

# UNIT 3

## INTRODUCTION TO PROGRAMMING IN C

Programming  
Grade in Industrial Technology Engineering  
2017-18  
Paula de Toledo. Maria Paz Sesmero. David Griol



Universidad  
Carlos III de Madrid  
[www.uc3m.es](http://www.uc3m.es)

### Contents

- Introduction to the C programming language
- Basic structure of a program
- Variables and constants
- Simple data types
- Strings and structured data types
- Expressions and instructions
- Operators
- Input/output: printf and scanf

## Algorithm, Program and programming language

- **Program**

- Set of orders (instructions) written in a given programming language that are given to a computer to solve a **problem** implementing an **algorithm**

### Algorithm

```
Ask for number 1
Get number 1
Ask for number 2
Get number 2
Result ← number 1 * number 2
Show result
```

### Program in C

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

int main(void)
{
    float num1, num2, result;
    printf("Insert first number: \n");
    scanf("%f", &num1);
    printf("Insert second number.: \n");
    scanf("%f", &num2);
    result=num1*num2;
    printf("result of the sum is %f \n", result);
    return 0;
}
```

C language

5

## History of C

- C is closely related to the development of the UNIX operating system at AT&T Bell Labs
- 1968-1971
  - First versions of UNIX
  - Unix users write programs in assembly language
  - Towards a more appropriate programming language: B, NB (1973)
- 1971-1972
  - C is created (K. Thompson)
  - UNIX is rewritten in C; versions of C are developed for other platforms (Honeywell 635, IBM 360/370)
- 1978
  - Kernighan and Ritchie
    - Publication of "The C programming language" book
  - Johnson
    - Development of pcc (portable C compiler)
- 1989
  - C becomes standard (ISO/IEC 9899-1990)
- New languages have been developed from C: Objective C, C++, C#, etc.

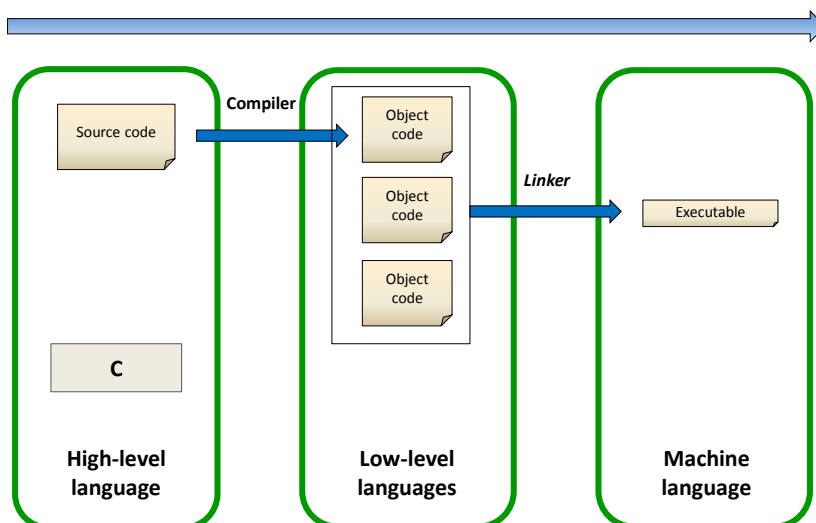
6

## ANSI C

- Different compilers, development platforms and language derivations may lead to C code targeted to a specific machine
  - E.g.: Win32 graphic libraries
- “Unambiguous and machine-independent definition of the language C”
  - A program in ANSI C must be compiled by any C compiler and must work in any platform
- ANSI C is a standard subset of the language:
  - Well-defined syntax
  - Restricted set of functions
- Several specifications: C89/C90, **C99**, C11

7

## Compilation + Linking process in C



8

## Programming language

- A programming language is defined by:
  - Alphabet
    - Allowed characters
  - Lexicon
    - Words
  - Syntax
    - Rules for word combination to make meaningful programs

## C alphabet

- Symbols used in C program
  - Letters
    - All but 'ñ' and accents
  - Numbers
  - Special characters (& , " , \ , ...)
- C is case sensitive: uppercase and lowercase letters are different

## C lexicon

- The lexicon includes the basic elements to build sentences
  - **Keywords**
    - Terms with a specific meaning for the compiler
      - Lowercase (include, define, main, if, etc.)
  - **Delimiters**
    - Blank spaces, tabs, line breaks
  - **Operators**
    - Represent operations: arithmetic, logic, assignment, etc. (+, -, \*, etc.)
  - **Identifiers**
    - Keywords cannot be used as identifiers
    - Variable names (user\_age) – cannot start with a number
    - Function names (printf, scanf)
  - **Literals**
    - Values that do not change:
    - Numbers: 2, 3.14159
    - Strings: "Hello world"
    - Characters: 'a'

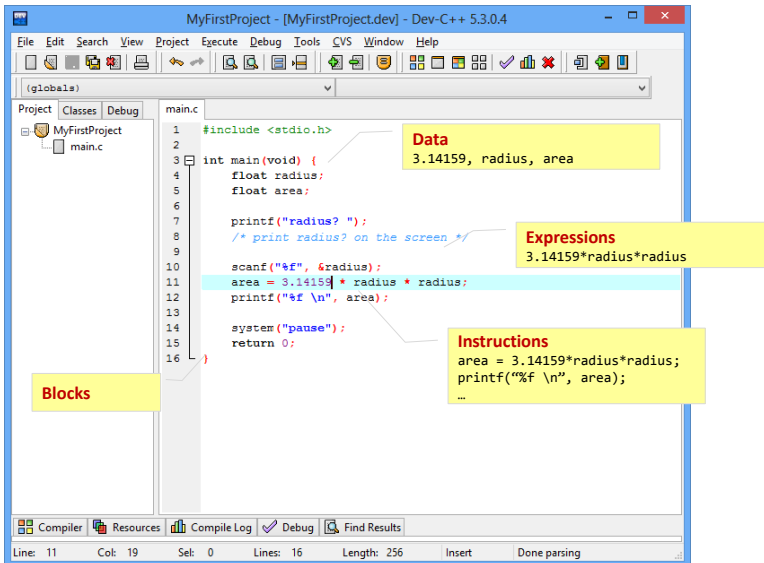
12

## Main components of a program (C or not)

- **Data**
  - Values processed by the program
  - *perimeter, radius, 9.8*
- **Expressions**
  - Combination of operands and operators with a single value as a result
    - `user_age >= 18`
    - `3.14159*radius*radius`
- **Statements/Instructions**
  - Complete *action*
    - `area=3.14159*radius*radius;`
    - `printf("Hello world");`
    - `int a;`
- **Blocks** or compound statements
  - Group of statements
  - Braces { }
    - The statements of the `main` function are enclosed in a block

13

## Components of a program



14

## 2. BASIC STRUCTURE OF A PROGRAM



## Basic C program structure

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

```
int main(void) {
```

```
...
```

```
return 0;
```

```
}
```

File inclusion  
(preprocessor directive)

Notice the parentheses ()  
and the braces {}

Main function:  
**void** = no input data  
**int** = output is int

**return 0;**  
Output is 0

Return is optional,  
but recommended

17

## Functions

Unit 6. Functions

- The basic building block in C is the **function**
  - A C program is a collection of functions
  - A function is a piece of code that performs a task when it is called/invoked
    - **Input values >> Output values**
- Functions include:
  - Data: information used by the function (variable)
  - Instructions: performing operations

18

## Main function

- All C programs have a main function
- Starting point of the program
  - Automatically started when the program is run

### main function structure

```
int main(void) {  
    ...  
    return 0;  
}
```

The simplest C program:

```
int main(void)  
{return 0;}
```

Valid, but useless

## Code reuse and functions – File inclusion

- “Real life” Programming is based in reusing code written by other programmers -- functions
  - C provides **libraries** with **functions** that we will use in our programs
  - *stdio.h* library implements input and output functions `printf()` and `scanf()`
- To use functions you need to add to your program the library file containing the function code
  - `#include "file.h"` searches for the file in the current folder
  - `#include <file.h>` searches for in the default compiler folder



## Pre-processor directives

- In C, the compiler calls an auxiliary program before starting the compilation process itself
- This program is the preprocessor, and does a first code translation
- It processes the pre-processor directives that start always with a # (hash symbol)
- Examples
  - `#include` to include function libraries
    - The preprocessor adds the functions code to our code
  - `#define` to define constant variables
    - the preprocessor substitutes the constant name by its value
- Not common in other programming languages, C specific

21

## Comments

- Comments are notes to the code that are meant for the programmer and not for the compiler
  - The compiler ignores comments (they are not real code)
  - They can be used at any point of the program
- Its very important to comment the code well:
  - Make the code readable and understandable
  - Although we now know perfectly what a program does, maybe we will have to reuse it in the future
  - Perhaps other programmers reuse our code and need to understand it
  - It is a good practice to introduce a comment at the beginning of each file describing what it does

22

## Syntax for comments

- Multi-line comments

- `/*` : Open comment block
- `*/` : Close comment block

```
/* print radius? on the screen */  
/* This program solves a  
   second grade equation. */
```

- In-line comments

- `//` : The remainder of the line is considered a comment

```
printf("%f \n", area);    // print area value
```

## 3. VARIABLES AND CONSTANTS

## Variables and constants

- A variable is a container for information processed by the program
  - Used to read, use in calculations, write
  - If the information can change, it's a variable; otherwise a constant

```
main.c
1  #include <stdio.h>
2
3  #define PI 3.14159
4
5  int main(void) {
6      float radius;
7      float area;
8
9      printf("radius? ");
10     /* print radius? on the screen */
11
12     scanf("%f", &radius);
13     area = PI * radius * radius;
14     printf("%f \n", area);
15
16     system("pause");
17     return 0;
18 }
```

25

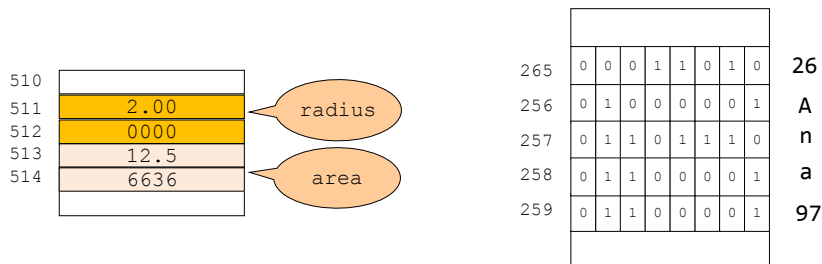
## Features of variables and constants: name, type, value

- **Variables and constants have:**
  - **Name**
    - Label or identifier of the variable or constant
      - radius, area, PI
  - **Type**
    - Determines which types of values the variable can take
      - Integer number, real number, single letter,...
  - **Value**
    - Value of the symbol at a given moment
      - 2, 12.566360

26

## Variables in memory

- Variables can be seen as a block of memory storing a piece of data
  - User-defined name for a group of cells of the memory
  - When the name (or identifier) of the variable is used in the program, the information at the address of the variable is accessed
  - The amount of memory allocated to the variable depends on its type, which must be set when the variable is declared



27

## Data types - intro

Type	Description	Size (bytes)	Range
int	Integer number	2 bytes	-32768 to 32767
float	Real number with simple precision (7 decimal values)	4 bytes	$3.4 \times 10^{-38}$ to $3.4 \times 10^{38}$
double	Real number with double precision (up to 16 decimal values)	8 bytes	$1.7 \times 10^{-308}$ to $1.7 \times 10^{308}$
char	Alphanumeric characters	1 byte	Unsigned: 0 to 255

28

## Variable declaration

- You need to declare variables before using them
  - The declaration instruction allocates a piece of the memory to store the value of the variable
  - In the declaration, we specify:
    - name of the variable
    - data type (int, char)
- Syntax
  - <data type> <variable name>;
- Examples

```
float average_mark;
int num1, sum;
char letter;
```

29

## Choosing good names for your variables

- Self-explanatory names
  - Lowercase
  - not too long
    - `counter = counter + 1;`
    - `num_registered_students = 56;`
  - Variables are declared at the beginning of the block in which they are used. They are valid only in this block (scope)!
- ```
int main(void) {
    int counter;
    int radius;

    a = 10;
    printf("%i", a);
}
```

30

## Assigning a value to a variable

- Assigning a value to a variable means that the value is stored in the memory cell allocated to the variable
  - `a=7;`
- If there was a previous value stored on the variable, it is deleted
- The assignment operator is =
  - A variable can be assigned an individual value or the result of evaluating an expression
    - `a=3;`
    - `x = y;`
    - `delta=0.001;`
    - `suma=a+b;`
  - In pseudocode the assignment is represented as  $\leftarrow$ 
    - *The variable on the left is assigned the value on the right*

31

## Variable initialization

- Initialize means assigning an initial value to a variable
- In the declaration:
  - `int a=8;`
- After the declaration:
  - `int a;`
  - `a = 8;`
- Multiple declaration/initialization in one line
  - `int a, b, c;`
  - `int a=5, b=4, c=8;`
  - `int a=1, b, c=a;`
- Uninitialized variables have junk values
  - We cannot assume that they are 0!!

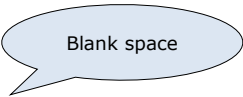
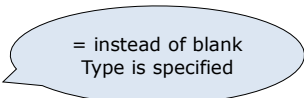
32

## Reading and printing variable values

- Reading (input):
  - float radius;
  - **scanf** ("%f", &radius); // gets a value and stores it in radius variable
- Printing (output):
  - int radius =23;
  - **printf** ("%i", radius); // prints value onto the screen

33

## Constants

- A **C constant** is a symbol whose value **is set at the beginning of the program and does not change later**
- Two alternatives:
  - #define directive
    - #define <name> <value>
      - #define PI 3.14159
      - #define KEY 'a'
      - #define MESSAGE "Press INTRO to continue..."
  - const qualifier to a variable
    - const <type> <name> = <value>;
      - const float PI = 3.14159;
      - const char KEY = 'a';
      - const char MESSAGE [] = "Press INTRO to continue...";
- Constant identifiers are usually written in **uppercase letters**

34

## Constant declaration with `const` vs `#define`

- **#define** is a preprocessor directive. Before compiling the code the preprocessor substitutes the constant by the value
- Advantages of **const** versus **#define**
  - The compiler generates more efficient code
  - The compiler can check if the type and the assigned value are compatible
- Advantages of **#define** versus **const**
  - `const` values cannot be used in places where the compiler expects a literal value (e.g., array definition)

37

## Literals

- Literals are values in our code
  - Constants are literals, but specific values too
- 
- `n1=5;` `//literal integer 5`
  - `r1=5.7;` `// literal float 5.7`
  - `myIntial='n';` `// literal character n`
  - `"wonderful day"` `//literal string of characters`

38



## 4. BASIC DATA TYPES IN C

### Basic data types in C

| Type   | Description                                                    | Size (bytes) | Range                                           |
|--------|----------------------------------------------------------------|--------------|-------------------------------------------------|
| int    | Integer number                                                 | 4 bytes      | [-2147483648, 2147483647]                       |
| float  | Real number with simple precision<br>(up to 7 decimal values)  | 4 bytes      | $3.4 \times 10^{-38}$ to $3.4 \times 10^{38}$   |
| double | Real number with double precision<br>(up to 16 decimal values) | 8 bytes      | $1.7 \times 10^{-308}$ to $1.7 \times 10^{308}$ |
| char   | Alphanumeric characters                                        | 1 byte       | 0 to 255                                        |

- Size in bytes may be different in different operating systems and platforms
- Other simple data types
  - void
  - Pointers
- Modifiers
  - int: signed, unsigned
  - int: long, short

## Integer datatype

- **int datatype** is used to represent integer values
  - int literals
  - int variables
  - int expressions
- int literals can be expressed with different notations
  - Decimal (base 10): **2013**
  - Octal (base 8): **011** (starting with 0)
  - Hexadecimal (base 16): **0x2B** (starting with 0x)
- 011      // 1\*8+1\*1 --> 9
- 0x2B     // 2\*16+11 --> 43

## float and double

- **float** and **double** data types are used to represent real values
  - double more precision, but also larger memory size
- The decimal separator for float and double literals is .
- Two options to represent a number
  - Regular: **82.3473**
  - Scientific notation: **2.4E-4**

## char type

- **char** data type is used to represent ASCII characters
- char literals are enclosed in single quotation marks ' '
- ```
char letter = 'b';
```
- ```
printf("%c", letter);
```
- Special and escape characters can be used
- ```
char lineBreak = '\n';
```

## Character coding system ASCII

Binario	Dec	Hex	Carácter	Binario	Dec	Hex	Carácter	Binario	Dec	Hex	Carácter
0010 0000	32	20	espacio	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	\$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

## Type matching when assigning values

- Assignments can must done between a variable and an expressions with same type or compatible types
  - `int <--- int`
  - `float <--- int` adds .0 to the int
  - `int <--- char` assigns the ASCII code of the char to the int
  - `char <--- int` assigns the ASCII value coded by the int to the char
  - `int <--- float` the decimal part of the float is truncated

```
int a=5, b;  
char c='Z';  
float x, y=3.1;
```

```
b=a;  
x=a;  
b=c;  
c=a;  
b=y;
```

46

## Void data type

- void data type is used to indicate that no value is expected in specific parts of the program
  - A function has no parameters
    - `int main(void)`
    - is equivalent to
    - `int main()`
  - A function does not return any value
    - `void main(void)`
- You cant declare a variable to be of void datatype

47

## 5. STRINGS AND STRUCTURED DATATYPES

### Structured data types

- Data can be structured or unstructured
  - **Unstructured (simple) data types**
    - Symbols with a single element and a single value
      - *Numbers*: integer , float
      - *Characters*: char
  - **Structured data types**
    - Symbols with an internal structure, not a single element
      - Character strings
      - Arrays and matrices
      - Structures

Unit 5. Structured data  
types

## Character strings

- Character **strings** are used to represent a sequence of characters
- String literals are enclosed in double quotation marks " "
- String variables and constants are declared as arrays (unit 6):
  - `char message [] = "Hello world";` // string constant
  - `char name[100];` // string variable of 100 characters

51

## Storing strings in memory

- Strings are stored in memory as a strip of characters ending with the null character `'\0'`
- `char myString [8]="Hola";`

0	1	0	0	1	0	0	0	'H'	72	} 8 bytes
0	1	1	0	1	1	1	1	'o'	111	
0	1	1	0	1	1	0	0	'l'	108	
0	1	1	0	0	0	0	1	'a'	97	
0	0	0	0	0	0	0	0	'\0'	0	
--	--	--	--	--	--	--	--			
--	--	--	--	--	--	--	--			
--	--	--	--	--	--	--	--			

52

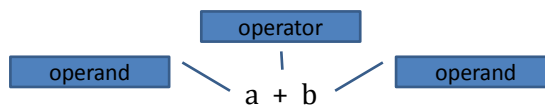
## Using special characters in strings

- A C string can contain any ASCII character
  - Including numbers, and blank spaces
- Some characters that have a specific meaning for C need to be “marked” by using an escape sequence
  - This tells the compiler that the character coming next has to be dealt with in a special way
  - In C the escape sequence is the \
  - Example: “
    - With no escape sequence the compiler would interpret this is the end of the string
    - Use \” instead
    - Char greeting [] = “He said \”Hello \”””

## 6. EXPRESSIONS AND INSTRUCTIONS

## Expression

- An **expression** is a combination of data by means of one or several **operators**
- Data can be literal values, variables, constants, and other expressions
  - Even calls to functions
- Data in an expression are called operands



- Expression composition is guided by rules
  - Each operation can only operate on operands of a given data type

56

## Types of operators

- Arithmetic : i.e. sum +
  - 7+3+4.5;
  - num1+num2;
- Relational: : i.e. "greater than" >
  - 7>3;
  - num1>num2;
- Logical: i.e. and
  - (pin =pin\_ok) and (attempts < 3)

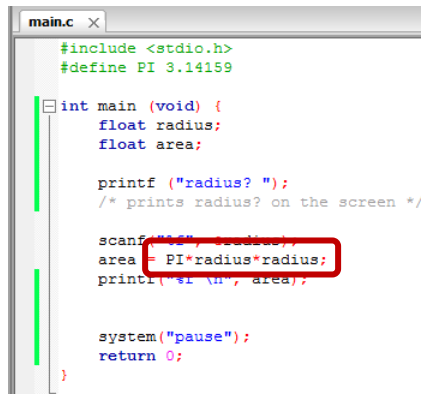
57



## Expression: examples

- Examples

- $a + b$
- $x == y$
- $x <= y$



```
main.c x
#include <stdio.h>
#define PI 3.14159

int main (void) {
    float radius;
    float area;

    printf ("radius? ");
    /* prints radius? on the screen */

    scanf ("%f", &radius);
    area = PI*radius*radius;
    printf ("%f\n", area);

    system("pause");
    return 0;
}
```

58

## Instructions or statements

- Orders of the program to accomplish a task
  - applied on operators and expressions
- Types
  - According to the function
    - Declaration  
int a;
    - Assignment  
a = 5;
    - Input and output  
printf, scanf
    - Control  
if, while,

59

## Instruction examples

```
#include <stdio.h>
# define PI 3.14
int main (void)
{
    float radius;
    printf ( "radius=?");
    scanf ("%f",&radius);
    area = PI* radius * radius;
    printf ("%f", area );
    return 0;
}
```

Variable declaration

Input

Assign result of the  
expression

Output value

60

## 7. OPERATORS

## Arithmetic operators

Operator	Operation
+	Addition
-	Substraction
*	Multiplication
/	Division
%	Remainder or Module

- Operands are numbers (int, float, double) and the result is a number
- Datatype of the result depends on datatype of operands
- The % operator requires two integer operands, being the second one different to 0
- The / requires the second operand to be different to 0.
  - When both operands are integers, the result is also an integer value (no decimals!)

62

## Datatype of the result : integer division

```
#include <stdio.h>

int main (void){
    int n1=7, n2=2, n3;
    float r1=7, r2=2, r3;

    printf ("integer divided by integer, assigned to integer \t");
    n3=n1/n2;
    printf ("%i \n", n3);

    printf ("float divided by float, assigned to float \t");
    r3=r1/r2;
    printf ("%f \n", r3);

    printf ("float divided by integer , assigned to integer \t");
    n3=r1/n2;
    printf ("%i \n", n3);

    printf ("float divided by integer , assigned to float \t");
    r3=r1/n2;
    printf ("%f \n", r3);

    printf ("integer divided by integer , assigned to float \t");
    r3=n1/n2;
    printf ("%f \n", r3);

    system ("PAUSE");
    return 0;
}
```

```
integer divided by integer, assigned to integer      3
float divided by float, assigned to float          3.500000
float divided by integer , assigned to integer      3
float divided by integer , assigned to float        3.500000
integer divided by integer , assigned to float      3.000000
```

63

## Power and square root

- There is no C operator for power and square root
- Library functions are used
  - pow y sqrt, in Library math.h

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

```
int main (void){
    int result;

    result=pow(2,3);
    printf ("2 to the power of 3 is %i \n", result);

    result=sqrt(16);
    printf ("16 squared is %i\n", result);

    system ("PAUSE");
    return 0;
}
```

```
2 elevado a 3 es 8
La raiz cuadrada de 16 es 4
Presione una tecla para continuar . . .
```

64

## Logical operators: AND OR NOT

- Operate on logic (boolean) operators or expressions
  - Logic operators: operators whose value can be 'true' or false'
- Truth tables

Negation is a unary operator – only one operand

conjunction : AND

Operands		T	F
T	T	T	F
F	F	F	F
		output	

disjunction : OR

	T	F
T	T	T
F	T	F

Negation : NOT

T	F
F	T

66

## Logical operators in C

C Operator	
and	&&
or	
not	!

In C – false is 0 and true is anything else – typically 1  
In other languages there may be specific boolean types

### Examples:

To pass the lecture, exam **and** lab parts must be passed

Pass = Pass\_Exer AND Pass\_Lab

To pass the lecture, **at least one** of the parts needs to be passed

Pass = Pass\_Exer OR Pass\_Lab

&& : AND

	T	F
T	T	F
F	F	F

|| : OR

	T	F
T	T	T
F	T	F

! : NOT

T	F
F	T

67

## Practice with logical expressions

- Write an expression that is true in this case
  - access to disco
    - Older than 18 and has 20 euros for the ticket
  - access to disco – 80's version
    - Older than 18 and has 20 euros for the ticket or is a girl
  - access to disco
    - Anyone can enter provided that number of people smaller than maximum capacity
  - Block the cell phone
    - Entered pin not correct and number of attempts 3 or more
  - Three numbers for an ascending sequence
    - num1 > num2 > num3
  - Pass the course
    - Exam is passed (mark >5) or mean exam and continuous assessment is greater than 5, and student had not been expelled

68

## Practice with logical expressions(II)

- Write an expression that is false if
  - Give a parking fine
    - Payment issued and within time limit or has special residents parking permit
  - Leave the country
    - valid passport (not expired) and no arrest warrant (search order by the police)

## Relational operators

- The result of relational operators is a boolean value
  - false: 0, true 1

Operator	Operation
<	Less than
<=	Less or equal than
>	Larger than
>=	Larger or equal than
==	Equals
!=	Different from

## Precedence rules for operators

- Conventions about which procedures to perform first in order to evaluate a given expression
- $a + b > c \ || \ c < 0$
- Precedence rules are similar in all programming languages
- Where it is desired to override the precedence conventions, or just for clarity, **parenthesis** are used
  - Expressions enclosed with parenthesis are evaluated first, from the inner-most to the outer-most
    - $( (a + b) > c ) \ || \ (c < 0)$

73

## Operator precedence order in C

Unary	!	NOT (negación lógica)	!a
	++	Increment	++a
	--	Decrement	--a
	-	Sign change	-b
	*	Indirection	*p
Multiplication	&	Address	&a
	*	Multiplication	a*b
	/	Division	a/b
Addition	%	Module	a%b
	+	Addition	a+b
Relational	-	Subtraction	a-b
	<	Less than	a<b
	<=	Less or equal than	a<=b
	>	Larger than	a>b
Equality	>=	Larger or equal than	a>=b
	==	Equals to	a == b
Logic	!=	Different to	a != b
	&&	AND	a && b
		OR	a    b
Assignment	=	assignment	a = b

- Expressions with operators of the same category are evaluated from left to right

74

## 8. INPUT AND OUTPUT: PRINTF AND SCANF

### Input and output instructions

- Programs interact with the user through input and output data
  - Standard input : keyboard
  - Standard output : screen
  - Other options: read from and print to file
- Input and output (I/O) instructions allow programs to read and display data
- C does not have input/output instructions
  - Library functions printf and scanf instead
  - include the library where printf and scanf are declared
    - #include <stdio.h>



## printf: print formatted

- Sends formatted output to the standard output stream (screen)
- Syntax
  - `printf("format string", arguments)`
    - The format string has two components: text and identifiers
    - The number of identifiers has to match the number of arguments to print

```
printf("Hello!, %s\tYou are %i, you are %f mts. tall\n", name, age, height);
```

```
printf("Your surname starts with a %c. ", initial);  
printf("Bye\n");
```

Format string

List of argument

```
Hello!,Sara  You are 21, you are 1.680000 mts. tall  
Your surname starts with a P. Bye
```

78

## Conversion specifiers (simplified version)

- Set the format to be used for printing data

Specifier	Format
<b>%i</b>	Integer (int)
<b>%f</b>	real (float), decimal notation
<b>%c</b>	character
<b>%s</b>	character string
<b>%d</b>	Integer, decimal format – same as %i

```
printf("number: %i \n", 2013);    // 2013  
printf("number: %i \n", -2013);  // -2013  
printf("number: %f \n", 82.3473); // 82.34730  
printf("number: %f \n", 2.4E-4);  // 0.000240
```

79

## Format modifiers

`%[flags][width][.precision]<type>`

- Used to modify how data are displayed
- Flags
  - - : left-aligns the output
- Field Width
  - Minimum size in characters of the output (pads if necessary)
  - Default width depends on the datatype
  - If this size is not big enough, this setting is overridden (ignored)
- Precision
  - In a string, maximum number of characters to plot
  - In float or double: decimals used to represent the number (rounds if necessary)

80

## Format modifier examples

`%[flags][width][.precision]<type>`

- `%5.2f`
  - Float using at least 5 spaces and two decimals
- `%-5.2i`
  - Integer, aligned to the left, using 5 spaces and 2 figures

```
printf ("%5.2f", 3.0);
```

	3	.	0	0
--	---	---	---	---

```
printf ("%5.2i", 3);
```

0	3			
---	---	--	--	--

81

## How to print special characters

- ñ
  - `printf ("Feliz a%co ",164);`
    - `%c` format descriptor for char
    - `ñ` = ascii code 164
- Characters that need escape sequence
  - `\'` single quotes `printf("\'");`
  - `\"` double quotes `printf("\'");`
  - `\\` backslash `printf("\\");`
- Special printable characters
  - `\n` end of line
  - `\t` tabulator
  - `\b` backspace

82

## Read data with format: scanf

- `scanf()` Reads information from the standard input channel
- Syntax
  - `scanf("format string", arguments)`
    - `scanf ("%i", &age)`
      - The argument is the address of the variable,
      - `&` indicates address
  - More than one variable can be read in the same `scanf` instruction

```
int n;
float mark;
```

```
printf ( "Enter student id and mark:\n");
scanf ("%i %f", & n, & mark);
printf ("\n The mark of the student %i is %f\n", n, mark);
```

83

## Reading and printing strings

- `scanf("%s", string_variable);`
    - Reads chars from the keyboard to form a string
    - NO NEED TO ADD **&**
      - A string is a type of array
      - In an array (unit 6) the name of the array contains the address of the variable
- ```
char name[100];  
scanf("%s", name);
```
- `printf("%s", string_variable);`

84

## Reading strings containing blank spaces

- `scanf "%s" stops reading when it finds a blank space`
  - `scanf("%s", name);`
  - Miguel de Cervantes
  - `printf("Hello %s", name);`
  - Hello Miguel
- To read a string including blank spaces we use a special format specifier
  - `scanf ("%^[\\n]", name);`
  - `%^[\\n]` means that `scanf` reads until a line break character is found

85

## Reading characters with scanf

- Try this code

```
int main(void)
{
    char c;

    printf("Input No.1\n");
    scanf("%c", &c);
    printf("c = %c\n", c);

    printf("Input No.2\n");
    scanf("%c", &c);
    printf("c = %c\n", c);

    printf("Input No.3\n");
    scanf("%c", &c);
    printf("c = %c\n", c);

    return 0;
}
```

- %i and %f take any input from the input channel (buffer) and empty the buffer
- Conversely %c takes one char but doesn't empty the buffer
- The "enter" char is left in the buffer
- Next scan instruction will read the 'enter' left over in the buffer
- What you will notice is that the second scan instruction is skipped
- The third scan will take the second character

86

## Reading chars with scanf

- Integers vs chars in scanf
  - When reading a number , everything in the buffer that can't be converted to a number is disregarded (blank spaces,...)
  - When chars are read, anything will be formatted as a char
- How to avoid this
  - Use format descriptor " %c"
    - NOTICE **leading blank space**
  - This changes the format so that blank spaces and enter are skipped

87

## Conversion specifiers (detailed list)

| Specifier     | Format                                                 |
|---------------|--------------------------------------------------------|
| <b>%i</b>     | Integer (int)                                          |
| <b>%f</b>     | real (float), decimal notation                         |
| <b>%e</b>     | real (float), scientific notation                      |
| <b>%lf</b>    | Long float (double)                                    |
| <b>%c</b>     | character                                              |
| <b>%s</b>     | character string                                       |
| <b>%d</b>     | Integer, decimal format – same as %i                   |
| <b>%o</b>     | Integer, octal                                         |
| <b>%x</b>     | Integer, hexadecimal                                   |
| <b>%p</b>     | pointer ( <b>memory address</b> stored in the pointer) |
| <b>%[^\n]</b> | Only for scanf. String including blanks                |
| <b>" %c"</b>  | Only for scanf. Characters excluding blanks and enter  |

# UNIT 3

## INTRODUCTION TO PROGRAMMING IN C

Programming  
Grade in Industrial Technology Engineering  
2017-18  
Paula de Toledo. Maria Paz Sesmero. David Griol



Universidad  
Carlos III de Madrid  
[www.uc3m.es](http://www.uc3m.es)