OPERATING SYSTEMS
— Laboratory 8 —

C PROGRAMMING

1. Data Types

Type	Memory size	Value range
char	1 byte	[-128, 127] or [0, 255]
int	2 or 4 bytes	[-32.768, 32.767] or [-2.147.483.648, 2.147.483.647]
float	4 bytes	[1,2E-38, 3,4E+38]
double	8 bytes	[2,3E-308, 1,7E+308]

- Data types char and int can be preceded by unsigned

- Data type int can be preceded by: short or long

Type	Memory size	Value range
unsigned char	1 byte	[0, 255]
unsigned int	2 or 4 bytes	[0, 65.535] or [0, 4.294.967.295]
short (int)	2 bytes	[-32.768, 32.767]
unsigned short	2 bytes	[0, 65.535]
long (int)	4 bytes	[-2.147.483.648, 2.147.483.647]
unsigned long	4 bytes	[0, 4.294.967.295]

2. REZERVED WORDS (Keywords)

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

3. CONSTANTS

- Can be defined 2 ways:
  - Using pre-processing directive #define:

    ```
    #define TEN 10
    #define NEWLINE '\n'
    ```

  - Using prefix const:

    ```
    const int TEN = 10;
    const char NEWLINE = '\n';
    ```

4. VARIABLES

- Defining a variable:

  type_variable **name_variable;**

  – *type_variable* can be: `char, int, short, long` etc. *(see point. 1)*

  name_variable can contain letters, digits and character „ _" *(underscore), but* the first character must be a digit

  – reserved words *(see pct. 2)* can not be used as variable names

  – C language is case sensitive *(case-sensitive)*

- examples:

```
int n;        int n = 10;
char c;       char c = 'a';
```

5. OPERATORS

- specify operations performed with variables and constants
- operator types:

  – *arithmetical:*   `+   -   *   /   %   ++   --`

  – *relational:*   `==   !=   >   <   >=   <=`

  – *logical:*   `&&  ||  !`

  – *bitwise operators:*   `&   |   ^   ~   <<   >>`

  – *assignment operators:*   `=   +=   -=   *=   /=   %=   <<=   >>=   &=   ^=   |=`

  – *other operators:*   `sizeof()   &   *   ?:`

- operators priority:

  https://en.cppreference.com/w/c/language/operator_precedence

6. DERIVED DATA TYPES

6.a. Unidimensional arrays (vectors)

  type_array **name_array**[dimension_array];

- examples:

```
int list[5];
int list[5] = {10, 20, 30, 40, 50};
double values[] = {100.0, 2.0, 300.0, 40.0, 50.0};
```

6.b. Strings (arrays of chars)

```
char msg[] = "Hello";
char msg[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

6.c. Pointers

- *pointer* = *a variable containing the address of an other variable*
- examples:

```
int *p;       // pointer to an int variable
char *c;      // pointer to a char variable
float *f;     // pointer to a float
```

```
double *d;    // pointer to a double
```

- how to obtain the address where a variable `x` is stored: `&x`

- how to obtain the value of variable `x`: `*p` *(if p is a pointer to variable `x`)*

6.d. Data structures

- defining a data structure:

```
struct Books
{
   int id;
   char author[50];
   char title[100];
}
```

- declaring and using a data structure:

```
int main(int argc, char** argv)
{
   struct Books book1;
   ...
   book1.id = 1000;
   strcpy(book1.author, "B.W. Kernighan, D.M. Ritchie");
   strcpy(book1.title, "The C Programming Language");
   ...
   return 0;
}
```

7. FUNCTIONS

- declare a function:

  tip_returnat **nume_functie***(tip_param* **param1***, tip_param* **param2***, ...);*

  where:

  – *tip_returnat* can be anny data type or `void`

  – **nume_functie** may contain letters, digits and the character „ _" *(underscore)*

  – first character must be a letter

  – reserved words *(see pct. 2)* can not be used as function names

  – *tip_param* **param1***, tip_param* **param2***, ...* is the list of formal parameters or `void`

  (if the function has no parameters)

- examples:

```
void afiseaza_matrice(int** matrice)
float media_aritmetica(int a, int b)
int** citeste_matrice(FILE* file)
```

- function `main()`:

  – is the main entry point in the program *(the program main entry point)*

  – recommended header entry point:

    `int main(int argc, char** argv)`

  because it allows access to the command line arguments provided by the user (parameters).

8. INPU/OUTPUT FUNCTIONS

```
int getchar(void)
int putchar(void)
char *gets(char *s)
int puts(const char *s)
int scanf(const char *format, ...)
int printf(const char *format, ...)
```

9. FUNCTIONS FOR WORKING WITH FILES

9.a. For text files:

```
FILE *fopen(const char *filename, const char *mode)
int fgetc(FILE *fp)
int fgets(char *buf, int n, FILE *fp)
int fputc(int c, FILE *fp)
int fputs(const char *s, FILE *fp)
int fclose(FILE *fp)
```

9.b. For binary files:

```
size_t fread(void *buf, size_t bsize, size_t nbyte, FILE *fp)
size_t fwrite(const void *buf, size_t bsize, size_t nbyte, FILE *fp)
```

or:

```
int open(const char *path, int oflag, ... )
ssize_t read(int fd, void *buf, size_t nbyte)
ssize_t write(int fd, const void *buf, size_t nbyte)
int close(int fd)
```

10. HELLO WORLD IN C IN UNIX

- UNIX text editors: `vim, nano, joe`

- example: `hello.c`

```c
#include <stdio.h>

// the program main entry point
int main(int argc, char** argv)
{
  printf("Hello world !\n");
  return 0;
}
```

- compilation: `gcc –Wall -g -o hello hello.c`

- program execution (run): `./hello arg1 arg2 arg3 …`

11. EXAMPLES

- print the numver and values of command line arguments:

  `wget http://www.cs.ubbcluj.ro/~dbota/SO/lab2/exemple/lab2_2.c`

- get environment variables:

  `wget http://www.cs.ubbcluj.ro/~dbota/SO/lab2/exemple/lab2_3.c`

- using unidimensional arrays:

  `wget http://www.cs.ubbcluj.ro/~dbota/SO/lab2/exemple/lab2_4.c`

- read and int from stdin:

```
wget http://www.cs.ubbcluj.ro/~dbota/SO/lab2/exemple/lab2_5.c
```

- read the content of a text file:
  ```
  wget http://www.cs.ubbcluj.ro/~dbota/SO/lab2/exemple/lab2_6.c
  ```
- read the content of a matrix from a text file:
  ```
  wget http://www.cs.ubbcluj.ro/~dbota/SO/lab2/exemple/lab2_7.c
  ```

12. COMPILE ERRORS AND WARNINGS

- syntax errors
- missing libraries
- use of undefined variable
- two variables with the same name
- use of undeclared function
- function call with wrong parameters (incorrect argument number or argument order etc.)

13. DETECTING MEMORY LEAKS

- detecting errors related to memory leaks using the tool `valgrind`:
  `valgrind ./myprog`

14. INFORMATION ABOUT C FUNCTIONS IN UNIX:

whatis functionname

man sectionnumber functionname

Most C functions are in section 3, with system calls in section 2.

Examples:

man 3 printf

man 2 open

REFERINȚE:

- Programare C: https://www.tutorialspoint.com/cprogramming/index.htm
- Manual Valgrind: http://valgrind.org/docs/manual/quick-start.html

Lab 8 – Homework C Programming 1

No.	Assignment
1	Write a C Program that receives a command line argument a filename, and prints to standard output the content of this text file, printing also an empty line after each line of text. Make no assumption regarding the maximum length of a line.
2	Write a C Program that deletes from a file the bytes from even offsets. The name of the file is provided as a command line argument.
3	Write a C Program that counts the number of words on each line of a file. Make no assumptions regarding the maxim length of a line.
4	Write a C Program that compares the content of two files provided as command line parameters. The comparison will be performed on blocks of 1KB each. The program will print the total number of blocks from each file and the number of identical blocks found.
5	Write a C Program that receives as command line arguments two words and the name of a file. The program will print at stdout the content of the file, replacing the occurrences of the first word with the second.
6	Write a C Program that receives as command line arguments a word and the name of a file. The program will delete from the file the occurrences of the provided word.
7	Write a C Program that receives as command line argument the name of a file, and reverses the order of the bytes in the file.
8	Write a C Program that receives as command line argument the name of a file and deletes the even numbers from this file.
9	Write a C Program that receives as command line arguments two file names and merges the lines of the two files (that are already sorted).
10	Write a C Program that receives as command line argument a name of a file. The program prints at stdout the content of this file, erasing all empty lines from the file. Make no assumptions regarding the maximum length of a line.
11	Write a C Program that receives as command line argument a name of a file. The program duplicates the bytes from even offsets in the file.
12	Write a C Program that counts all the digits on each line of a text file. Make no assumptions regarding the maximum length of a line.
13	Write a C Program that receives as command line arguments a name of a file and several words. The program will print at stdout the content of the file, deleting the occurrences of all words provided as arguments.
14	Write a C Program that counts the number of characters on each line of a text file. Make no assumptions regarding the maximum length of a line.
15	Write a C Program that counts the number of letters on each line of a text file. Make no assumptions regarding the maximum length of a line.
16	Write a C Program that receives as command line arguments a number and a filename, and deletes from the file the even occurrences of the number.