PROGRAMMING LANGUAGES

Laboratory 2

BASIC FEATURES OF THE "C" PROGRAMMING LANGUAGE

I. THEORETICAL BACKGROUND

1. Short term dictionary:

OBJECT	= the content of a memory zone
LVALUE	= expression referring to an object
POINTER	= a variable containing the address of another variable
EXPRESSION	= combination of variables and constants for producing new values
( )	= list of arguments for a function or for an arithmetic expression
{ }	= groups composed instruments, functions
[ ]	= borders masive dimensions or indices of masive elements
" "	= borders character strings
' '	= borders a character or an avoiding sequence
;	= terminates an instruction
/*...*/	= comment

2. Constants

2.1. Integer constants (generated on 2 bytes)
They could be of the following types:

- decimal
- octal – is an integer constant starting with 0
  E.g.: 8 => 010
- hexadecimal – is an integer constant starting with 0x or 0X

2.2. Long explicite constants (generated on 4 bytes)
This is an integer constant followed by l or L letters.

2.3. Float constants
They are comprised of an integer part, a decimal point, a fractionary part, an „e” or „E” and an integer signed exponent. The integer part, the fractionary part, the decimal point, the „e” or the signed exponent are not always present.

2.4. Character constants
They are characters written between ' '.
E.g.: 'x'

2.5. Symbolic constants
These are the constant value identifiers. They are introduced using "#define".
E.g.: # define MAX 1000

3. Variables

3.1. Memory classes
Depending on the working mode, the next classification is imposed:

a) Automatic variables
They are declared implicitly in the context or using the "auto" identifier.
These variables are local to each block and are automatically destroyed when leaving the block.

b) External variables
These variables are declared using the „extern” identifier or implicitly in the context. The external variables values are available throughout the entire program.

c) Static variables
These variables are declared using the „static" identifier and are available only in the file they were declared.
There are two types of static variables:
- internal – inside a function, and they are not destroyed when the function is left;
- external – in the entire program.

d) Register variables
They are declared using the "static" identifier. They are similar to the „auto" variables, just that they are used more frequently.

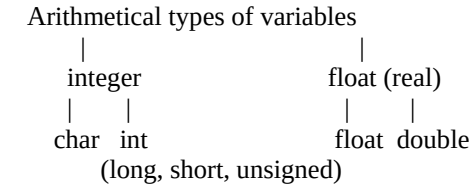
3.2. Variable types

a) Character variables => they are declared with "char"
They have 1 byte (-128 - 127)

b) Integer variables => "int"
They could be: "short" (2 bytes)
"long" (4 bytes)
"unsigned" (no sign)

E.g.: short int x;
long int y;
unsigned int z;

c) Float variables
"float" => simple precision
"double" => double precision



Obs.: Integers are always signed quantities.

4. Conversions

The basic „C" accepted conversions are:

char => int (Caution! A signed integer could be introduced)

int => char (superior order bits are lost)

short => int

long => int (truncation: excedent superior-order bits are lost)
=> short
=> char

float => double (zeros are added to the fractionary part)

int => float
=> unsigned

char => int

short => int

Obs.:
a) If an operand is "double" type, the other operand is automatically converted and the result is also "double".
The same goes for "long" and "unsigned".
b) Assigning conversions: the right member values is converted to the left member value. And the result has the same type.
c) Logical conversions are also possible.

5. Format specifiers:

%u => unsigned int
%d => int
%ld => long int
%p => pointer
%f => float
%e => f loat with exponential format
%c => char
%s => string char
%x sau %X => integer in hexadecimal format

Obs.: When also digits appear in the format specifiers.
E.g.: %5d => integer on 5 spaces with right alignment
%-5d => integer on 5 spaces with left alignment

6. Avoiding sequences:

\n => new line
\t => the tab character
\b => the backspace character
\\ => the backslash character (\)
\/ => the slash character (/)
\xhhh => the character represented by the ASCII hhh code, where hhh represents 1-3 hexadecimal digits.

7. The prefix file "stdio.h"

This file contains a few standard preset rutines, some useful definitions for working with files, symbolic constant definitions and also the redefinition of some special characters that do not exist on all types of keyboards.
Including this file in all "C" programs is highly recommended.

8. Executing a program

All C programs have a "main" function. Generally, a program's execution is finished at the end of the "main" function.

9. Standard routines

9.1. For printing

a) The "printf" routine – prints a message on the screen

Format: printf (<string_format>, <object>, <object>, ...)

Obs.: <string_format> has to be bordered by quotation marks.

E.g.: printf("result %d \n", result);
| |
Format specifier Avoiding sequence

Program :

```
/*  HELLO.C  --  Hello, world */

# include <stdio.h>

main ( )
{
      printf ("Hello, world  \n");
}
```

Obs.:

Ctrl + F9 => compiling and running the program
ALT + F5 => screen visualisation

b) The "puts" routine
Prints a string on the scrren, followed by the new line sequence (\n).

Therefore: printf("Hello \n"); <=> puts("Hello");

c) The "putchar" routine
Writes a character without jumping to a new line.

Therefore: printf("%c", 'x'); <=> putchar('x');

printf("%c \n", 'x'); <=> putchar('x');
putchar('\n');

Obs.: The code of the "printf" rutine is big and that is why "puts" and "putchar" routines are preferred for optimization.

9.2. Reading routines

a) The "scanf" routine – reads a message from the keyboard.

Format: scanf(<string_format>, <address>, <address>, ...);

E.g.: scanf("%d %d", &x, &y);
|
This space means that between the two values could be one ore more blank spaces (blank, tab, new line).

Obs.: scanf("%d , %d", &x, &y);
|
This comma means that the two values are going to be written separated by comma.

Program (sending an address):

```
/*  HELLO.C  --  Hello, world */

# include <stdio.h>
main ( )
{
      char nume [25];
      printf("Nume: ");
      scanf("%s", nume);
      printf ("Hello, %s \n", nume);
}
```

Obs.: The program will display only the last name of the person, and not also the first name. That is because the blank space indicates the ending of the string to be read.

A solution for this drawback:

```
      /*  HELLO.C  --  Hello, world */

# include <stdio.h>
main ( )
{
      char nume [15], prenume [15];
      printf("Nume:");
      scanf("%s    %s", nume, prenume);
      printf ("Salut, %s %s \n", nume, prenume);
}
```

b) The "gets" routine

Read the entire string from the keyboard, till the Enter key is hit.

Format: gets (string);

Another solution to the previously presented problem:

```
      /*  HELLO.C  --  Hello, world */

# include <stdio.h>
main ( )
{
      char nume [25];
      printf("Nume:");
      gets(nume);
      printf ("Salut, %s \n", nume);
}
```

c) The "getch" si "getche" routines

These routines read a single character from the keyboard. The two routines don't have parameters and return a char value.

Format: getch( ); works with no echo
getche( ); first displays the introduced character

E.g.:

```
#include <stdio.h>
main ( )
{
  char c;
  printf("Introduce a character: ");
  c = getch( );                |
  putchar (c);                 | <=>  putchar(getch( ));
}
```

Or:

```
 #include <stdio.h>
 main ( )
  {
    printf("Introduce a character");
    getche( );
  }
```

II. Assignment Workflow

1. Write and then run all the programs from paragraphs 9.1. and 9.2. Make conclusions.
2. Write a program that asks your name and the address. Afterwards the program will display a message like "the student … has the address …".