (i  ==  j)  /*  (!)  Point  inside  the  main
diagonal */
63                      {
64                              printf (".");;
65                      }
66                      else /* (!) Point outside the main diagonal */
67                      {
68                              if  (i  +  j  ==  lines  -  1)  /*  (+)  Point
within the secondary diagonal */
69                              {
70                                      printf (".");;
71                              }
72                              else /* (+) Point outside the secondary
diagonal */
73                              {
74                                      printf (" ");;
75                              }
76                      }
77              }
78      }
79      printf ("\n");;
   80}
81
82return      0;
83 }
```

NOTE "Stars.c

REQUESTED

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

a) Reads from the "standard" input "N" number of lines to display "4" shapes with asterisks.

b) Plots this shape, for example, for "5" lines:

```
*
**
***
****
*****
```

c) Illustrate this figure, for example, for "5" lines :

```
 **
 ***
****
*****
```

d) Illustrate this figure, for example, for "5" lines :

```
   *
  ***
 *****
*******
*********
```

e) Illustrate this figure, for example, for "5" lines:

```
*****
*. .*
* . *
*. .*
*****
```

MODEL

In order to implement the request, the libraries :

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

In addition, the characteristic operators :

(a) numeric : -, +, *

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

c) assignment : =

(d) logical : ||

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

(f) scaling : variable++

(g) diminishing : variable--

The control commands :

(a) if - else

The iteration commands :

(b) for

VARIABLES

Integer variables (of type "int")

lines (the number of lines of the shapes)

i (Counter for loop control)

j (Counter for loop control)

k (Counter for loop control)

DISCUSSION

Declaration of variables (lines 8-9)

The program starts with the declaration of the variables in lines '8-9' (see subsection 'Variables' for details).

Program title (line 12)

On line "12" with a "printf()" conjunction, the program title (Shapes with asterisks) and two escape characters of this
line change (\n) are printed from the "standard" output.

Enter the number of lines (line 15)

Line '15' is read from the 'standard' input with a function "scanf()" the number of lines of the "4" shapes to be plotted accompanied by the appropriate message printed from the "standard" output with a "printf()" function (line 14). The result is entered in the variable "lines" and it is worth noting that the "4" shapes to be plotted will have the same number of lines as the user enters (see subsection "Required").

1ο figure (lines 17-24)

In lines "17-24" the syntax for the visualization of the first schema is implemented.
In more detail, we have a loop with a loopback instruction "for" (lines 17-24) that is executed as long as the representation "i <= lines" results in a value of "True" (condition true). Inside the loop, there is another loop with a loopback instruction "for" (lines 19-22), as long as the representation "j <= i" produces a result of a value "True" (condition true), and a function "printf()" (line 23) that prints the escape character of the line change (\n). In the second loop (lines 19-22), a 'printf()' function (line 21) is executed which prints the asterisk character (*)
. The loops work as follows :

(a) First loop

```
  17 for (i = 1 ; i <= lines ; i++)
18 {
19for  (j = 1 ; j <= i ; j++)
   20{
21           printf ("*");;
22     }
23printf     ("\n");;
24 }
```

In the first loop (lines 17-24) we have the variable "i" (of type "int") used as a helper to control it. Its initial value is "1" and the loop is executed until "i", which is incremented by "1" with the representation "i++" each time an execution of the loop ends, contains a value greater than the number of rows of the shapes entered by the user (for details see. "Entering the

number of lines (line 15)"). At each execution of the loop, the second loop (lines 19-22) and the function "printf()" (line 23) are executed, which prints the escape character of the line change (\n). The loop is executed as many times as the number of lines of the schemas entered by the user from the "standard" input.

b) Second loop

```
19 for   (j = 1 ; j <= i ; j++)
20 {
21       printf ("*");;
22 }
```

In the second loop (lines 19-22) we have the variable "j" (of type "int") used as a helper to control it. Its initial value is "1" and the loop is executed until "j", which is incremented by "1" with the representation "j++" each time an execution of the loop ends, contains a value greater than the auxiliary "i" of the first loop (lines 17-24). The loop is executed each time the first loop is executed, and the number of times the loop is executed increases by "1" as the auxiliary "i" is increased by "1" by the "i++" representation from the first loop. At each execution of the loop, the function "printf()" (line 21) is executed which prints the asterisk character (*).

For example, the user enters "5" lines which is entered in the variable "lines". Therefore, the first loop is executed "5" times, while the second loop is executed :

a) In the 1η iteration of the first loop, "1" time (*\n).

b) In the 2η iteration of the first loop, "2" times (**\n).

(c) In the 3η iteration of the first loop, "3" times (***\n).

d) In the 4η iteration of the first loop, "4" times (****\n).

e) In the 5η iteration of the first loop, "5" times (*****\n).

The figure is illustrated as follows :

```
*
**
***
****
*****
```

2º figure (lines 26-37)

In lines "26-37" the syntax for the representation of the second scheme.
In more detail, we have a loop with a loopback instruction "for" (lines 26-37) that is executed as long as the representation "i < lines" results in a value of "True". Within the loop, two other loops are executed with the repeat instruction "for". One for as long as the representation 'j > i' (lines 28-31) produces a result of a value 'True' (condition true) and the other for as long as the representation 'k <= j' (lines 32-35) produces a result of a value 'True' (condition true). A function 'printf()' (line 36) is also executed which prints the line change escape character (\n). In the second loop (lines 28-31) a function 'printf()' (line 30) is executed which prints the space character (" "), while in the third loop a function 'printf()' (line 34) is also executed which prints the asterisk character (*).
The loops operate as follows :

(a) First loop

```
26 for (i = 0 ; i < lines ; i++)
27 {
28 for   (j = lines - 1 ; j > i ; j--)
29 {
30       printf (" ");;
31 }
32 for   (k = 0 ; k <= j ; k++)
33 {
34       printf ("*");;
35 }
36 printf ("\n");;
37 }
```

In the first loop (lines 26-37) we have the variable "i" (of type "int") used as a helper to control it.
Its initial value is "0" and the loop is executed until the auxiliary, which is incremented by "1" with the representation "i++" each time an execution of the loop ends, contains a value equal to or greater than the number of rows of shapes entered by the user (line 15). At each execution of the loop, the second loop (lines 28-31), the third loop (lines 32-35) and a "printf()" function (line 36) are executed which prints the escape character of the line change (\n).
The loop is executed as many times as the number of lines of the shapes entered by the user from the "standard" input.

(b) Second loop

```
28        for (j = lines - 1 ; j > i ; j--)
29        {
30              printf (" ");;
31        }
```

In the second loop (lines 28-31) we have the variable "j" (of type "int") used as a helper to control it.
Its initial value is the value "lines - 1", i.e., the number of lines entered by the user minus "1" and the loop is executed until the auxiliary, which is decremented by "1" with the representation "j--" each time an execution of the loop ends, contains a value equal to or less than the auxiliary "i" of the first loop, which is incremented by "1" with the representation "i++".

The loop at the beginning is executed as many times as the number of lines of the schemas entered by the user from the "standard" input reduced by "1" (line 15) and, of course, when the first loop is executed (lines 26-37). The times it is executed are reduced by "1", as from the second loop the auxiliary "j" is reduced by "1" by the representation "j--". At each execution of the loop, the function "printf()" (line 30) is executed which prints the blank character (" ").

(c) Third loop

```
32 for (k = 0 ; k <= j ; k++)
33        {
34              printf ("*");;
35        }
```

In the third loop (lines 32-35) we have the variable "k" (of type "int") used as an auxiliary to control it.
Its initial value is "0" and the loop is executed until the auxiliary which is incremented by "1" with the representation "k++" each time an execution of the loop ends, contains a value greater than the auxiliary "j" of the second loop which is decremented by "1" with the representation "j--". The loop is executed each time the first loop is executed (lines 26-37), and the number of times the loop is executed increases by "1" as the auxiliary "i" is increased by "1" from the first loop by the "i++" representation, and after the second loop is executed so that the auxiliary "j" is decreased by "1" by the "j--" representation found in the "k <= j" representation controlling the third loop. At each execution of the loop, the function "printf()" (line 34) is executed which prints the asterisk character (*).

For example, the user enters "5" lines which is entered in the variable "lines". Therefore, the first loop is executed "5" times, while, the second and third loops are executed :

a) In the 1η iteration of the first loop it is executed
:
"4" times the 2ος loop, "1" times the 3ος loop (*\n).

b) In the 2η iteration of the first loop it is executed :

"3" times the 2ος loop, "2" times the 3ος loop (**\n).

c) In the 3η iteration of the first loop, it is executed :

"2" times the 2ος loop, "3" times the 3ος loop (***\n).

(d) In the 4η iteration of the first loop, it is executed :

"1" times the 2ος loop, "4" times the 3ος loop (****\n).

(e) In the 5η iteration of the first loop, it is executed :

"0" times the 2ος loop, "5" times the 3ος loop (*****\n).

The scheme is illustrated as follows :

```
    *
   **
  ***
 ****
*****
```

3ο figure (lines 39-50)

In lines "39-50" the syntax for the illustration of the third figure is implemented.

In more detail, we have a loop with a loopback instruction "for" (lines 39-50) that is executed as long as the representation "i < lines" results in a value of "True" (condition true). Within the loop, two other loops are executed with the repeat instruction "for". One for as long as the instance "j > i" (lines 41-44) produces a result of a value "True" (condition true) and the other for as long as the instance "k <= j * 2" (lines 45-48) produces a result of a value "True" (condition true). A function 'printf()' (line 49) is also executed which prints the line change escape character (\n). In the second loop (lines 41-44) a function 'printf()' (line 43) is executed which prints the space character (" "), while in the third loop a function 'printf()' (line 47) is also executed which prints the asterisk character (*).

The loops work as follows :

a) First loop

```
39 for (i = 0 ; i < lines ; i++)
40 {
41 for   (j = lines - 1 ; j > i ; j--)
42 {
43       printf (" ");;
44 }
45 for (k = 0 ; k <= j * 2 ; k++)
46 {
47       printf ("*");;
48 }
49 printf ("\n");;
50 }
```

In the first loop (lines 39-50) we have the variable "i" (of type "int") used as a helper to control it.
Its initial value is "0" and the loop is executed until the auxiliary "i", which is incremented by "1" with the representation "i++" each time an execution of the loop ends, contains a value equal to or greater than the number of rows of shapes entered by the user (line 15). At each execution of the loop, the second loop (lines 41-44), the third loop (lines 45-48) and a 'printf()' function (line 49) are executed which prints the escape character of the line change (\n).

The loop is executed as many times as the number of lines of the shapes entered by the user from the "standard" input.

(b) Second loop

```
41       for (j = lines - 1 ; j > i ; j--)
42       {
43             printf (" ");;
44       }
```

In the second loop (lines 41-44) we have the variable "j" (of type "int") used as a helper to control it.
Its initial value is the value "lines - 1", i.e., the number of lines entered by the user minus "1" and the loop is executed until the counter decremented by "1" with the representation "j--" each time an execution of the loop ends, contains a value equal to or less than the auxiliary "i" of the first loop which is incremented by "1" with the representation "i++" after each execution.
The loop at the beginning is executed as many times as the number of rows

of shapes entered by the user from the "standard" input minus by "1" (line 15) and, of course, when the first loop is executed (lines 39-50). The times it is executed are decremented by "1", as from the first loop the counter "j" is decremented by "1" by the representation "j- -". At each execution of the loop, the function "printf()" (line 43) is executed, which prints the blank character (" ").

(c) Third loop

```
45 for (k = 0 ; k <= j * 2 ; k++)
46        {
47              printf ("*");;
48        }
```

In the third loop (lines 45-48) we have the variable "k" (of type "int") used as an auxiliary to control it.
Its initial value is "0" and the loop is executed until the counter incremented by "1" with the representation "k++" each time an execution of the loop ends, contains a value greater than the representation "j * 2" conditional on the auxiliary "j" of the second loop decremented by "1" with the representation "j--".

The loop is executed each time the first loop is executed (lines 39-50) and the number of times executed is increased by "2" as from the first loop the auxiliary "i" is increased by "1" by the representation "i++" and after the second loop is executed, so that the auxiliary "j" is decreased by "1" by the representation "j--" which is in the representation "k <= j * 2" controlling the third loop. At each execution of the loop, the function "printf()" (line 47) is executed which prints the asterisk character (*). For example, the user enters "5" lines which is entered in the variable "lines". Therefore, the first loop is executed "5" times, while, the second and third loops are executed :

a) In the 1η iteration of the first loop,
the 2ος loop is executed : "
4" times, the 3ος loop (*\n) is executed "1" time.

b) In the 2η iteration of the first loop it is executed :

"3" times the 2ος loop, "3" times the 3ος loop (***\n).

c) In the 3η iteration of the first loop, it is executed :

"2" times the 2ος loop, "5" times the 3ος loop (*****\n).

d) In the 4η iteration of the first loop, it is executed :

"1" times the 2ος loop, "7" times the 3ος loop (******\n).

e) In the 5η iteration of the first loop it is executed :

"0" times the 2ος loop, "9" times the 3ος loop (*******\n).

The scheme is illustrated as follows :

```
  *
 ***
*****
*******
*********
```

4ο figure (lines 52-80)

In lines "52-80" the syntax for the illustration of the fourth figure is implemented.
More specifically, we have a loop with a loopback instruction "for" (lines 52-80) that is executed as long as the representation "i < lines" results in a value of "True" (condition true). Inside the loop there is another loop with a loopback instruction 'for' (lines 54-78), as long as the representation 'j < lines' produces a value 'True' (condition true), a function 'printf()' (line 79) which prints the escape character of the line change (\n) and a control instruction 'if-else' (lines 56-76).

The loops work as follows :

(a) First loop

```
52 for (i = 0 ; i < lines ; i++)
53 {
54       for (j = 0 ; j < lines ; j++)
55       {
56       if (i == 0 || j == 0 || i == lines - 1 || j == lines - 1)
/* Point within the perimeter of the shape */
57             {
58                   printf ("*");;
59             }
60             else // Point outside the perimeter of the shape
61             {
62                   if (i==j) /* Point within the main diagonal */
63                   {
64                         printf (".");;
65                   }
```

```
66                    els
e     /* Point outside the main diagonal */
67                    {
68                          if (i + j == lines - 1) /* Point within
   the secondary diagonal */
69                          {
70                                printf (".");;
71                          }
72                          else
   /* Point outside the secondary diagonal */
73                          {
74                                printf (" ");;
75                          }
76                    }
77              }
78        }
79        printf ("\n");;
   80}
```

In the first loop (lines 52-80) we have the variable "i" (of type "int") used as an auxiliary to control it.
Its initial value is "0" and the loop is executed until the auxiliary, which is incremented by "1" with the representation "i++" each time an execution of the loop ends, contains a value equal to or greater than the number of rows of shapes entered by the user (line 15). At each execution of the loop, the second loop (lines 54-78) and the function "printf()" (line 79) are executed, which prints the escape character of the line change (\n).
The loop is executed as many times as the number of lines of the schemas entered by the user from the 'standard' input.

(b) Second loop

```
54        for (j = 0 ; j < lines ; j++)
55        {
56        if (i == 0 || j == 0 || i == lines - 1 || j == lines - 1)
/* Point within the perimeter of the shape */
57              {
58                    printf ("*");;
59              }
60              else // Point outside the perimeter of the shape
61              {
```

```
62                    if (i==j)
     /* Point within the main diagonal */
63                    {
64                          printf (".");;
65                    }
66                    else
   /* Point outside the main diagonal */
67                    {
68                          if (i + j == lines - 1)
   /* point within the secondary diagonal */
69                          {
70                                printf (".");;
71                          }
72                          else
   /* Point outside the secondary diagonal */
73                          {
74                                printf (" ");;
75                          }
76                    }
77              }
78       }
```

In the second loop (lines 54-78) we have the variable "j" (of type "int") used as a helper to control it.
Its initial value is "0" and the loop is executed until the counter, incremented by "1" with the representation "j++" each time an execution of the loop ends, contains a value equal to or greater than the number of rows of shapes entered by the user (line 15). At each execution of the loop, the "if-else" command is executed that checks the position of a point in the square matrix that is the fourth shape with coordinates [i,j] to print the appropriate characters (for details see "(~) Point inside the perimeter of the shape (lines 56-59)", "Point outside the perimeter of the shape (lines 60-77)", "Point inside the main diagonal (lines 62-65)", "Point outside the main diagonal (lines 66-76)", "Point inside the secondary diagonal (lines 68-71)", "Point outside the secondary diagonal (lines 72-75)"). The loop is executed as many times as the number of lines of the shapes entered by the user from the "standard" input each time the first loop is executed (lines 52-80). Therefore, it is executed "lines * lines" times. ***(Page 59)**

(~) Point inside the perimeter of the shape (lines 56-59)
(~) Point outside the perimeter of the shape (lines 60-77)

On lines '56-77' a single 'if-else' control is implemented with an 'if-else' control.

check for the case where the point with coordinates [i,j] is located inside or outside the perimeter of the shape of the square matrix.

In detail, if the statement "i == 0 || j == 0 || i == lines - 1 || j == lines - 1" results in a value of "True" (true condition), then the commands of subsection "(~) Point inside the perimeter of the shape" on lines "56-59" will be executed, otherwise if it results in a value of "False", then the commands of subsection "(~) Point outside the perimeter of the shape" on lines "60-77" will be executed.

(~) Point inside the shape perimeter (lines 56-59)

On lines '56-59', the commands of 'if (i == 0 || j == 0 || i == lines - 1 || j == lines - 1)' (line 56) are executed for the case where the point with coordinates [i,j] is within the perimeter of the shape of the square matrix.
More specifically, if the statement 'i == 0 || j == 0 || j == 0 || i == lines - 1 || j == lines - 1' produces a value of 'True', then the function 'printf()' (line 58) will be executed which prints the asterisk character (*). This is a logical representation with the logical operator "||" (OR), so it will result in a value of "True" if at least one of the statements "i == 0, j == 0, i == lines - 1, j == lines - 1" results in a value of "True".
For example, points belonging to the perimeter of a square matrix of dimensions "5X5" are points with coordinates [0.0], [0.1], [0.2], [0.3], [0.4], [1.4], [2.4], [3.4], [4.4], [4.3], [4.2], [4.1], [4.0], [3.0], [2.0], [1.0].

0 1 2 3 4

0 [0.0] [0.1] [0.2] [0.3] [0.4]

1 [1.0] [1.1] [1.2] [1.3] [1.4]

2 [2.0] [2.1] [2.2] [2.3] [2.4]

3 [3.0] [3.1] [3.2] [3.3] [3.4]

4 [4.0] [4.1] [4.2] [4.3] [4.4]

(~) Point outside the perimeter of the figure (lines 60-77)

In lines '60-77' the commands of 'else' (line 60) corresponding to 'if (i == 0 || j == 0 || i = lines-1 || j == lines - 1)' (line 56) are executed in case the point with coordinates [i,j] is outside the perimeter of the shape of the square matrix.
More specifically, if the statement "i == 0 || j == 0 || i == lines - 1 || j == lines - 1" produces a value of "False" (false condition), then the command "if-else" (lines 62-76) shall be executed to check whether the point with coordinates [i,j] belongs to a primary or secondary diagonal. This is a logical representation with the logical operator "||" (OR), so it will result in a value of "False" when, all instances "i == 0, j == 0, i == lines - 1, j

== lines - 1" result in a value of "False".
For example, points that do not belong to the perimeter of a "5X5" square matrix are points with coordinates [1.1], [1.2], [1.3], [2.1], [2.2], [2.3], [3.1], [3.2], [3.3].

0 1 2 3 4

0 [0.0] [0.1] [0.2] [0.3] [0.4]

1 [1.0] [1.1] [1.2] [1.3] [1.4]

2 [2.0] [2.1] [2.2] [2.3] [2.4]

3 [3.0] [3.1] [3.2] [3.3] [3.4]

4 [4.0] [4.1] [4.2] [4.3] [4.4]

(!) Point inside the principal diagonal (lines 62-65)
(!) Point outside the principal diagonal (lines 66-76)

In lines "62-76" a check is implemented with an "if-else" control command for the case where the point with coordinates [i,j] is inside or outside the main diagonal and at the same time outside the perimeter of the shape of the square matrix.

In particular, if the statement 'i == j' results in a value of 'True' (condition true), then the commands of the subsection '(!) Point inside the principal diagonal' in lines '62-65' shall be executed, otherwise if it results in a value of 'False', then the commands of the subsection '(!) Point outside the principal diagonal' in lines '66-76' shall be executed.

(!) Point inside the principal diagonal (lines 62-65)

On lines '62-65', the commands of 'if (i == j)' (line 62) are executed for the case where the point with coordinates [i,j] is inside the principal diagonal of the shape of the quadratic table.
More specifically, if the statement "i == j" produces a value of "True" (condition true), then the function "printf()" (line 64) will be executed which prints the dot character (.). That is, when, the variables "i" and "j" contain the same value.
For example, points belonging to the main diagonal and outside the perimeter of a square matrix of dimensions "5X5" are points with coordinates [1.1], [2.2], [3.3].

0 1 2 3 4

0 [0.0] [0.1] [0.2] [0.3] [0.4]

1 [1.0] [1.1] [1.2] [1.3] [1.4]

2 [2.0] [2.1] [2.2] [2.3] [2.4]

3 [3.0] [3.1] [3.2] [3.3] [3.4]

4 [4.0] [4.1] [4.2] [4.3] [4.4]

(!) Point outside the principal diagonal (lines 66-76)

Lines '66-76' execute the commands of 'else' (line 66) corresponding to 'if (i == j)' (line 62) for the case where the point with coordinates [i,j] is outside the main diagonal of the shape of the square matrix.

More specifically, if the representation "i == j" produces a value of "False" (condition true), then the command "if-else" (lines 68-75) shall be executed which checks whether the point with coordinates [i,j] belongs to the secondary diagonal or not. That is, when, the variables "i" and "j" do not contain the same value. For example, points not belonging to the primary diagonal and outside the perimeter of a square matrix are points with coordinates [0.0], [0.1], [0.2], [0.3], [0.4], [1.0], [1.2], [1.3], [1.4], [2.0], [2.1], [2.3], [2.4], [3.0], [3.1], [3.2], [3.4], [4.0], [4.1], [4.2], [4.3], [4.4].

0 1 2 3 4

0 [0.0] [0.1] [0.2] [0.3] [0.4]

1 [1.0] [1.1] [1.2] [1.3] [1.4]

2 [2.0] [2.1] [2.2] [2.3] [2.4]

3 [3.0] [3.1] [3.2] [3.3] [3.4]

4 [4.0] [4.1] [4.2] [4.3] [4.4]

(+) Point within the secondary diagonal (lines 68-71)
(+) Point outside the secondary diagonal (lines 72-75)

In lines '68-75' a check is implemented with an 'if-else' control command for the case where the point with coordinates [i,j] is inside or outside the secondary diagonal and at the same time outside the perimeter of the shape of the square matrix.

More specifically, if the representation "i + j == lines - 1" results in a value of "True" (condition true), then the instructions of the subsection "(+) Point inside the secondary diagonal" in lines "68-71" will be executed, otherwise, if it results in a value of "False",

then the commands in the subsection "(+) Point outside the secondary diagonal" on lines "72-75" will be executed.

(+) Point inside the secondary diagonal (lines 68-71)

On lines '68-71', the commands of 'if (i + j == lines - 1)' (line 68) are executed for the case where the point with coordinates [i,j] is inside the secondary diagonal of the shape of the quadrilateral table.
More specifically, if the representation "i + j == lines - 1" produces a value of "True" (condition true), then the function "printf()" (line 64) will be executed which prints the dot character (.). That is, when, the sum of the variables "i" and "j" contain the same value as the number of lines minus "1".
For example, points belonging to the minor diagonal and outside the perimeter of a square matrix of dimensions '5 × 5' are points with coordinates [1.3], [2.2], [3.1].

0 1 2 3 4

0 [0.0] [0.1] [0.2] [0.3] [0.4]

1 [1.0] [1.1] [1.2] [1.3] [1.4]

2 [2.0] [2.1] [2.2] [2.3] [2.4]

3 [3.0] [3.1] [3.2] [3.3] [3.4]

4 [4.0] [4.1] [4.2] [4.3] [4.4]

(+) Point outside the secondary diagonal (lines 72-75)

In lines '72-75' the commands of 'else' (line 72) corresponding to 'if (i + j == lines - 1)' (line 68) are executed for the case where the point with coordinates [i,j] is outside the secondary diagonal of the shape of the square matrix.
More specifically, if the representation "i + j == lines - 1" produces a value of "False" (condition true), then the function "printf()" (line 74) will be executed which prints the blank character (" "). That is, when, the sum of the variables "i" and "j" do not contain contain the same value as the number of lines minus "1".

For example, points not belonging to the minor diagonal and outside the perimeter of a square matrix are points with coordinates [0.0], [0.2], [0.3], [0.4], [1.0], [1.1], [1.2], [1.4], [2.0], [2.1], [2.3], [2.4], [3.0], [3.2], [3.3], [3.4], [4.0], [4.1] [4.2], [4.3], [4.4].

0 1 2 3 4

0 [0.0] [0.1] [0.2] [0.3] [0.4]

1 [1.0] [1.1] [1.2] [1.3] [1.4]

2 [2.0] [2.1] [2.2] [2.3] [2.4]

3 [3.0] [3.1] [3.2] [3.3] [3.4]

4 [4.0] [4.1] [4.2] [4.3] [4.4]

***** For example, the user enters "5" lines entered in the variable "lines". So, the first loop is executed "5" times, while, the second loop is executed :

(a) In the 1$^{\eta}$ iteration of the first loop, "5" times.

b) In the 2$^{\eta}$ iteration of the first loop, "5" times.

c) In the 3$^{\eta}$ iteration of the first loop, "5" times.

d) In the 4$^{\eta}$ repetition of the first loop, "5" times.

(e) In the 5$^{\eta}$ repetition of the first loop, "5" times.

The figure is illustrated as follows :

*****

. .

* . *

. .

*****

EXAMPLES

Example 1 (8 lines)

==========

Shapes with asterisks

==========

Enter number of lines : 8

```
*
**
***
****
*****
******
*******
********
```

```
       *
      **
     ***
    ****
   *****
  ******
 *******
********
```

```
      *
     ***
    *****
   *******
  *********
 ***********
*************
***************
```

```
********
*.    .*
* .  . *
* .. *
* .. *
* .  . *
*.    .*
********
```

Example 2 (2 lines)

Active code page: 1253

==========

Shapes with asterisks

==========

Enter number of lines : 2

```
*
**

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


 *
**

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


 *
***

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


**
**
```

OBSERVATIONS

In detail, we have two examples where the user enters "8" lines and "2" to represent the figures.

Example 1 (5 lines)

The four shapes are represented by 5 lines.

Example 2 (2 lines)

The four shapes are represented by 2 lines. It is worth noting that in the fourth figure there is no point outside the perimeter of the figure and inside the primary or secondary diagonal at the same time, which is why the dot character (.) is not printed.

ITEM 5

NOTE "SinTaylor.c"

The "SinTaylor.c" (Source Code) and the "Documentation "SinTaylor.c"" (Question, Structure, Variables, Crossing, Examples, Observations) answer the question of the question "Topic 5".

PROGRAM "SinTaylor.c"

```
1 #include <stdio.h>
2 #include <math.h>
3 #define pi 3.14159
4
5 int main (int argc, char **argv)
6 {
7   system ("chcp 1253"); system ("chcp 1253");
8
9 double deg, rad; // Declaration of variables
10  double sine;
11  double    term,    next_term,    diff_terms,    abs_diff_terms,
first_sin_T;
12  double diff_sin_T, abs_diff_sin_T;
13  double sin_T = 0.0; // Initialization of variables
14  int i;
 15 int j = 1; // Initialize variables
16 int sign = -1; // Initialize variables
17
18  printf
("============================================================
\n\n");;
19  printf ("Calculate the sine of an angle\n\n"); // The program
title
20  printf
("============================================================
\n\n");;
```

```
21  printf ("Insert angle in the interval of the 1st circle [0,360]\n\n");
 22 printf ("Degrees : ");

23  scanf ("%lf", &deg); // Enter the angle in degrees

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

25  rad = (pi * deg) / 180; // Angle conversion from degrees to radians

26  printf ("Degrees : [%20.6lf]\n", deg); // Print the angle in degrees

27  printf ("Radians : [%20.6lf]\n\n", rad); // Print the angle in radians

28  sine = sin (rad); // Calculate the sine of the angle with the function "sin (ω)"

29  do /* 1st Loop */

30  {

31        term = 1; // Initialize variable of the first term, the second ...

32        for (i = 1 ; i <= j ; i++) /* 2nd loop */

33        {

34              term = term * (rad / i); // Calculate the first term, the second ... (ω^1 / 1!, ω^3 / 3! ...)

35        }

36        j = j + 2; // Increase the auxiliary variable to calculate the second term, the third ...

37        next_term = term * ((rad * rad) / (j * (j - 1))); // Calculate the second term, the third ... (ω^3 / 3!, ω^5 / 5! ...)

38        diff_terms = next_term - term; // Calculate the difference between the second term and the first, the third term and the second ...

39        abs_diff_terms = fabs (diff_terms); // Calculate the absolute value of the difference

40        if (j == 3) /* (~) 1st repetition of loop 1 */

41        { /* ω^1 / 1! - ω^3 / 3! */

42              first_sin_T = term + (sign * next_term); // Calculate the first sum with the appropriate sign "sign"

43              sin_T = sin_T + first_sin_T; // Enter the sum in the variable "sin_t"

44              sign = sign * (-1); // Change sign

45        }

46        else /* (~) 2nd, 3rd ... repeat 1st loop */
```

```
47          { /* (ω^1 / 1! - ω^3 / 3!) + ω^5 / 5! ... */
48                  sin_T = sin_T + (sign * next_term); // Calculate
the second sum, the third ... with the appropriate sign "sign"
49                  sign = sign * (-1); // Change sign
50          }
51  }
52  while (abs_diff_terms > 0.000001);;
53
54  printf ("Sine : [%20.6lf]\n", sine); // Print the sine of the
angle with the ready function "sin (ω)"
55  printf ("Taylor : [%20.6lf]\n\n", sin_T); // Print the sine of
the angle with the "Taylor" infinite series
56  diff_sin_T = sine - sin_T; // Calculate the difference of the
sine with the "Taylor" infinite series
57  abs_diff_sin_T = fabs (diff_sin_T); // Calculate the absolute
value of the difference between the sine and the "Taylor" infinite
series
58  if (abs_diff_sin_T <= 0.0000009) /* (~) Acceptable absolute
value of the difference of the sine with the infinite series
"Taylor" */
59  {
60          printf ("Sine ~= Taylor\n");;
61          printf ("The two numbers are almost equal\n");
62  }
63  else /* (~) Rejected absolute value of the difference of the
sine with the infinite series "Taylor" */
64  {
65          printf ("Sine != Taylor\n");;
66          printf ("The two numbers are not nearly equal\n");
67  }
68
69  return 0;
70 }
```

DOCUMENTATION "SinTaylor.c"

REQUESTED

The program "SinTaylor.c" achieves the following operations:

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

b) Converts it into radians (ω).

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

(d) Calculate 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 if they are "almost"

equal or not.

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

MODEL

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

a) "stdio.h": contains the ready-made functions "scanf(...)" and "printf(...)" which are linked to the input and output channels respectively for reading and printing contents of the respective variables. Also, "printf(...)" was used to print feature 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 precompilation 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) relative : <=, >

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

VARIABLES

Integer variables (of type "int")

i (The auxiliary variable to control the second loop of lines "30-33")

j (The auxiliary variable for calculating the second term, the third term, ...)

sign (The auxiliary variable to change the sign)

Double precision real variables (of the "double" type)

deg (The angle in degrees)

rad (The angle in radians)

sine (The sine of the angle in radians)

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 second term and the first, the third and the second ...)

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

sin_T (The sum of the first with the second term, the sum of this with the third

term ... with the appropriate sign)

diff_sin_T (The difference of the sine with the infinite series "Taylor")

abs_diff_sin_T (The absolute value of the difference of the sine with the infinite series "Taylor")

DISCUSSION

Declaration of variables (lines 9-16)

The program starts with the declaration of variables on lines "9-16" (see subsection "Variables" for details).

Initialisation of variables (lines 13, 15, 16)

Some initialisation of variables is done on the above lines. Πιο αναλυτικά, στη γραμμή «13» η τιμή «0.0» εκχωρείται στη μεταβλητή «sin_T» για το άθροισμα του ημιτόνου της γωνίας σε ακτίνια με την απειροσειρά «Taylor», στη γραμμή «15» η τιμή «1» στη «j» ως βοηθητική μεταβλητή για τον υπολογισμό του δευτέρου όρου, του τρίτου κ.and so on, and in 'sign' the value '-1' as an auxiliary variable for changing the sign in the addition of the terms of the Taylor infinite series.

Program title (line 19)

On line '19' with a 'printf()' function, the title of the program (Calculation of the sine of an angle) and two escape characters of this line change (\n) are printed from the 'standard' output.

Enter the angle in degrees (line 23)

At line '23' an angle in degrees is read from the 'standard' input by a function 'scanf()' accompanied by the appropriate messages printed from the 'standard' output by the function 'printf()' (lines 21, 22). The result is entered in the variable 'deg'.

Conversion of angle from degrees to radians (line 25)

On line '25', the angle is converted to radians using the arithmetic expression '(pi * deg) / 180' and the result is entered in the variable 'rad'.

Print the angle in degrees (line 26)

At line '26', the contents of the variable 'deg' (the angle in degrees) 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.

Printing the angle in radians (line 27)

At line '27' the content of the variable 'rad' (the angle in radians) 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, since, this is a double precision real. This is to ensure uniform alignment of the results.

Calculation of the sine of the angle with the function 'sin (ω)' (line 28)

In line "28", the sine of the angle in radians is calculated with the function "sin(rad)" imported from the library "math.h" and the result is entered in the variable "sine".

1st loop (lines 29-52)

On lines "29-52" 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, in the first loop, 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 condition, the second ... (line 31)

In line "31" an initialization of the variable "term" characterizing the first term, the second term, etc. of the infinite string 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 32-35)

In lines "32-35" 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 :

Calculating the first term, the second ... (line 34)

In line "34" the "term * (rad / i)" representation is implemented to calculate the first term, 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 31)").

Increasing the auxiliary variable for calculating the second term, the third ... (line 36)

In line "36", the auxiliary variable "j" for calculating the second term, third term, etc. of the infinite series is increased by "2" with the representation "j + 2" each time the "1ος loop" is executed.

Calculation of the second term, the third ... (line 37)

In line "37", the "term * ((rad * rad) / (j * (j - 1)))" representation is implemented for calculating the second term, the third term and so on (ω^3 / 3!, ω^5 / 5! ...) of the infinite series. The result is registered 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 38)

In line "38", the difference between the second term and the first, the third term and the second, and so on, of the infinite series (next_term - term) is entered in the variable "diff_terms", each time the "1ος loop" is executed.

Calculation of the absolute value of the difference (line 39)

Line '39' contains 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. The absolute value is calculated with the help of the function "fabs(...)" introduced with the library "math.h".

(~) 1st iteration of loop 1 (lines 40-45)
(~) 2nd, 3rd ... iteration of loop 1 (lines 46-50)

On lines "40-50" 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 statement "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 differ from each other from the second sum onwards. Obviously, the sum of the first and second terms is the one that differs little from the other sums.

(~) 1st repetition of loop 1 (lines 40-45)

Lines "89-94" execute the instructions of "if (j == 3)" (line 40) for the case where "j" contains the value "3", i.e., that the program is in the first execution of the "1ου loop". Consequently, the first sum of the infinite series that differs least 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 42)

In line "42", 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 as initial value, the value "-1" (cf. "Initialisation of variables (line 16)". The result is entered in the variable "first_sin_T".

Enter the sum in the variable 'sin_T' (line 43)

On line '43', the first sum with the appropriate sign is entered in the variable 'sin_T' with the representation 'sin_T + first_sin_T', the initial value of which is '0.0' (see point '0.0' in the table below). "Initialization of variables (line 13)"

.

Change sign (line 44)

In line "44" 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 thereafter.

(~) 2nd, 3rd ... repetition of the 1st loop (lines 46-50)

Lines "46-50" execute the instructions of "else" (line 46) corresponding to "if (j == 3)" (line 40), 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 after. 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 48)

In line "48" the representation "(sin_T + (sign * next_term)" is implemented for the second sum, the third and so on 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 49)

In line "49", 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.

Printing the sine of the angle with the ready-made function 'sin (ω)' (line 54)

At line '54' the content of the variable 'sine' (the sine of the angle in radians with the ready-made 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.

Printing the sine of the angle with the "Taylor" infinite series (line 55)

At line '55', the contents of the variable 'sin_T' (the sine of the angle in radians with the infinite 'Taylor' series) 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.

Calculation of the difference of the sine with the infinitesimal 'Taylor' (line 56)

Line '56' implements the representation 'sine - sin_T' for calculating the difference of the sine of the angle in radians with the ready-made function 'sin()' (sine) and the infinite series 'Taylor' (sin_T). The result is entered in the variable 'diff_sin_t'.

Calculation of the absolute value of the difference of the sine with the infinite series "Taylor" (line 57)

In line "57" the representation "fabs (diff_sin_T)" is implemented to calculate the absolute value of the difference of the sine of the angle in radians

with the ready-made function "sin()" (sine) and the infinite series "Taylor" (sin_T). The function "fabs()" is imported from the library "math.h". The result is entered in the variable 'abs_diff_sin_t'.

(~) Acceptable absolute value of the difference between the sine and the infinite series "Taylor" (lines 58-62)
(~) Rejected absolute value of the difference between the sine and the infinite series "Taylor" (lines 63-67)

In lines "58-67" a check is implemented with the command "if - else" as to the deviation of the absolute value of the difference between the two functions that yield the sine of the angle. The control condition is the statement "abs_diff_sin_T <= 0.0000009". The test is performed to determine if the two numbers are "nearly" equal.

(~) Acceptable absolute value of the difference between the sine and the "Taylor" infinitesimal (lines 58-62)

Lines "58-62" execute the commands of "if (abs_diff_csts)" (line 58) for the case when the absolute value of the difference between the two numbers is less than or equal to "0.0000009" and, therefore, they are "almost" equal. If, the representation "abs_diff_sin_T <= 0.0000009" results in a "True" value (a value other than "0"), then, the characteristic messages will be printed from the "standard" output with the "printf()" function.

(~) Rejected absolute value of the difference between the sine and the "Taylor" infinite series (lines 63-67)

In lines "63-67" the commands of "else" (line 63) corresponding to "if (abs_diff_sin_T)" (line 58)
are executed for the case where the absolute value of the difference between the two numbers is greater than the value "0.0000009" and, therefore, the two numbers are not "nearly" equal. If, the representation "abs_diff_sin_T <= 0.0000009" results in the value "False" (0), then, the characteristic messages will be printed from the "standard" output with the function "printf()".

EXAMPLES

Example 1 (ω == 30)º

==========

Calculating the sine of an angle

==========

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

Fates : 30

Fates : [30.000000]

Radius : [0.523598]

Sine : [0.500000]

Taylor : [0.500000]

Sine ~= Taylor

The two numbers are almost equal

Example 2 (ω == 360)°

==========

Calculating the sine of an angle

==========

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

Degrees : 360

Fates : [360.000000]

Actinia : [6.283180]

Sine : [-0.000005]

Taylor : [-0.000005]

Sine ~= Taylor

The two numbers are almost equal

Example 3 (ω == 167.5)°

==========

Calculating the sine of an angle

==========

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

Degrees : 167.5

Fates : [167.500000]

Acinias : [2.923424]

Sine : [0.216442]

Taylor : [0.216442]

Sine ~= 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 a difference was found in "Example 2 (ω == 360º). In detail, we have "3" examples corresponding to an angle belonging to the interval of the first circle [0,360].

Example 1 (ω == 30)º

In "Example 1" the angle of "30" 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". 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 : 30.000000

radians : 0.523598

sin(...) : 0.500000

Taylor : 0.500000

Example 2 (ω == 360)º

In "Example 2" the angle of "360" 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". 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 : 360.000000

radians : 6.283180

sin(...) : -0.000005

Taylor : -0.000005

The observation is that at the sine the result obtained by both

numbers, that it is the value "-0.000005", while the standard calculator puts out that the sine of "360" degrees is the value "0.000000".

Example 3 (ω == 167.5)º

In "Example 3" the angle of "167.5" 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". 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 : 167.500000

radians : 2.923424

sin(...) : 0.216442

Taylor : 0.216442

COMPUTER PROGRAMMING

Thank you for your attention.