Laboratory 1: “The Basics of the C Language”

Arkadiusz Chrobot, PhD
Grzegorz Łukawski, PhD

September 4, 2026

1 Introduction

This manual describes the basic algorithmic structures in the C language. The first section explains how comments can be used in the program's source code. The second section covers the topics of variables and data types. In the third section the operators available in the C language are discussed. The fourth section describes the ternary (conditional) operator. Finally, in the last section functions that make user – computer communication possible are discussed. The manual ends with a set of programming exercises for solving during the laboratory classes.

2 Comments

There are three ways of making a comment in the source code that are available in the C language. The content of a comment may be placed between the following pairs of characters: `/*` and `*/`. Such a comment may expand for several lines, but it cannot be nested in other comment of the same type. The next type of comments comes from the C++ language and it has become available in the C since the ISO C99 standard. Such a comment starts with the following characters: `//` and ends with the end of the line. The third type of comments is used scarcely, usually when the other two cannot be applied. Such comments are created with the use of preprocessor directives. Listing 1 shows several examples of comments in the C language.

```c
int main(void)
{
    #if 0
            This is the first type of comments in the C language.
            As you can see, such a comment can expand for several lines
            of code.
            /* Comments of the first type can be nested in the comments of
               the first type. */
    #endif
    //And this is a comment of the second type.
    return 0;
}
```

Listing 1: Several examples of comments in the C language

3 Variables

Declarations of variables in the C language have the following form:

`data_type name_of_variable;`

All variables declared outside the `main()` function have *global scope* and are called *global variables*. The table 1 contains basic information about several data types in the C language that are commonly used. In PC and most other computers the `int`, `long int`, `short int` and `char` allow variables to store signed numbers by default. They can be converted to types for natural numbers only by using the `unsigned` keyword before their names. There is a complementary keyword (`signed`) but it is usually used together with the `char` type and also in case of some special types of computer systems.

The actual size of the data types depends on the hardware of the computer for which the program is compiled. The information in the table 1 is only valid for the 32-bit and 64-bit PC computers. In the program, the size of a variable or a data type can be acquired with the use of the `sizeof` operator. Most of the compilers permits using the `long` and `short` keywords as the sole specifiers of variable's data type. In that case the variable, declared int such a way, stores integer numbers. Aside from the data types listed in the table 1 there are other in the C language, among them the `void` type. However, the latter cannot be used to declare variables, except for pointers that will be discussed in another laboratory

Name	Size (in bytes)	Values
`int`	4	integer numbers
`short int`	2	integer numbers
`long int`	8	integer numbers
`char`	1	characters or integer numbers
`float`	4	floating-point numbers
`double`	8	floating-point numbers
`long double`	12	floating-point numbers

Table 1: Basic data types in the C language

manual. The `void` type is used also in definitions of functions that also will be a topic of other laboratory classes.

The size of the `void` data type returned by the `sizeof` operator is one byte. However, compiling a program that contains a declaration of a `void` type variable results in an error, and the compiler informs the programmer that the size of the variable is unspecified.

4 Operators

In this section of the manual the basic operator of the C language are discussed. One of the has been already described in the previous section. It is the `sizeof` operator that determines the size of a variable or a data type. The value it reruns is a natural number. Also the assignment statement, represented by the `=` symbol, is an operator. Due to the fact it can be used the way it is shown in the listing 2.

```
int a, b, c;
int main(void)
{
    a=b=c=4;    //The value 4 is assigned to the a, b and c variables.
    return 0;
}
```

Listing 2: An example of using the assignment statement

Tables 2, 3, 4 and 5 gather information about the following types of operators: arithmetic, bitwise, boolean and relational.

Operators	Description
`++`, `--`	Unary increment and decrement operators. They can be used in-front or behind the argument, for example: `++a;` (pre-increment), `a++` (post-increment), `--a;` (pre-decrement), `a--;` (post-decrement). These operators also return the value of the argument. The main difference between the "post" and the "pre" operators lies in the way they return this value (explained later in the text).
`+`, `-`	Those operators can be either binary (and in that case they represent the operations of adding and subtracting numbers) or unary. In the latter case the `-` operator changes the sign of a number and the `+` does nothing.
`*`, `/`	The multiplication and division operators. **Attention**: If both arguments of the division operators are integers then the result is also an integer, regardless of the variable's data type where the outcome is stored.
`%`	The reminder or *modulo* operator. In the C language this operator can be applied for negative arguments.

Table 2: Arithmetic operators

When a value of an expression that contains a binary arithmetic operator has to be assigned to a

variable that is used in the expression then a shorthand notation of that operator can be applied. Listing 3 shows some examples. Note however, that this form of arithmetic operators is interpreted by the compiler and the program as *compound assignments*.

```
int a=2, b=2;
int main(void)
{
    b+=a; //Instead of b=b+a;
    b-=a; //Instead of b=b-a;
    b*=a; // Instead of b=b*a;
    b/=a; // Instead of b=b/a;
    b%=a; // Instead of b=b%a;
    return 0;
}
```

Listing 3: A shorthand notation for binary arithmetic operators

The program in the listing 4 shows the different outcomes of the "pre" and "post" versions of the increment and decrement operators.

```
int a=4, b;
int main(void)
{
    b=++a; //After: the value of "b" is 5, the value of "a" is 5.
    a=4;
    b=a++; //After: the value of "b" is 4, the value of "a" is 5.
    a=4;
    b=--a; //After: the value of "b" is 3, the value of "a" is 3.
    a=4;
    b=a--; //After: the value of "b" is 4, the value of "a" is 3.
    a=4;
    a++; //After: the value of "a" is 5.
    a=4;
    ++a; //After: the value of "a" is 5.
    a=4;
    a--; //After: the value of "a" is 3.
    a=4;
    --a; //After: the value of "a" is 3.
    return 0;
}
```

Listing 4: The increment and decrement operators

For the bitwise operators a shorthand notation similar to the one presented in the listing 3 can be applied.

Looking at the tables 3 and 4 we can notice that the bitwise OR and AND operators and the boolean OR and AND operators are represented by very similar symbols. To avoid confusion we can memorize a mnemonic created by Bruce Eckel: *Because bit are "small", there is only one character in bitwise operators.*

Standards of the C language earlier than ISO C99 had not defined a separated data type for boolean values, hence in this language every value that is different from zero is also true and the zero is interpreted as false. However, ISO C99 defines a `bool` data type together with two constants: `true` and `false`. They are available if the `stdbool.h` header file is included in the program.

Operators	Description
`|`, `&`,`^`	Bitwise sum (OR), product (AND) and exclusive-or (XOR) operators.
`~`	The unary bitwise complement operator.
`>>`, `<<`	The bitwise right shift and left shift operators. ATTENTION: in the C language the first argument of these operators can be negative. In case of the right shift operator the most significant bits of the result are padded with the value of the argument's most significant bit, for some computers (like the PCs) while for the others they are padded always with zeros.

Table 3: Bitwise operators

Operators	Description
`||`, `&&`	The boolean sum (OR) and boolean product (AND) operators.
`!`	The unary boolean negation operator.

Table 4: Boolean operators

If boolean operators are used in expressions then a short-circuit evaluation is applied. It is explained by the program in the listing 5. The `&&` operator returns true if and only if both its arguments are true. This means that when the first argument is false then evaluating of the second argument is not necessary — the entire expression is false, there is no other possibility. Similarly, the `||` operator gives false in result if and only if both of its arguments are false, so if the first is true then there is no point in evaluating the second one – the entire expression is true.

```
int a=0,b=0;
int main(void)
{
    (a=0)&&(b=4); //The value of both variables is zero.
    (a=4)&&(b=3); //The value of "a" is 4 and the value of "b" is 3.
    (a=0)||(b=0); //The value of both variables is zero.
    (a=3)||(b=4); //The value of "a" is 3 and the value of "b" is 4.
    return 0;
}
```

Listing 5: Short circuiting

Operators	Description
`==`, `!=`	The *equal* and *not equal* operators. The first one returns true if both its arguments have the same value. The second one returns true if its arguments have different values.
`<`, `>`, `<=`, `>=`	The "less than", "greater than", "less than or equal" and, "greater or equal" operators.

Table 5: Relational operators

In this manual the *precedence* of the operators is not discussed. Information on this issue can be found in many other sources. However it is worth to mention that, like in mathematical expressions, the precedence of operators may be changed with the use of parentheses.

Some mathematical operations can be performed with the use of functions and other elements available if the `math.h` header file is included in the program. The 6 table lists some of these elements.

Functions and constants	Description
`sin(x)`, `cos(x)`, `tan(x)`	Sine, cosine and tangent. **Attention**: The argument (the `x` variable) is an angle measured in radians.
`pow(x,y)`	Power: x^y.
`sqrt(x)`	Square root of `x`.
`M_PI`	The π constant. It is not available in the ISO C99, but it is a part of the GNU C extension.

Table 6: Functions and constants defined in the math.h file

5 Ternary Operator

The ternary (or conditional) operator has the following form:

`condition?expression_1:expression_2;`

If the condition is satisfied then the operator evaluates the `expression_1` and returns its value, otherwise it evaluates the `expression_2` and also returns its value. The condition can be a simple or a complex expression created with the use of such types of operators as boolean and relational.

Attention: the C language permits using in the condition the assignment operator (`=`) that is very similar to the equals operator (`==`). These operators perform different operations, hence they give different results and often the use of assignment instead of the equal operator in a condition is a programmer mistake. Sometimes though programmers use the assignment operator in the condition deliberately, to make the code more concise.

The program in the listing 6 uses the `?` operator to find the greater value from these that are stores in two variables. In this case the condition is defined as: `a>b`. The assignment operator is used outside the condition and it stores the value returned by the ternary operator in the `max` variable.

```
int a=5, b=4, max;
int main(void)
{
    max=(a>b)?a:b;
    return 0;
}
```

Listing 6: The ternary operator

6 User – Computer Communication

Aside from the `puts()` function that has been described in the first manual, the `scanf` and `printf` functions are used for interaction with the user. Both of them are declared in the `stdio.h`.

The `scanf()` function allows the user to input a value, using the keyboard, and stores it in a variable. The program in the listing 7 show how to use it.

```
int a;
int main(void)
{
    scanf("%d",&a);
    return 0;
}
```

Listing 7: The usage of the `scanf() function.`

The function takes (at least) two arguments. The first one is so-called *formatted string*, that contains

the *conversion specifier*. The specifier determines how the value entered by the user should be interpreted by the function. The second argument of the `scanf()` function is the address of the variable where the value has to be stored. To get the address the *address operator* (represented by the `&` character) has to be applied. This is the same character that represents the bitwise AND operator. The compiler recognises which of them is used in an expression by checking the number of their arguments. The address operator is unary and the bitwise AND operator is binary. The table 7 describes some of the `scanf()` conversion specifiers.

Conversion Specifier	**Description**
`%d`	Decimal integer of the `int` type.
`%ld`	Decimal integer of the `long int` type.
`%u`	Decimal natural number of the `unsigned int` type.
`%c`	Character.
`%f`	Floating-point number of the `float` type.
`%lf`	Floating-point number of the `double` type.

Table 7: Conversion specifiers for the `scanf()` function

The `printf()` function can display on the screen the values of expressions. Like the `scanf()` function it requires using the formatted strings containing conversion specifiers (table 8) that determine how the values have to be presented on the screen. Additionally so-called *escape sequences* can be used in the formatted string (table 9). The program in the listing 8 demonstrates some examples of using the `printf()` function.

Conversion Specifier	**Description**
`%d`	Decimal integer of the `int` type.
`%ld`	Decimal integer of the `long int` type.
`%u`	Decimal natural number of the `unsigned int` type.
`%c`	Character.
`%f`	Floating-point number of the `double` or `float` type. The specifier may contain additional information about formatting, for example: `%.3f` — only the tree digits of fraction will be displayed on the screen.
`%lf`	Floating-point number of the `double` type. It also may contain some additional information about formatting.
`%Lf`	Floating-point number of the `double` type. It also may contain some additional information about formatting.
`%e`, `%E`	Floating-point number of the `double` or `float` typed displayed with the use of the scientific notation, for example: `3e-9` (in case of the first specifier) or `3E-9` (in case of the second specifier).
`%le`, `%lE`	As above, but the number is of the `long double` type.
`%Le`, `%LE`	As above.
`%x`, `%X`	Natural hexadecimal number displayed like `a5` if the first specifier is used or like `A6` if the second specifier is used.
`%o`	Natural octal number.

Table 8: Conversion specifiers for the `printf()`

Escape Sequence	Description
\n	New line character.
\r	Beginning of the line character.
\\	The \ character.
\"	The quotation mark.
\t	The tabulator.

Table 9: Escape sequences for the `printf()`

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

int main(void)
{

    printf("%d",a); //Displays the value of "a".
    printf("%d\n",a); //Display the value of "a" and moves the cursor
                      //to the next line.

    /*
     * The printf() function can print more complex messages.
     * In these messages the conversion specifiers are replaced by values of
     * expressions.
     */

    printf("The value of \"a\" is: %d\n",a);
    printf("The value of \"a\" is: %d, and the value of \"b\" is: %d\n",a,b);
    return 0;
}
```

Listing 8: Examples of using the `printf()` function

7 Exercises

1. Write a program that asks the user for three integers and displays the greatest among them.

2. Repeat the previous exercise, but this time your program should ask the user for three characters and display the one that is the latest in the alphabet. **Attention!** If you call several times in row the `scanf()` function with the `"%c"` formatted string then you will notice that the `scanf()` functions does not allow you to enter all the characters that you need. The reason for this is that the `scanf("%c");` statement reads not only the "visible" characters, but also the new line characters emitted by the `Enter` key. To get read of the problem use in the second and subsequent `scanf()` calls the `" %c"` formatted string (it has a space between the `"` and `%` characters.

3. Write a program that asks the user for two numbers, stores them in two `unsigned char` variables and displays results of bitwise AND, OR and XOR operations, where the numbers are arguments. The results should be displayed in hexadecimal.

4. Declare in a program three variables of the following types: `int`, `short int` and `double`. Display their values without initializing them.

5. Write a program that asks the user for tow numbers of the `int`, divides one by the other and prints the result. Is the outcome always the same as you expected?

6. Write a program that asks the user for two integer numbers and displays the first one when its divisible by the second one, or the second one if the first one is not divisible by it.

7. How can you check if a number is even without using the reminder operator? Write a program that applies your idea.

8. Declare in a program an `unsigned char` variable. Subtract one from its default value. Display the outcome. Can you explain the result?

9. (a little harder) Find on the Internet the information about the De Morgan's laws. Write a program that calculates the bitwise product of two values without using the `&` operator. Repeat the task for the bitwise sum.