Edube.or

C Beginner course

**On Natural language vs programming language**

Such languages are, among others, **programming languages**. You are probably familiar with this concept already. A programming language is defined by a set of certain rigid rules, much more inflexible than any natural language.

For example, these rules determine which symbols (letters, digits, punctuation marks, and so on) could be used in the language. This part of the definition of the language is called **lexicon**.

Another set of rules determines the appropriate ways of collating the symbols – this is the **syntax** of the language.

We would also like to be able to recognize the meaning of every statement expressed in the given language – and this is what we call **semantics**.

Any program we write must be correct in these three ways: lexically, syntactically and semantically, otherwise it will neither run nor produce any acceptable results

Lexic = symbols

Syntactic = order of those symbols

Semantics = recognition of statements meanings

**Machine Code and High-level Lenguage**

A computer responds only to a predetermined **set of known commands**.

A complete set of well-known commands is called an **instruction list**, sometimes abbreviated to **IL**

The IL is in fact the alphabet of a language, commonly known as a **machine language**

All these circumstances led to a need for some kind of **bridge** between the human language (natural language) and the computer language (machine language). That bridge is also a language – an intermediate common language for humans and computers to work together. Such languages are often called **high-level programming languages**.

programs written in high-level languages could be translated into any number of different machine languages and thus make them usable on many different computers. This feature of high-level programming languages is called **portability**.

**Compilators**

The translation of a high-level language is made by a specialized computer program called a **compiler**. The process of translating from a high-level language into a machine language is called **compilation**.

a program (which in fact is just text) is called the source code, or simply source, while the file which contains the source is called the **source file**.

it’s common for a file containing the source code in the “C“ language to have its name ending with the suffix “**.c**”

Next, your source code needs to be **compiled.**The compiler reads your code, does some complex analysis and its first goal is to determine whether or not you made any errors during the coding.

Not all errors are detected by compilers, using “#” instead of “+”, the compiler will inform you of your error. However, if you typed a "-" instead of a "+", the compiler will no longer be able to guess that your intention was to add two numbers, rather than to subtract them. **Do not expect the compiler to think for you.**

If the compiler doesn’t notice any mistakes in your source, the result of its work will be a file containing your program translated into machine language. That file is commonly called an **executable file**.

**Executable files**

The name of the file depends on the compiler you use and the operating system. Unix/Linux: create an output file named “*a.out*” by default. MS Windows changes the suffix of the program file from “*.c*” to “*.exe*”.

**On Compiling phases**

Source code might be comprehensive and divided among several or even dozens of source files, especially when working as a team of developers. In such cases, the compiling splits into two phases – a **compilation** of your source, in order to translate it into machine language, and a joining (or gluing) of your executable code with the executable code derived from other developers into a single and unified product. The phase of “gluing” the different executable codes is commonly known as **linking** while the program which conducts the process is called a **linker**.

**A few words about C**

The “C” language is one of a huge number of programming languages currently in use, and one of the oldest. It was created in the early seventies of the twentieth century by [Dennis Ritchie](https://en.wikipedia.org/wiki/Dennis_Ritchie) while he was working in Bell Laboratories.

What is the most common use of “C”? It is the so-called **general-purpose programming language**, i.e., suitable for almost any programming project and at the same time not particularly predestined to any specific, narrow class of applications

# "The C Programming Language" book

The book is called simply **The C Programming Language.** We recommend it without hesitation. Without any doubt, it is the best book on C programming ever written – its high quality can be proven by the fact that one of its authors is **Dennis Ritchie**.

#include <stdio.h>

int main(void)

{

puts("It's me, your first program.");

return 0;

}

he character # (hash) means that the content of this line is a **preprocessor directive.** The prefix “pre” suggests that these operations are performed **before** the full processing (compilation) takes place. The changes the **preprocessor** introduces are controlled entirely by its **directives**.

his preliminary information needed by the compiler is included in the files whose names usually end with “*.h*” (**header**). These files are commonly called **header files**.

The **stdio.h** file (defined by the standard of the “C” language) contains a collection of preliminary information about ready-made blocks which can be used by a program to write text on the screen or to read letters from the keyboard.

**Functions**

Every function in “C” begins with the following set of information:

* what is the **result** of the function?
* what is the **name** of the function?
* how many **parameters** does the function have and what are their **names**?

Return 0 and 1 in main: 0 The program executed correctly, 1 mean that something had gone wrong, it didn’t allow your program to be successful and the operating system could then use that information to react in the most appropriate way.

**Variables**

There are special “**containers**” for this purpose and these containers are called **variables**

What does every variable have?

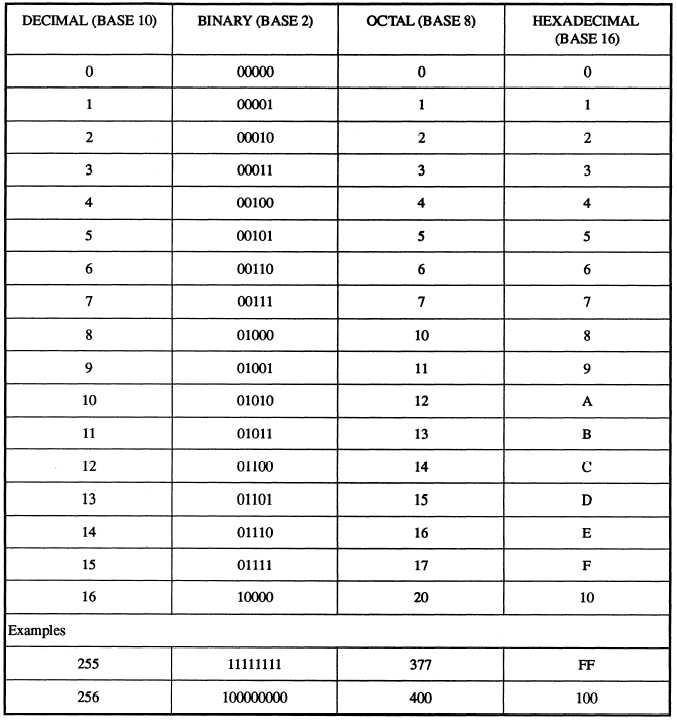
* a **name**;
* a **type**;
* a **value**;

If you want to give a name to a variable you must follow some strict rules:

* the name of the variable must be composed of **upper-case or lower-case Latin letters, digits and the character** \_ (*underscore*);
* the name of the variable must **begin with a letter**;
* the **underline character is a letter** (strange but true);
* upper- and lower-case letters are treated as **different** (a little differently than in the real world – Alice and ALICE are the same given names but they are two different variable names, consequently, two different variables);

A number that starts with 0 is an Octal. Like 0123 with a decimal value equal to 83.

A number like 0x123 is a **hexadecimal number** with a decimal value equal to 291.



#include <stdio.h>

int main()

{

printf("The value of twenty-one is: %d \n", 0x15);

printf("The value of twenty-two is: %d \n", 0x16);

printf("The value of sixty-two is: %d \n", 0x3E);

printf("The value of seventy-four is: %d \n", 0x4A);

return 0;

}

Prints:

The value of twenty-one is: 21

The value of twenty-two is: 22

The value of sixty-two is: 62

The value of seventy-four is: 74

Types

The type is an attribute that uniquely defines which values can be stored inside the variable.

The variable comes into existence as a result of a declaration.

Int myInteger; (declaration)  
myInteger = 0; (initialisation)  
Int myInteger = 0; (declaration and initialisation)



Excercice:

#include <stdio.h>

int main(void){

//Q1

int ene = 31;

int feb = 28;

int mar = 31;

//Q2

int abr = 30;

int may = 31;

int jun = 30;

//Q3

int jul = 31;

int ago = 31;

int sep = 30;

//Q4

int oct = 31;

int nov = 30;

int dic = 31;

printf("Days in Q1 of the current year: %d\n", ene+feb+mar);

printf("Days in Q2 of the current year: %d\n", abr+may+jun);

printf("Days in Q3 of the current year: %d\n", jul+ago+sep);

printf("Days in Q4 of the current year: %d\n", oct+nov+dic);

return 0;

}