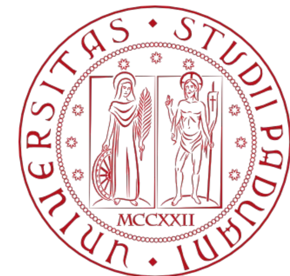
COMPUTER ENGINEERING LABORATORY

Luigi Rizzo

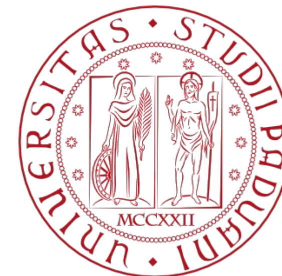
luigi.rizzo@unipd.it

October 2023-January 2024



UNIVERSITÀ DEGLI STUDI DI PADOVA

Getting started



Università degli Studi di Padova

To start

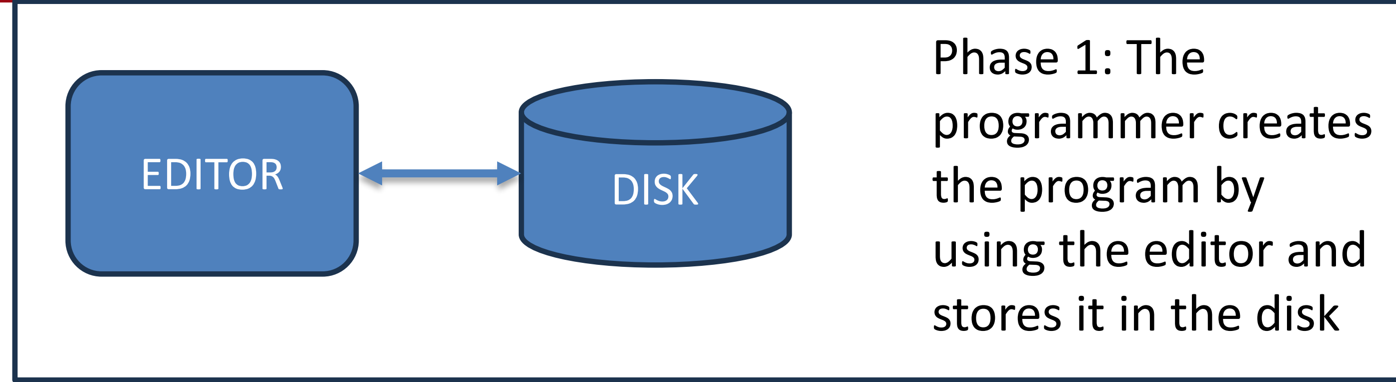
A quick introduction to C language illustrating the essential elements of the language in real programs, focusing on the fundamentals: variables and constants, arithmetic, control structures, functions, input/output elementary principles.

The first program to write is the same for every language: print the words “Hello world!”

What shall be done:

- create the program text
- compile it correctly
- run it
- know where the output is sent

Hello world



printf command is avalibale in stdio library

```
#include <stdio.h>

int main () {
	/* our first program that
	prints the string ”Hello world!”
	*/
	printf(”Hello world!\n”); // \n represents a newline
}
```

start of the function: {

comments

character sequences (strings) must becontained between ” ”

commands separated by ;

end of the function: }

every C program shall have function main

Hello world

The way to run the program depends on the system you are using. As a specific example on a Linux system you shall create the source program in a file with .c extension like helloWorld.c and compile it with the command gcc

For example the command

gcc helloWorld.c –o helloWorld

if no errors have been made in writing the program, it will compile, creating an executable file with the name helloWorld

Running the program via helloWorld command will produce the output

Hello world!

In other system the rules may be a bit different but more or less very similar

Hello world: C language main characteristics

- A C program consists of functions and variables. A function contains statements that specify the computing operations to be done, and variables store values used during the computation
- Programmers may give functions whatever names they like, but "main" is special - every program begins executing at the beginning of main
- #include <stdio.h>
  - tells the compiler to include information about the standard input/output library; the line appears at the beginning of many C source files
- One method of communicating data between functions is for the calling function to provide a list of values, called arguments, to the function it calls. The parentheses after the function name surround the argument list. In this example, main is defined to be a function that expects no arguments, which is indicated by the empty list ( )
- The statements of a function are enclosed in braces { }

Notes

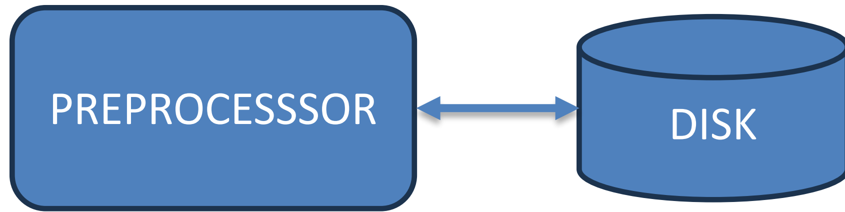
Some considerations.

- \n represents only a single character. An escape sequence like \n provides a general and extensible way for representing invisible characters. Other escape sequences that C provides are \t for tab, \b for backspace, \" for the double quote and \\ for the backslash itself.

Exercises

- Run the ``hello, world'' program on your system. Experiment with leaving out parts of the program, to see what error messages you get.
- Experiment what happens when prints's argument string contains \c, where c is some character not listed above.

Translation



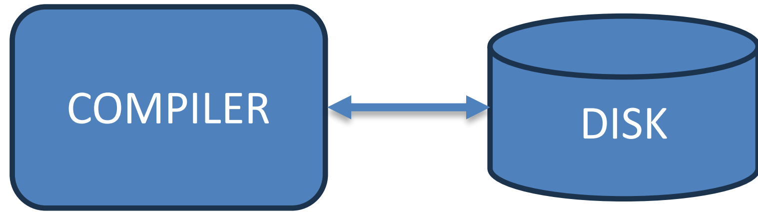
Phase 2: the preprocessor processes the code

- Comments removal
- Each line starting with # indicates a preprocessor directive
- #include <x>: the contents of file x are copied to this point in the file
  - #include <x> allows access to the commands made available by the x library
  - Eg stdio.h allows you to use the printf command
- Expanding macros (we'll see them later)
  - #define X 3, replaces every occurrence of X in the file with 3

```
#include <stdio.h>

int main () {
        printf(”Hello world!\n”);
}
```

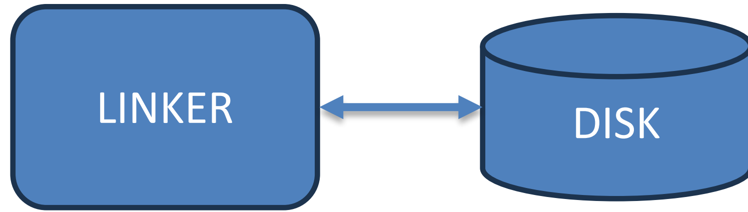
Translation



Phase 3: the compiler creates the object code and stores it in the disk

- The compiler analyzes the code file including the code and translates it into low-level language instructions
- Instructions shall strictly follow the syntax defined by the C language
- An error is generated if the compiler fails to parse the code
- If successful, a file with low-level language instructions is generated

Translation



Phase 4: the linker links the object code to the libraries, creates an executable file and and stores it in the disk

- A program is generally made up of many files and makes use of functions already written by others (for example printf)
- To avoid duplicating the code of these functions, they are loaded into memory once and linked to our program
- The linker is invoked by passing it the file that uses printf and the file where printf is defined (both compiled)

Comments

- High level description of what (or how) some fragment of code or an entire program does
- Purpose of comments: let those who read the program understand the code as quickly as possible
- Comments shall not be trivial: 3+2; //add 3 and 2
- When using an unusual algorithm for solving a problem, it’s better to indicate such usage
- Every program and function should indicate how shall be invoked
- Any characters between /* and */ are ignored by the compiler; they may be used freely to make a program easier to understand. Comments may appear anywhere where a blank, tab or newline can

Variables and Arithmetic Expressions

UNIVERSITÀ DEGLI STUDI DI PADOVA

Using the formula °C=(5/9)(°F-32) print a table of Fahrenheit temperatures and their centigrade or Celsius equivalents, for values from 0 to 100 °F

The program can still consist of the definition of a single function named main.

Several new ideas, including comments, declarations, variables, arithmetic expressions, loops , and formatted output are introduced

```
#include <stdio.h>
main()
{
        int fahr, celsius;
        int lower, upper;
        lower = 0;
        upper = 100;
        fahr = lower;
        while (fahr <= upper) {
                celsius = 5 * (fahr-32) / 9;
                printf("%df\t%dc\n", fahr, celsius);
                fahr = fahr + 1;
        }
}
```

Variables and arithmetic expressions

In C, **all variables must be declared before they are used**, usually at the beginning of the function before any executable statements.

A declaration announces the properties of variables; it consists of a type and a list of variables, such as

int fahr, celsius;

int lower, upper;

The type int means that the variables listed are integers; by contrast with float, which means floating point, i.e., numbers that may have a fractional part.

The range of both int and float is machine-dependant.

Computation in the temperature conversion program begins with the **assignment** statements which set the variables to their initial values.

Individual statements are terminated by semicolons

lower = 0; upper = 100;

Other data types

C provides several other data types besides int and float, including:

char character - a single byte
short short integer
long long integer
double double-precision floating point

The size of these objects is also machine-dependent. There are also *arrays*, *structures* and *unions* of these basic types, *pointers* to them, and *functions* that return them, all of which we will meet in the course.

Output


```
0f     -17c
1f     -17c
2f     -16c
3f     -16c
4f     -15c
5f     -15c
6f     -14c
7f     -13c
8f     -13c
9f     -12c
10f    -12c
.....................
96f    35c
97f    36c
98f    36c
99f    37c
100f   37c
```

Some problems

- The output isn't pretty because the numbers are not justified. We can augment each %d directive in the printf statement with a width, therefore the printed numbers printed will be right-justified. For example

    printf("%3d %6d\n", fahr, celsius);

So, the first number of each line will be printed in a field three digits wide, and the second in a field six digits wide

```
 0f   -17c
 8f   -13c
 9f   -12c
...............
10f   -12c
```

Output


```
0 f     -17.7778 c
1 f     -17.2222 c
2 f     -16.6667 c
3 f     -16.1111 c
4 f     -15.5556 c
5 f     -15.0000 c
6 f     -14.4444 c
7 f     -13.8889 c
8 f     -13.3333 c
9 f     -12.7778 c
10 f    -12.2222 c
…………………………..
96 f    35.5556 c
97 f    36.1111 c
98 f    36.6667 c
99 f    37.2222 c
100 f   37.7778 c
```

Another (more serious) problem is that because we have used integer arithmetic, the Celsius temperatures are not accurate; for instance, 0ºF is actually about -17.8ºC, not -17. We shall use floating-point arithmetic instead of integer in order to produce more accurate outputs. Here the ouput produced by a second version using floating arithmetic

Output

0 f	-17 c
1 f	-17 c
2 f	-16 c
3 f	-16 c
4 f	-15 c
5 f	-15 c
6 f	-14 c
7 f	-13 c
8 f	-13 c
9 f	-12 c
10 f	-12 c
…………………	
96 f	35 c
97 f	36 c
98 f	36 c
99 f	37 c
100 f	37 c

0f	-17.7778c
1f	-17.2222c
2f	-16.6667c
3f	-16.1111c
4f	-15.5556c
5f	-15.0000c
6f	-14.4444c
7f	-13.8889c
8f	-13.3333c
9f	-12.7778c
10f	-12.2222c
…………………………	
96f	35.5556c
97f	36.1111c
98f	36.6667c
99f	37.2222c
100f	37.7778c

```c
#include <stdio.h>

int main()
{
        float celsius, fahr;
        float lower, upper;
        lower = 0;
        upper = 100;
        fahr = lower;
        while (fahr <= upper) {
                celsius = 5 * (fahr - 32) / 9;
                printf("%3.0ff\t%6.4fc\n", fahr, celsius);
                fahr = fahr + 1;
        }
}
```

With comments

UNIVERSITÀ DEGLI STUDI DI PADOVA

```c
#include <stdio.h>

int main()
{
	/* print Fahrenheit-Celsius table
	for fahr = 0, 1, ..., 100 */
	float celsius, fahr;
	float lower, upper;
	lower = 0;
	upper = 100;
	fahr = lower;
	while (fahr <= upper) {
		celsius = 5 * (fahr - 32) / 9;
		printf("%3.0ff\t%6.4fc\n", fahr, celsius);
		fahr = fahr + 1;
	}
}
```

Arithmetic operations

- If an arithmetic operator has integer operands, an integer operation is performed.
- If an arithmetic operator has one floating-point operand and one integer operand, the integer will be converted to floating point before the operation is done.

Writing floating-point constants with explicit decimal points emphasizes their floating-point nature for human readers.
The assignment ***fahr = lower;*** and the test ***while (fahr <= upper)*** work in the natural way - the int is converted to float before the operation is done.

Printf conversion specifications

The printf conversion specification %3.0f says that a floating-point number (here fahr) is to be printed at least 3 characters wide, with no decimal point and no fraction digits. %6.4f describes another number (celsius) that is to be printed at least 6 characters wide, with 4 digits after the decimal point. The output looks like this:

```
 96f   35.5556c
 97f   36.1111c
 98f   36.6667c
 99f   37.2222c
100f   37.7778c
```

Width and precision may be omitted from a specification.

Printf conversion specifications

SPECIFICATION	RESULT
%d	print as decimal integer
%4d	print as decimal integer, at least 4 characters wide
%f	print as floating point
%6f	print as floating point, at least 6 characters wide
%.4f	print as floating point, 2 characters after decimal point
%6.4f	print as floating point, at least 6 wide and 2 after decimal point

printf also recognizes %o for octal, %x for hexadecimal, %c for character, %s for character string and %% for itself.

Exercises

- *Modify the temperature conversion program to print a heading above the table.*
- *Write a program to print the corresponding Celsius to Fahrenheit table*.

Loop

Each line of the temperature table is computed in the same way, so we use a loop that repeats once per output line; this is the purpose of the while loop

```
while (fahr <= upper) {
...
}
```

The while loop operates as follows: The condition in parentheses is tested. If it is true, the body of the loop is executed. Then the condition is re-tested, and if true, the body is executed again. When the test becomes false the loop ends, and execution continues at the statement that follows the loop.

The body of a while loop may be one or more statements enclosed in braces, or a single statement without braces.

Loop

A cycle executes a block of instructions, called a cycle block (composed of the instructions instructionLoop1, instructionLoop2, ..), 0, 1 or more times, depending on the value of a **cond** condition.

The program evaluates a condition. As long as the condition is true, the program executes the loop block and checks the condition again.

When the condition becomes false, the program ends the execution of the block.

A single execution of the loop block is called an **iteration**.

For example, when the loop block is executed three times, it is said that three iterations have been executed.

A function (or algorithm) containing a loop is said to be iterative.

Loop

There are three types of cycles:

- **Counter cycle**.
    - We need to use it when we know the number of iterations before executing the loop.
- **Cycle at initial condition**.
    - We must use it when:
        - we don't know the number of iterations, and
        - there is at least one input that the loop does not have to process even once.
- **Cycle with final condition**.
    - We must use it when:
        - we don't know the number of iterations, and
        - the loop must process each input to the problem at least once.

Loop

Of course, in the loop block we shall insert an instruction that shall make the condition cond false sooner or later.
If this instruction were not there, the condition would always be true and therefore the cycle would be repeated an infinite number of times.
There are two particular situations regarding the value of the cond condition:

1. The cond condition is false immediately, that is, when the program checks it for the first time. In this case, the block is executed 0 or 1 time, depending on the type of loop (as we will see later in the course).
2. The cond condition is always true. In this case, there is an **infinite** loop, because the program always executes the block and, therefore, will never execute instructions placed after the end of the block.

Loop

Sometimes we could purposely design an infinite loop. But, quite often, there is an infinite loop because we have made a logical error. An iterative function requires more complicated tracing than a generic function because it modifies the same variables multiple times. For this reason, we need to set up its trace table by considering the following rules:

- We need to insert a column into the table containing the value of the loop condition.
- We must insert a row into the table for each iteration, where we insert the values of the variables or expressions modified by the iteration.

We will see some examples later in the course.

The for statement

There are many different ways to write a program for a particular task. Let's try a variation about the temperature converter.

```
#include <stdio.h>
/* print Fahrenheit-Celsius table */
main()
{
        float fahr;
        for (fahr = 0; fahr <= 100; fahr = fahr + 1)
                printf("%3.0ff\t%6.4fc\n",fahr, (5.0/9.0)*(fahr-32));
}
```

This produces the same answers, but it certainly looks different

The for statement

The syntax of the for statement is

```
for (statement1; cond2; statement3) {
        /* This is where the for block begins. */
        instructionLoop1;
        instructionLoop2;
        ..
        } /* for */
```

cond2 is a condition. statement1, cond2 and statement3 are optional.

The effect of the for loop is:

The program executes statement1.

As long as cond2 is true, the program executes the loop block and then executes statement3. When the loop block has only one statement, we can eliminate its curly braces.

The for statement: some hints

Università degli Studi di Padova

Use the for loop as follows:

Use statement1 to initialize a counter i to an initial value;

Use cond2 to check if counter i is less than the final value;

Use statement3 to increment (or decrement) the counter i.

```
for (i = ini; i < end; i++) {
        /* The for block begins here. */
        instructionLoop1;
        instructionLoop2;
        ..
        } /* for */
```

The program loops for every i from ini to end - 1 inclusive.

The for statement: some hints

So, if ini < end, the program executes the loop end – ini times.
If, ini >= end, the program does not execute the loop even once.
The loop defined using the for statement is called a counter loop, because the variable i "counts" the iterations performed.
If we use the variable i only to count the iterations, it is better to initialize it with the value 0, for a reason that you will see when studying arrays.
So, to execute a loop n times, we should use the loop:

```
for (i = 0; i < n; i++) {
        ..
        } /* for */
```

The do – while loop

We have already encountered the while and for loops.

while (expression)
statement

the expression is evaluated. If it is non-zero, statement is executed and expression is reevaluated.

This cycle continues until expression becomes false, at which point execution resumes after statement.

for (expr1; expr2; expr3)
statement

the while and for loops test the termination condition at the top.

By contrast, the third loop in C, the do-while, tests at the bottom after making each pass through the loop body; the body is always executed at least once.

The do – while loop

The syntax of the do – while loop is

do

statement

while (expression);

The statement is executed, then expression is evaluated. If it is true, statement is executed again, and so on. When the expression becomes false, the loop terminates.

Experience shows that do-while is much less used than while and for. Nonetheless, from time to time it is valuable