Laboratory 2: “Loops and Conditionals“

Leszek Ciopiński, MSc
Arkadiusz Chrobot, PhD
Grzegorz Łukawski, PhD

September 4, 2026

1 Introduction

In the second section of this manual the conditional statements are discussed. The third section describes iteration statements (loops). The third section is about several numerical methods. This manual ends with a set of programming exercises for solving during the laboratory classes.

2 Conditional Statements

Conditional statements include the `if` statement (which is called a conditional statement by itself), the `switch` statement and ternary operator.

An example of the `if` statement could be found in the listing 1.

```
if(a==b)
        a=5;
else
        b=5;
```

Listing 1: The `if` statement example

The condition in parentheses don't have to be simple. It can be build with the use of any operators, but usually the relational and boolean operators are applied. **Be careful**: the C allows programmers to use in the conditions the `=` operator (the assignment operator) that looks very similar to the equal operator (`==`). They perform different operations, hence they give different results. Usually using the first one (the assignment) in a condition is a programmer mistake. However, advanced programmers deliberately use such an operator in conditions, if they can simplify the code in that way. Compilers tend to warn programmers if they detect an assignment operator inside the condition.

The `else` keyword, together with its entire branch of the `if` statement can be skipped if there is no need to use it. Please note, that in the example, the statements that precede and follow the keyword are ended with a semicolon (`;`).

Apart from single statements blocks of statements may be placed behind the condition and the `else` keyword. A block of statement starts with an opening curly brace, usually contains several statements (although it may contain only one statement or even be empty), and ends with a closing curly brace. Some C experts recommend using always blocks in `if` statements, even if they would only have a single statement inside. Sometimes such blocks prevent programmers from making mistakes in the code.

The `if` statement can be replaced by the ternary operator. If the condition that precedes the question mark is met then the operator returns the value of the expression that follows the question mark, otherwise it returns the value of the expression that follows the colon. An example that uses the `if` statement and the ternary operator to perform the same operation — finding the maximal value of the variables `a` and `b` — is given in the listing 2.

```
int a=5, b=4, max;

if(a>b)
        max=a;
else
        max=b;

max=(a>b)?a:b;
```

Listing 2: The `if` statement and the ternary operator

An example of using the switch statement is given in the listing 3.

```c
int a;

switch(a) {
        case 1: puts("One");
                break;
        case 2: puts("Two");
                break;
        case 3: puts("Three");
                break;
        default: puts("Unrecognized number");
}
```

Listing 3: The `switch` Statement

The `default` keyword (which corresponds to the `else` keyword in the `if`) and the statements related to it don't have to be used, if they are not necessary. Please note, that each case ends with a `break` statement (keyword). If its is not present then the program will preform statements in the other cases until it finally finds the `break` or the `switch` statement ends.

Not using the `break` statement in the `switch` is not necessarily a mistake. Sometimes it is done deliberately by a programmer to achieve a specific behaviour of the code.

The `break` keyword may be replaced by the `return` keyword or similar statement if we want to terminate the execution of a function or a program.

3 Loops

A three types of iteration statements, or loops, are available in the C: `for`, `while` and `do…while`[1].

3.1 The `for` Loop

The `for` loop has the following overall form:

`for(initialization;condition;afterthought) body`

where `initialization` means assigning an initial value to the loop counter, `condition` is an expression that if true the loop continues otherwise the loop stops and the `afterthought` is an expression that changes the value of the loop counter in each iteration (repetition) of the loop. The `body` is the part of the loop that is repeated and it can be a single statement or a block of statements (or it can even be empty). The part of the `for` loop above the body is called a *header*. In C a loop counter can be variable of any type that we have learned about. Some examples of the loops are given in the listening 4.

[1] The "loops" build with the use of the `goto` statement can be considered as the forth type of loop, but we won't discuss it in this manual

```c
int a;
for(a=0;a<5;a++)  /* This loop performs 5 iterations. Although its body is
        ;            empty, the loop has a side effect: the final value of
                     "a" is 5. */

for(a=1;a<=5;a++)              // This loop performs 5 iterations. The final
        printf("%d\n",a);      // value of "a" is 6. Inside a loop the pritnf()
                               // function is called that displays the value
                               // of the loop counter in each iteration.

for(a=0;a<7;a+=2)              // The new thing in this loop is that the loop
        printf("%d\n",a);      // counter in incremented by 2 in each iteration.
                               // Its final value is 8 (which is not displayed
                               // on the screen).

for(a=7;a>0;a--)               // In this loop the loop counter is decremented
        printf("%d\n",a);      // by 1 in each iteration.

int i,j;
for(i=7,j=0;i>j;j++,i--)        // In this loop two loop counters are used. The
        printf("%d %d\n",i,j); // first (the "i" variable) is decremented and
                               // the second (the "j" variable) - incremented.

double x;
for(x=0.0;x<0.5;x+=0.01)        // In this loop the loop counter of the double
        printf("%.10lf\n",x);  // type is applied.

a=0;
for(i=0; i<5; i++) {            // Often inside a loop a block of statements
        a+=i;                   // instead of single statement is performed.
        printf("%d\n",a);
}
```

Listing 4: The `for` loop examples

3.2 The `while` And `do…while` Loops

The `while` loop is performed as long as its condition stays true. If the condition is never satisfied the loop is never performed. The `do…while` is always performed at least once, because its condition is verified after its body is executed. The body of each of these loops should include a statement that makes the condition eventually false. Several examples of both loops is given in the listing 5.

```c
char a=0;
while(a!='q')                // This loop is performed until the user presses
        scanf("%c",&a);      // the "q" key and confirm it by pressing Enter.

int x=0,y=0;
while(y>=0) {                // This loop is performed until the user inputs
        scanf("%d",&y);      // a negative integer. Inside the body of the
        x+=y;                // loop two statements are performed.
}

while(getchar()!='q')        // This loop performs the same operation as the first
        ;                    // one. The getchar() function returns the  ASCII code
                             // of a character read from the keyboard. Please
                             // note the semicolon that ends the loop.

do                            // This loop performs the same operation as the
        scanf("%c",&a);       // first one and the second one. Please note that
while(a!='q');                // it is not necessary to initialize the "a" variable
                              // because it will always be given value before the
                              // condition is checked.

x=y=0;
do {                          // In this loop's body two statements are performed.
        x+=1;
        y*=y;
} while(x!=10);
```

Listing 5: Examples of `while` and `do…while` loops.

The conditions for both loops are build in the same way as for the `if` statement.

3.3 The `break` And `continue` Statements

The `break` statement can be used inside a loop, together with the `if` statement or the ternary operator. In that case the `break` terminates the loop. An example is given in the listening 6.

```c
int a;
for(a=0;a<10;a++) {
        printf("%d\n",a);
        if(a==5)
                break;
}
```

Listing 6: An example of using the `break` statement

The `break` statement can be also applied in the `while` or `do…while` loop Please note, that if used inside the `switch` statement, nested in a loop, the `break` keyword ends a single case and don't terminate the loop. An example of such a case is given in the listing 7.

```
int i;

for(i=0;i<3;i++)
        switch(a) {
                case 0: puts("zero");
                        break;
                case 1: puts("one");
                        break;
                case 2: puts("two");
       }
```

Listing 7: An example of using the `switch` statement inside the `for` loop

In a similar way the `continue` statement can be used inside the loop's body. However this statement works differently than the `break` — it terminates the current iteration of the loop and stars another one, if it is possible i.e. the loop's condition is still satisfied. An example is given in the listing 8.

```
int i;
for(i=0;i<11;i++) {             // Only even numbers are printed in this loop.
        if(i%2)
                continue;
        printf("%d\n",i);
}
```

Listing 8: An example of using the `continue` statement in a loop

The `continue` statement can be applied inside the `while` and `do…while` loops.

3.4 Infinite Loops

Examples of infinite loops in C are given in the listing 9. Despite their names these loops can be useful, but contrary to the examples they have to include in their bodies some statements (like the `break`) that will eventually terminate them.

```
for(;;);   // An empty infinite for loop.
while(1);  // An empty infinite while  loop.
do; while(1); // An empty infinite do...while loop.
```

Listing 9: Examples of infinite loops

4 Introduction to Numerical Methods

In this section an overview of several numerical methods that find the square root of a given number or a value of a given function are given.

4.1 The MacLaurin Series

The McLaurin series is a special case of the Taylor series. In mathematics it is used for evaluating such functions as $sin(x), cos(x)$ and e^x.

For the $sin(x)$ function the series takes the following form :

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \tag{1}$$

where x is an angle expressed in radians.

From the programmer point of view it should be noted that each subsequent term of this series differs from the previous one by the $(-1) \cdot \frac{x^2}{(2 \cdot i) \cdot (2 \cdot i + 1)}$ factor, where $i = 1, 2, \ldots$ is the number of the position of the term in the series (where $i = 1$ is for the $\frac{x^3}{3!}$ term).

4.2 Leibniz Series

Using the Leibniz series can be applied for finding the value of π. The series is defined as follows:

$$\sum_{n=1}^{\infty} (-1)^{n-1} \frac{1}{2n-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4} \tag{2}$$

The algorithm for calculating π that is based on the series is quite slow, so it can take it several dozen of seconds to output the answer, even if it's performed by a modern computer.

4.3 Newton's Method

The Newton's Method is an algorithm that finds the root of a given function. Particularly, it can be applied for finding the square root of a non-negative real (in case of computer science — floating–point) number. Let's assume that y is that number and x is an approximated value of its square root. The following formula can be applied to find the approximation of the square root:

$$x_{k+1} = \frac{(x_k + \frac{y}{x_k})}{2} \tag{3}$$

where k is the number of the approximation. As the initial value of the x one can be assumed ($x_0 = 1$). The calculation should be performed inside a loop, as long as the precision of the approximation is not satisfying.

5 Exercises

1. Write a program that counts down with words to the launch of a rocket. The counting should start with a number provided by the user. For example, the user inputs 5 and the program displays: "five", "four", "three", etc. in each line on the screen.

2. Write a program that uses any type of loop to find the sum of an arithmetic progression. The fist and the last term and also the common difference should be given by the user. You are not allowed to apply the Gaussian formula for the sum of the arithmetic progression.

3. Write programs that perform the same calculations as the program from the problem 2, but use the other types of loops that are described in this manual.

4. Write a program that uses any type of loop to find the sum of the geometric progression. The initial and the last term and also the common ratio should be given by the user, but they should be also verified by the program before using. You are not allowed to use the formula for geometric progression sum.

5. Write programs that perform the same calculation as the program from the problem 4, but use other types of loops that are described in this manual.

6. Using the Leibniz series write a program that calculates an approximated value of π.

7. Using the MacLaurin series write a program that calculates the value of the $sin(x)$ function for $x = \frac{\pi}{3}$. The program should allow the user to determine how many terms of the series should be used.

8. Write a program that takes a non–negative floating–point number from the user and finds the square root of that number using the Newton's Method.