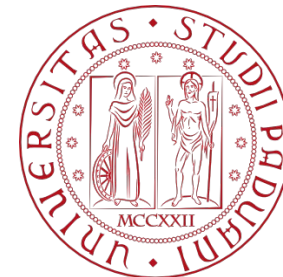


# COMPUTER ENGINEERING LABORATORY

**Luigi Rizzo**

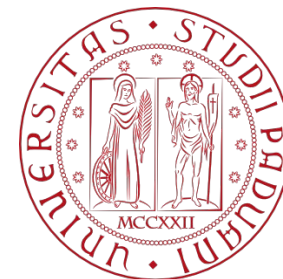
[luigi.rizzo@unipd.it](mailto:luigi.rizzo@unipd.it)

October 2024-January 2025



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

# Exercises: loops, arrays, structs, enums, input/output



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

Input and output are not part of the C language itself.

We shall use the standard library, a set of functions that provide input and output, string handling, storage management, mathematical routines, and a variety of other services for C programs.

The library implements a simple model of text input and output. A text stream consists of a sequence of lines; each line ends with a newline character.

The simplest input mechanism is to read one character at a time from the standard input, normally the keyboard, with **getchar**:

```
int getchar(void)
```

getchar returns the next input character each time it is called, or EOF when it encounters end of file. The symbolic constant EOF is defined in `<stdio.h>`.

In many operating systems, a file may be substituted for the keyboard by using the `<` convention for input redirection: if a program uses getchar, then the command line *program <infile* causes program to read characters from infile instead of standard input.

The input may come also from another program via a pipe mechanism: on many systems, the command line *otherprogram | program* runs the two programs otherprogram and program and pipes the standard output of otherprogram into the standard input for program.

The function

*int putchar(int)*

is used for output: `putchar(c)` puts the character `c` on the standard output, which is by default the screen. `putchar` returns the character written, or EOF if an error occurs. Again, output can usually be directed to a file with *>filename*: if program uses **putchar**,

*program >outfile*

will write the standard output to `outfile` instead. If pipes are supported,

*program | anotherprogram*

puts the standard output of `program` into the standard input of `anotherprogram`.

Each source file that refers to an input/output library function must contain the line

**`#include <stdio.h>`**

before the first reference. When the name is bracketed by < and > a search is made for the header in a standard set of places (for example, on UNIX systems, typically in the directory /usr/include).

An example, considering the program lower, that converts its input to lower case.

```
#include <stdio.h>
#include <ctype.h>
main() /* lower: convert input to lower case */
{
    int c
    while ((c = getchar()) != EOF)
        putchar(tolower(c));
    return 0;
}
```

The function `tolower` is defined in `<ctype.h>`; it converts an upper case letter to lower case and returns other characters untouched.

``functions" like `getchar` and `putchar` in `<stdio.h>` and `tolower` in `<ctype.h>` are often macros, thus avoiding the overhead of a function call per character.



Write a program that asks the user to type a string of N characters, reads the string from stdin until carriage return is pressed, inverts it and displays the inverted string (e.g. "Computer" becomes "retupmoC")

Some suggestions:


- Define a function for each subproblem
- Solve one subproblem at a time and check the solution before proceeding with the next subproblem.

# Exercise



```
int main()
{
    char s[N], c;
    int i = 0;
    printf("Type the word to reverse:\n");
    while ((c = getchar()) != '\n')
    {
        s[i++] = c;
        if (i == (N - 1))
            break;
    }
    s[i] = '\0';
    printf("\nString to be inverted: %s", s);
    return(0);
}
```

*reverse(s);*  
*printf("\nReversed string: %s", s);*



# Exercise



```
#include <stdio.h>
#include <string.h>
#define N 41
```

```
void reverse(char v[])
{
    int i, l;
    l = (int)strlen(v);
    for (i = 0; i < l / 2; i++) exchange(&v[i], &v[l - i - 1]);
}
```

```
void exchange(char* a, char* b)
{
    char aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

# Exercise 1



Given a string, transform it in a new string, in which every character is located offset positions further in the alphabet

- The alphabet considered is:
    - The one included between the characters n.32 and n.126 of the ascii table
  - The alphabet is cyclical: after the character n.126 there is the one n.32
  - For example, with offset = 4
    - the character 'a' becomes 'e'
    - the letter 'X' becomes '\'
    - the letter 'x' becomes '|'
1. The string to be transformed and the offset (can be a positive or a negative integer) are passed as arguments to the program
  2. The string to be transformed shall be read from a file whose name and the offset (can be a positive or a negative integer) are passed as arguments to the program

# Exercise 1 continue



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>

Source: [www.LookupTables.com](http://www.LookupTables.com)

Some suggestions:

- Define as many subproblems as possible
- Solve one subproblem at a time
- Write the main program with the instructions solving the 1st subproblem, check that the execution is correct than proceed with the instructions solving the 2nd subproblem and so on.

# Exercise 2



The Physical Characteristic data type represents the characteristic of a person.  
Each piece of information consists of  
weight (in ounces)  
height (in cm)  
hair colour (blond, brown, black, white)  
age

Define a data type that can contain the data above illustrated.  
Suggestion: you should use both struct and enum data types to manage the above illustrated data.

Write a program that

- prints how much space (in bytes) is occupied by each instance of the structure
- read from a file, redirected as std input to the program, at most 20 instances (one instance per row) of data (in any row each element is separated by the next one by the character ';')
- prints information for all people whose age value is  $\leq 20$
- prints information for all people whose hair colour is blond

Hair colours are represented as follows (there could be other colours too)

- blond 1
- brown 0
- black 3
- white 2



Some suggestions:

- Define as many subproblems as possible
- Solve one subproblem at a time defining a function for every subproblem
- Write the main program invoking the first function (that solves the first subproblem), check that the execution is correct than proceed with the other functions.