COMPUTER PROGRAMMING

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

DEPARTMENT OF INFORMATION AND COMPUTER ENGINEERING

PROJECT 6
TABLES-POINTERS-FILES

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: 31/1/2022

COMPUTER PROGRAMMING

SOURCE CODES / DOCUMENTATION

ITEM 2

NOTE "Combinations.c"

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

PROGRAM "Combinations.c"

```
#include <stdio.h>

#include <stdlib.h>

/* Function declaration */

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

int Read_N_Numbers (int, int); // Get the number "N" of numbers,
where "N" belongs to the interval (6, 49]

void Print_N (int N); // Print the number of numbers in the interval
(6, 49]

void Read_Matrix (int *, int); // Insert the "N" numbers in the
interval [1, 49] and assign them to a dynamically bound matrix

int Search (int *, int, int, int, int); // Search the dynamically
bound table of numbers to possibly find identical registered numbers

int *Read_Even_Lims (); // Insert the interval [0, X1]U[X1, X2]U[X2,
6] for the number of even numbers of each combination

int *Read_Sum_Lims (); // Insert the interval [21, Y1]U[Y1, Y2]U[Y2,
279] for the sum of the numbers of each combination

void Check_Memory (int *); // Check in memory for dynamic table
bindings

void Sort (int *, int); // Sort the table of numbers in ascending
order

void Print_Matrix (int *, int); // Print the matrix with the "N"
numbers in the interval [1, 49]

void Print_Even_Lims (int *); // Print the interval [0, X1]U[X1,
X2]U[X2, 6] for the number of even numbers of each combination

void Print_Sum_Lims (int *); // Print the interval [21, Y1]U[Y1,
Y2]U[Y2, 279] for the sum of the numbers of each combination

int *Combos (int *, int, int *, int *); // Create the combinations

int Search_Evens (int *, int *, int); // Search for the number of
even numbers of each combination
```

```
int Search_Sum (int *, int *, int); // Search for the sum of the
numbers of each combination

void Print_Combos (int *); // Print the combinations

void Frequency (int *, int *); // Calculate the frequency of
occurrence of each number in the printed combinations

void Print_Freq (int *); // Print the frequency of occurrence of each
number in the printed combinations

void Print_num_Combos (int *); // Print the number of combinations
that meet certain conditions

void Free_Memory (int *); // Free memory



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

{

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


      int *Mat; // Variable declaration

      int N;

      int *Lim_s, *Lim_e;

      int *c;


      N = Read_N_Numbers (6, 49); // Call the function
"Read_N_Numbers (6, 49)"

      Mat = (int *) malloc (N * sizeof (int)); // Dynamic memory
binding to create a one-dimensional table "Mat" with the inserted
numbers in the interval [1, 49]

      Check_Memory (Mat); // Call the function "Check_Memory (Mat)"

      Read_Matrix (Mat, N); // Call the function "Read_Matrix (Mat,
N)"

      Sort (Mat, N); // Call the function "Sort (Mat, N)"

      Lim_e = Read_Even_Lims (); // Call the function "Read_Even_Lims
()"

      Lim_s = Read_Sum_Lims (); // Call the function "Read_Sum_Lims
()"

      Print_Matrix (Mat, N); // Call the function "Print_Matrix (Mat,
N)"

      Print_N (N); // Call the function "Print_N (N)"

      Print_Even_Lims (Lim_e); // Call the function "Read_Even_Lims
(Lim_e)"
```

```
	Print_Sum_Lims (Lim_s);;	// Call the function
"Read_Sum_Lims (Lim_s)"

	c = Combos (Mat, N, Lim_e, Lim_s); // Call the function "Combos
(Mat, N, Lim_e, Lim_s)"


	return 0;

}


void Title () /* Title () */

{

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

	printf ("CONNECTIONS\n\n"); // The title of the program

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

}


int Read_N_Numbers (int L1, int L2) /* Read_N_Numbers (6, 49) */

{

	int x; // Declaration of variables

	do /* Loop */

	{

		system ("cls");

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

		printf ("Enter the number of numbers 'N' in the interval
(6, 49] : ");;

		scanf ("%d", &x); // Insert the number "N" of numbers,
where "N" belongs to the interval (6, 49]

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

	}

	while (x <= L1 || x > L2);;


	return x; // Return the number "N" of numbers, where "N"
belongs to the interval (6, 49]

}
```

```
void Print_N (int N) /* Print_N (N) */

{

        printf ("N : %3d\n\n", N); // Print the number of numbers in
the interval (6, 49]

}


void Read_Matrix (int *M, int N) /* Read_Matrix (Mat, N) */

{

        int i; // Declaration of variables

        for (i = 0; i < N; i++) /* 1st loop */

        {

                do /* 2nd Loop */

                {

                        printf ("Enter number : ");;

                        scanf ("%d", R + i); // Insert the "N" numbers in
the interval [1, 49] and assign them to the dynamically bound table
"Mat"

                }

                while (Search (M, i, *(M + i), 1, 49) != -1); // Call the
function "Search (Mat, 0..N, *(Mat + 0..N), 1, 49)"

        }

}


int Search (int *M, int N, int Num, int O1, int O2) /* Search (Mat,
0..N, *(Mat + 0..N), 1, 49) */

{

        int j = 0; // Declaration of variables

        if (N == 0) /* (~) Table position *Mat */

        {

                if (Num >= O1 && Num <= O2) /* (!) The number belongs to
the interval [1, 49] */

                        return -1; // Return confirmation value (acceptable
number)

                return j; // Return error value (discarded number

        }

        else /* (~) Table positions from *(Mat + 1) onwards */
        {

                while (Num >= O1 && Num <= O2 && j < N && *(M + j) !=
Num) /* Search loop to identify possible equalities in the contents
of the table and contents not belonging to the interval [1, 49]*/

                        j++; // Increase the auxiliary variable for
violating the constraints of a number entry in the "Mat" table

                if (j == N) /* (+) The auxiliary variable is equal to the
position of the table reached by the search */

                        return -1; // Return confirmation value (acceptable
number)

                return j; // Return error value (discarded number)

        }

}


int *Read_Even_Lims () /* Read_Even_Lims () */

{

        int *Evens; // Variable declaration

        Evens = (int *) malloc (2 * sizeof (int)); // Dynamic memory
binding to create a one-dimensional table "Evens" with the two
inserted numbers in the interval [0, X1]U[X1, X2]U[X2, 6] for the
number of even numbers of each combination

        Check_Memory (Evens); // Call the function "Check_Memory
(Evens)"

        do /* Loop */

        {

                system ("cls");

                printf ("Enter the bounds of the even numbers of each
combination in the interval [0, X1]U[X1, X2]U[X2, 6]\n\n");

                printf ("X1 : ");;

                scanf ("%d", Evens); // Insert the first boundary "X1" of
the interval [0, X1]U[X1, X2] for the number of even numbers of each
combination

                printf ("X2 : ");;

                scanf ("%d", Evens + 1); // Insert the second boundary
"X2" of the interval [X1, X2]U[X2, 6] for the number of even numbers
of each combination

                printf ("\n");;

        }

     while (*Evens < 0 || *Evens > *(Evens + 1) || *(Evens + 1) > 6);;
```

```
	return Evens; // Return the table "Evens" with the limits "X1"
and "X2"

}


int *Read_Sum_Lims () /* Read_Sum_Lims () */

{

	int *Sum; // Declaration of variables

	Sum = (int *) malloc (2 * sizeof (int)); // Dynamic memory
binding for 1D table "Sum" with the two inserted numbers in the
interval [21, Y1]U[Y1, Y2]U[Y2, 279] for the sum of numbers of each
combination

	Check_Memory (Sum); // Call the function "Check_Memory (Sum)"

	do /* Loop */

	{

		system ("cls");

		printf ("Enter the limits of the sum of the numbers of
each combination in the interval [21, Y1]U[Y1, Y2]U[Y2, 279]\n\n");

		printf ("Y1 : ");;

		scanf ("%d", Sum); // Insert the first boundary "Y1" of
the interval [21, Y1]U[Y1, Y2] for the sum of the numbers of each
combination

		printf ("Y2 : ");;

		scanf ("%d", Sum + 1); // Insert the second boundary "Y2"
of the interval [Y1, Y2]U[Y2, 279] for the sum of the numbers of each
combination

		printf ("\n");;

	}

	while (*Sum < 21 || *Sum > *(Sum + 1) || *(Sum + 1) > 279);;


	return Sum; // Return the table "Sum" with the limits "X1" and
"X2"

}


void Check_Memory (int *Array) /* Check_Memory (Array) */

{

	if (Array == NULL) /* (~) The "Array" index points to the
"NULL" index */

	{

		system ("cls");
```

```
            printf ("\nMemory binding problem\n\n\n");;
            exit (1); // Terminate the program with a return value of
"1"
      }
}

void Sort (int *M, int N) /* Sort (Mat, N) */
{
      int i, tmp, j; // Variable declaration
      for (i = 0; i < N - 1; i++) /* 1st loop */
            for (j = i + 1; j < N; j++) /* 2nd loop */
                  if (*(M + i) > *(M + j)) /* (~) *M == 21 > *(M + 1)
== 3 */
                  {
                        tmp = *(M + i); // tmp == *M ==> tmp == 21
                        *(M + i) = *(M + j); // *M == *(M + 1) ==> *M
== 3
                        *(M + j) = tmp; // *(M + 1) == tmp ==> *(M +
1) == 21
                  }
}

void Print_Matrix (int *M, int N) /* Print_Matrix (Mat, N) */
{
      int i; // Declaration of variables
      system ("cls");
      for (i = 0; i < N; i++) /* Loop */
      {
            printf ("Element [%2d] Value : %3d\n", i + 1, *(M + i));
// Print the elements of the "Mat" table with the "N" numbers in the
interval [1, 49] sorted
      }
      printf ("\n");;
}

void Print_Even_Lims (int *E) /* Print_Even_Lims (Evens) */
{
	int i; // Declaration of variables
	for (i = 0 ; i < 2; i++) /* Loop */
	{
		if (i == 0) /* (~) Position *Evens, where X1 belongs */
		{
			printf ("X1 : %3d\n", *(E + i)); // Print the first
boundary "X1" of the interval [0, X1]U[X1, X2] for the number of even
numbers of each combination
		}
		else /* (~) Position *(Evens + 1) and then */
		{
			if (i == 1) /* (!) Position *(Evens + 1), where X2
belongs */
			{
				printf ("X2 : %3d\n\n", *(E + i)); // Insert
the second boundary "X2" of the interval [X1, X2]U[X2, 6] for the
number of even numbers of each combination
			}
		}
	}
}

void Print_Sum_Lims (int *S) /* Print_Sum_Lims (Sum) */
{
	int j; // Declaration of variables
	for (j = 0 ; j < 2; j++) /* Loop */
	{
		if (j == 0) /* (~) Position *Sum, where Y1 belongs */
		{
			printf ("Y1 : %3d\n", *(S + j)); // Print the first
limit "Y1" of the interval [21, Y1]U[Y1, Y2] for the sum of the
numbers of each combination
		}
		else /* (~) Position *(Sum + 1) and then */
		{
			if (j == 1) /* (!) Position *(Sum + 1), where Y2
belongs */
			{
```

```
                        printf ("Y2 : %3d\n\n", *(S + j)); // Print
the second boundary "Y2" of the interval [Y1, Y2]U[Y2, 279] for the
sum of the numbers of each combination

                  }

            }

      }


}


int *Combos (int *M, int N, int *E, int *S) /* Combos (Mat, N, Evens,
Sum) */

{

      int i, j, k, l, m, n; // Declaration of variables

      int *Com, *Cnt, *Freq;

      int evens, sum;

      Com = (int *) malloc (6 * sizeof (int)); // Dynamic memory
binding for 1D table "Com" with each combination created

      Check_Memory (Com); // Call the function "Check_Memory (Com)"

      Cnt = (int *) calloc (5, sizeof (int)); // Dynamic memory
binding for 1D table "Cnt" (with initial value "0" in each position
of the table) with counters to calculate the number of combinations
that meet specific conditions

      Check_Memory (Cnt); // Call the function "Check_Memory (Cnt)"

      Freq = (int *) calloc (49, sizeof (int)); // Dynamic memory
binding for 1D table "Freq" (with initial value "0" in each position
of the table) with counters to calculate the frequency of occurrence
of each number in the printed combinations

      Check_Memory (Freq); // Call the function "Check_Memory (Freq)"

      for (i = 0; i < N - 5; i++) /* 1st loop */

            for (j = i + 1; j < N - 4; j++) /* 2nd loop */

                  for (k = j + 1; k < N - 3; k++) /* 3rd loop */

                        for (l = k + 1; l < N - 2; l++) /* 4th loop
*/

                              for (m = l + 1; m < N - 1; m++) /* 5th
loop */

                                    for (n = m + 1; n < N; n++) /*
6th loop */

                                    {
                                        *(Com + 0) = *(M + i); //
1st number of the combination in position *(Com + 0) of the table
"Com"

                                        *(Com + 1) = *(M + j); //
2nd number of the combination in position *(Com + 1) of the "Com"
table

                                        *(Com + 2) = *(M + k); //
3rd number of the combination in position *(Com + 2) of the "Com"
table

                                        *(Com + 3) = *(M + l); //
4th number of the combination in position *(Com + 3) of the "Com"
table

                                        *(Com + 4) = *(M + m); //
5th number of the combination in position *(Com + 4) of the "Com"
table

                                        *(Com + 5) = *(M + n); //
6th number of the combination in position *(Com + 5) of the "Com"
table

                                        *Cnt = *Cnt + 1; //
Increase the counter to calculate the number of combinations "N" per
6

                                        evens = Search_Evens (Com,
E, 6); // Call the function "Search_Evens (Com, Evens, 6)"

                                        sum = Search_Sum (Com, S,
6); // Call the function "Search_Sum (Com, Sum, 6)

                                        if (evens == 1) /* (~)
Number of edges of the combination within the limits [X1, X2] */

                                        {

                                              if (sum == 1) /* (!)
'sum of the numbers of the combination within the limits [Y1, Y2] */

                                              {

                                                    Print_Combos
(Com);      // Call the function "Print_Combos (Com)"


                                                    *(Cnt + 3) =
*(Cnt + 3) + 1; // Increase the counter to calculate the number of
combinations printed

                                                    Frequency
(Freq, Com); // Call the function "Frequency (Freq, Com)"

                                              }

                                              else /* (!) 'Sum of
the numbers of the combination outside the limits [Y1, Y2] */

                                              {
```

```
                                                                    *(Cnt + 2) = 
*(Cnt + 2) + 1; // Increase the counter to calculate the number of combinations that met the first condition but did not meet the second condition
                                                              }
                                                        }
                                                        else /* (~) Number of edges of the combination outside the limits [X1, X2] */
                                                        {
                                                              *(Cnt + 1) = *(Cnt + 1) + 1; // Increase the counter to calculate the number that did not meet the first condition
                                                        }
                                                  }
                                                  Print_num_Combos (Cnt); // Call the function "Print_num_Combos (N, Cnt)
                                                  Print_Freq (Freq); // Call the function "Print_Freq (Freq)"
                                                  Free_Memory (M); // Call the function "Free_Memory (Mat)"
                                                  Free_Memory (E); // Call the function "Free_Memory (Evens)"
                                                  Free_Memory (S); // Call the function "Free_Memory (Sum)"
                                                  Free_Memory (Com); // Call the function "Free_Memory (Com)"
      return Cnt; // Return the table "Cnt" with the counters
}


int Search_Evens (int *C, int *E, int N) /* Search_Evens (Mat, Evens, N) */
{
      int j = 0; // Declaration of variables
      int evens = 0;
      while (j < N) /* Loop */
      {
            if (*(C + j) % 2 == 0) /* (~) 'even number */
            {
                  evens++; // Increase the auxiliary variable for the number of even numbers
```

```
                        j++; // Increase the auxiliary variable to access
the combination
                }
                else /* (~) Redundant number */
                {
                        j++; // Increase the auxiliary variable to access
the combination
                }
        }

        if (evens >= *E && evens <= *(E + 1)) /* (!) The set of even
numbers belongs to the limits [X1, X2] */
                return 1; // Return confirmation value
        return 0; // Return reject value
}

int Search_Sum (int *C, int *S, int N) /* Search_Sum (Mat, Sum, N) */
{
        int sum = 0; // Variable declaration
        int j;
        for (j = 0; j < N; j++) /* Loop */
                sum = sum + *(C + j); // Calculate the sum of the numbers
of the combination

        if (sum >= *S && sum <= *(S + 1)) /* (~) The sum of the numbers
of the combination belongs to the limits [Y1, Y2] */
                return 1; // Return confirmation value
        return 0; // Return reject value
}

void Print_Combos (int *C) /* Print_Combos (Com) */
{
        printf ("%2d %2d %2d %2d %2d %2d\n", *(C + 0), *(C + 1), *(C +
2), *(C + 3), *(C + 4), *(C + 5)); // Print the valid combination
}

void Frequency (int *F, int *C) /* Frequency (Freq, Com) */
```

```
{
       *(F + (*(C + 0) - 1)) = *(F + (*(C + 0) - 1)) + 1; // Increase the counter for the frequency of occurrence of the "N" number of the 1st position of the combination

       *(F + (*(C + 1) - 1)) = *(F + (*(C + 1) - 1)) + 1; // Increase the counter for the frequency of occurrence of the "N" number of the 2nd position of the combination

       *(F + (*(C + 2) - 1)) = *(F + (*(C + 2) - 1)) + 1; // Increase the counter for the frequency of occurrence of the "N" number of the 3rd position of the combination

       *(F + (*(C + 3) - 1)) = *(F + (*(C + 3) - 1)) + 1; // Increase the counter for the frequency of occurrence of the "N" number of the 4th position of the combination

       *(F + (*(C + 4) - 1)) = *(F + (*(C + 4) - 1)) + 1; // Increase the counter for the frequency of occurrence of the "N" number of the 5th position of the combination

       *(F + (*(C + 5) - 1)) = *(F + (*(C + 5) - 1)) + 1; // Increase the counter for the frequency of occurrence of the "N" number of the 6th position of the combination
}


void Print_Freq (int *F) /* Print_Freq (Freq) */
{
       int i; // Declaration of variables

       printf ("\n");;

       for (i = 0; i < 49; i++) /* Loop */
       {
              if (*(F + i) != 0) /* (~) Print an element of the "Freq" table that does not contain the value "0" */
              {
                     printf ("Frequency of [%2d] : [%8d]\n", i + 1, *(F + i)); // Print the frequency of occurrence of each "N" number selected to create combinations
              }


       }

       Free_Memory (F); // Call the function "Free_Memory (Freq)"
}


void Print_num_Combos (int *Cnt) /* Print_num_Combos (Cnt) */
```

```
{
      printf ("The number of combinations N per 6 is : [%8d]\n", N,
*Cnt); // Print the number of combinations N per 6 is

      printf ("The number of combinations that did not satisfy the
first condition : [%8d]\n", *(Cnt + 1)); // Print the number of
combinations that did not satisfy the first condition

      printf ("The number of combinations that satisfied the first
but not the second condition : [%8d]\n", *(Cnt + 2)); // Print the
number of combinations that satisfied the first but not the second
condition

      printf ("The number of combinations printed : [%8d]\n", *(Cnt +
3)); // Print the number of combinations printed

      Free_Memory (Cnt); // Call the function "Free_Memory (Cnt)"

}


void Free_Memory (int *Array) /* Free_Memory (Array */

{

      free (Array); // Free the memory used to create the dynamic
array

}
```

DOCUMENTATION "Combinations.c"

REQUESTED

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

a) Reads from the "standard" input a set of numbers "N" in the interval (6, 49].

(b) Reads from the "standard" input "N" numbers in the range [1, 49].

c) Reads from the "standard" input two numbers "X1", "X2" in the interval [0, X1]U[X1, X2]U[X2, 6] for the number of even numbers of each combination.

d) Read from the "standard" input two numbers "Y1", "Y2" in the range [21, Y1]U[Y1, Y2]U[Y2, 279] for the sum of the numbers of each combination.

(e) Create dynamically bound tables for data "b", "c" and "d" respectively.

f) Searches the table of "N" numbers for numbers not belonging to the interval

[1, 49] that have already been entered in the table.

g) Checks whether there is space in memory to bind them.

(h) sorts the table of 'N' numbers in the interval [1, 49] in ascending order.

(i) prints the number "N" of numbers in the interval [6, 49]

(j) prints the table of 'N' numbers in the interval [1, 49] sorted in ascending order.

k) Prints the numbers "X1" and "X2" of the interval [0, X1]U[X1, X2]U[X2, 6] for the number of even numbers of each combination.

(l) Print the numbers "Y1" and "Y2" of interval [21, Y1]U[Y1, Y2]U[Y2, 279] for the sum of the numbers of each combination.

m) Generates combinations of "N" every 6.

n) Creates a dynamically bound table to cache each combination created.

o) Searches the table with each combination for the number of even numbers of each combination.

p) Searches the table with each combination for the sum of the numbers of each combination.

q) Prints the combinations for which the number of even numbers belongs to the interval [X1, X2] and the sum of the numbers belongs to the interval [Y1, Y2].

(r) Calculate the frequency of occurrence of each number in the printed combinations.

(s) Prints the frequency of occurrence of each number in the printed combinations.

(t) Prints the number of combinations that satisfy specified conditions.

(v) Release the dynamically bound memory.

MODEL

In order to implement the requirement, 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 to read and print contents of the respective variables. Also, "printf(...)" was used to print feature messages for optimal understanding of the source code.

b) "stdlib.h": contains the ready-made functions "malloc(...)" and "calloc(...)" for dynamic memory binding to create dynamic tables.
In addition, pointers (of type "int") and the characteristic operators :

(a) numeric : +

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

(c) assignment : =

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

(e) scaling : variable++
The control commands :

(a) if - else

The iteration commands :

(a) do - while

(b) for

(c) while
The tables :

a) Dynamically one-dimensional

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

FUNCTIONS

Of type 'void' (do not return a value)

Title () 'The title of the program'.

Print_N (int) "Print the number of numbers in the interval (6, 49]"

Check_Memory (int *) "Check in memory for dynamic table bindings"

Sort (int *, int) "Sort the table of numbers in ascending order"

Print_Matrix (int *, int) "Print the matrix with the "N" numbers in the interval [1, 49]"

Print_Even_Lims (int *) "Compare the functions "Sin (ω)" and "Taylor_S (ω)" that calculate the sine of the angle in radians for whether they are "almost" equal"

Print_Sum_Lims (int *) "Comparison of the functions "Cos (ω)" and "Taylor_C (ω)" which compute the cosine of the angle in radians to see if they are "almost" equal"

Print_Combos (int *) "Print combinations"

Frequency (int *, int *) "Calculate the frequency of occurrence of each number in the printed combinations"

Print_Freq (int *) "Printing the frequency of occurrence of each number in the printed combinations"

Print_num_Combos (int, int *) "Printing the number of combinations that satisfy certain conditions"

Free_Memory (int *) "Free memory"

<u>Of type "int" (return an integer value)</u>

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

Read_N_Numbers (int, int) "Input the number "N" of numbers, where "N" belongs to the interval (6, 49]"

Search (int *, int, int, int, int) "Search the dynamically bound table of numbers to possibly locate identical registered numbers"

Search_Evens (int *, int *, int) "Search for the number of even numbers of each combination"

Search_Sum (int *, int *, int) "Search for the sum of the numbers of each combination"

<u>Of type 'int *' (return integer pointer)</u>

***Read_Even_Lims()** "Insert the interval [0, X1]U[X1, X2]U[X2, 6] for the number of even numbers of each combination"

***Read_Sum_Lims ()** "Enter the interval [21, Y1]U[Y1, Y2]U[Y2, 279] for the sum of the numbers of each combination"

***Combos (int *, int, int *, int *)** "Create combinations"

VARIABLES

main (int argc, char **argv)

Integer variables (of type "int")

*Mat (pointer pointing to the first position of the dynamically bound integer matrix "Mat" containing the "N" numbers in the interval [1, 49])

N (The number "N" of numbers in the interval (6, 49])
*Lim_e (Index pointing to the first position of the dynamically bound integer table generated by the function "Read_Even_Lims(...)" for the interval [0, X1]U[X1, X2]U[X2, 6] of the number of even numbers of each combination)
*Lim_s (Index pointing to the first position of the dynamically bound integer table generated by the function "Read_Sum_Lims(...)" for the interval [21, Y1]U[Y1, Y2]U[Y2, 279] of the sum of numbers of each combination)
*c (Index pointing to the dynamically bound integer table created by the function "Combos(...)" containing the combinations one by one)

Read_N_Numbers (int L1, int L2)

Parameters

L1 (The constant "6")

L2 (The constant "49")

Integer variables (type "int")

x (The number 'N' of numbers in the interval (6, 49])

Print_N (int N)

Parameter

N (The number 'N' of numbers in the interval (6, 49])

Read_Matrix (int *M, int N)

Parameters

*M (Index pointing to the first position of the dynamically bound integer matrix 'Mat' containing the 'N' numbers in the interval [1, 49])

N (The number of 'N' numbers in the interval (6, 49])

Integer variables (of type "int")

i (Auxiliary variable for loop control)

Search (int *M, int N, int Num, int O1, int O2)

Parameters

*M (Index pointing to the first position of the dynamically bound integer matrix 'Mat' containing the 'N' numbers in the range [1, 49])

N (The number of 'N' numbers in the interval (6, 49])

Num (The number read from the 'standard' input)

O1 (The constant '1')

O2 (The constant '49')

Integer variables (of type 'int')

j (Auxiliary variable for loop control)

***Read_Even_Lims ()**

Integer variables (type 'int')

*Evens (Index pointing to the first position of the dynamically bound integer matrix "Evens", for the interval [0, X1]U[X1, X2]U[X2, 6] of the number of even numbers of each combination)

***Read_Sum_Lims ()**

Integer variables (type "int")

*Sum (Index pointing to the first position of the dynamically bound integer table 'Sum', for the interval [21, Y1]U[Y1, Y2]U[Y2, 279] of the sum of numbers of each combination)

Check_Memory (int *Array)

Parameter

*Array (pointer pointing to the first position of a dynamically bound integer array)

Sort (int *M, int N)

Parameters

*M (Pointer pointing to the first position of a dynamically bound integer array 'Mat' containing the 'N' numbers in the interval [1, 49])

N (The number of 'N' numbers in the interval (6, 49])

Integer variables (of type "int")

i (Auxiliary variable to control the first loop)

j (Auxiliary variable to control the second loop)

tmp (Auxiliary variable for sorting the elements of the table)

Print_Matrix (int *M, int N)

Parameters
*M (Index pointing to the first position of the dynamically bound integer matrix 'Mat' containing the 'N' numbers in the interval [1, 49])

N (The number of 'N' numbers in the interval (6, 49])

Integer variables (of type "int")

i (Auxiliary variable for loop control)

Print_Even_Lims (int *E)

Parameter

*E (Index pointing to the first position of the dynamically bound integer matrix 'Evens' for the interval [0, X1]U[X1, X2]U[X2, 6] of the number of even numbers of each combination)

Integer variables (of type "int")

i (Auxiliary variable for loop control)

Print_Sum_Lims (int *S)

Parameter

*S (Index pointing to the first position of the dynamically bound integer table 'Sum' for the interval [21, Y1]U[Y1, Y2]U[Y2, 279] of the sum of numbers of each combination)

Integer variables (of type "int")

j (Auxiliary variable for loop control)

***Combos (int *M, int N, int *E, int *S)**

Parameters
 *M (Index pointing to the first position of the dynamically bound integer matrix 'Mat' containing the 'N' numbers in the interval [1, 49])

N (The number of 'N' numbers in the interval (6, 49])
 *E (Index pointing to the first position of the dynamically bounded integer matrix "Evens" for the interval [0, X1]U[X1, X2]U[X2, 6] of the number of even numbers of each combination)
 *S (Index pointing to the first position of the dynamically bounded integer matrix 'Sum' for the interval [21, Y1]U[Y1, Y2]U[Y2, 279] of the sum of numbers of each combination)

Integer variables (of type "int")

i (Auxiliary variable to control the first loop)

j (Auxiliary variable to control the second loop)

k (Auxiliary variable to control the third loop)

l (Auxiliary variable to control the fourth loop)

m (Auxiliary variable to control the fifth loop)

n (Auxiliary variable to control the sixth loop) evens (The confirmation value returned by the function 'Search_Evens(...)' for the number of even numbers of each combination)

sum (The confirmation value returned by the function 'Search_Sum(...)' for the sum of the numbers of each combination)

Integer index variables (of type 'int *')
*Com (Pointer pointing to the first position of the dynamically bound integer table "Com" with each combination created, temporarily)
*Cnt (Pointer pointing to the first position of the dynamically bound integer table 'Cnt' with the appropriate counters, for the calculation of the number of combinations satisfying specified conditions)
*Freq (Indicator pointing to the first position of the dynamically bound integer table 'Freq' with the appropriate counters, to calculate the frequency of occurrence of each 'N' number in the combinations printed)

Search_Evens (int *C, int *E, int N)

Parameters
*C (Counter pointing to the first position of the dynamically bound integer array "Com" with each combination generated, temporarily)

N (The number 'N' of numbers in the interval (6, 49])
*E (Index pointing to the first position of the dynamically bounded integer matrix "Evens" for the interval [0, X1]U[X1, X2]U[X2, 6] of the number of even numbers of each combination)

Integer variables (of type "int")

j (Auxiliary variable for loop control)

evens (The confirmation value for the number of even numbers of each combination)

Search_Sum (int *C, int *S, int N)

Parameters

*C (Index pointing to the first position of the dynamically bound integer table "Com" with each combination created, temporarily)

N (The number 'N' of numbers in the interval (6, 49])
*S (Index pointing to the first position of the dynamically bounded integer table

"Sum" for the interval [21, Y1]U[Y1, Y2]U[Y2, 279] of the sum of numbers of each combination)

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

j (Auxiliary variable for loop control)

sum (The confirmation value for the sum of the numbers of each combination)

<u>Print_Combos (int *C)</u>

<u>Parameter</u>
*C (Index pointing to the first position of the dynamically bound integer table "Com" with each combination created, temporarily)

<u>Frequency (int *F, int *C)</u>

<u>Parameters</u>

*F (Index pointing to the first position of the dynamically bound integer table 'Freq' with the frequency counters of occurrence of each 'N' number in the combinations printed)
*C (Index pointing to the first position of the dynamically bound integer table 'Com' with each combination generated, temporarily)

<u>Print_Freq (int *F)</u>

<u>Parameter</u>

*F (Index pointing to the first position of the dynamically-bound integer table 'Freq' with the frequency counters of occurrence of each 'N' number in the combinations printed)

<u>Integer variables (of type 'int')</u>

i (Auxiliary variable for loop control)

Print_num_Combos (int N, int *Cnt)

<u>Parameter</u>
*Cnt (Pointer pointing to the first position of the dynamically bound integer table 'Cnt' with the appropriate counters, to calculate the number of combinations satisfying specified conditions)

Free_Memory (int *Array)

Parameter
 *Array (Pointer pointing to the first position of a dynamically bound integer array)

DISCUSSION

 See. Comments "Program "Combinations.c""

EXAMPLES

Example 1

```
Element [ 1] Value : 2

Item [ 2] Price : 6

Item [ 3] Price : 7

Item [ 4] Price : 12

Item [ 5] Price : 21

Item [ 6] Price : 32

Item [ 7] Price : 43

Item [ 8] Price : 45


N : 8


X1 : 2

X2 : 4


Y1 : 21

Y2 : 141


 2 6 7 12 21 32

 2 6 7 12 21 43
```

```
2 6 7 12 21 45
2 6 7 12 32 43
2 6 7 12 32 45
2 6 7 12 43 45
2 6 7 21 32 43
2 6 7 21 32 45
2 6 7 21 43 45
2 6 7 32 43 45
2 6 12 21 32 43
2 6 12 21 32 45
2 6 12 21 43 45
2 6 12 32 43 45
2 7 12 21 32 43
2 7 12 21 32 45
2 7 12 21 43 45
2 7 12 32 43 45
6 7 12 21 32 43
6 7 12 21 32 45
6 7 12 21 43 45

The number of combinations N per 6 is : 28
The number of combinations that did not meet the first condition : 0
The number of combinations that met the first but did not meet the second condition : 7
The number of combinations printed : 21

Frequency of [ 2] : 18
Frequency of [ 6] : 17
Frequency of [ 7] : 17
Frequency of [12] : 17
Frequency of [21] : 15
Frequency of [32] : 14

Frequency of [43] : 14

Frequency of [45] : 14
```

OBSERVATIONS

The limits on the sum of [21, Y1]U[Y1, Y2]U[Y2, 279] of the numbers a combination must be to be a candidate for printing have a logical explanation.

The "smallest" combination per "6" that can be created from numbers in the interval [1, 49] is "1, 2, 3, 4, 5, 6". If we add them up, their sum is equal to the value "21", so it is defined as the minimum limit for the sum of a combination per "6" with numbers in the interval [1, 49].

On the other hand, the 'largest' combination per '6' that can be generated by numbers in the interval [1, 49] is '44, 45, 46, 46, 47, 48, 49'. If we add them up, their sum is equal to the value "279", so it is defined as the maximum limit of the sum of a combination per "6" with numbers in the interval [1, 49].

COMPUTER PROGRAMMING

modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier

Thank you for your attention.