PROGRAMMING LANGUAGES

Laboratory 6

ASPECTS OF THE C++ PROGRAMMING LANGUAGE

I. THEORETICAL BACKGROUND

Remarks regarding the C++ programming language

The C++ programming language is a superset of the C language and is universally used. It permits writting object oriented programs.

The fact that the C++ programming language is a superset of the C language means that any program written in C language is in the same time a program compatible in C++. This assertion is true with few exceptions. For instance, the C++ compiler makes a few additional control checks. These controls mainly refer to the parameter types for the used functions and also to their return values types.

Editing functions

The functions which are called in a program can be edited in the following way:

* **The main function "main" and the called functions are written in the same file.**
* **The functions can be edited in separate files.**

If the functions are edited in separate files, there are two possible compilation ways:

1. Using the preprocessor directive #include in the main program in order to add the function files, which will have the .cpp (C++) file extension.

2. Using a Project file (with the .prj file extension). Such a file contains the name of each file (together with its extension) which is compiled and link-edited so as for the executable file to be generated.

In this file, not only source files can be indicated (.cpp), but also object type files (with the .obj extension) or even files with function libraries, others than the ones of the system.

Obs.: The "project" files for the Turbo C and C++ languages are not compatible. They are built using the Project menu of the two integrated development systems.

The advantages of the Project files refer to the fact that, at each launching, only the source files where changes occurred are automatically compiled. This is why, in the case of large source programs it is recomended to divide them in many source files which are compiled using Project files.

Generally, in a source file „related" functions are grouped, that is to say functions that process the same data subsets or are logically linked to one another.

NOTE: Since our assignment programs are generally not too large, we shall frequently use the file including method using the preprocessing construction #include.

II. ASSIGNMENT WORKFLOW

1. Write a program that calculates and lists m! for

m = 0, 1, 2, ..., 170.

The program will call the factorial function (from the previous assignment) in order to determine m! for a given m value inside the interval (0, 170). If m is not situated inside the interval (m<0 or m>170), the function will return -1. The "main" and "factorial" functions will be placed inside the same file.

2. Modify the above program so as to stop its course each time 23 new values of the factorial are computed. The program continues computing values when any key is hit by the user.

3. Divide the program from the previous point in two files with the ".cpp" extension. One of them should contain the "main" function and the other one the "factorial" function.

In the file with the main program, from which will be launched the compiling, the "stdio.h" and "conio.h" files will be included and also the file containing the "factorial" function will be included. Therefore, the include list in the file of the main program will look as follows:

```
#include<stdio.h>
#include<conio.h>
#include "factorial.cpp"
```

Obs.: **All the files created from now on will have the .cpp extension.**

4. Write in a separate file a function which has two integer parameters x and y, computes and returns the number of arrangements of x taken by y. The function returns the arrangements result.

First, the function tests whether x belongs to [1, 170] and y belongs to [1, x].
In error case, the function returns -1.

The following formula will be used:

$$A(x,y) = x * (x-1) * (x-2) * ... * (x-y+1)$$

5. Write a program that calculates and displays the number of arrangements of n taken by k, for:

n = 1, 2, ...30 and k = 1, 2, ... n .

Two "for" loops will be used. The development of the program will be stopped from place to place, every other 23 listings, in a similar way as in point 2.
This program calls the **arrangements** function defined in the above exercise.

III. SOLUTIONS:

1. L6_1.C

```
#include<stdio.h>

double factorial (int n);   /* the protoype of the factorial function */

 /* the main function*/
main( )   /* displays m! for m = 0, 1, 2, ..., 170 */
 {
   int m;

   for(m=0; m<171; m++)
      {
       printf ("m=%d\tm!=%g\n", m, factorial (m) );
      }
    getch( );
  }

double factorial (int n)
/* determines and returns n! if n is inside [0,170], otherwise returns -1 */
  {
   double f;
   int i;
   if ( n<0  ||  n>170)
     return (-1.0);
   for ( i=2, f=1; i<=n; i++)
     f*=i;
   return (f);
  }
```

2. L6_2.C

```
#include<stdio.h>

double factorial (int n);   /* the protoype of the factorial function */

 /* main program function*/
main( )   /* displays m! for m = 0, 1, 2, ..., 170 */
 {
   int m;

   for (m=0; m<171; m++)
```

```
      {
       printf ("m=%d\tm!=%g\n", m, factorial (m) );
       if ((m+1)%23==0)
            {
             printf ("Press any key to continue \n");
             getch( );
            }
      }
   getch( );
 }

double factorial (int n)
/* calculates and returns n! if n is situated inside the interval [0,170], otherwise returns -1 */
 {
  double f;
  int i;
  if ( n<0  ||  n>170)
    return (-1.0);
  for ( i=2, f=1; i<=n; i++)
    f*=i;
  return (f);
 }
```

3.1. L6_3_1.CPP

```
double factorial (int n)
/* calculates and returns n! if n is situated inside the interval [0, 170], otherwise returns -1 */
 {
  double f;
  int i;
  if ( n<0  ||  n>170)
    return (-1.0);
  for( i=2, f=1; i<=n; i++)
    f*=i;
  return (f);
 }
```

3.2. L6_3_2.CPP

```
#include<stdio.h>
#include<conio.h>
#include "l6_3_1.cpp"

main( )   /* displays m! for m = 0, 1, 2, ..., 170 */
 {
  int m;

  for (m=0; m<171; m++)
     {
      printf ("m=%d\tm!=%g\n", m, factorial(m) );
      if ((m+1)%23==0)
           {
            printf ("Press any key to continue \n");
            getch( );
           }
     }
```

```
   getch( );
 }
```

4. L6_4.CPP

```
double arrangements (int x, int y)
/* calculates and returns the arrangements of (x, y) */

{
 double  a;
 int i;

 if (x<1  ||  x>170)
       return -1.0;
 if (y<1  ||  y>x)
       return -1.0;
 a=1.0;
 i=x-y+1;
 while (i<=x)
       a*=i++;
 return a;
}
```

5. L6_5.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l6_4.cpp" /* contains the arrangements function */

main( ) /* calculates and returns arrangements(n, k), if n is inside [1, 30] and k is inside [1, n] */

{
 int k, n;

 for (n=1; n<=30; n++)
      {
       printf ("Press any key to continue\n");
       getch( );
       printf ("\nn= %d\n", n);
       for (k=1; k<=n; k++)
        {
         printf ("\tk=%d\tA(n, k)=%g\n", k, arrangements(n, k));
         if (k%20==0)
          {
           printf ("Press any key to continue \n");
           getch( );
          } /* end if*/
        }  /* end interior for */
      }   /* end exterior for */
 printf ("End of the program\n");
 getch( );
```

```
}      /* end of main */
```