PROGRAMMING LANGUAGES

Laboratory 7

CLASSES OF VARIABLES

I. THEORETICAL BACKGROUND

1. Classes of variables (review, completing)

When compiling a source program (file), the compiler allocates memory for program variables. You can assign variables on the stack, in the computer registers and in other areas of memory.

The program defines, with statements, the allocation of variables. You can define variables (visible) in the entire program and variables that are useful locally (with limited visibility). Variables that can be used throughout the entire program are called global variables, and those that can be used only in a certain part of the program are called local variables.

1.1. Global variables

The global variables have a definition and possibly one or more external declarations of variable.

The definition of global variables coincides with the usual statement that is written out of the body of any of the program functions. Such a definition is usually written at the beginning of a source file. This is because it is valid from the place where is written to the end of that source file.

If the program consists of several source files, a global variable can be used in a spurce file where it is not defined, if it is declared as external variable in that file. An external variable declaration coincides with a common variable declaration preceded by the keyword **extern**.

Example:

A program consists of two source files f1.cpp and f2.cpp. In f1.cpp file we define global variables day, month, and year as integers (int). These variables are used both in f1.cpp and in f2.cpp. If the f1.cpp file is included in the f2.cpp file, those variables are declared as extern in the f2.cpp file functions.

1.2. Local variables

Local variables, as opposed to global ones, are not valid in the entire program. They are available locally in the unit where declared.
Local variables are allocated on the stack. In this case they are called automatic variables. They are usually declared in a function body or at the beginning of a block instruction.

When a function is called, the automatic variables (usually declared before the first instruction from that function body) are allocated on the stack. At the return from that function, the automatic variables allocated to the stack are eliminated (clean stack). Result that the automatic variables lose their existence at the returning from the function where are declared. Therefore, an automatic variable is valid only in body of the function where was declared.

Note: The same way are automatic variables declared at the beginning of a block instruction. Such a variable is allocated on the stack when the control of the program get to the block instruction where is declared the respective variable and it is eliminated from the stack when the control of the program goes to the next following instruction.

The local variables may not be allocated on the stack. For this purpose they are declared to be static. A static variable declaration is a common statement preceded by the keyword **static**.

Static variables can be declared both in the body of a function and outside any function. Unlike automatic variables, a static variable is not allocated on the stack at the execution, but at compilation in a memory zone intended for them.
* A static variable that is declared in a function body is valid only in that body function.

* A static variable, which is declared outside functions bodies, it is valid in the whole source file where it was defined.

Unlike the global variables, such a variable can't be declared as external. Therefore, it can't be used in other files if they compile separately or they are included before the respective declaration statement.

2. The initialization

Often we want to initialize some variables in the program baseline. Thus, for global variables we can give initial values to define them. Similarly, other classes of variables can be initialized at their declaration.

2.1. Initialization of simple variables

A simple variable can be initialized by a construction like:

type name = expression;

for automatic or global variables, or:

static type name = expression;

if the variable is static.

Global or static variables can be assigned with initial values at compiling, so they have the respective values at the start of the program. The fact that these variables are initialized at compiling requires that the values used for initialization to be constant.

* Global and static variables are initialized by default with zero.

* An automatic variable, uninitialized, has an unpredictable value when the function calls this variable. Thus, its value remains undefined until it has assigned a value by a statement.

In all cases, there will be conversions if the expression type that initializes the variable is not similar with the type of variable.

Examples:

a. int n=10;
/* defining a global variable. n is set to 10 at the beginning of the program */

b. #define MAX 100

int i=MAX*2;

/* i is a global variable initialized to 200 */

c. int y;
/* y is a global variable and is 0 at launch */

d. static int h;

/* h is a static variable and is 0 at launch */

e.
```
type function (int m)
{
 int z=5;
 int a=z+m;
 ...
}
```
/* z and a are automatic variables initialized with values 5 and z + m */

II.ASSIGNMENT WORKFLOW

Note: All files that will be elaborated in this laboratory will have a .cpp extension.

1. Write a function to a file

double factorial (int n)

which calculates and returns n! for n in the range [0, 170]. If the factorial function finds parameter n<0 or n>170, it will return -1.0.
The automatic variables will be initialized at declaration.

2. Write a function in a file that asks to introduce two integers x and y, then calculate and return the arrangements of x taken by y (more specific A(x, y)). At first, the function checks if x belongs to [1, 170] interval, and y to [1, x] interval. In case of error, the function returns -1.
The formula for the arrangements calculation is:

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

Automatic variables are initialized at declaration.

3. Write a function in a file that asks to introduce two integers x and y, then calculate and return the y-combinations of x (more specific C(x, y)). At first, the function checks if x belongs to [1, 170], and y to the [0, x] interval. In case of error, the function returns -1.
The formula is:

C(x, y) = A(x, y) / y!

For this purpose, the function calls permutations and factorial functions defined in the previous tasks. Automatic variables are initialized at declaration
How do you solve the cases y==0 and y==x ?

4. Write a program that calculates and displays k-combinations of n, for

n = 1, 2, ...30 and k = 0, 2, ... n .

Use two cycles of "for". The program will be stopped, from time to time, for viewing the screen. Continuation of the program will be made by pressing a key. This program calls the function defined in the previous task.

5. A faster computation of the combination function is achieved if one takes into account the relationship:

$$\mathbf{C(x,y) = C(x,x-y)}$$

This relationship is useful to apply for

$$\mathbf{y > x/2}$$

Change the file from section 3. Respect the new computing idea and then test the validity changes by using the program from section 4.

6. Another way to calculate the number of combinations is the relationship:

$$\mathbf{C(x, y) = (x/1) * ((x-1)/2) * ((x-2)/3) * ... * ((x-y+1)/y)}$$

In this case, we avoid the calculation of P (x, y) and y! which are growing rapidly with increasing values of x and y.

Change the file from section 3. Respect the new computing idea and then test the validity changes by using the program from section 4.

Note that for this version is no longer necessary to call permutation and factorial functions.

7. According to the example of the theoretical aspects (paragraph 1.1.), create two files that contain common integer variables day, month and year, but in one of them there are global variables, and in other external. The global variables are initialized at declaration.

The file where those variables are declared as extern will contain the main function which will display these variables values.

III.SOLUTIONS:

1. L7_1.CPP

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

2. L7_2.CPP

```
double arrangements (int x, int y)
/* compute  y-permutations of x (more specific A(x,y))*/

{
 double a=1.0;
 int i=x-y+1;

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

3. L7_3.CPP

```
#include "l7_1.cpp"
#include "l7_2.cpp"

double combinations (int x, int y)
/* compute y-combinations of x (more specific C(x,y))*/

{
 if (x<1  ||  x>170)
    return -1.0;
 if (y<0  ||  y>x)
    return -1.0;
 if (y==0  ||  y==x)
    return 1.0;
 return arrangements(x, y)/factorial(y);
}
```

4. L7_4.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_3.cpp" /* contain combinations function */

main( ) /* Compute k-combinations of n, for n in the range [1, 30] and k in the range [0, n] */

{
 int k, n;

 for(n=1;  n<=30;  n++)
       {
        printf ("Push a key\n");
        getch( );
        printf ("\nn= %d\n", n);
        for (k=0;  k<=n;  k++)
         {
          printf ("\tk=%d\tC(n, k)=%g\n", k, combinations(n, k) );
          if ((k+1)%20==0)
           {
            printf ("Push a key \n");
            getch( );
           } /* end if*/
         }  /* end interior for */
       }   /* end exterior for */
 printf ("End\n");
 getch( );
}      /* end main */
```

5.1. L7_5.CPP

```
#include "l7_1.cpp"
#include "l7_2.cpp"

double combinations (int x, int y)
/* compute y-combinations of x */
{
 if (x<1  ||  x>170)
    return -1.0;
 if (y<0  ||  y>x)
    return -1.0;
 if (y==0  ||  y==x)
    return 1.0;
 if (y>x/2)
    return arrangements(x, x-y)/factorial(x-y);
 else
    return arrangements(x, y)/factorial(y);
}
```

5.2. L7_5MAIN.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_5.cpp" /* contains the combinations function */

main( ) /* compute k-combinations of n, for n in the range [1, 30] and k in the range [0, n]  */
{
 int k, n;

 for (n=1; n<=30; n++)
       {
        printf ("Push a key \n");
        getch( );
        printf ("\nn= %d\n", n);
        for (k=0; k<=n; k++)
         {
          printf ("\tk=%d\tC(n, k)=%g\n", k, combinations (n, k));
          if ((k+1)%20==0)
            {
             printf ("Push a key \n");
             getch( );
            } /* end if*/
         }  /* end  interior for*/
       }   /* end exterior for*/
 printf ("End\n");
 getch( );
}      /* end main */
```

6.1. L7_6.CPP

```
double combinations (int x, int y)
/* compute y-combinations of x (more specific C(x,y)) */
{
 int i;
 double c=1.0;
 if (x<1  ||  x>170)
      return -1.0;
 if (y<0  ||  y>x)
      return -1.0;
 if (y==0  ||  y==x)
     return 1.0;
 for (i=1; i<=y; i++)
 c=(c*(x-i+1))/i;
 return c;
}
```

6.2. L7_6MAIN.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_6.cpp" /* contains the combinations function*/

main( )/* compute k-combinations of n, for n in the range [1, 30] and k in the range [0, n]  */

{
 int k, n;

 for (n=1; n<=30; n++)
       {
        printf ("Push a key\n");
        getch( );
        printf ("\nn= %d\n", n);
        for (k=0; k<=n; k++)
         {
          printf ("\tk=%d\tC(n, k)=%g\n", k, combinations(n, k) );
          if ((k+1)%20==0)
            {
             printf ("Push a key \n");
             getch( );
            } /* end if*/
         }  /* end interior for */
       }   /* end exterior for*/
 printf ("end\n");
 getch( );
}      /* end main */
```

6.1. L7_7_1.CPP

```
int day=7, month=5, year=1999;
/* global variables definitions*/
```

6.2. L7_7_2.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_7_1.cpp"
main( )
{
 extern int day, month, year;
 printf ("\nDay=%d\tMonth=%d\tYear=%d\n", day,  month, year);
 getch( );
}
```