

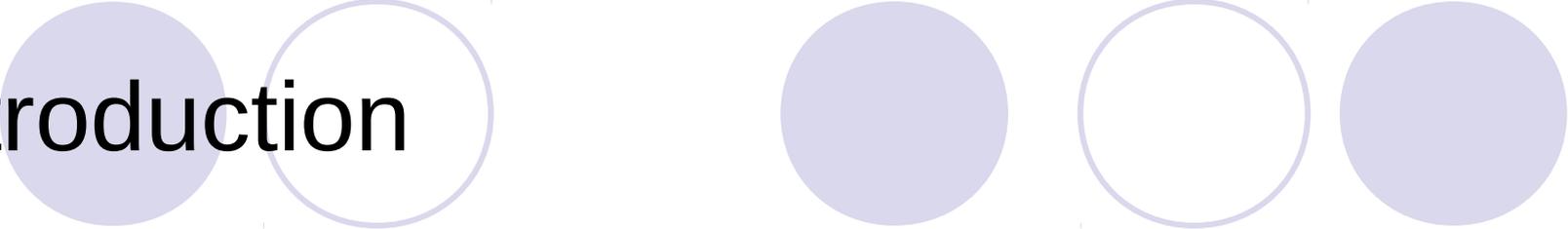
Informática
*Ingeniería en Electrónica y Automática
Industrial*

Introduction to C programming language

Introduction to C programming language

- Introduction
- Main features
- Functions
- Variables
- Compilation
- Libraries and linkage
- Examples

Introduction



- Developed in the 70's by Dennis Ritchie for a PDP-11 computer running with Unix
- Although developed in Unix is not linked to any OS and works with all of them
- For long the standard version, developed by Rirtchie and Kernighan, was delivered with version 5 of Unix
- Proliferation of different versions forced the creation of a standard one: **ANSI C** (American National Standard Institute)

C Language main Features (I)

- It was very succesful since its creation:
 - Compact
 - Structured
 - Portable
 - Flexible
 - Medium tipe
 - Very popular

C Language main Features (II)

COMPACT

- Just 32 reserved words in ANSI standard:

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

- It allows all algebraic and logic operations in conventional maths
- Any program could be written just with the reserved words and operators (difficult though).

C Language main Features (III)

STRUCTURED

- The structural basic component is the **function**
- Does not allow to write functions inside other functions
- It allows independent code parts with their own data: functions that can be used by other programs
- It allows *code blocks*: statements and propositions grouped inside brackets «{ }» forming a logical unit
- Different conditional and iteration sentences
- `goto` exists but strongly NOT recommended.

C Language main Features (IV)

PORTABLE

- Executables are independent from hardware if standard libraries are used, ie,
- Same source code can be compiled in different architectures
- C is relatively simple
- There are compilers for all systems

FLEXIBLE

- Created and tested by professional programmers, so it has few restrictions and gives the programmer a lot of control
 - Advantage for advanced programmers, disadvantage for begginers...
- It allows different data types, conversions among them and creation of new types.

C Language main Features (V)

MEDIUM TYPE

- It combines elements of high level languages with elements of low level ones:
 - Powerful sentences (high level)
 - Bit-level operations, register, ports and memory managment (low level)

VERY POPULAR

- C compilers are relatively simple, so they are the first to be developed when a new system is launched
- Very used among professional and amateur programmers
- Very used to program OS, interpreters, compilers, assemblers, drivers....
- Powerful extensions: C++

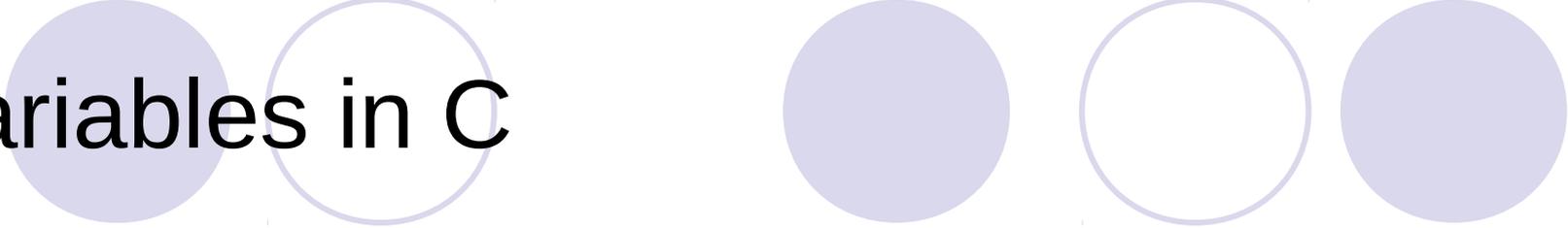
Functions in C

- **Functions** are the primary programming objects in C: it is where the program activity occurs
- Each function contains an independent and portable code block
- Generic form:

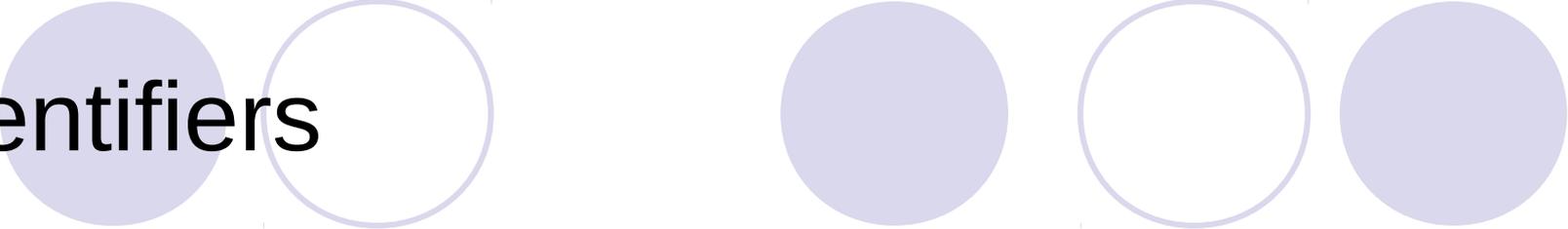
```
return-type function-name(Parameter list)
{
    declarations
    ...
    statements
}
```

/* Function beginning*/
/* Función body */
/* Funtion end */

Variables in C



- **Variables** in C are *memory parts with a name*
- Are used to store values that can be modified by the program
- They must be *declared* before use
 - Declaration sets the data type it will contain
- C supports all basic variable types (character, int, float, etc.) and allows to:
 - Modify defined types
 - Create new types



Identifiers

- **Identifiers** are the names to identify
 - Variables
 - Constants
 - Functions
- Features:
 - Must start with alphabetic character and can contain alphanumeric characters and underscore «_»
 - Reserved words are forbidden
 - Upper case and lower case letters are DifFEreNT
- Recommendations
 - Functions created by the programmer starts with upper case
 - Identifiers of defined or symbolic constants are written with upper case.

Statements in C

- Situated in any position in the line (no *fields* established as columns)
- Always end with «;»
- Indent is optional but recommended
 - Hierarchical
 - Facilitates understanding
 - Ignored by compiler
- Statesmen blocks
 - Statement groups between brackets «{ }» forming a unit
- Comments
 - Recommended but optional explanatory text
 - Starts with «//» until end of line (no ANSI standard)
 - Between «/* */» symbols in any number of lines
- Preprocessor directives
 - Special orders for the compiler that are not part of the C language (but included in all compilers)
 - Always start with «#»
 - Facilitate programming
 - Examples: #include, #define

Libraries and Linkage

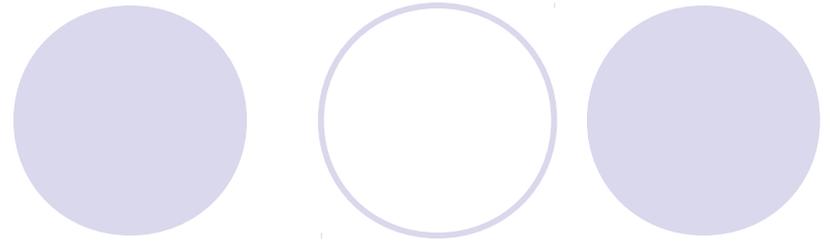
- Along with the compiler, function *libraries* containing basic function are necessary:
 - They can be used in any statement
 - ANSI specifies a minimal set of functions: the *standard library*
 - Compilers usually include many more
 - User can create its own function libraries
- The **linker** merges user code with necessary libraries to create the executable machine code.
- Examples: `stdio.h` (input/output, `printf()`), `math.h` (mathematical functions, `sin(x)`)



C program full development

- Steps:
 - Algorithm design
 - Program creation and writing in a text file
 - Compilation to obtain the *object file*
 - Linkage of the object file with the called libraries to obtain the final machine code *executable*
- For big developments the program is divided in many files that along with libraries form a *project*. Each part can be compiled and tested separately and linked with the rest at the final stage to produce the total executable program (i.e, any OS like Linux or Windows).

Examples (I)

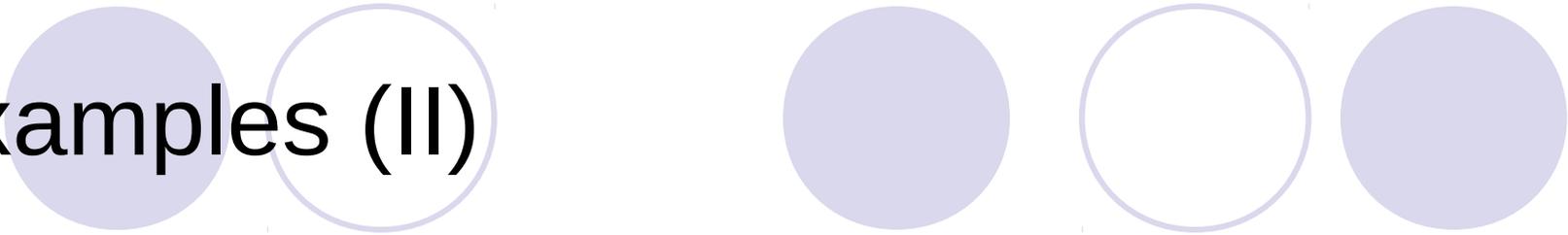


- The simplest program
- Notice
 - Preprocessor directive
 - Main function
 - Comment
 - Function call
 - Character chain

```
#include <stdio.h>

main()          /*main function */
{
    printf("hello, world \n");
}
```

Examples (II)



- Program to convert a Fahrenheit temperature to Celsius
- Notice
 - Variable declaration
 - Assignment statements and arithmetic operations
 - Data types
 - Comments
 - `Printf()` function
 - Return

Examples (III)

```
/* Fahrenheit to celsius conversion */

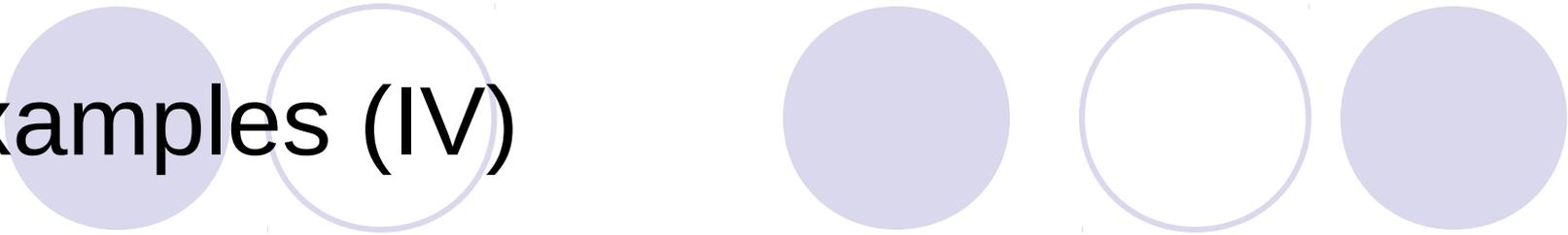
#include <stdio.h>

main()
{
int fahrenheit, celsius;    /* Entire variables */

    printf("Conversion °F to °C:\n");

    fahrenheit = 100;        /* Fahrenheit
temperature */
    celsius = 5*(fahrenheit-32)/9;    /* Conversion formula */
    printf("%d °F = %d °C\n", fahrenheit, celsius);    /* Result*/
    return 0;
}
```

Examples (IV)



- Program to convert *any* Fahrenheit temperature to Celsius
- Notice
 - Reading input from keyboard
 - Real numbers declaration
 - Arithmetical operations with real numbers

Examples (V)

```
/* Fahrenheit-Celsius conversion with input introduced by the user
   and with real numbers. */

#include <stdio.h>

main()
{
float fahren, celsius; /* Real variables */

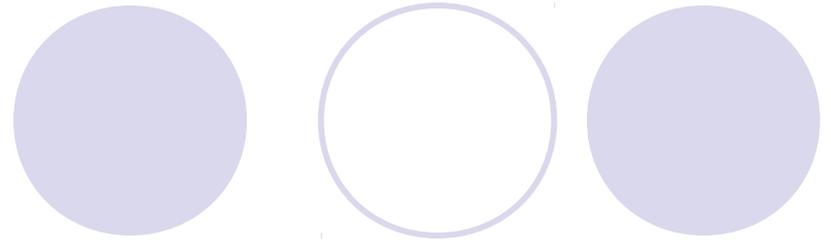
    printf("°F to °C conversion:\n");
    printf("Introduce Fahrenheit temperature: ");

    scanf("%f", &fahren); /* real data input */

    celsius = (5.0/9.0)*(fahren-32); /* Formula */
    printf("%f °F = %f °C\n",fahren, celsius); /* Result*/

    return 0;
}
```

Examples (VI)



- Program to print a table of Fahrenheit temperatures and their Celsius equivalent using “for” loop
- Notice:
 - Different data types
 - Indent
 - for sentence
 - Formats in `printf()`

Examples (VII)

```
/* Conversion table Fahrenheit-Celsius. */
/* With "for" loop */

#include <stdio.h>

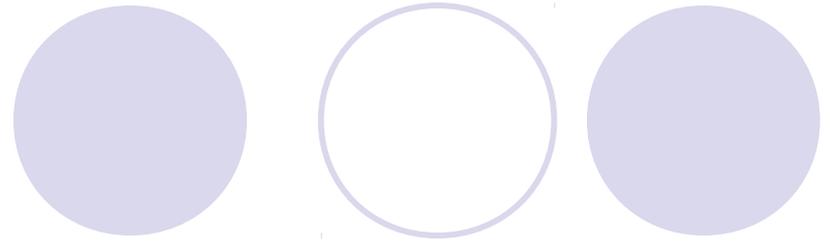
main()
{
float fahren, celsius;           /* Variables */
int liminfe, limsup, increm;

    liminfe = 0;                /* lower limit */
    limsup = 100;               /* Upper limit */
    increm = 10;                /* Step size */

    printf(" °F\t   °C\n");     /* Table header*/
    printf("=====\n");

    for (fahren=liminfe ; fahren<=limsup ; fahren=fahren+increm)
    {
        celsius = (5.0/9.0)*(fahren-32.0);
        printf("%3.0f\t%6.1f\n",fahren, celsius);
    }
    return 0;
}
```

Examples (VIII)



- Program to print a table of Fahrenheit temperatures and their Celsius equivalent using “while” loop
- Notice:
 - #define symbolic constants
 - System() call
 - while sentence
 - Comparison «<=» (lower or equal than)

Examples (IX)

```
/* Conversion table Fahrenheit-Celsius. */
/* With "while" loop and symbolic constants*/

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

#define LIMINFE    0          /* Lower limit */
#define LIMSUP    100       /* Upper limit */
#define INCREM    10        /* Step size */

main()
{
    float fahren, celsius;          /* Variables */
    fahren = LIMINFE;              /* Lower limit */
    system("clear");               /* Clear terminal (Linux) */
    printf(" °F\t    °C\n");       /* Table header */
    printf("=====\n");
    while (fahren <= LIMSUP)
    {
        celsius = (5.0/9.0)*(fahren-32.0);
        printf("%3.0f\t%6.1f\n", fahren, celsius);
        fahren = fahren + INCREM;
    }
    return 0;
}
```