Table of Contents

[2.2 First C code: calculating and printing the sum of two numbers 1](#_Toc440874132)

[**2.2 First C code: calculating and printing the sum of two numbers** 1](#_Toc440874133)

[2.3 Comments in the C language 3](#_Toc440874134)

[2.5 From source code to a functional program 4](#_Toc440874135)

[Exercise 2.1 4](#_Toc440874136)

[Exercise 2.2 5](#_Toc440874137)

[3. Variables in the C language 6](#_Toc440874138)

[3.1 How does computer memory operate? 6](#_Toc440874139)

[3.2 Using and naming variables 7](#_Toc440874140)

[3.3 Numerical variable types in the C language 8](#_Toc440874141)

[3.4 Variable declaration 9](#_Toc440874142)

[3.5 Variable initialisation 9](#_Toc440874143)

[3.6 The printf() function 10](#_Toc440874144)

[3.6.1 Printing several variables 11](#_Toc440874145)

[3.6.2 Printing formatted text 11](#_Toc440874146)

[3.7 The scanf() function 12](#_Toc440874147)

[Exercise 3.1 12](#_Toc440874148)

[Exercise 3.2 13](#_Toc440874149)

[4. Program statements, expressions and operators 14](#_Toc440874150)

[4.1 Program statements in the C language 14](#_Toc440874151)

[4.1.1 White space in program statements 14](#_Toc440874152)

[4.2 Expressions 15](#_Toc440874153)

[4.3 Operators 15](#_Toc440874154)

[4.3.1 Assignment operator 16](#_Toc440874155)

[4.3.2 Mathematical operators 16](#_Toc440874156)

[4.3.3 Precedence of mathematical operators 17](#_Toc440874157)

[4.3.4 Comparison operators 18](#_Toc440874158)

[4.3.6 Logical operators 19](#_Toc440874159)

[4.3.7 Combinations and precedence of logical operators 20](#_Toc440874160)

[Exercise 4.1 20](#_Toc440874161)

[Exercise 4.2 21](#_Toc440874162)

## 2.2 First C code: calculating and printing the sum of two numbers

**2.2 First C code: calculating and printing the sum of two numbers**

1 #include <stdio.h>

2

3 int main()

4 {

5 int first\_number, second\_number, sum;

6

7 printf("Enter a number: ");

8 scanf("%d", &first\_number);

9

10 printf("Enter another number: ");

11 scanf("%d", &second\_number);

12

13 sum = first\_number + second\_number;

14

15 printf("The sum of the numbers you entered is %d", sum);

16

17

18 return 0;

19 }

Did this look horrible? Next, we will go through the entire source code in enough detail so that you will be able to understand everything required at this stage.

**Line 1:** It is probably wrong to say in the very beginning that you do not need to understand the first line of your program yet. At this stage, it is enough that you remember to include the following as the first line of all your programs "#include < stdio.h>". You will learn the purpose of this line later.

**Line 3:** Line 3 declares the main program in our example. In this case, the main program is the only program in our source code. Every program written in C includes a main program as the very minimum. Commands that make the program do something are written in the main program.

**Line 4:**In order for the computer to know where the main program starts, we have written a curly bracket immediately following the declaration of the main program. The main program ends on line 19, where a matching curly bracket indicates the end of the program. The commands that make the program calculate and print the sum are included between lines 4 and 19.

**Line 5:** In the algorithm that we wrote earlier, we instructed the computer to store numbers in "boxes". In the world of programming, these boxes are called variables. Our program needs three boxes or variables. In the C language, we must separately tell the computer which boxes we intend to use later in the program. The notation "int" indicates that the variables "first\_number", "second\_number" and "sum" can be used to store integers. For the time being, you will not need any variables other than integer variables, and you can declare them as indicated in the example. If you only need one variable, you can type: int variable\_name;

**Line 7:** When you have a look at line 7, you will probably get an idea of its purpose in the code. Correct! The line uses the printf() function that can conveniently print text on the screen. In this case, we printed out the text "Enter a number:". From now on, you will be able to write programs that print text on the screen.

**Line 8:** Line 8 uses the scanf() function that can be used to read a number entered by the user and store it in a variable. The special designation "%d" in the function indicates that we want to read an integer in particular. On the other hand, the designation "&first\_number" indicates that the integer shall be stored in the variable "first\_number".

Note that in the algorithm written in English, we instructed the computer to wait until a number is entered. The C language function "scanf()" waits until the user types something on the keyboard and presses the return key. You should also note that the character & must be typed before the name of the variable! (see line 8)

**Lines 10 & 11:** These are similar to lines 7 & 8.

**Line 13:** On line 13, we add up the numbers entered by the user:

sum = first\_number + second\_number;

The computer adds up the variables and stores the result in the variable called sum. You will learn some other operations associated with variables later.

**Line 15:** The printf() function was explained above on lines 7 and 10. The use of the function on line 15 is slightly different from the previous instances where we only printed out text.

This time, in addition to the text, we want to print the value of the variable "sum". When printing, the computer replaces the designation "%d" with the value of the variable. The variable is separated by a comma. In other words, the text to be printed is written inside the quotation marks. If the printout includes variables, these are indicated after the printout. If you wanted to print the values of two variables, you could type the following:

printf("The first variable is "%d" and the second one is "%d", first\_var, second\_var);

**Line 18:** The following designation on line 18: "return 0;" indicates that we want to get out of the program, in other words end it.

|  |  |
| --- | --- |
| https://vw4.viope.com/existing-content/pics/info.gif | **NOTE!** Statements written inside the main program are separated by semicolons. See lines 5, 7, 8, 10, 11, 13, 15 and 18! One of the most typical errors is to forget a semicolon where one should be typed. |

**Summary of Section 2.2**

All the matters introduced in our example will be revisited later, so there is no need to worry. For the time being, it is enough that you remember the following:

* Until you proceed further in your studies, every program you write will start with the designation "#include <stdio.h>".
* Every program written in the C language includes at least a main program called "main()". The "events" of the main program are written between curly brackets "{" and "}".
* Statements in the C language are separated by semicolons.
* Operators are the very essence of programming. In the C language, every variable has to be declared before being used. An integer variable is declared as follows: "int variable\_name;". You can declare several variables of the same type on the same line (see Example 1).
* printf() is a function that makes it easy to print text on the screen.
* scanf() is a function for receiving and storing a number entered by the user.

## 2.3 Comments in the C language

One issue that you will encounter is the readability of your programs after time has passed. The longer your programs, the more difficult it will become to review them. When authoring a program, you know the exact meaning of every line you type. What if you read your program again after a week, for example? You may already have forgotten half of the contents.

For this reason, the C language provides a comment facility for including any text in your program.

#include <stdio.h>

/\* This is a fine program with comments \*/

int main()

{

/\* Declare the variable number \*/

int number;

/\* Ask the user to enter a number \*/

printf("Enter a number: ");

/\* Get the number to the variable "number" \*/

scanf("%d", &number);

/\* This comment demonstrates a

comment on several lines.

Now, print out the entered number \*/

printf("The number was: %d\n", number);

return 0;

}

As you saw, comments are written between the characters "/\*" and "\*/". These characters tell the compiler to ignore the text between them. You can type anything that helps you to improve the readability of the program. It is very important to comment your programs, as comments make it possible for others to understand the operation of the program.

The examples in Viope C are not commented. This is due to the fact that you need to learn to read the C language first, and comments would make the examples longer and more difficult for the beginner to read

## 2.5 From source code to a functional program

Next, you should try how to compile C language source code into a functional program. For this purpose, you need a special program called a compiler.

You do not need a compiler to complete the programming exercises in Viope C. It is enough that you can write the source code correctly. However, it would be beneficial to practice programming by testing your programs "in real life". You can find several compilers for different operating systems free of charge on the Internet.

Try to compile the previous example into a functional program. After this, you can start with the multiple-choice questions and exercises in Chapter 2. Once you have completed the questions and exercises, you can continue to the next chapter, where variables are introduced.

### Exercise 2.1

**Printing on the screen**

Write a functioning program in C that, upon execution, prints the following on the screen: "Hello everybody!", followed by a newline character \n after the exclamation point. After printing, the program closes itself.  
  
**Hint:**  
Everything required for creating the program can be found in the example program in Chapter 2. Once you have understood the operating principle of the example, it should be easy to write your program. You do not need any boxes or variables in this program.

*Example output:*

Hello everybody!

*The output of the program must be exactly the same as the example output (the most strict comparison level)*

**Solution:**

#include <stdio.h>

int main(void)

{

printf("Hello everybody!\n");

return 0;

}

### Exercise 2.2

**Reading from the keyboard and printing**

You have completed this exercise

Submission time has expired

Write a program that prompts the user for an integer, calculates the square and prints the result on the screen (also print out a newline character \n after the result). After printing, the program closes itself.  
  
**Hint:**  
The program can be written in quite much the same way as the example in the chapter. You can implement the program in many ways, but you will probably need two variables: one for the number entered by the user and one for the square. In the example, the addition operator + was used for calculating a sum. In this program, you need the multiplication operator \*.

*Example output:*

Enter an integer: 11 The square of the number you entered is 121

*The output of the program must be exactly the same as the example output (the most strict comparison level)* *The verification of program output does not account for whitespace characters like "\n", "\t" and " "*

**Solution:**

#include<stdio.h>

int main(void)

{

int number = 0;

printf("Enter an integer: ");

scanf("%d",&number);

printf("The square of the number you entered is %d\n",number\*number);

return 0;

}

# 3. Variables in the C language

In the previous section, we wrote our first C language program. You probably noticed that the "boxes" or variables used in the program had a central role with regard to program operation. The variables were used for storing numbers, and it was possible to calculate a new number (the sum of the numbers) and print out the desired result (the sum of the numbers) on the screen.

Because variables are one of the most important elements in the C language, we will now examine them in more detail. In the example we created, only integer variables were used. In this section, you will learn that variables in C can also have other types.

We shall first examine the operations of computer memory in order to gain the best possible understanding on what a variable is.

## 3.1 How does computer memory operate?

Computers are built of a variety of components, most often including a hard disk, display controller, RAM, diskette drive etc. All the components are attached to the motherboard in one way or another; the motherboard can be considered the core of the computer's operations. When speaking about variables, we must understand the operations of RAM attached to the computer.

RAM (Random Access Memory) is the location in a computer where data can be stored during program execution, removed or replaced with new data. Every computer has a certain amount of memory, and the amount is most often indicated in megabytes (for example, 256 MB). RAM can be used when the computer is turned on. All data in the memory will be destroyed when the computer is turned off. If some data needs to be stored permanently, you need a hard disk, for example, where the data remains stored also when the computer is turned off.

The smallest unit of data in the world of computing is a byte. One megabyte corresponds to 1024 kilobytes, and one kilobyte corresponds to 1024 bytes. If your computer has 256 megabytes of memory, it can hold 268,435,456 units of data.

The table below includes some examples of different "data chunks" and the amount of memory required:

|  |  |
| --- | --- |
| **Data** | **Number of bytes** |
| The letter x | 1 |
| The number 100 | 2 |
| The number 120.145 | 4 |
| The expression: Viope C | 7 |
| A page of text | approximately 3000 |

Every memory location in RAM has a size of one byte and a unique address. We shall leave the addressing operations to the C compiler and examine variables instead.

## 3.2 Using and naming variables

We have already learned that a variable can be used for convenient storage of data in a C language program. You also know that a variable must be declared before use. Actually, the purpose of declaration is to tell the computer that you intend to use its memory for a certain purpose; on the basis of this, the computer is able to reserve a location for the data in its memory. The name of the variable refers to this very location. The computer uses the variable name to find the correct memory location among thousands of locations.

You must learn which names can be assigned to variables in order to be able to use them in your programs. The C language has a number of rules that limit the names of variables:

* A variable name may include letters, digits and underscores.
* The first character in a variable name must always be a letter.
* Variable names are case-sensitive, which means that the variables "First\_Number" and "first\_number" are not equal.
* The reserved words of the C language cannot be used as variable names. These include, among others, for, if, int, double and char. A complete list of the reserved words can be found in the C manual.

The following table presents some examples of valid and invalid variables with comments.

|  |  |  |
| --- | --- | --- |
| **Variable name** | **Valid/invalid** | **Comment** |
| dollar\_rate | Valid | A good variable name, because the name indicates the purpose. |
| yfjCWWr04jkg\_ | Valid | Nonsensical variable, because it does not indicate the purpose. |
| 3rdmonth | Invalid | Starts with a number and is therefore invalid. |
| \_firstname\_ | Valid | A valid variable name, but it is not recommended to use an underscore as the first character. |
| email@ | Invalid | Includes a special character and is therefore invalid. |
| good variable | Invalid | Includes a blank character and is therefore invalid. |

Most C language compilers limit the length of variable names to a maximum of 31 characters.

**3.2.1 Guidelines for naming variables**

You should use common sense when naming variables! I have often encountered students who name their variables by "hit or miss". Careless naming of variables will often significantly hamper the readability of your program. The following guidelines help you in naming your variables in a fashion that allows others to understand your program:

1. Name your variables using lower-case letters. There are some cases where upper case is appropriate, but you should use lower case for the time being.
2. The names of your variables should describe their purposes in as much detail as possible. For example, if you need a variable for storing the number of days in January, "days\_in\_january" is much better than "somevariable".
3. As indicated in item 2, you should use combinations of several words separated with underscores.
4. Whatever you do, stick to your practice! By using a constant style of variable naming, you can make your own programming easier.

## 3.3 Numerical variable types in the C language

Now you know how variables operate and how you should name them. Next, we shall examine the various types of variables that the C language has to offer. Until now, you have encountered just one type, the integer.

Why are different types of variables actually required? Would it not be easier if all the required data could be stored in any variable? The use of a single type of variable in all situations could be slightly easier, but we cannot do it this way. The C language has several different types of variables, because different kinds of data, more precisely types of numbers require different amounts of memory.

Small integers (1, 32, 100 etc...) take up much less memory than large integers (234577883, 98723112 etc...) or floating-point numbers, also known as decimal numbers (0.00078109, 0.243645646 etc...). In addition to the amount of memory required, the type of a number also affects the time required for calculations on the number. The greater the number, the more time it takes to complete an operation (calculation) associated with the number.

The following table presents the numerical data types in the C language. Note that a character is also processed as a number in C. When a character is stored in a program, the computer actually stores a number corresponding to the character.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable type** | **Reserved word** | **Bytes** | **Range** |
| Character | char | 1 | -128 - 127 |
| Integer | int | 2 | -32768 - 32767 |
| Short integer | short | 2 | -32768 - 32767 |
| Long integer | long | 4 | -2147483648 - 2147483647 |
| Unsigned character | unsigned char | 1 | 0 - 255 |
| Unsigned integer | unsigned int | 2 | 0 - 65535 |
| Unsigned short integer | unsigned short | 2 | 0 - 65535 |
| Unsigned long integer | unsigned long | 4 | 0 - 4294967295 |
| Single-precision floating-point number | float | 4 | 1.2E-38 - 3.4E38(\*) |
| Double-precision floating-point number | double | 8 | 2.2E-308 - 1.8E308(\*\*) |

The numbers of bytes in the third column of the table represent the amount of memory required by each variable type in a 16-bit computer architecture (this does not play any significant role in programming, so we shall not provide any more detailed description).

\*     Estimated range; precision =   7 digits  
\*\*   Estimated range; precision = 19 digits

## 3.4 Variable declaration

I have mentioned several times that every variable must be declared before using it in a program. Let us examine the issues related with variable declaration in some more detail.

Variable declaration in the C language complies with the following syntax (syntax refers to the rules that prescribe how to write different things in C):

data\_type variable\_name;

In the above, data\_type defines the data type of the variable, and naturally, variable\_name is a name for the variable defined by the programmer. Several variables of the same data type can be declared simultaneously on the same line. Some examples of variable declarations:

int number1;

float floatingnumber1;

int first\_number, second\_number, sum\_of\_numbers;

...

Almost all programs require some variables. The variables required in a program are most often declared before the main program or immediately in the beginning of the main program.

## 3.5 Variable initialisation

When you declare a variable in your program, you instruct the compiler to allocate space in the computer memory for the variable. A novice programmer will often think that the value of a freshly declared variable is zero until something else is added to the variable. However, this is not true; upon declaration, the variable receives a value that may be zero or any other value.

If you forget this, you may encounter interesting (?!?) situations in your upcoming programming exercises. For example, you may be writing a cash register program that adds up the prices of your purchases. You can be sure that you have forgotten to initialise the variable "total" when the cash register prints the following on the screen: "total 109840398301 euros". Did you understand what I mean? No problem, let us go back to variable initialisation.

The following is a first example of declaring a variable and initialising it:

int sum;

sum = 0;

|  |  |
| --- | --- |
| https://vw4.viope.com/existing-content/pics/info.gif | **NOTE!** In the C language, the equal sign has a different meaning compared to what you have learned in mathematics. In the C language, the character "=" is an assignment operator for assigning a value to a variable.  **For example:**"y = 7" has the following meaning in algebra: "y is equal to 7", while in the C language, the expression means: "assign the value 7 to the variable y". |

After initialisation, you can start using the variable sum with no problems and be sure that at the end of your program the variable will not hold any astronomical number. The same initialisation can be done on a single line when declaring the variable:

int sum = 0;

In addition, you may declare and initialise several variables on the same line as follows:

float swedish\_krona\_rate = 9.1589, dollar\_rate = 1.2006;

When you initialise variables, do not forget to make sure that the initial value you provide is in the range of values allowed for the variable. The following designation in your program would surely cause some error messages:

int small\_integer = 100000000;

## 3.6 The printf() function

As you know, the printf() function can be used for printing the values of variables. Let us have a deeper look at how this is done with different types of variables.

If you want to print the value of an integer variable, you could type the following:

int number = 1;

printf("The number is %d", number);

However, this will not work if you use a floating-point variable float in place of the integer variable, for example. The following expressions would cause an error condition:

float number = 1;

printf("The number is %d", number);

The designation %d in the examples is called a formatting specifier that indicates the data type of the variable to be printed. The following table presents the most common formatting specifiers in the C language.

|  |  |
| --- | --- |
| **Formatting specifier** | **Meaning** |
| %c | Prints a single character |
| %d | Prints a signed integer |
| %f | Prints a float or double type in decimal notation |
| %s | Prints a string |
| %u | Prints an unsigned integer |

If you want to print the value contained in a decimal variable float, you can type the following:

float number = 1.9282;

printf("The number is %f", number);

You can also print decimal numbers with a specified precision. If we want to print the above with a precision of two decimal places, we can supplement the printing command as follows:

float number = 1.9282;

printf("The number is %.2f", number);

Correspondingly, "%.3f" would have given us a precision of three decimal places.

### 3.6.1 Printing several variables

You will often encounter a situation where you want to print the value of several variables in the same command. If you have three variables whose values you want to print in sequence, you can type the following:

printf("%d ", number1);

printf("%d ", number2);

printf("%d", number3);

However, you can make your work easier by typing the same thing as follows:

printf("%d %d %d", number1, number2, number3);

The above prints integers only, but naturally, you can print variables of different data types in the same statement.

### 3.6.2 Printing formatted text

Even though the purpose of the third chapter is to teach you to use variables, we will include a few more issues regarding the use of the printf() function.

As you know, the text to be printed using the printf() function is delimited by quotation marks:

printf("Text between quotation marks");

What if you want to print a quotation mark as part of the text? Does the following example work?

printf("Mike said: "Hello!".");

The answer is no. The compiler does not understand that you want to print the second quotation mark on the line ("Hel..) and interprets the character as ending the printout. A method exists for printing a quotation mark:

printf("Mike said: \"Hello!\".");

The compiler will understand what you mean if you type the character "\" before the quotation mark. This character can be used for printing other "special cases" as well. The most common of these are presented in the following table:

|  |  |
| --- | --- |
| **Character combination** | **Meaning** |
| \" | Prints a quotation mark |
| \' | Prints an apostrophe |
| \n | Prints a newline |
| \t | Prints a horizontal tab |
| \\ | Prints a backslash |

The following example prints formatted text:

printf("First sentence.\tSecond sentence.\nThird sentence.\tFourth sentence.");

The printout would appear on the screen as follows:

First sentence. Second sentence.

Third sentence. Fourth sentence.

## 3.7 The scanf() function

The same rules that apply to the printing of different data types also cover the reading of user input into different types of variables. The table in Section 3.6 presents the formatting specifiers used by the scanf() function. The following example demonstrates the matter:

int integer;

float floating\_point\_number

printf("Enter an integer: ");

scanf("%d", &integer);

printf("Enter a floating-point number: ");

scanf("%f", &floating\_point\_number);

In this chapter, you have learned to declare, initialise and use variables. Once you have completed the associated exercises, you can start learning all the operations that you can do on variables.

## Exercise 3.1

**Numbers and processing them**

You have completed this exercise

Submission time has expired

Write a program that first asks the user for an integer and after that, a floating-point number. Finally, the program prints both numbers on the screen. The floating-point number shall be printed with two decimal places of precision.

*Example output:*

Enter an integer: 14 Enter a decimal number:3.123 You entered the integer: 14 You entered the decimal number, rounded to two decimal places: 3.12

*The output of the program must be exactly the same as the example output (the most strict comparison level)* *The verification of program output does not account for whitespace characters like "\n", "\t" and " "*

**Solution:**

#include<stdio.h>

int main(void)

{

int integer = 0;

float floating\_point = 0;

printf("Enter an integer: ");

scanf("%d", &integer);

printf("Enter a decimal number:");

scanf("%f", &floating\_point);

printf("\nYou entered the integer: %d\n", integer);

printf("You entered the decimal number, rounded to two decimal places: %.2f\n", floating\_point);

return 0;

}

## Exercise 3.2

**Processing and printing values received as input**

You have completed this exercise

Submission time has expired

Write a program that prompts the user for an amount in Finnish markka and converts it to euro. Finally, the program prints the amount on the screen in euro with two decimal places of precision. The euro conversion factor is 5.94573.  
  
**Hint:**  
You need three variables: one for the conversion factor, one for markka and one for euro. The required division can be accomplished by using the "/" character.

*Example output:*

Enter an amount in FIM:9.90 FIM converted to euro: 1.67

*The output of the program must be exactly the same as the example output (the most strict comparison level)* *The verification of program output does not account for whitespace characters like "\n", "\t" and " "*

**Solution:**

#include <stdio.h>

int main(void)

{

float euroFactor = 5.94573, markka = 0, euro = 0;

printf("Enter an amount in FIM:");

scanf("%f",&markka);

euro = (markka/euroFactor);

printf("FIM converted to euro: %.2f\n",euro);

return 0;

}

# 4. Program statements, expressions and operators

I have said several times that variables are a very important element in C programming. We shall now start to study the use of variables in our programs. In figurative terms, we could say that we are now equipped with a bat and a ball. Next, we shall learn the rules of the game. :)

This chapter contains a lot of new information. You may not be able to internalise everything at the first reading, so allow yourself the option to return to this chapter (and any other chapters) later.

## 4.1 Program statements in the C language

A program written in C comprises program statements. Each program statement is an independent instruction telling the computer to perform a certain task. Every instruction that you type for the computer must be perfect. The computer is such a stupid machine that the instructions you prepare must be unambiguous from the computer's viewpoint, just like the instructions we prepared for Tahvo.

A single program statement is usually typed on its own line. However, in some cases a statement may be so long that it is more reasonable to type it on two or more lines.

Every program statement ends with a semicolon, with the exception of preprocessor commands such as #include, which will be discussed later. Our first program had many simple program statements such as:

printf("The sum of the numbers you entered is %d", sum);

int first\_number, second\_number, sum;

sum = first\_number + second\_number;

You will be seeing a great lot of program statements, and before you can notice, you will be able to create many kinds of statements yourself.

### 4.1.1 White space in program statements

White space refers to characters that are not actually shown in the program code. These include blanks, tabs and newlines. The C language compiler ignores these characters. For this reason, every program statement must end with a semicolon so that the compiler can identify the end of the statement. The following three examples of a program statement are totally equal from the compiler's viewpoint.

sum = first\_number + second\_number;

sum

=

first\_number

+

second\_number;

sum=first\_number+second\_number;

Each of the statements above instructs the computer to calculate the sum of the variables first\_number and second\_number and assign it to the variable sum. The only difference is the readability of the statements; you probably share my opinion that the first one is the best? When writing programs, you can freely format the statements as you wish. It is most important that your code is as readable as possible!

## 4.2 Expressions

A program statement may include expressions and operators. Do not let the words statement and expression confuse you. A program statement is a larger entity that may include several expressions. At its simplest, an expression includes just one item of data. For example, the variable first\_number is an expression that comprises the value assigned to it. Generally speaking, an expression refers to the use of a numerical value.

Most expressions comprise several simple expressions that are combined using operators. The following example contains three expressions and two operators, which make up a complex expression:

5 + 7 - 1

When an expression contains more than one operator, the value of the expression depends on the precedence - that is, the order of executing the operations. We shall return to this in the next section.

## 4.3 Operators

An operator refers to a symbol that instructs the compiler to execute a certain operation. Operators can be divided into four different classes, including:

* Assignment operator
* Mathematical operators
* Comparison operators
* Logical operators

In the now familiar statement "sum = first\_number + second\_number;" you have encountered the addition operator, which is a mathematical operator. The same expression uses the assignment operator to assign the calculated sum to the variable sum.

We will now examine all the operators in the C language in more detail.

### 4.3.1 Assignment operator

The assignment operator can be used to assign the value of an expression to a variable. In the C language, the equal sign (=) is used as the assignment operator (=). It is quite common that a novice programmer confuses the meaning of this designation with its purpose familiar from algebra. For the sake of revision, let us repeat that the program statement

z = 70;

means "assign the value 70 to the variable z". This is different from algebra where the meaning would be "z is equal to 70".

When using the assignment operator, the right side of the operator can be any expression, and the left side can be a variable name. When the statement is executed, the computer will first calculate the value of the expression on the right side and assign the result to the desired variable.

variable\_where\_to\_assign\_the\_result = expression;

### 4.3.2 Mathematical operators

The mathematical operators of the C language perform calculations, such as addition and subtraction. Operators are divided into unary and binary operators.

**Unary operators**

The word "unary" refers to the fact that the operation only applies to one operand. (The expression first\_number + second\_number has two operands.) Despite the difficult-sounding name, it is easy to understand the operation of unary operators. The following table presents the most common unary operators.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Symbol** | **Operation** | **Example** |
| Increment | ++ | Increments the value of the operand by one | ++x x++ |
| Decrement | -- | Decrements the value of the operand by one | --y y-- |

These operators can only be used in association with variables. The following example demonstrates how the operators work:

int number = 10;

number = number + 1;

number++;

The first line of the example initialised the variable number with a value of 10. The second line incremented the value of the variable by one. The third line used the increment operator, also incrementing the value of the variable by one. After executing the third statement, the value of the variable is 12.

The table above demonstrates how operators can be used before or after an operand. The placement of the operator is significant and is illustrated in the following:

number = 10;

number2 = number++;

In the above, the variable number received the value 10. On the second line, the value of the variable number was assigned to the variable number2, after which number increased by one. After executing the statements, the values of the variables number and number2 are 11 and 10. On the other hand, if we type the following:

number = 10;

number2 = ++number;

the value of the variable number increases before the assignment and the value of both variables will be 11.

**Binary operators**

Unlike unary operators, binary operators use two operands for performing the calculation. In subtraction, for example, we need both the subtrahend and the minuend. The following is a description of the binary operators in the C language.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Symbol** | **Operation** | **Example** |
| Addition | + | Adds up two operands | x + y |
| Subtraction | - | Subtracts one operand from another | x - y |
| Multiplication | \* | Multiplies two operands | x \* y |
| Division | / | Divides one operand by another | x / y |
| Remainder | % | Calculates the remainder after a division between the two operands | x % y |

Remainder is probably the only operator whose operation should be clarified by means of an example:

x = 9 % 4;

In the above, the variable x gets the value one. The remainder operator divides the first operand by the second one and returns the remainder. In the following example, the variable remainder gets the value 0.

int dividend = 10, divisor = 5, remainder;

remainder = dividend % divisor;

Thus if the left-side operand of the remainder operator (dividend) is divisible by the right-side operand (divisor), the result is zero. If the left-side operand is not divisible by the right-side operand, the result is the remainder after the calculation.

### 4.3.3 Precedence of mathematical operators

In association with expressions having several operators, it is important to understand the order in which the operations are executed. This is nothing new - there are certain rules in mathematics as well that indicate the precedence of calculations. In order to illustrate the problem, you may attempt to deduce the value of the variable result in the following statement:

result = 2 - 7 \* 3;

If subtraction is performed first, the variable result gets the value -15. On the other hand, if multiplication is performed first, the value is -19.

The operators in the C language have a certain order of precedence for calculations. The following table specifies the priorities of the operators that can be used for unambiguous calculation:

|  |  |
| --- | --- |
| **Operator** | **Priority** |
| ++, -- | 1 |
| \*, /, % | 2 |
| +, - | 3 |

If an expression includes several operators at the same level of priority, the calculations will be performed from left to right. In the expression that we just reviewed,

result = 2 - 7 \* 3;

the variable result gets the value -19. (The multiplication 7 \* 3 is calculated first, and the result is subtracted from the number 2.)

In the following example, result gets the value 7:

result = 3 + 2 \* 6 / 3;

The expression includes both multiplication and division, and the one to the left (2\*6) is calculated first. After this, the result is divided by 3 (12/3), and the number is added to the number 3 (3+4).

### 4.3.4 Comparison operators

As the name suggests, comparison operators are used for comparing expressions in the C language. They can be used for asking questions such as: "Is x less than 10?" or "Is y equal to z?". The comparison operators return either the value 1 (true) or 0 (false). The result can be assigned to a variable just like the results of mathematical operators. In the example

int true\_or\_false, first\_number = 5, second\_number = 1;

true\_or\_false = (first\_number < second\_number);

the variable true\_or\_false gets the value 0 (false). The following table lists all the comparison operators of the C language:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Symbol** | **Question** | **Example** |
| Greater than | > | Indicates if operand1 is greater than operand2 | x > y |
| Less than | < | Indicates if operand1 is less than operand2 | x < y |
| Greater than or equal | >= | Indicates if operand1 is greater than or equal to operand2 | x >= y |
| Less than or equal | <= | Indicates if operand1 is less than or equal to operand2 | x <= y |
| Not equal | != | Indicates if operand1 is inequal to operand2 | x != y |
| Equal | == | Indicates if operand1 is equal to operand2 | x == y |

|  |  |
| --- | --- |
| https://vw4.viope.com/existing-content/pics/info.gif | **NOTE!** Novice programmers often confuse the comparison operator (==) and the assignment operator (=). I hope that you will not make the same mistake. |

### 4.3.6 Logical operators

When writing programs, we may encounter a situation where several comparisons are needed at the same time. In real life, such a situation could be described as follows, for example:

"If you are hungry AND you have money, have a meal at a restaurant."

The C language has three logical operators that you can use:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Symbol** | **Example** |
| AND | && | comparison1 && comparison2 |
| OR | || | comparison1 || comparison2 |
| NOT | ! | !comparison1 |

**The AND operator**

The AND operator can be used to combine comparisons so that the combined comparison is true if both the comparisons are true:

(5 > 2) && (6 != 8) is **true**, because 5 is greater than 2 **AND** 6 is inequal to 8.  
(1 == 2) && (3 > 2) is **false**, because 1 is not equal to 2.

**The OR operator**

The OR operator can be used for comparisons that return the value true if either of the comparisons is true:

(5 < 2) || (6 < 8) is **true**, because 6 is less than 8.  
(1 == 2) || (3 < 2) is **false**, because neither of the comparisons is true.  
(1 == 1) || (3 != 6) is **true**, because both comparisons are true.

**The NOT operator**

The NOT operator gets a value of true when the operand is false. See the examples to gain a better understanding of the matter.

!(6 < 8) is **false**, because the operand is true.  
!(3 < 2) is **true**, because the operand is false.

|  |  |
| --- | --- |
| https://vw4.viope.com/existing-content/pics/info.gif | **NOTE!** It may be difficult to understand the purpose of all the operators described here. Later, you will discover how useful the operators are. At this stage, it is most important to gain a general understanding of all the available "tricks". You should also remember that you can always return and revise the sections that you have already read! |

### 4.3.7 Combinations and precedence of logical operators

In many situations, not even a combination of two comparisons is enough. In real life, we could need the following:

"If you are hungry **AND** you have money in your wallet **OR** you have a debit card in your wallet, have a meal at a restaurant."

In a similar way, we can combine several comparisons in the C language:

(hunger == 1) && (money == 1) || (debit\_card == 1)

What about the precedence? Just like mathematical operators and comparison operators, logical operators have their own hierarchy. The following table specifies the order of precedence between logical comparison operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Priority level in relation to other logical operators** | **Priority level in relation to other operators** |
| ! | 1 | Same priority as the operators ++ and --. That is, higher than that of comparison operators and binary operators. |
| && | 2 (higher than OR) | Lower than that of mathematical operators and comparison operators. |
| || | 3 | Lower than that of mathematical operators and comparison operators. |

You have now gained a general understanding of operators in the C language. All the programs you write are mostly based on variables and operations applied to them. The following chapter will illustrate the "power" of comparison operators, for example.

## Exercise 4.1

**Calculations**

You have completed this exercise

Submission time has expired

Write a program that prompts the user for two integers and prints the sum, difference and product of the numbers on the screen.  
  
**Hint:**  
This exercise is an excellent opportunity for practicing how to print several variables in one printf() statement. The variables in the program are integer variables.

*Example output:*

Enter the first number:83

Enter the second number:78

83+78=161

83-78=5

83\*78=6474

*The output of the program must be exactly the same as the example output (the most strict comparison level)* *The verification of program output does not account for whitespace characters like "\n", "\t" and " "* *The verification of program output is not case-sensitive* *The verification of program output does not account for whitespace and is not case-sensitive (the least strict comparison level)*

*Example query result:*

#include <stdio.h>

int main(void)

{

int first = 0, second = 0;

printf("Enter the first number:");

scanf("%d",&first);

printf("Enter the second number:");

scanf("%d",&second);

printf("\n"); /\* print an empty line \*/

printf("%d+%d=%d\n",first,second,first+second);

printf("%d-%d=%d\n",first,second,first-second);

printf("%d\*%d=%d\n",first,second,first\*second);

return 0;

}

## Exercise 4.2

**Checking if a number is even**

You have completed this exercise

Submission time has expired

Write a program that prompts the user for an integer and checks whether it is even or odd. If the number is even, print the value 0, and if it is odd, print the value 1.  
  
**Hint:**  
A number is even if the remainder from division by two is zero.

*Example output:*

Enter an integer: 83 The number is 1

*The output of the program must be exactly the same as the example output (the most strict comparison level)* *The verification of program output does not account for whitespace characters like "\n", "\t" and " "* *The verification of program output is not case-sensitive* *The verification of program output does not account for whitespace and is not case-sensitive (the least strict comparison level)*

**Solution:**

#include <stdio.h>

int main(void)

{

int number = 0;

printf("Enter an integer: ");

scanf("%d",&number);

printf("The number is %d\n",(number%2));

return 0;

}