Introduction to Programming Language (C)

(EEF 110E)

Dr. Emre DİNCEL

LECTURE NOTES

(WEEK 1)

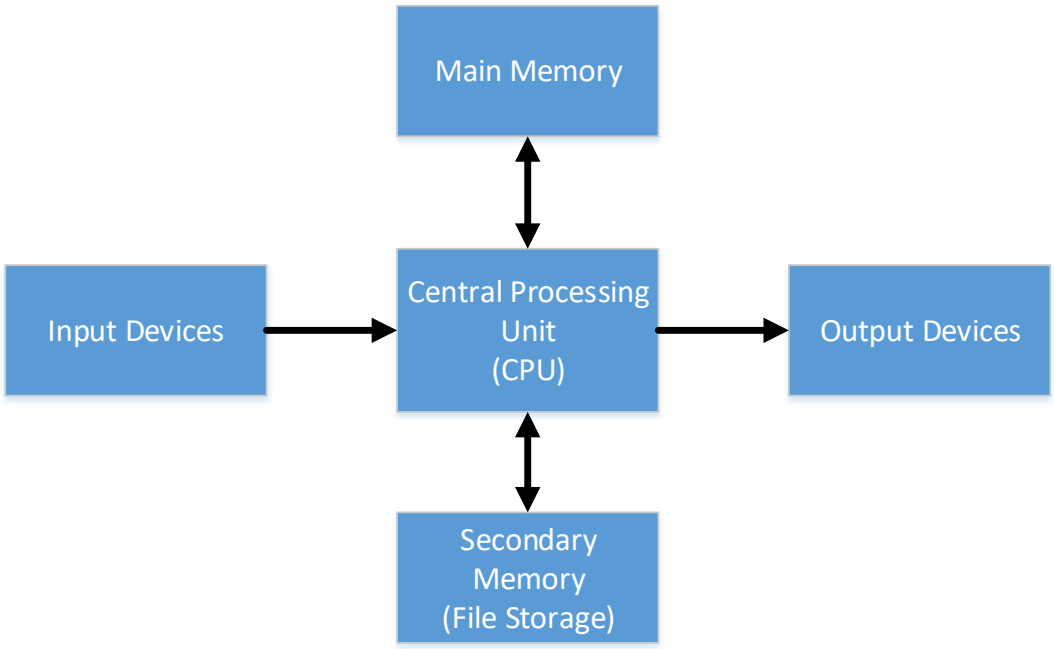
❖ Introduction to Programming Language C

- Why C?

It is possible to say that C is the lowest high level language. It also has relatively flexible syntax rules. C is one of the most widely used programming languages, and many great programs and operating systems (such as Unix) are written in C. Also, note that C++ extensions allow object oriented programming.

- Main Components of a Computer:

In the below figure, we see the basic components of a computer based system.



The central processing unit (CPU) can be called as the brain of the computer. It coordinates all computer operations (controller unit), and performs arithmetic and logical operations (arithmetic-logic unit: ALU). The CPU follows instructions contained in a program which has been loaded into memory. These instructions specify which operations should be carried out and in what order.

The CPU does the followings in an infinite loop:

- Retrieve the instruction (fetch the instruction)
- Interpret (decode) the instruction to determine what to do
- Retrieve any data necessary to carry out the instruction
- Perform (execute the instruction) the actual manipulation of the data

- If necessary store the results back in memory
- Move to the next instruction

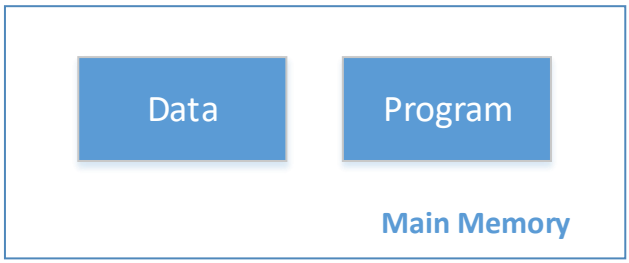
The CPU can perform basic arithmetic and logic operations on data. Based on simple comparisons it can also make simple decisions about which instruction to execute next.

Main memory is used to store both data and programs. All kind of information is stored as combinations of 1 and 0s. The smallest memory cell is called as bit (1 or 0).

1 Byte = 8 Bit

1 Kbyte = 2^{10} = 1024 Bytes

1 Mbyte = 1024 Kbyte = 1,048,576 Bytes



It is possible to imagine memory as a list of bytes. The data used in programs is usually stored in memory as one or more bytes. In order to reach the data, we use labels of addresses where the data is stored. These labels are called as variables.

Address	Data (8 bit)
Byte 0	xxxxxxxx
Byte 1	xxxxxxxx
…	…
Byte N-1	xxxxxxxx

- Main parts of a program:

We can list the main parts of a program as follows:

- Beginning of the program
- Definitions and declarations (of variables, constants, functions, etc.)
- Execution
  - Initialization
  - Reading data from input

- Necessary calculations
- Writing results to output

➢ End of the program

Consider the above program written in C:

```
#include <stdio.h> // Import library file to be used

int main() // Main function
{
   printf("Hello World!\n\r");
   return 0;
}
```

This program prints "Hello World!" on the screen. Note that the program starts after the curly parenthesis that follows **main()**. All C (and also C++) programs have a main function. The program then ends with the **return** command.

The pre-processor statement **#include** causes the library called *stdio* to be used. Here, stdio.h is called a header file. Header files are used to be able to use predefined functions in C. For instance, the function **printf** is defined in stdio.h.

Note that \n inside a print statement means the new line character, and in UNIX environment, in order to go to the beginning of a line, the carriage return character \r can be used. Also, the return statement in the main function causes the program to stop the execution. Here, 0 indicates that the program has terminated normally.

Please note that, C is a case sensitive language. Generally, all predefined statements and functions are in lower-case letters. Also, there is a semi-colon (;) after all statements.

- Compiling and Linking:

In order to convert a C program to machine code we need to compile it. The program that does this job is called a C compiler. The compiler takes the C program (source code) and all the necessary header files required by the program, and then produces an object file (.obj). This object file, if necessary together with other object files (libraries), is converted to an executable file by another program called linker. The resulting executable file can be run on the machine.

Please note that, in most of the software development environments such as MS Visual C++ the compiler and linker can be together.

- Errors:

It is possible to divide the programming errors into two categories:

- Syntax errors (errors that result in not obeying the grammar (syntax) rules of the language – compile time errors).
- Semantic (logic) errors (errors on the logic of the program – run time errors).

- Basic Data Types:

The followings are the basic data types supported by C language:

- Integers (int, char)
- Real Numbers (float, double)
- Addresses (pointers)

In addition to these basic data types, there exists more complex data types such as **arrays**, **structs**, **unions** which will be introduced later in the course.

Integers are whole numbers without any fractional part. For example: 1, 0, -12, 25, … are all integers.

Real or floating point numbers are numbers with fractional part such as 4.5, -12.287, 11.0, …

Note that an integer cannot have a fraction, whereas a real number can. On a 32-bit computer an integer can have values between (-2^{31}) to $(2^{31}-1)$, whereas a real number can be between -10^{38} to $+10^{38}$ with 6 or 7 significant digits.

However, note that integers take less space in memory compared to real numbers. Also, arithmetic operations on integers are much faster than those on real numbers.

❖ Variables

In C programming language, we need to define the names and types of variables before we start to use them. The syntax for declaring a variable is

type name1, name2, … ;

So in order to declare an integer variable called k we simply type

int k;

Similarly, to declare three real numbered variables with labels x, y and z, we type,

float x, y, z;

In the following example, we can see that an integer variable named as “number” is defined. Right after the definition, please note that the variable does not have any value; therefore, it is initialized in the next line as *number = 2 + 5*. Thus, the value of the variable is now “7”. In the next line, we print the content of the variable on the screen via printf function. Note that “%d” is used to print integer type variables on the screen.

```
#include <stdio.h>

int main() // Main function
{
   int number;
   number = 2 + 5;
   printf("%d\n", number); // Prints 7 on the screen
   return 0;
}
```

It is also possible to assign the initial value while declaring the variable such as,

double speed = 5.25, acceleration = 1.0;

As you may already noticed, the assignments are done in C in the form of

name = expression;

Here, **name** is the name of a variable and **expression** is an arithmetic, or other, expression. If necessary, expression is first calculated to determine the value to be assigned to the variable. For example,

x = y + 2;

Please do not confuse the equation sign used here with mathematical equations. It is possible to read this assignment in plain English as “let x be y plus 2” or similarly “assign y plus 2 to x”.

On the other hand, in an assignment statement, the same variable can appear in both sides of the statement. For example,

x = x + 1;

is a meaningful assignment and would cause the value of x to be incremented by 1.

- Reserved Words (Keywords):

There are a number of names called reserved words (keywords), which may not be used for any purpose other than they are intended to. There are 32 keywords in ANSI C:

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

- Identifies:

There are several syntax rules an identifier (a variable, function or type name) has to obey. For instance,

- It consists of characters containing the letters (A-Z and a-z), the digits (0-9) and the underscore character (_).
- It cannot start with a digit.
- It is case sensitive.
- If the identifier consists more than 31 characters, all characters beyond the 31st character may be ignored by any given compiler.
- Reserved words cannot be used as identifiers.
- Identifiers used before for identifying a different variable, function, etc., cannot be used again.

It is also important for an identifier to be chosen to indicate the meaning of its use!

- Arithmetic Operations:

There exist 5 arithmetic operations listed below:

+	Addition
-	Subtraction
*****	Multiplication
/	Division
%	Reminder (or modulus)

Also, - and + operators can be used as sign operators. For example,

```
y = -x;      // Means that y = -1*x
y = +x;      // Means that y = x
z = x - -y; // Means that z = x – (-y) = x + y
```

Note that division of the integers result in an integer such as 5/3 = 1, 7/8 = 0, etc.

There can be more than one arithmetic operation in an arithmetic expression. In such cases, operations in an arithmetic expression are done from left to right unless the operation on the right has a priority over the operation over left.

Operator	**Priority**
Unary (+, -)	Highest
*, / and %	Medium
Addition and subtraction (+, -)	Lowest

Nevertheless, priorities can be overwritten by the use of parentheses. For example,

(1.4 + 2.8) / 0.7 = 6.0

Note that the result would be 5.4 without the parentheses. Therefore, use parentheses when you are not sure to be on the safe side, and to increase the readability of your code. You can also add spaces between the variables and operators so as to increase the readability.

When an operation takes place on a real number and an integer, the integer is first converted to a real number so the result is real. When an integer is assigned to a real number, it is simply converted to a real number, whereas, when a real number is assigned to an integer, it is truncated to an integer.

- Storing Characters:

Characters are stored as 8-bits of binary data using the ASCII codes. For instance,

A	0100 0001 (decimal 65)
B	0100 0010 (decimal 66)
C	0100 0011 (decimal 67)
D	0100 0100 (decimal 68)
…	…

a	0110 0001 (decimal 97)
b	0110 0010 (decimal 98)
c	0110 0011 (decimal 99)

Character variables can be defined as follows.

char var1, var2, … ;

Let us examine the following code segment:

```
char ch = 'A';
printf("%d\n", ch);  // prints 65 on the screen
printf("%c\n", ch);  // prints A on the screen
```

As you can see from the above example, characters are actually integer type variables. Please, however, note that in order to see them on the screen, %c is used in the printf function. We can even perform arithmetic operations on character variables as in the below program example:

```
#include <stdio.h>

int main()
{
   char ch;
   ch = 'A' + 4;
   printf("%c\n", ch);  /* prints E on the screen */
   return 0;
}
```

- Combining Arithmetic Operators with =:

It is possible to combine the arithmetic operators with = expression as follows.

```
x += y;    ⟶    x = x + y;
x -= y;    ⟶    x = x - y;
x *= y;    ⟶    x = x * y;
x /= y;    ⟶    x = x / y;
x %= y;    ⟶    x = x % y;
z *= x + y;    ⟶    z = z * x + y;
```

Increment (++) and decrement (--) operators can also be used.

```
x++;  ⟶  ++x;  ⟶  x += 1;  ⟶  x = x + 1;
x--;  ⟶  --x;  ⟶  x -= 1;  ⟶  x = x - 1;
```

Be careful about the position of the increment (++) and decrement (--) operators since they affect your code. We will provide an example about this issue later in the course.

- Comments:

In a C program anything put in between /* and */ are comments (similarly // on the line). These are not considered by the compiler and have no effect on the execution of the program. These, however, help the future programmers to understand what that particular part of the program is doing. Therefore, they increase the readability and portability of the programs.