

# Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informàtica de Sistemes y Computadoras (DISCA)

*Universitat Politècnica de València*

## Bloque Temático 1: Introducción

### Seminario 1

### Lenguaje C

fSO

DISCA



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

- **Objetivos:**

- Introducir el **lenguaje de programación C** remarcando sus diferencias con Java
- Entender el proceso de compilación y enlazado
- Presentar el concepto de **punteros en C**
- Manejar **las llamadas a funciones** y el paso de **parámetros por referencia y valor**

- **Bibliografía:**

- C reference card: la “chuleta”.
- “El Lenguaje de Programación C”, B.W. Kernighan y D.M. Ritchie, Prentice-Hall Hispanoamericana (1991).
- “C++ Estándar”, Enrique Hernández et al. Paraninfo, 2002.
- “Lenguaje C”. 2006, Moldes, F. Javier

- **Introducción**
- Proceso de Compilación y Enlazado
- Elementos de un Programa C
- Sentencias de Control de Flujo
- Tipos de Datos Derivados
- Funciones
- Preprocesador y Librerías

- **Características del Lenguaje C**

- Lenguaje de programación **de propósito general**, muy adecuado para programación de sistemas y sistemas empujados (UNIX fue escrito en C).
- Lenguaje relativamente **pequeño**: solo ofrece sentencias de control sencillas y funciones.
- Muy **portable**: existen compiladores para todos los procesadores
- **Código** generado **muy eficiente**: tanto en velocidad como en tamaño.

- **Lenguaje compilado**

- Compilar: a partir de código fuente (texto) se genera el ejecutable.
  - Java es interpretado (se necesita una máquina virtual para ejecutar el programa)
- Si el ejecutable no es para la misma máquina se denomina compilación cruzada

- Historia del C
  - C fue diseñado originalmente en 1972 para el SO UNIX en el DEC PDP-11 por Dennis Ritchie en los Laboratorios Bell.
  - En los 80, gran parte de la programación se realiza en C.
  - En 1983 aparece C++ (orientado a objetos).
  - El lenguaje C está estandarizado:
    - ANSI C, ISO C.
- **Java y C**
  - Java está basado en C/C++
  - Las estructuras de control son iguales a C
  - Las clases son similares a C++
  - **Java elimina la gestión de memoria directa (punteros)**

## C Reference Card (ANSI)

### Program Structure/Functions

<code>type fnc(type<sub>1</sub>,...)</code>	function declarations
<code>type name</code>	external variable declarations
<code>main()</code>	main routine
<code>declarations</code>	local variable declarations
<code>statements</code>	
<code>}</code>	
<code>type fnc(arg<sub>1</sub>,...) {</code>	function definition
<code>declarations</code>	local variable declarations
<code>statements</code>	
<code>return value;</code>	
<code>}</code>	
<code>/* */</code>	comments
<code>main(int argc, char *argv[])</code>	main with args
<code>exit(arg)</code>	terminate execution

### C Preprocessor

include library file	<code>#include &lt;filename&gt;</code>
include user file	<code>#include "filename"</code>
replacement text	<code>#define name text</code>
replacement macro	<code>#define name(var) text</code>
Example. <code>#define max(A,B) ((A)&gt;(B) ? (A) : (B))</code>	
undefine	<code>#undef name</code>
quoted string in replace	<code>##</code>
concatenate args and rescan	<code>##</code>
conditional execution	<code>#if, #else, #elif, #endif</code>
is <code>name</code> defined, not defined?	<code>#ifdef, #ifndef</code>
<code>name</code> defined?	<code>defined(name)</code>
line continuation char	<code>\</code>

### Data Types/Declarations

character (1 byte)	<code>char</code>
integer	<code>int</code>
float (single precision)	<code>float</code>
float (double precision)	<code>double</code>
short (16 bit integer)	<code>short</code>
long (32 bit integer)	<code>long</code>
positive and negative	<code>signed</code>
only positive	<code>unsigned</code>
pointer to int, float,...	<code>*int, *float,...</code>
enumeration constant	<code>enum</code>
constant (unchanging) value	<code>const</code>
declare external variable	<code>extern</code>
register variable	<code>register</code>
local to source file	<code>static</code>
no value	<code>void</code>
structure	<code>struct</code>
create name by data type	<code>typedef typename</code>
size of an object (type is <code>size_t</code> )	<code>sizeof object</code>
size of a data type (type is <code>size_t</code> )	<code>sizeof (type name)</code>

### Initialization

initialize variable	<code>type name=value</code>
initialize array	<code>type name[]={value<sub>1</sub>,...}</code>
initialize char string	<code>char name[]="string"</code>

### Constants

long (suffix)	<code>L</code> or <code>l</code>
float (suffix)	<code>F</code> or <code>f</code>
exponential form	<code>e</code>
octal (prefix zero)	<code>0</code>
hexadecimal (prefix zero- <code>0x</code> )	<code>0x</code> or <code>0X</code>
character constant (char, octal, hex)	<code>'a', '\ooo', '\xhh'</code>
newline, cr, tab, backspace	<code>\n, \r, \t, \b</code>
special characters	<code>\\, \?, \', \"</code>
string constant (ends with <code>'\0'</code> )	<code>"abc...de"</code>

### Pointers, Arrays & Structures

declare pointer to <code>type</code>	<code>type *name</code>
declare function returning pointer to <code>type</code>	<code>type *f()</code>
declare pointer to function returning <code>type</code>	<code>type (*pf)()</code>
generic pointer type	<code>void *</code>
null pointer	<code>NULL</code>
object pointed to by pointer	<code>*pointer</code>
address of object <code>name</code>	<code>&amp;name</code>
array	<code>name[dim]</code>
multi-dim array	<code>name[dim<sub>1</sub>][dim<sub>2</sub>]...</code>
<b>Structures</b>	
<code>struct tag {</code>	structure template
<code>declarations</code>	declaration of members
<code>};</code>	
create structure	<code>struct tag name</code>
member of structure from template	<code>name.member</code>
member of pointed to structure	<code>pointer-&gt;member</code>
Example. <code>(*p).x</code> and <code>p-&gt;x</code> are the same	
single value, multiple type structure	<code>union</code>
bit field with <code>b</code> bits	<code>member : b</code>

### Operators (grouped by precedence)

structure member operator	<code>name.member</code>
structure pointer	<code>pointer-&gt;member</code>
increment, decrement	<code>++, --</code>
plus, minus, logical not, bitwise not	<code>+, -, !, ~</code>
indirection via pointer, address of object	<code>*pointer, &amp;name</code>
cast expression to type	<code>(type) expr</code>
size of an object	<code>sizeof</code>
multiply, divide, modulus (remainder)	<code>*, /, %</code>
add, subtract	<code>+, -</code>
left, right shift [bit ops]	<code>&lt;&lt;, &gt;&gt;</code>
comparisons	<code>&gt;, &gt;=, &lt;, &lt;=</code>
comparisons	<code>==, !=</code>
bitwise and	<code>&amp;</code>
bitwise exclusive or	<code>^</code>
bitwise or (incl)	<code> </code>
logical and	<code>&amp;&amp;</code>
logical or	<code>  </code>
conditional expression	<code>expr<sub>1</sub> ? expr<sub>2</sub> : expr<sub>3</sub></code>
assignment operators	<code>+=, -=, *=, ...</code>
expression evaluation separator	<code>,</code>

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

### Flow of Control

statement terminator	<code>;</code>
block delimiters	<code>{ }</code>
exit from switch, while, do, for	<code>break</code>
next iteration of while, do, for	<code>continue</code>
go to	<code>goto label</code>
label	<code>label:</code>
return value from function	<code>return expr</code>

### Flow Constructions

if statement	<code>if (expr) statement</code> <code>else if (expr) statement</code> <code>else statement</code>
while statement	<code>while (expr) statement</code>
for statement	<code>for (expr<sub>1</sub>; expr<sub>2</sub>; expr<sub>3</sub>) statement</code>
do statement	<code>do statement</code> <code>while (expr);</code>
switch statement	<code>switch (expr) {</code> <code>case const<sub>1</sub>: statement<sub>1</sub> break;</code> <code>case const<sub>2</sub>: statement<sub>2</sub> break;</code> <code>default: statement</code> <code>}</code>

### ANSI Standard Libraries

<code>&lt;assert.h&gt;</code>	<code>&lt;ctype.h&gt;</code>	<code>&lt;errno.h&gt;</code>	<code>&lt;float.h&gt;</code>	<code>&lt;limits.h&gt;</code>
<code>&lt;locale.h&gt;</code>	<code>&lt;math.h&gt;</code>	<code>&lt;setjmp.h&gt;</code>	<code>&lt;signal.h&gt;</code>	<code>&lt;stdarg.h&gt;</code>
<code>&lt;stddef.h&gt;</code>	<code>&lt;stdio.h&gt;</code>	<code>&lt;stdlib.h&gt;</code>	<code>&lt;string.h&gt;</code>	<code>&lt;time.h&gt;</code>

### Character Class Tests <ctype.h>

alphanumeric?	<code>isalnum(c)</code>
alphabetic?	<code>isalpha(c)</code>
control character?	<code>iscntrl(c)</code>
decimal digit?	<code>isdigit(c)</code>
printing character (not incl space)?	<code>isgraph(c)</code>
lower case letter?	<code>islower(c)</code>
printing character (incl space)?	<code>isprint(c)</code>
printing char except space, letter, digit?	<code>ispunct(c)</code>
space, formfeed, newline, cr, tab, vtab?	<code>isspace(c)</code>
upper case letter?	<code>isupper(c)</code>
hexadecimal digit?	<code>isxdigit(c)</code>
convert to lower case?	<code>tolower(c)</code>
convert to upper case?	<code>toupper(c)</code>

### String Operations <string.h>

`s,t` are strings, `cs,ct` are constant strings

length of <code>s</code>	<code>strlen(s)</code>
copy <code>ct</code> to <code>s</code>	<code>strcpy(s,ct)</code>
up to <code>n</code> chars	<code>strncpy(s,ct,n)</code>
concatenate <code>ct</code> after <code>s</code>	<code>strcat(s,ct)</code>
up to <code>n</code> chars	<code>strncat(s,ct,n)</code>
compare <code>cs</code> to <code>ct</code>	<code>strcmp(cs,ct)</code>
only first <code>n</code> chars	<code>strncmp(cs,ct,n)</code>
pointer to first <code>c</code> in <code>cs</code>	<code>strchr(cs,c)</code>
pointer to last <code>c</code> in <code>cs</code>	<code>strrchr(cs,c)</code>
copy <code>n</code> chars from <code>ct</code> to <code>s</code>	<code>memcpy(s,ct,n)</code>
copy <code>n</code> chars from <code>ct</code> to <code>s</code> (may overlap)	<code>memmove(s,ct,n)</code>
compare <code>n</code> chars of <code>cs</code> with <code>ct</code>	<code>memcmp(cs,ct,n)</code>
pointer to first <code>c</code> in first <code>n</code> chars of <code>cs</code>	<code>memchr(cs,c,n)</code>
put <code>c</code> into first <code>n</code> chars of <code>cs</code>	<code>memset(s,c,n)</code>

## C Reference Card (ANSI)

### Input/Output <stdio.h>

#### Standard I/O

standard input stream `stdin`  
 standard output stream `stdout`  
 standard error stream `stderr`  
 end of file `EOF`  
 get a character `getchar()`  
 print a character `putchar(chr)`  
 print formatted data `printf("format", arg1, ...)`  
 print to string `s` `sprintf(s, "format", arg1, ...)`  
 read formatted data `scanf("format", &name1, ...)`  
 read from string `s` `sscanf(s, "format", &name1, ...)`  
 read line to string `s` (< `max` chars) `gets(s, max)`  
 print string `s` `puts(s)`

#### File I/O

declare file pointer `FILE *fp`  
 pointer to named file `fopen("name", "mode")`  
 modes: `r` (read), `w` (write), `a` (append)  
 get a character `getc(fp)`  
 write a character `putc(chr, fp)`  
 write to file `fprintf(fp, "format", arg1, ...)`  
 read from file `fscanf(fp, "format", arg1, ...)`  
 close file `fclose(fp)`  
 non-zero if error `ferror(fp)`  
 non-zero if EOF `feof(fp)`  
 read line to string `s` (< `max` chars) `fgets(s, max, fp)`  
 write string `s` `fputs(s, fp)`

#### Codes for Formatted I/O: "%-+ 0w.pmc"

- left justify  
 + print with sign  
 space print space if no sign  
 0 pad with leading zeros  
 w min field width  
 p precision  
 m conversion character:  
     h short, l long, L long double  
 c conversion character:  
     d,i integer u unsigned  
     c single char s char string  
     f double e,E exponential  
     o octal x,X hexadecimal  
     p pointer n number of chars written  
     g,G same as f or e,E depending on exponent

### Variable Argument Lists <stdarg.h>

declaration of pointer to arguments `va_list name`  
 initialization of argument pointer `va_start(name, lastarg)`  
     *lastarg* is last named parameter of the function  
 access next unnamed arg, update pointer `va_arg(name, type)`  
 call before exiting function `va_end(name)`

### Standard Utility Functions <stdlib.h>

absolute value of int `n` `abs(n)`  
 absolute value of long `n` `labs(n)`  
 quotient and remainder of ints `n,d` `div(n,d)`  
     returns structure with `div_t.quot` and `div_t.rem`  
 quotient and remainder of longs `n,d` `ldiv(n,d)`  
     returns structure with `ldiv_t.quot` and `ldiv_t.rem`  
 pseudo-random integer [0, `RAND_MAX`] `rand()`  
 set random seed to `n` `srand(n)`  
 terminate program execution `exit(status)`  
 pass string `s` to system for execution `system(s)`

#### Conversions

convert string `s` to double `atof(s)`  
 convert string `s` to integer `atoi(s)`  
 convert string `s` to long `atol(s)`  
 convert prefix of `s` to double `strtod(s, endp)`  
 convert prefix of `s` (base `b`) to long `strtol(s, endp, b)`  
     same, but unsigned long `strtoul(s, endp, b)`

#### Storage Allocation

allocate storage `malloc(size)`, `calloc(nobj, size)`  
 change size of object `realloc(pts, size)`  
 deallocate space `free(ptr)`

#### Array Functions

search array for key `bsearch(key, array, n, size, cmp())`  
 sort array ascending order `qsort(array, n, size, cmp())`

### Time and Date Functions <time.h>

processor time used by program `clock()`  
     Example. `clock()/CLOCKS_PER_SEC` is time in seconds  
 current calendar time `time()`  
 time<sub>2</sub>-time<sub>1</sub> in seconds (double) `difftime(time2, time1)`  
 arithmetic types representing times `clock_t`, `time_t`  
 structure type for calendar time comps `tm`  
     tm\_sec seconds after minute  
     tm\_min minutes after hour  
     tm\_hour hours since midnight  
     tm\_mday day of month  
     tm\_mon months since January  
     tm\_year years since 1900  
     tm\_wday days since Sunday  
     tm\_yday days since January 1  
     tm\_isdst Daylight Savings Time flag

convert local time to calendar time `mktime(tp)`  
 convert time in `tp` to string `asctime(tp)`  
 convert calendar time in `tp` to local time `ctime(tp)`  
 convert calendar time to GMT `gmtime(tp)`  
 convert calendar time to local time `localtime(tp)`  
 format date and time info `strftime(s, smax, "format", tp)`  
     *tp* is a pointer to a structure of type `tm`

### Mathematical Functions <math.h>

Arguments and returned values are double

trig functions `sin(x)`, `cos(x)`, `tan(x)`  
 inverse trig functions `asin(x)`, `acos(x)`, `atan(x)`  
     `atan2(y, x)`  
 hyperbolic trig functions `sinh(x)`, `cosh(x)`, `tanh(x)`  
 exponentials & logs `exp(x)`, `log(x)`, `log10(x)`  
 exponentials & logs (2 power) `ldexp(x, n)`, `frexp(x, *e)`  
 division & remainder `modf(x, *ip)`, `fmod(x, y)`  
 powers `pow(x, y)`, `sqrt(x)`  
 rounding `ceil(x)`, `floor(x)`, `fabs(x)`

### Integer Type Limits <limits.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system.

CHAR_BIT	bits in char	(8)
CHAR_MAX	max value of char	(127 or 255)
CHAR_MIN	min value of char	(-128 or 0)
INT_MAX	max value of int	(+32,767)
INT_MIN	min value of int	(-32,768)
LONG_MAX	max value of long	(+2,147,483,647)
LONG_MIN	min value of long	(-2,147,483,648)
SCHAR_MAX	max value of signed char	(+127)
SCHAR_MIN	min value of signed char	(-128)
SHRT_MAX	max value of short	(+32,767)
SHRT_MIN	min value of short	(-32,768)
UCHAR_MAX	max value of unsigned char	(255)
UINT_MAX	max value of unsigned int	(65,535)
ULONG_MAX	max value of unsigned long	(4,294,967,295)
USHRT_MAX	max value of unsigned short	(65,536)

### Float Type Limits <float.h>

FLT_RADIX	radix of exponent rep	(2)
FLT_ROUNDS	floating point rounding mode	
FLT_DIG	decimal digits of precision	(6)
FLT_EPSILON	smallest $x$ so $1.0 + x \neq 1.0$	( $10^{-5}$ )
FLT_MANT_DIG	number of digits in mantissa	
FLT_MAX	maximum floating point number	( $10^{37}$ )
FLT_MAX_EXP	maximum exponent	
FLT_MIN	minimum floating point number	( $10^{-37}$ )
FLT_MIN_EXP	minimum exponent	
DBL_DIG	decimal digits of precision	(10)
DBL_EPSILON	smallest $x$ so $1.0 + x \neq 1.0$	( $10^{-9}$ )
DBL_MANT_DIG	number of digits in mantissa	
DBL_MAX	max double floating point number	( $10^{37}$ )
DBL_MAX_EXP	maximum exponent	
DBL_MIN	min double floating point number	( $10^{-37}$ )
DBL_MIN_EXP	minimum exponent	

May 1999 v1.3. Copyright © 1999 Joseph H. Silverman

Permission is granted to make and distribute copies of this card provided the copyright notice and this permission notice are preserved on all copies.

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. (jhs@math.brown.edu)



- Introducción
- **Proceso de Compilación y Enlazado**
- Elementos de un Programa C
- Sentencias de Control de Flujo
- Tipos de Datos Derivados
- Funciones
- Preprocesador y Librerías





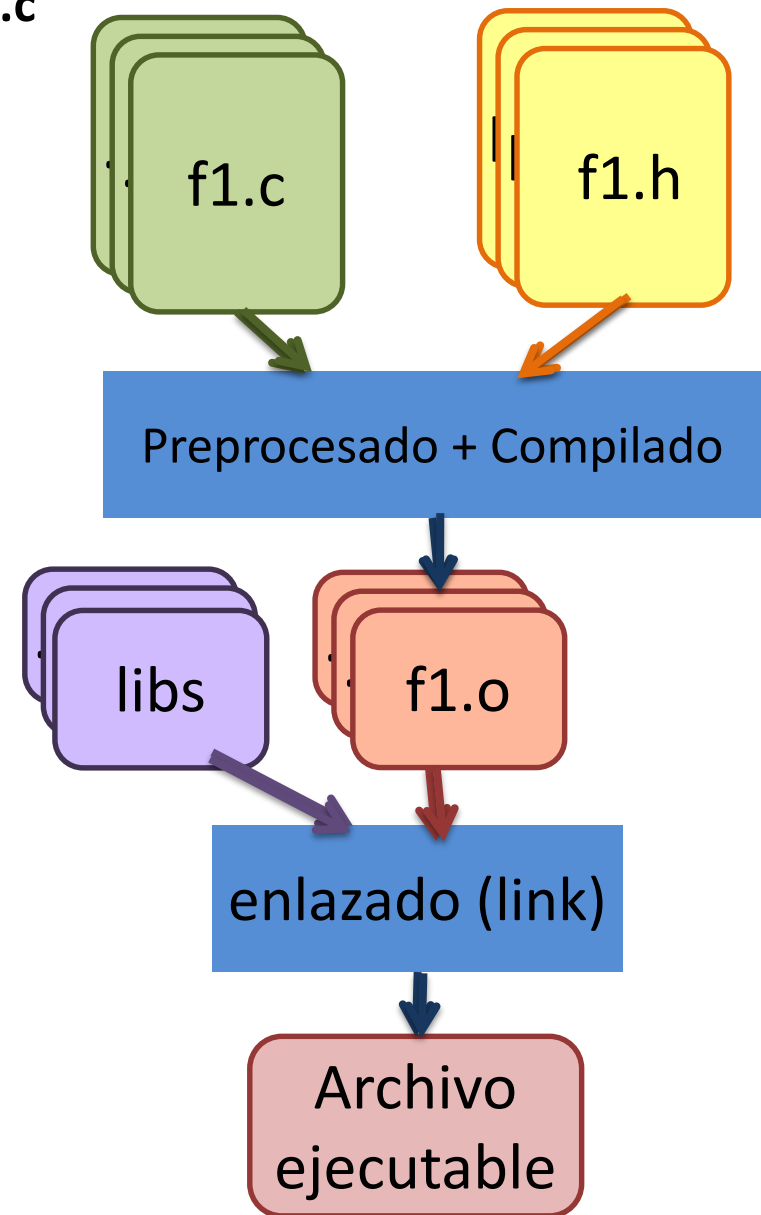
- **Ficheros con código fuente** con extensión “.c “

```
/* fichero_fuente.c */  
#include <stdio.h>  
main()  
{  
    printf ("Hola mundo!\n");  
}
```

- Ficheros con declaraciones y prototipos extensión “.h”

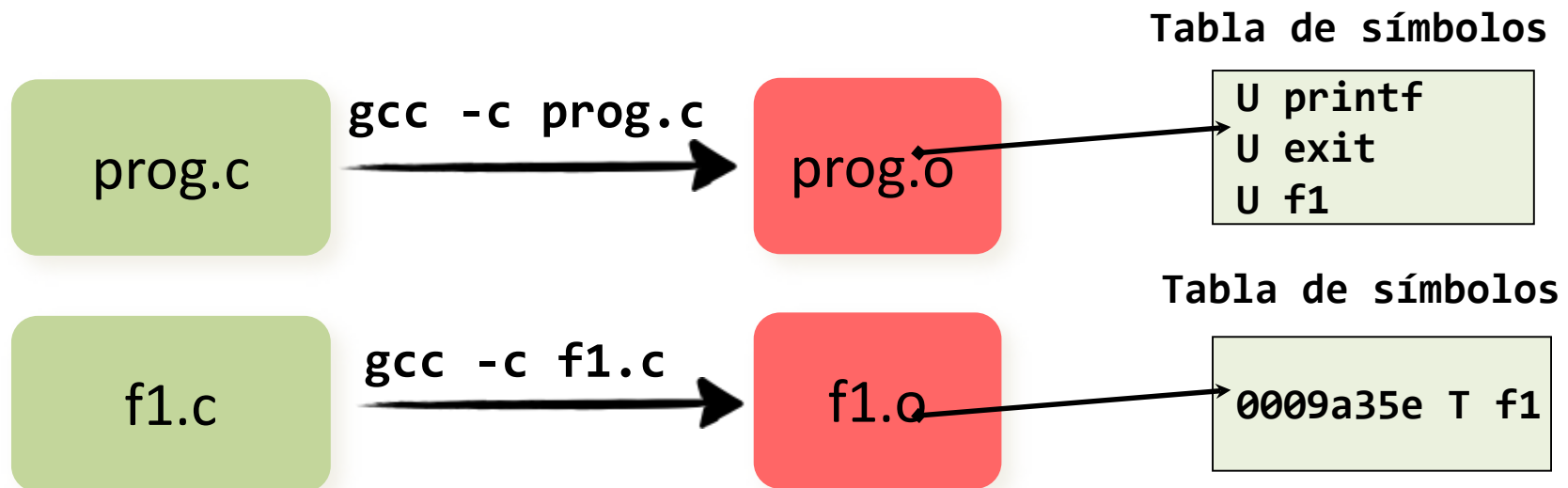
```
/* cabecera.h */  
#define MI_CADENA "Hola Mundo"  
#define PI 3.1415925  
#define MAX(A,B) ((A)>(B)?(A):(B))
```

- Es necesario **compilar y enlazar**  
\$ gcc holamundo.c -o ejemplo
- Existen entornos de desarrollo (IDE) que integran editor, compilador, depurador, etc.
  - Ejemplo Dev C++, Eclipse



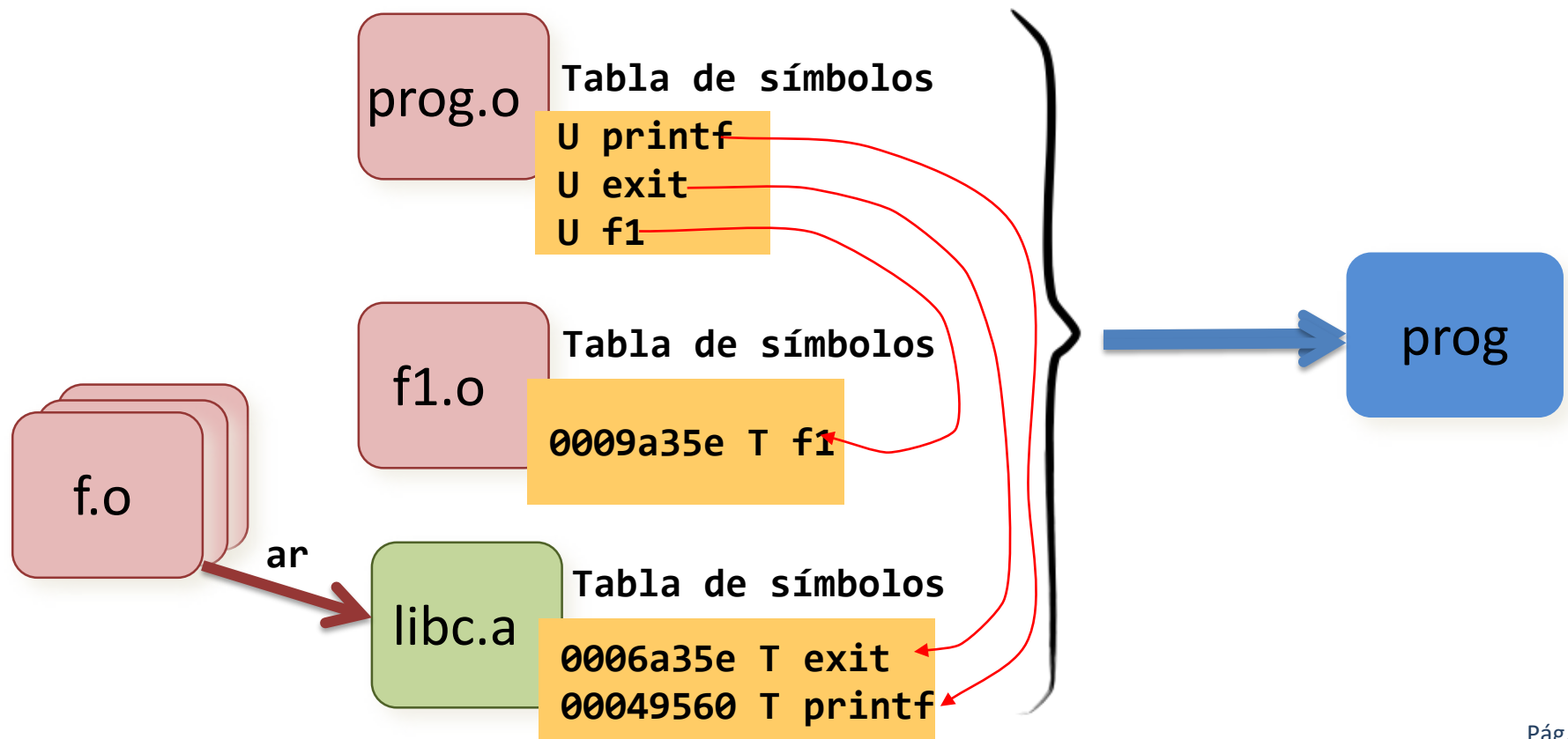
## • Compilación

- Generación de **código máquina reubicable**
- Generación de la tabla de símbolos con las dependencias de otros módulos



- Proceso de **enlazado o montaje** (*link*)
  - **Resolución de referencias cruzadas:** enlazado de símbolos no resueltos con otros módulos y bibliotecas.
  - Generación del fichero ejecutable

```
$gcc -o prog prog.o f1.o -lc
```



- Introducción
- Proceso de Compilación y Enlazado
- **Elementos de un Programa C**
- Sentencias de Control de Flujo
- Tipos de Datos Derivados
- Funciones
- Preprocesador y Librerías

- Básicamente **C** está compuesto por los siguientes elementos:
  - Comentarios
  - Identificadores
  - Palabras reservadas
  - Variables y tipos de datos
  - Constantes
  - Operadores
  - Sentencias



Java==C

- **Comentarios**
  - Dos tipos `/*y */` y `//` hasta final de línea
- **Identificadores**
  - Nombre que se asigna a los distintos elementos del programa (variables, funciones)
    - No valido que comiencen por numero y guión (-)
    - No utilizar caracteres reservados como `{},()`, etc.
- **Palabras reservadas**
  - Todas las palabras propias de lenguaje como: *if, else, float, ....*
- **DISTINGUE ENTRE MAYÚSCULAS Y MINÚSCULAS**

```
/* Comentario de varias lineas  
esto también  
*/
```

```
// Comentario hasta fin linea
```

```
// Identificadores válidos
```

```
a
```

```
b12
```

```
la_variable_es_larga
```

```
// Identificadores NO válidos
```

```
3b
```

```
b-s
```



- **Literales**

- **Enteros**

- Hexadecimal: 0x2f, 0xFFFF, 0x123
    - Octal: 027, 035

- **Reales** (notación exponencial)

- 1.04E-12, -1.2e5

- **Caracteres**

- 'c', 'b', '\n', '\t', '\0'

- **Cadenas** "Juan"

- **Constantes**

- Les precede “#define”

```
#define PI 3.141593
#define CIERTO 1
#define FALSO 0
#define AMIGA "Marta"
```



Java==C





Java~C

- **Variables**
  - Tienen un tipo y modificadores.  
`tipo var [=valor];`
- **Tipos**
  - Carácter **char** [= 1 byte] (tipo **byte** en Java, en Java **char** son 2 bytes UNICODE)
  - Entero  
`int` [= 2/4 bytes], `long` [= 4/8 bytes]
  - Real (IEEE 754)  
`float` [= 4 bytes], `double` [= 8 bytes]
  - Conversión (**casting**)
    - `var_ta=(tipo)var_tb`
- **Modificadores (no definidos en Java)**
  - **signed**: con signo (por defecto)
  - **unsigned**: sin signo
- **Definición tipos**
  - `typedef tipo nuevo_tipo;`

```

char c;
unsigned char b; // un byte
int b; // entero con signo
unsigned int c; // sin signo
long l;
signed long l; // lo mismo
unsigned long l2;
float f1;
double s2;
int d= 5; // valor inicial 5

b = (int)c;
f1 = (float)c;

// definición de tipos
typedef float kg;
typedef int altura;

```

Java==C

## • Operadores

- Asignación =
- In/Decrementales ++,--
- Aritméticos +,-,\*,/,% (resto)
- Relacionales ==,<,>,<=,>=,!=
- Lógico || (or), && (and)
- Unarios: -,+,!
  - coma , : permite varios expresiones
  - sizeof: tamaño en bytes de una variable
  - dirección (&) e indirección (\*)

## • Expresiones

- lógicas: devuelven valor lógico
- aritméticas: devuelven valor numérico

```
int a;  
a=5; //asignación  
  
a++; // a vale 6  
a--; // vuelve a valer 5  
b=5%2; //devuelve 1  
  
(2==1) // resultado=0  
(3<=3) // resultado=1  
(1!=1) // resultado=0  
(2==1) || (-1==1) // 1  
((2==2) && (3==3)) || (4==0) // 1  
  
sizeof(a); // devuelve 4  
  
a = ((b>c)&&(c>d))||((c==e)||e==b)); //  
expresión lógica  
  
x=(-b+sqrt((b*b)-(4*a*c)))/(2*a);  
// exp. aritmética
```

Java==C

- **Sentencias**

- simples: expresión terminada por ;

```
float real;  
espacio = espacio_inicial + velocidad * tiempo;
```

- vacía o nula: ;


```
;
```

- bloques: empiezan por “{ “ y terminan por “}”

```
{  
    int i = 1, j = 3, k;  
    double masa;  
    masa = 3.0;  
    k = y + j;  
}
```

- **Entrada y salida**

- scanf (formato,argumentos)
- printf (formato,argumentos)



```
printf("Texto: %s, Entero: %d, Float: %f\n", "rojo", 5, 3.14);
```

SACA POR CONSOLA

Texto: rojo, Entero: 5, Float: 3.14);

- **EJEMPLOS**

```
printf("Hola mundo\n");  
printf("El numero 28 es %d\n", 28);  
printf("Imprimir %c %d %f\n", 'a', 28, 3.0e+8);
```

```
scanf("%f", &numero);  
scanf("%c\n", &letra);  
scanf("%f %d %c", &real, &entero, &letra);  
scanf("%ld", &entero_largo);  
scanf("%s", cadena);
```



- Practica con ejemplos:
  - Calcula el cuadrado de un número

## cuadrado.c

```
#include <stdio.h>
main()
{
    int numero;
    int cuadrado;
    printf("Introduzca un número: ");
    scanf("%d", &numero);
    cuadrado = numero * numero;
    printf("El cuadrado de %d es %d\n", numero,cuadrado);
}
```

- Introducción
- Proceso de Compilación y Enlazado
- Elementos de un Programa C
- **Sentencias de Control de Flujo**
- Tipos de Datos Derivados
- Funciones
- Preprocesador y Librerías

Java==C

- **Sentencia if**

```
if (expresion)
    sentencia;
```

- **Sentencia if ... else**

```
if (expresion)
    sentencia1;
else
    sentencia2;
```

- **Sentencia if ... else múltiple**

```
if (expresion1)
    sentencia1;
else if (expresion2)
    sentencia2;
[else
    sentencia3;]
```

```
if (a > 4)
```

```
b = 2;
```

```
if (b > 2 || c < 3)
```

```
{
```

```
    b = 4; c = 7;
```

```
};
```

```
// opción con else
```

```
if (d < 5)
```

```
{
```

```
    d++;
```

```
}
```

```
else
```

```
{
```

```
    d--;
```

```
};
```

```
// opción con else
```

```
if (IMC < 20)
```

```
    printf("delgado");
```

```
else if (IMC <= 25)
```

```
    printf("normal");
```

```
else
```

```
    printf("obeso");
```



Java==C

- **Sentencia “switch”**

```
switch (expresion) {  
    case expresion_cte_1:  
        sentencia_1;  
        break;  
    case expresion_cte_2:  
        sentencia_2;  
        break;  
    ...  
    case expresion_cte_n:  
        sentencia_n;  
        break;  
    [default:  
        sentencia;]
```

- “expresión” tiene que ser un tipo entero (int, long) o carácter (char)

```
switch (a) {  
    case 5:  
        printf("aprobado");  
        break;  
    case 6:  
        printf("bien");  
        break;  
    case 7:  
    case 8:  
        printf("notable");  
        break;  
    case 9:  
        printf("sobresaliente");  
        break;  
    case 10:  
        printf("Matricula");  
        break;  
    default:  
        printf("suspendido");  
}
```

Java==C

- **Bucle “while”**

```
while (expresion)
    sentencia;
```

- **Bucle “do... While”**

```
do
    sentencia;
while (expresion);
```

- **Sentencias “break”, “continue”**

- `break;`
  - termina el bucle
- `continue;`
  - salta al fin del bucle

```
x = 1;
while (x < 10)
    x++;
x = 1; z = 2
while (x < 10 && z != 0) {
    b = x/z;
    x++;
    z--;
}
// Otro bucle
x = 1;
do {
    x++;
} while (x < 20)

while (x < 10) {
    x++; z--;
    if (z==0) break;
    b = x/z;
}
```

- **Bucle “for”**

```
for (inicializacion; expresion_de_control; actualizacion)
    sentencia;
```

```
int i;
for (i=0; i< 10; i++) {
    total = total + a[i];
}

int numero;
for (numero=0; numero <100; numero++)
{
    printf("%d\n", numero);
}
```

- Practica con ejemplos

## sumaserie.c

```
#include <stdio.h>

main()
{
    int N;
    int suma = 0; /* leer numero N */
    printf("N: ");
    scanf("%d", &N);
    while (N > 0) {
        suma = suma + N;
        N = N - 1; /* equivale a N-- */
    }
    printf("1 + 2 + ... + N = %d\n", suma);
}
```

## sumaserie2.c

```
#include <stdio.h>

main()
{
    int N, suma, j;
    do
    {
        /* leer el numero N */
        printf("Introduzca N: ");
        scanf("%d", &N);
        suma = 0;
        for (j = 0; j <= N; j++)
            /* bucle anidado */
            suma = suma + j;
        printf("1 + 2 + ... + N = %d\n", suma);
    } while (N > 0); /* fin del bucle do */
}
```

- Introducción
- Proceso de Compilación y Enlazado
- Elementos de un Programa C
- Sentencias de Control de Flujo
- **Tipos de Datos Derivados**
- Funciones
- Preprocesador y Librerías

- **Arrays** (listas o vectores)

- definición

- `tipo var[tam];`

- Acceso

- `var[entero];`

- Índice comienza en 0

- **Matrices**

- `tipo var[tam1][tam2];`

- **Copia de vectores**

- utilizar función “**memcpy**”

- `memcpy(v1, v2, tam)`

- Se copia el vector v2 en v1

```
//arrays definición y acceso
```

```
int v[10]={1,4,3,11,0,6,7,8,9,0};
```

```
int v2[10];
```

```
suma = 0;
```

```
int i;
```

```
v[1] = 5;
```

```
for (i=0; i< 10; i++)
```

```
{
```

```
    suma = suma + v[i];
```

```
}
```

```
int m[10][5];
```

```
int i,j;
```

```
for (i=0; i < 10; i++)
```

```
{
```

```
    for (j=0; j < 5; j++)
```

```
    { suma = suma + m[i][j];
```

```
    }
```

```
}
```

```
// copia de vectores
```

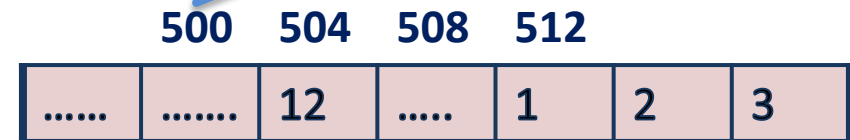
```
memcpy(v2,v, sizeof(v));
```

## • Punteros

- Un puntero es una variable que contiene la dirección de otro objeto
- Se define como:  
**tipo \*nombre;**
- El **operador &** obtiene la dirección de una variable
- Se denomina **indirección** devolver el dato apuntado por el puntero  
**operador \***

```
int b;  
int x = 12;  
int *p;  
int N[3] = { 1, 2, 3 };  
char *pc; // Puntero a carácter  
p = &x;   // p vale 504 (apunta a x)  
b = *p;   // Indirección: b = 12  
*p = 10;  // Modifica contenido x  
p = N;    // p apunta a N (vale 512)
```

Suponga que el compilador asigna memoria a partir de la dirección 500 y que los enteros ocupan 4 bytes





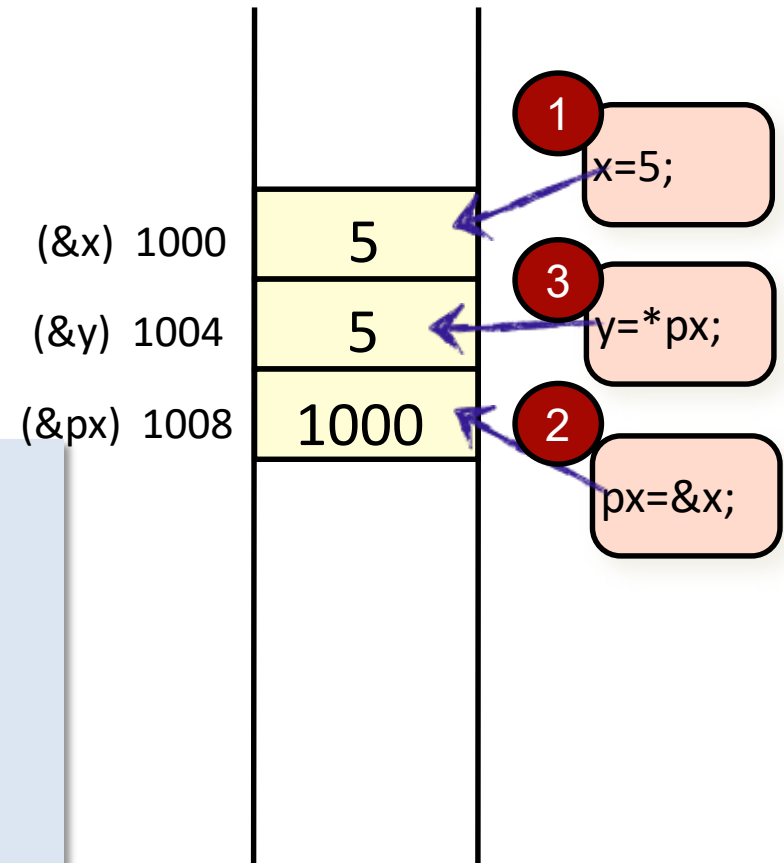
- **Punteros**
- Practica con ejemplos

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

**punteros.c**

```
main()
{ int x; /* variable entera */
  int y; /* variable entera */
  int *px; /* puntero a entero */
  x = 5;
  px = &x; /* px = direccion de x */
  y = *px; /* y = contiene la direccion almacenada en px */

  printf("x = %d\n", x);
  printf("y = %d\n", y);
  printf("*px = %d\n", *px);
  printf("px (&x) = %p\n", px);
}
```



- **Cadenas**

- Array de caracteres (**char**)

- Se declara indicando tamaño o con puntero

- `char cadena[10];`

- `char* cadena;`

- Finalizan con carácter nulo “\0”

- Función para copiar cadenas

- strcpy(cad1, cad2)**

- Se copia cadena cad2 en cad1

```
#include <stdio.h>
#define TAM_CADENA 80
main() {
    char cadena[TAM_CADENA];
    printf("Introduzca una cadena: ");
    scanf("%s", cadena); // No requiere &
    printf("La cadena es %s\n", cadena);
}
```

```
char ciudad[20] = "San Sebastián";
char nombre[] = "Pepito Perez";
char nombre2[20];
strcpy(nombre, nombre2);
```

P	E	P	I	T	O		P	E	R	E	Z	\0	
---	---	---	---	---	---	--	---	---	---	---	---	----	--

- Aritmética de punteros

- C permite realizar varias operaciones con variable punteros
- Operadores incremento/decremento (++/--).
- Suma y resta (desplazamiento de la posición)
- El desplazamiento es siempre del tipo al que apunta la variable
- Ejemplos:

```
int Datos[5] = {1,2,3,4,5};
```

```
int *p;
```

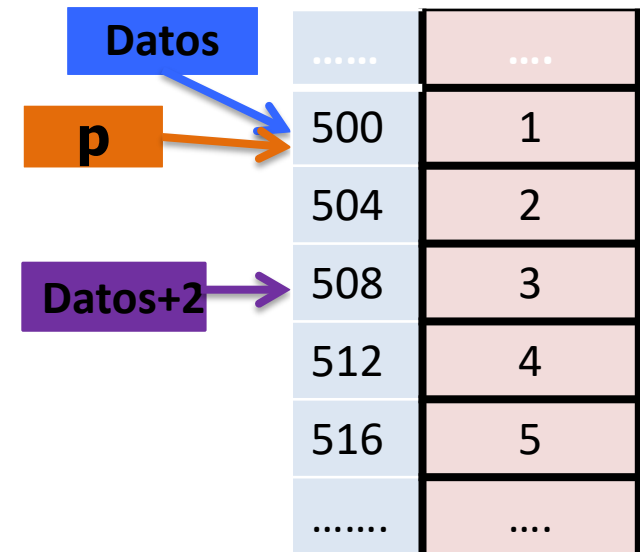
```
int i;
```

```
int b;
```

```
p = Datos+2; // p apunta al 3er elemento(508)
```

```
p = Datos; // p apunta a Datos (500)
```

```
for (i = 0; i < 5; i++) {  
    printf("Datos[%u]=%u", i, Datos[i]);  
    printf("Datos[%u]=%u", i, *p++);  
}
```



- Otra clase de punteros
  - **Puntero genérico**: su tipo es “**void**” y pueden apuntar a cualquier tipo de dato
  - **Puntero nulo**: es una variable puntero cuyo valor es 0, se utiliza **NULL**
    - El valor NULL se utiliza para indicar que ha ocurrido algún error
    - El valor NULL se utiliza para indicar que o para indicar que el puntero no apunta a ningún dato

```
void *v;  
int i[10];  
int a;  
v = i;  
a = *((int *)v);    // Hay que hacer casting  
v = malloc(100000000); // Seguramente dará error  
if (v == NULL) exit(-1);
```

- **Estructuras**

- definición

```
struct nombre_estructura
{ tipoDato1 miembro_1;
  tipoDato2 miembro_2;
  ...
  tipoDatoN miembro_N;
};
```

- declarar una variable

```
struct nombre_estruc v;
```

- Acceso a los miembros

```
v.miembro
v->miembro
```

```
//definición de una estructura
struct CD
{
    char titulo[100];
    char artista[50];
    int num_canciones;
    int anyo;
    float precio;
};
//declaración de una variable
struct CD cd1;
//accediendo a la estructura
strcpy(cd1.titulo, "La Boheme");
strcpy(cd1.artista, "Puccini");
cd1.num_canciones = 2;
cd1.anyo = 2006;

struct CD *pcd;
pcd = &cd1;
pcd->precio = 16.5;
```

- **Vector de estructuras**

- declarar como vector

```
struct nombre ve[tam];
```

- Acceso a los miembros

```
v[i].miembro
```

```
v->miembro
```

- **Paso de estructuras a funciones**

- Por valor -> muy costoso
- Mejor por referencia (el puntero)

```
function imprime_cd (struct cd *pcd) {  
    printf("Precio = %d\n", pcd->precio);  
    // Se puede modificar  
}
```

```
// Invocando a la función imprime  
imprime_cd(&cd1);  
imprime_cd(&col[10]);
```

//definición de una estructura

```
struct CD  
{  
    char titulo[100];  
    char artista[50];  
    int num_canciones;  
    int anyo;  
    float precio;  
};  
struct CD col[100];  
  
for (i = 1; i < 100; i++)  
    total = col[i].precio;  
pcd = &col[10];  
//pcd apunta al undécimo cd  
pcd->precio = 16.5;
```

- Practica con ejemplo (I): Gestión de un almacén

```
main() {
    registro_piezas cajas[NUMCAJAS];
    int registro=0;
    int i;
    /* Lee los datos desde teclado */
    do {
        /* Lee el nombre de la pieza. */
        printf("Nombre de la pieza => ");
        scanf("%s", cajas[registro].pieza);

        /* Lee el número de piezas. */
        printf("Numero de piezas => ");
        scanf("%d", &cajas[registro].cantidad);

        /* Lee el precio de cada pieza. */
        printf("Precio de cada pieza => ");
        scanf("%f", &cajas[registro].precio_unidad);

        /* Indica que el registro tiene datos, V */
        cajas[registro].existe = 'V';
        registro ++;
    } while (registro < NUMCAJAS);
}
```

**almacen.c**

```
#include <stdio.h>
#include <string.h>
#define NUMCAJAS 3

typedef struct {
    char pieza[20]; // Tipo de pieza.
    int cantidad;   // Número de piezas.
    float precio_unidad; // Precio de pieza
    char existe;    // Comprueba si existe registro
} registro_piezas;
```

```
/* Imprimir la información. */
for(registro = 0; registro < NUMCAJAS; registro++) {
    if(cajas[registro].existe == 'V') {
        printf("La caja %d tiene:\n", registro + 1);
        printf("Pieza => %s\n", cajas[registro].pieza);
        printf("Cantidad => %d\n", cajas[registro].cantidad);
        printf("Precio => %f euros\n", cajas[registro].precio_unitario);
    }
} /* Fin for. */
} /* fin main*/
```



- Introducción
- Proceso de Compilación y Enlazado
- Elementos de un Programa C
- Sentencias de Control de Flujo
- Tipos de Datos Derivados
- **Funciones**
- Preprocesador y Librerías

- C es un lenguaje basado en **funciones**
  - tiene que haber siempre una función main
- Java es un lenguaje basado en **objetos....**

- **Definición de una función en C**

```
tipo_retorno nombre_funcion(tipo1 arg1,..., tipoN
argN)
{
    [declaración de variables locales;]
    codigo ejecutable
    [return (expresión);]
}
```

- **Declaración de una función en C**

- Para poder usar una función antes de definirla, se debe declarar.

```
tipo_retorno nombre_funcion(tipo1 arg1,..., tipoN
argN)
```

- **Llamada a una función**

```
ret = funcion(arg1, arg2, ..., argN);
```

- **Paso de parámetros por valor**

## main.c

```
#include <stdio.h>

// declaración

double valor_abs(double dato);

void main (void) {
    double z, y;
    y = -30.8;
    z = valor_abs(y) + y*y;
}
```

## valor.c

```
// definición
double valor_abs(double x)
{
    if (x < 0.0)
        return -x;
    else
        return x;
}
```

- Paso de parámetros **por referencia**

- en la definición de función, **al argumento le precede “\* ”**

```
tipo_retorno nombre_funcion(tipo1 *arg1, ..., tipoN argN)
```

- en llamada, **al argumento le precede “&”**

```
retorno= nombre_funcion(&arg1, ..., tipoN argN)
```

```
void permutar(double *x, double *y)
{
    double temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

void main(void) {
    double a=1.0, b=2.0;
    printf("a = %f, b = %f\n");
    permutar(&a, &b);
    printf("a = %f, b = %f\n");
}
```

- **Ámbito de las variables**
  - **Globales**
    - se declaran fuera de cualquier función
    - se puede acceder desde cualquier función del fichero
  - **Locales**
    - se definen dentro de las funciones
    - solo accesibles dentro de la función
  - **Estáticas**
    - son locales pero guardan su valor

## Global\_local.c

```
#include <stdio.h>

void funcion1(void);

int a = 10; /* variable global */

main()
{
    int b = 2; /* variable local */
    a++;
    funcion1();
    a++;
    printf("a= %d, b= %d\n", a, b);
    a++;
    funcion1();
}

void funcion1(void)
{
    static int c = 4; /* variable estática */
    printf("a= %d, c= %d\n", a, c);
    c++;
    return;
}
```

- Practica con ejemplos

## hipotenusa.c

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

void hipotenusa(float a, float b, float *h)
{
    *h = sqrt(pow(a,2) + pow(b, 2));
}

void leer (float *a, float *b) {
    printf("Dame valores a y b:\n");
    scanf("%f %f", *a, *b);
}

main() {
    float a, b, h;
    leer (&a,&b);
    hipotenusa(a,b,&h);
    printf("La hipotenusa es %f\n", h);
}
```

- **Parámetros de entrada en línea de comandos**

- Al invocar un programa por línea de comandos, podemos pasarle parámetros

- Ejemplo: **\$suma 2 3**

- Se define el main de la siguiente forma.

- **int main (int argc, char \*argv[])**

donde “**argc**” es el número de argumentos y “**argv**” un array con los argumentos. El primer argumento **argv[0]** es el nombre del programa

## suma.c

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    int sum1,sum2;
    if (argc == 3) {
        sum1 = atoi(argv[1]);
        sum2 = atoi(argv[2]);
        printf("La suma es %d\n", sum1+sum2);
    }
    else {
        printf("Uso de la orden: %s arg1 arg2\n", argv[0]);
    }
}
```

- Introducción
- Proceso de Compilación y Enlazado
- Elementos de un Programa C
- Sentencias de Control de Flujo
- Tipos de Datos Derivados
- Funciones
- **Preprocesador y Librerías**



- Preprocesador I

- Antes de compilar, existe una fase que se llama preprocesar:

- Sustituye macros (`#define`, `#undef`)
    - Compilaciones condicionales (`#if`, `#ifdef`)
    - Incluir archivos (`#include`)
    - Ordenes directas al compilador (`#pragma` y `#error`)

- Comando **#define**

- Permite **definir macros** (constantes)

```
#define E 2.7182
```

```
#define g 9.81
```

- Funciones **inline**

```
#define SUMA(c,d) (c + d)
```

```
#define MAX(a, b) ((a) > (b)) ? (a) : (b))
```

```
if (MAX(altura1, altura2) == 5) {
```

- Comando **#undef MACRO** -> elimina la definición de una macro

- Preprocesador II

- Compilación condicional

```
#ifdef MACRO
    // Compila si está definida MACRO
#else
    // Se compila si no está definida
#endif
```

- Permite compilar bloque de código de forma opcional

- Más opciones

- #ifndef
    - #elif

```
#ifdef MODO_64BITS
    // Código para 64bits
    int x = 5;
#else
    // Código para 32bits
    long x = 5;
#endif
```

```
#ifdef CPU_ARM7
    // Código para ARM
    xARM.b = 5;
#elif CPU_INTEL
    // Código para INTEL
    x.b = 5;
#else
    // Resto CPU
#endif
```

- Librerías o bibliotecas
  - Conjunto de funciones de uso común
    - matemáticas, entrada/salida, tiempo, etc.
  - Las funciones están declaradas en un **fichero “.h”** llamada cabecera (header)
  - Para usar una función hay que incluir el fichero:

```
#include <nom_fich.h> // Librería del sistema
#include "nom_fich.h"  // Directorio local
```
  - Por ejemplo printf está en la librería “stdio” que está definida en la cabecera **“stdio.h”**

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

- Biblioteca string (string.h)
  - **char \*strcat (char \*cad1, char \*cad2)**
    - Concatena cad2 a cad1 devolviendo la dirección de cad1. Elimina el nulo de terminación de cad1 inicial
  - **char \*strcpy (char \*cad1, char \*cad2)**
    - Copia la cadena cad2 en cad1, sobrescribiéndola. Devuelve la dirección de cad1. El tamaño de cad1 debe ser suficiente para albergar a cad2
  - **int strlen (char \*cad)**
    - Devuelve el número de caracteres que almacena cad (sin contar el nulo final)
  - **int strcmp (char \*cad1, char \*cad2)**
    - Devuelve
      - > 0 : cad1 > cad2
      - 0: cad1 == cad2
      - <0: cad1 < cad2

- **Gestión de memoria** (stdlib.h o malloc.h)
  - `void *malloc(int bytes)`
    - Reserva n bytes de memoria. Devuelve un puntero al principio del espacio reservado.
  - `free(void *p)`
    - Libera un bloque de memoria al que apunta p.
- y muchas, muchas más....