STATEMENTS in C

1. Overview

The learning objective of this lab is:

- To understand and proper use statements of C/C++ language, both the simple and structured ones: the expression statement, the empty statement, the composed statement, the "**if"** statement, the "**switch"** statement and the repetitive statements.
- To correctly develop small programs using C statements

2. Brief theory reminder

The structured program is a program having a control structure built only based on:

- Sequential structure;
- Alternative and selective structure;
- Repetitive structure.

In the C/C++ languages there are also statements like **return, break, continue** and **goto**, that assures a great flexibility in writing C/C++ programs.

2.1. The expression statement:

The syntax of this statement is:

expression;

i.e. after the expression it is written "**;**".
This statement is used as assignment statement or as a function call statement. A usage example is:

```
/* Program L04Ex1.c */

/*The program displays the maximum of two integers */
#include <stdio.h>
int main()
{
    int a, b, c;
    printf("\nPlease input two integers, a and b\n");
    scanf("%d %d", &a, &b);
    c=a>b? a: b;
    printf("\nThe maximum of a=%d and b=%d is c=%d\n",a,b,c);
    return 0;
}
```

2.2. The empty statement

This statement consists of a single **';'** (the semicolon character), and has no effect. This statement is used when an dummy statement is needed according the syntax requirements of a composed or repetitive statement.

Usage example:

```
for( i = 0, s = 0; i < n; s = s + a[i], ++i);
```

2.3. The composite statement

A composite statement consists of a sequence of statements enclosed between braces (i.e. '{' and '}'), sometimes preceded by declarations, as follows:

```
{
   declarations;
   statements;
}
```

This statement is used when only one statement is needed according the syntax requirements of a program context, but the computational process requires more than one statement.

An usage example is given in L04Ex2.c :

```
/*  Program L04Ex2.c  */
/*  Computes of the roots of the equation a*x^2 +b*x +c =0   */
#include <stdio.h>
#include <math.h>
int main()
{
    float a, b, c, delta, x1, x2;
    printf("\nComputes of the roots of the equation\n\ta*x^2 +b*x +c =0\n");
    printf("Please input values for a, b, and c\n");
    scanf("%f %f %f", &a, &b, &c);
    if (a!=0)
    {
        delta=b*b-4*a*c;
        if (delta >= 0)
        {
            x1=(-b-sqrt(delta))/(2*a);
            x2=(-b+sqrt(delta))/(2*a);
            printf("\nThe equation has the roots x1=%g and  x2=%g\n", x1, x2);
        }
        else
        {
            x1=-b/(2*a);
            x2=sqrt(-delta)/(2*a);
            printf("\n\nThe equation has complex roots x1=%g-j*%g\
and  x2=%g+j*%g\n", x1, x2, x1, x2);
        }
    }
    else printf("\nEquation is not of second order (i.e. a=0)\n");
    return 0;
}
```

2.4. The conditional statement „if"

This statement has two forms:

a) **if (expression)**
   statement

b) **if (expression)**
   statement_1
   else statement_2

The effect is:

- the evaluation of the expression "expression".
- if the result of the evaluation is **true** the statement **„statement"** it is executed in case a) and, in case b) the statement **"statement_1"**, after the control is switched to the statement that immediately follows the **if** statement.

- if the result of the evaluation of the expression **"expression**" is **false**, there are the following cases:
  a) control is switched to the statement that follows immediately the **if** statement, and,
  b) statement **"statement_2"** is executed; then control is switched to the statement that logically follows the „**if"** statement.

  Notes:
  a) The statements **"statement"**, **"statement_1"**, **"statement_2"** may contain any jump statement (i.e. statements that transfer the control directly to statements that are not necessarily placed immediately after the **„if"** statement).
  b) The statement **„if"** may contain other **„if"** statements. In such a case it is important to handle the **„else"** branch carefully, in order to combine it correctly with the logically corresponding **„if"** statement.

  Usage example: Program L04Ex2.c (see 2.3).

2.5. The selection statement „switch"

The syntax of the „**switch"** statement is:

```
switch ( expression )
{
 case C1:  statement_sequence_1;
           break;
 case C2:  statement_sequence_2;
           break;
 ..........................................
 case Cn:  statement_sequence_n;
           break;
 default:  statement_sequence;
}
```

The effect of **switch** statement is:
a) Evaluation of the expression "expression";
b) The result of the evaluation is compared with the integer constants **C1, C2,** …, **Cn**. If this result is equal with constant **Ci**, it is executed the statement " **statement_sequence_i**", then it is executed the statement placed immediately after the „**switch**" statement. If the result of the evaluation does not match any of the constants C1, C2, …, Cn, the sequence „**statement_sequence"**, placed after the label "**default**", is executed.

Notes:
a) The **default** branch is optional. If the branch **„default"** is missing and the value of the expression **„expression"** does not match none of the constants **C1, C2, ..., Cn**, the statement **switch** has no effect.
b) If the keyword **break** is missing, all the statements that follow there are executed, until either a **break** keyword or the end of the switch statement is encountered.
c) The structured **switch** statement may be replaced by an appropriate set of nested **if** statements.

Usage example:

```
/*  Program L04Ex3.c  */

/* Operations on integers of the form OPERAND1 operator OPERAND2        */
 #include <stdio.h>
 #include  <stdlib.h>
  int main()
 {
    int operand1, operand2, result;
    char operation;
```

```
    printf("\nInput an infix expression without spaces\n");
    scanf("%d%c%d", &operand1, &operation, &operand2);
    switch(operation)
    {
        case '+':   result=operand1+operand2;
                    break;
        case '-':   result=operand1-operand2;
                    break;
        case '*':   result=operand1*operand2;
                    break;
        case '/':   if (operand2!=0) result = operand1/operand2;
                    break;
        default:    exit(1);
    }
    printf("\n%d %c %d = %d\n", operand1, operation, operand2, result);
    return 0;
}
```

2.6. The loop statement „while"

The general syntax of the statement „**while**" is:

```
while ( expression )
    statement
```

The effect of the statement **while** is:

a) Evaluation of the expression „expression";
b) If the result of the evaluation is **true**, it is executed the statement „statement" and it is performed again the step a). If the result of the evaluation is **false,** it is executed the statement that is placed immediately after the statement **while**.
   a. Observations:
c) In case that the expression „expression" is **false** from the beginning, then the statement "statement" is never executed.
d) In the body of the statement **while** it is necessary to have some statements which change the value of the variables from the expression "expression".

Usage example:

```
/*  Program L04Ex4.c  */

/* Computes of the greatest common divisor (gcd) and
   the smallest commom multiple (scm) of two natural numbers, a and b  */
#include <stdio.h>
int main()
{
    int a, b, a1, b1, gcd, scm, remainder;
    printf("Computes of the greatest common divisor (gcd) and\n");
    printf( "the smallest commom multiple (scm) of two natural numbers, a and b"0;
    printf("Input value for a=");
    scanf("%d", &a);
    printf("Input value for  b=");
    scanf("%d", &b);
    /* Computation of gcd */
    a1=a;
    b1=b;
```

```
    while ((remainder=a1%b1)!=0)
    {
        a1=b1;
        b1=remainder;
     }
     gcd=b1;
     scm=a*b/gcd;
     clrscr();
     printf("a=%d  b=%d  gcd(a,b)=%d  scm=%d", a, b, gcd, scm);
    return 0;
  }
```

2.7. The loop statement „for"

The syntax of the statement **for** is:

```
for ( expr1; expr2; expr3 )
    statement
```

where:

- expr1, expr2, expr3 are expressions;
- **„statement"** is the body of the **„for"** loop.

The **for** loop, can also be expressed using the **while** loop as follows:

```
expr1;
while ( expr2 )
{
    statement;
    expr3;
 }
```

Note: **expr1, expr2, expr3** may be empty, but the presence of **";"** is mandatory.

Usage example:

```
/*  Program L04Ex5.c   */

/* Computes the arithmetic mean value of n real numbers */
#include <stdio.h>
int main()
{
    float a[100], mean, sum;
    int i, n;

    printf("Computes the arithmetic mean value of n (<100) real numbers\n");
    printf("\nInput the number of terms, n=");
    scanf("%d",&n);
    printf("\nInput the terms\n");
    for( i=0, sum=0; i<n; ++i)
    {
        printf("a[%2d]=", i);
        scanf( "%f",&a[i]);
        sum+=a[i];
     }
    mean=sum/n;
    printf("\nMEAN=%g\n", mean);
    return 0;
```

```
}
```

2.8. The loop statement „do-while"

The loop statement "**do-while"** executes the test at the end of the loop. Its syntax is:

```
do
    statement
while ( expression );
```

The effect in terms of the statement „**while**" is:

```
statement;
while( expression )
    statement;
```

Note that the body of the loop is executed at least once.
Usage example:

```
/*  Program L04Ex6.c  */

/* Computes the greatest common divisor (gcd) and smallest common multiple (smc)
    Of two natural numbers, a and b */
#include <stdio.h>
int main()
{
    int a, b, a1, b1, gcd, smc, remainder;

    printf("Computes of the greatest common divisor (gcd) and\n");
    printf( "the smallest commom multiple (scm) of two natural numbers, a and b");
    printf("Input a value for a=");
    scanf("%d",&a);
    printf("Input a value for  b=");
    scanf("%d",&b);
    /* Find the gcd */
    a1=a;
    b1=b;
    do{
            remainder=a1%b1;
            a1=b1;
            b1=remainder;
       }
     while (remainder != 0);
     gcd=a1;
     smc=a*b/gcd;
     clrscr();
     printf("a=%d  b=%d  gcd(a,b)=%d smc=%d", a, b, gcd, smc);
     return 0;
 }
```

2.9. The statements „continue" and „break"

The statements „**continue"** and „**break"** may be used only in the body of a loop.
The statement "**continue"** makes control leave the current iteration and continue with the first statement in the case of the statement **for.** For the **while** and **do-while** statements: execution continues with the evaluation of the expression which determines whether the loop will be executed again or not.

The statement „**break"** breaks the loop (**for, while, do while**), and the statement immediately following the loop gets executed.

2.10. The unconditional jump statement „goto"

The statement **goto** is used to jump between different locations in a program. The destination is always a labeled statement.

A label is a name, followed by the colon character **":"** , such as:

```
label:
```

The syntax of the statement **goto** is:

```
goto label;
```

Example:

```
...
goto alfa;
   ...
alfa: if ( ) ...
...
```

2.11. The standard function „exit"

The prototype of the standard function **exit** is described in **stdlib.h** and **process.h** as:

```
void exit(int code);
```

The function **exit** causes the end of the program. The value of the integer **„code"** is zero for a normal termination and different from zero in case of errors.

3. Lab Tasks

3.1 Analyze and execute of the programs given as examples in Section 2.

3.2. Four pairs of real numbers are acquired from the standard input, representing the vertices of a polygon with 4 edges. Establish the nature of this polygon.

3.3. Real numbers of a sequence of size ***n*** are read from the standard input. Find and print to standard output: the minimum and the maximum values of this sequence, and their positions (indices) in the sequence.

3.4. Write a program to generate all the prime numbers less than or equal to a natural number, ***n.***

3.5. Read from the standard input a natural number, ***n***. Find the greatest perfect square that is less than or equal to ***n***. Then find the least prime number that is greater than or equal to ***n***.

3.6. Read from the standard input a natural number, ***n***. Check if this number is palindrome.

3.7. Read from the standard input the hexadecimal digits of an integer hexadecimal number. Find and display the equivalent decimal number.

3.8. Read from the standard input the degree and the coefficients of the polynomial $p(x) = a_0+a_1x^1+ ... + a_nx^n$. Compute and display the value of the polynomial for $x = x_0$ (x_0 is read from the standard input).

3.9. Write a program to perform the operations +,-,×, / on two polynomials:

$A(x)=a_0+a_1x^1+...+a_nx^n$
$B(x)=b_0+b_1x^1+.....+b_mx^m$
The degrees and the coefficients are read from the keyboard.

3.11. Given a system of ***n*** equations with ***n*** unknown variables, write a program to solve this system using a numeric method.

3.12. Compute the polynomials ***P***(***x***) and ***Q***(***x***) from the relation:

$$\frac{Q(X)}{P(X)}=\sum_{i=1}^{n}\frac{a_i}{b_ix+c_i}$$

n, a_i , b_i , c_i are read from the standard input.

3.13. Given a sequence of ***n*** real numbers sorted in ascending order, verify if a given value, ***x***, exists in the given sequence, and display this value and its position.

3.14. Given a sequence of ***n*** integer numbers, extract the maximum length subsequence which is in ascending order.

3.15. For a capability test there exists a set of ***n*** questions, each question, ***i*** , having a value of p_i points. Write a program which generates all the tests with ***q*** questions, each such test having assigned between ***a*** and ***b*** points.

3.16. Given two strings of ***n*** and ***m*** integer elements, compute:
a) The string that contains all the elements belonging to both strings.
b) The string of all the elements of the two given strings, written once.
c) The string of the elements from the first string, without the elements that are also in the second string.

3.17. Given a real number ***a*** written in base 10, write a program to convert this number in base ***B***, where $B \leq 16$.

3.18. Given a natural number ***n***,
a) Find the number obtained by eliminating those digits that appear more than once in that number.
b) Find the number obtained by switching the first digit with the last one, the second with the next to last one, and so on.
c) Find the biggest number that could be obtained by a combination of its digits.

3.19. Given a matrix of $n \times n$ elements all 0 or 1, verify if this matrix is symmetric.

3.20. Read a sentence from the standard input. Compute the number of the words and find and display the longest one.