Fonaments dels Sistemes Operatius (FSO)

Departament d'Informàtica de Sistemes i Computadors (DISCA)
Universitat Politècnica de València

Bloc Temàtic 1: Introducció

Seminari 1

Llenguatge C

fSO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Seminari llenguatge C

- **Objectius:**
  - Introduir el l**lenguatge de programació C** remarcant les seues diferències amb Java
  - Entendre el procés de compilació i enllaçat
  - Presentar el concepte de **punters en C**
  - Manejar **les crides a funcions** i el pas de **paràmetres por referència i valor**

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

Contingut

- **Introducció**
- Procés de compilació i enllaçat
- Elements d'un programa C
- Sentències de control de flux
- Tipus de dades derivats
- Funcions
- Preprocessador i biblioteques

Introducció

- **Característiques del llenguatge C**
  - Llenguatge de programació **de propòsit general**, molt adequat per a programació de sistemes i per a sistemes empotrats (UNIX va ser escrit en C).
  - Llenguatge relativament **reduit**: només ofereix sentències de control senzilles i funcions.
  - Molt **portable**: hi ha compiladors per a tots els processadors
  - **Codi** generat **molt eficient,** tant en velocitat como en grandària.
- **Llenguatge compilat**
  - Compilar: a partir d'un codi font (text) es genera l'executable.
    - Java és interpretat (cal una màquina virtual per a executar el programa)
  - Si l'executable no és per a la mateixa màquina la traducció es denomina compilació creuada

- Història del C
  - C va ser dissenyat originalment en 1972 per al SO UNIX en el DEC PDP-11 per Dennis Ritchie en els Laboratoris Bell.
  - En els 80, gran part de la programació es fa en C.
  - En 1983 apareix C++ (orientat a objectes).
  - El llenguatge C està estandarditzat:
    - ANSI C, ISO C.
- **Java i C**
  - Java està basat en C/C++
  - Les estructures de control són iguals a C
  - Les classes són semblants a C++
  - **Java elimina la gestió de memòria directa (punters)**

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$,...)	function declarations
type name	external variable declarations
main() {	main routine
declarations	local variable declarations
statements	
}	
type fnc(*arg*$_1$,...) {	function definition
declarations	local variable declarations
statements	
return *value*;	
}	
/* */	comments
main(int argc, char *argv[])	main with args
exit(*arg*)	terminate execution

C Preprocessor

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

Data Types/Declarations

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

Initialization

initialize variable	*type name=value*
initialize array	*type name*[]={*value*$_1$,...}
initialize char string	char *name*[]="*string*"

Constants

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

Pointers, Arrays & Structures

declare pointer to *type*	*type* **name*
declare function returning pointer to *type*	*type* **f*()
declare pointer to function returning *type*	*type* (**pf*)()
generic pointer type	void *
null pointer	NULL
object pointed to by *pointer*	**pointer*
address of object *name*	&*name*
array	*name*[*dim*]
multi-dim array	*name*[*dim*$_1$][*dim*$_2$]...

Structures

struct *tag* {	structure template
declarations	declaration of members
};	
create structure	struct *tag name*
member of structure from template	*name.member*
member of pointed to structure	*pointer* -> *member*
Example. (*p).x and p->x are the same	
single value, multiple type structure	union
bit field with *b* bits	*member* : *b*

Operators (grouped by precedence)

structure member operator	*name.member*
structure pointer	*pointer->member*
increment, decrement	++, --
plus, minus, logical not, bitwise not	+, -, !, ~
indirection via pointer, address of object	**pointer*, &*name*
cast expression to type	(*type*) *expr*
size of an object	sizeof
multiply, divide, modulus (remainder)	*, /, %
add, subtract	+, -
left, right shift [bit ops]	<<, >>
comparisons	>, >=, <, <=
comparisons	==, !=
bitwise and	&
bitwise exclusive or	^
bitwise or (incl)	|
logical and	&&
logical or	||
conditional expression	*expr*$_1$? *expr*$_2$: *expr*$_3$
assignment operators	+=, -=, *=, ...
expression evaluation separator	,

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

Flow of Control

statement terminator	;
block delimeters	{ }
exit from switch, while, do, for	break
next iteration of while, do, for	continue
go to	goto *label*
label	*label*:
return value from function	return *expr*

Flow Constructions

if statement	if (*expr*) *statement* else if (*expr*) *statement* else *statement*
while statement	while (*expr*) *statement*
for statement	for (*expr*$_1$; *expr*$_2$; *expr*$_3$) *statement*
do statement	do *statement* while(*expr*);

switch statement	switch (*expr*) {
	case *const*$_1$: *statement*$_1$ break;
	case *const*$_2$: *statement*$_2$ break;
	default: *statement*
	}

ANSI Standard Libraries

<assert.h>	<ctype.h>	<errno.h>	<float.h>	<limits.h>
<locale.h>	<math.h>	<setjmp.h>	<signal.h>	<stdarg.h>
<stddef.h>	<stdio.h>	<stdlib.h>	<string.h>	<time.h>

Character Class Tests <ctype.h>

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

String Operations <string.h>

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

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

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*",arg_1,...)
print to string s	sprintf(s,"*format*",arg_1,...)
read formatted data	scanf("*format*",&$name_1$,...)
read from string s	sscanf(s,"*format*",&$name_1$,...)
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*",arg_1,...)
read from file	fscanf(*fp*,"*format*",arg_1,...)
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: "%-+ 0*w.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 unamed 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)
retursn 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()
$time_2$-$time_1$ in seconds (double)	difftime($time_2$,$time_1$)
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)
arctan(y/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	

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

Contingut

- Introducció
- **Procés de compilació i enllaçat**
- Elements d'un programa C
- Sentències de control de flux
- Tipus de dades derivats
- Funcions
- Preprocessador i biblioteques

Procés de compilació i enllaçat

- **Fitxers de codi font** amb extensió **“.c “**
- Fitxers de declaracions i prototipus amb l’extensió **“.h”**

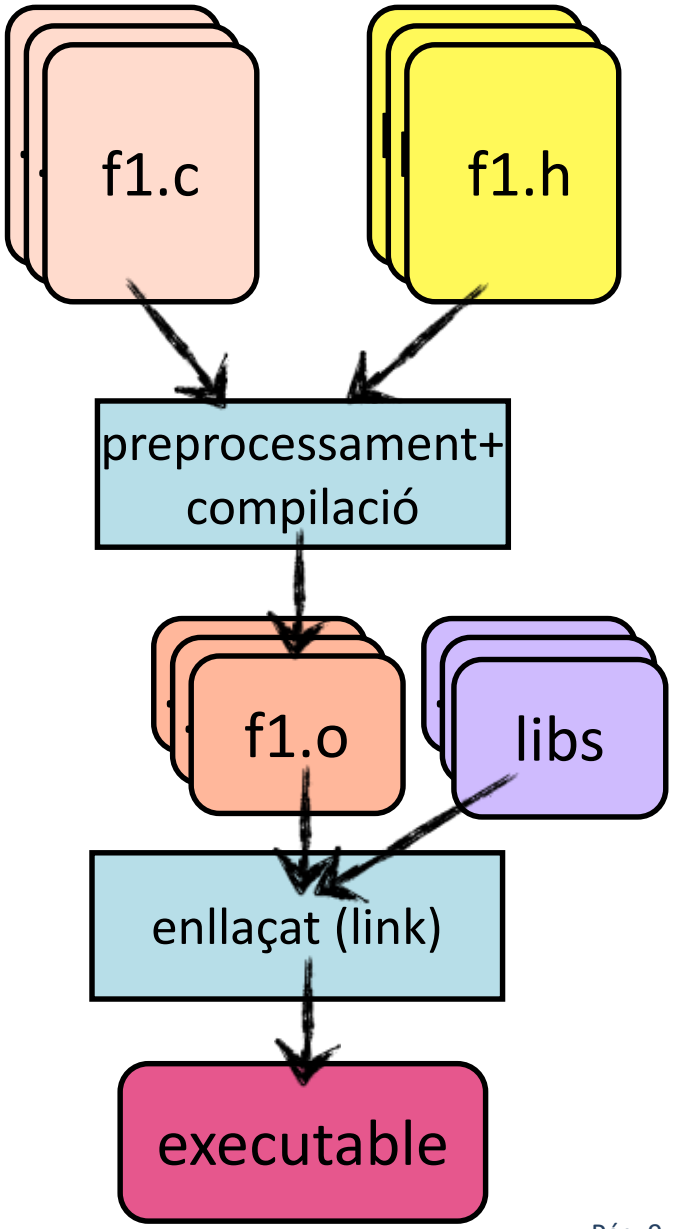
```
#include <stdio.h>

main()
{
    printf("Hola, mundo!\n");
}
```

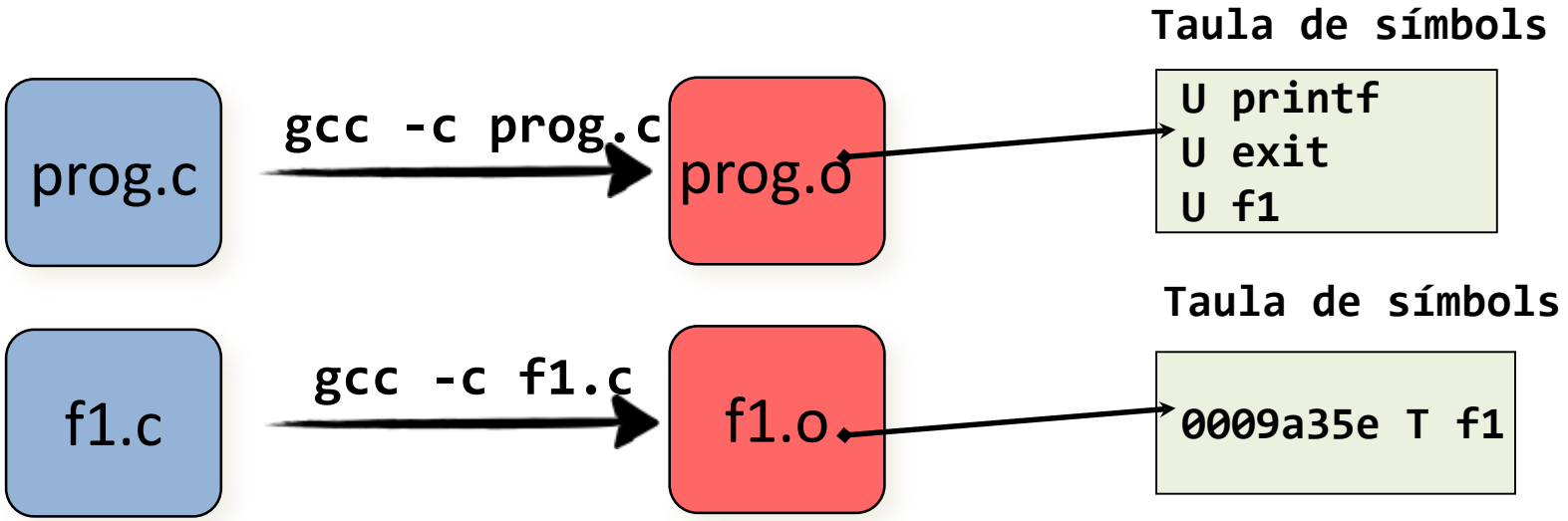
- Cal **compilar i enllaçar**

  $gcc holamundo.c -o ejemplo

- Hi ha entorns de desenvolupament (IDE) que integren editor, compilador, depurador, etc.
  - Exemple: Dev C++, Eclipse



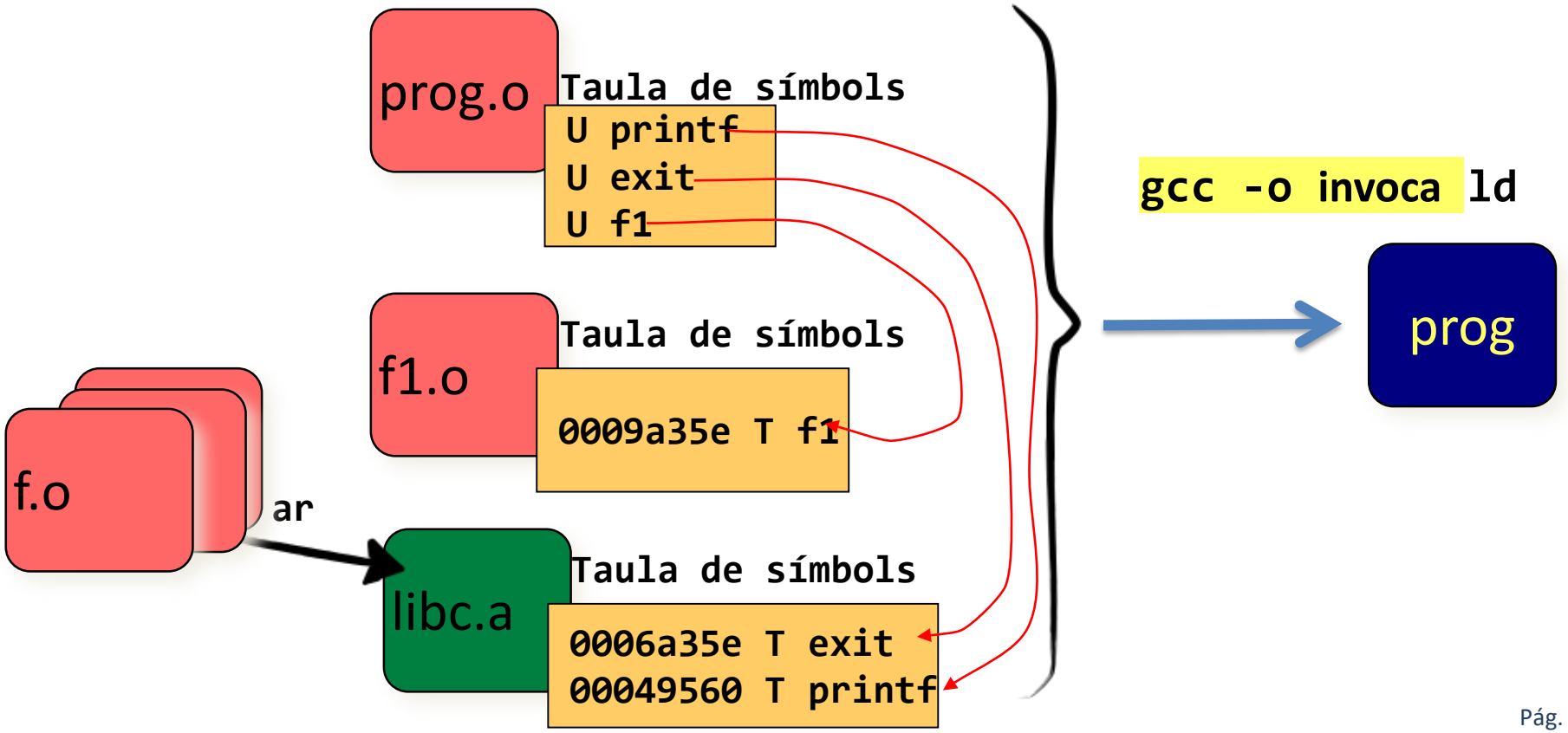
- **Compilació**
  - Generació de **codi màquina reubicable**
  - Generació de la taula de símbols amb les dependències d'altres móduls



Procés de compilació i enllaçat

- Procés d'**enllaçat o muntatge** (*link*)
  - **Resolució de referències creuades**: enllaçat de símbols no resolts amb altres mòduls i biblioteques.
  - Generació del fitxer executable

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



Contingut

- Introducció
- Procés de compilació i enllaçat
- **Elements d'un programa C**
- Sentències de control de flux
- Tipus de dades derivats
- Funcions
- Preprocessador i biblioteques

Elements d'un programa en C

- Bàsicament el *C* està compost pels elements següents :
  - Comentaris
  - Identificadors
  - Paraules reservades
  - Variables i tipus de dades
  - Constants
  - Operadors
  - Sentències

Elements d'un programa en C

- **Comentaris**
  - Dos tipus **/*i */** i **//** fins el final de línia
- **Identificadors**
  - Nom que s'assigna als distints elements del programa (variables, funcions)
    - No vàlid que comencen per nombre o guió (-)
    - No utilitzar caràcters reservats com {},(), etc.
- **Paraules reservades**
  - Totes les paraules pròpies del llenguatge com: *if, else, float,* ….
- DISTINGEIX ENTRE MAJÚSCULES I MINÚSCULES

Java==C

```
/* Comentari de moltes línies
 açò també
 */

// Comentari fins final de línia

// Identificadors vàlids
a
b12
la_variable_es_llarga
// Identificadors NO vàlids
3b
b-s
```

Elements d'un programa en C

- **Literals**
  - **Enters**
    - Hexadecimal: 0x2f, 0xFFFF, 0x123
    - Octal: 027, 035
  - **Reals** (notació exponencial)
    - 1.04E-12, -1.2e5
  - **Caracters**
    - 'c', 'b', '\n','\t','\0'
  - **Cadenes**
    - "Joan"
- **Constants**
  - Precedides per **"#define"**

```
#define PI 3.141593
#define CERT 1
#define FALS 0
#define AMIGA "Marta"
```

Java==C

Elements d'un programa en C

Java~=C

- **Variables**
  - Tenen un tipus i modificadors.
    `tipus var``[=valor];`
- **Tipus**
  - Caràcter **char** [=1 byte]
  - Enter
    int [=2/4 bytes ó long: 4/8 bytes]
  - Real
    Float [= 4 bytes ó double: 8 bytes]
  - Conversió (***casting***)
    - `var_ta=(tipus)var_tb`
- **Modificadors**
  - **signed**: amb signe (per defecte)
  - **unsigned**: sense signe
- **Definició de tipus**
  - `typedef tipus nou_tipus;`

```
char c;
unsigned char b; // un byte
int b;  // enter amb signe
unsigned int c; // sense signe
long l;
signed long l;  // el mateix
unsigned long l2;
float f1;
double s2;
int d= 5; // valor inicial 5

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


// definició de tipus
typedef float kg;
typedef int altura;
```

Elements d'un programa en C

- **Operadors**
  - Assignació =
  - in/decrementals ++,--
  - Aritmètics +,-,*,/,% (resto)
  - Relacionals ==,<,>,<=,>=,!=
  - Lògic ||(or), && (and)
  - Unaris: -,+,!
    - coma , : permet vàries expresions
    - sizeof: grandària en bytes d'una variable
    - adreça (&) i indirecció (*)
- **Expressions**
  - lógiques: tornen valor lògic
  - aritmétiques: tornen valor numèric

Java==C

```
int a;
a=5; //assignació

a++; // a val 6
a--; // torna a valdre 5
b=5%2; // torna 1

(2==1) // resultat=0
(3<=3) // resultat=1
(1!=1)  // resultat=0
(2==1) || (-1==-1) // 1
((2==2) && (3==3)) || (4==0) // 1

sizeof(a); // torna 4

a = ((b>c)&&(c>d))||((c==e)||(e==b)); //
expressió lògica

x=(-b+sqrt((b*b)-(4*a*c)))/(2*a);
// exp. aritmètica
```

Java==C

- **Sentències**
  - simples: expressió terminada per ;

```
float real;
espai = espai_inicial + velocitat * temps;
```

  - buida o nul·la: ;

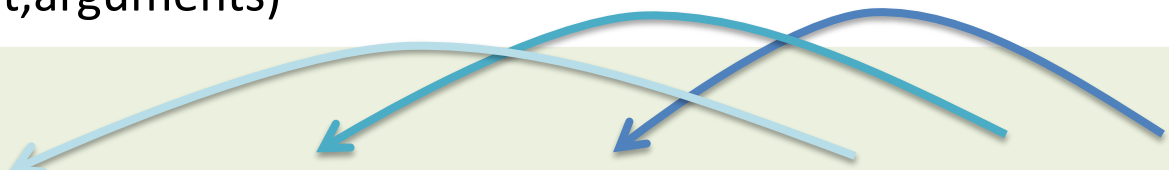
```
;
```

  - blocs: comencen per { i acaben per }

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

Elements d'un programa en C

- **Entrada i eixida**
  - scanf (format,arguments)
  - printf (format,arguments)



```
printf("Text: %s, Enter: %d, Float: %f\n", "roig", 5, 3.14);

MOSTRA POR CONSOLA
Text: roig, Enter: 5, Float: 3.14);
```

  - EJEMPLOS

```
printf("Hola món");
printf("El nombre 28 és %d\n", 28);
printf("Imprimir %c %d %f\n", 'a', 28, 3.0e+8);

scanf("%f", &nombre);
scanf("%c\n", &lletra);
scanf("%f %d %c", &real, &enter, &lletra);
scanf("%ld", &enter_llarg);
scanf("%s", cadena);
```

- Practica amb exemples:
  - Calcula el quadrat d'un nombre

quadrat.c

```
#include <stdio.h>
main()
{
    int nombre;
    int quadrat;
    printf("Introduiu un nombre: ");
    scanf("%d", &nombre);
    quadrat = nombre * nombre;
    printf("El quadrat de %d és %d\n", nombre, quadrat);
}
```

- Introducció
- Procés de compilació i enllaçat
- Elementos de un programa C
- **Sentències de control de flux**
- Tipus de dades derivats
- Funcions
- Preprocessador i biblioteques

Sentències de control de flux

- **Sentència if**

```
if (expressio)
    sentencia;
```

- **Sentència if ... else**

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

- **Sentència if ... else múltiple**

```
if (expressio1)
   sentencia1;
else if (expressio2)
   sentencia2;
[else
   sentencia3;]
```

Java==C

```
if (a > 4)
b = 2;

if (b > 2 || c < 3)
{
   b = 4; c = 7;
};

// opció amb else
if (d < 5)
{
   d++;
}
else
{
   d--;
};

// opció amb else
if (IMC < 20)
      printf(”prim");
else if (IMC <= 25)
      printf("normal");
else
      printf(”gros");
```

Sentències de control de flux

- **Sentència "switch"**

```
switch (expressio) {
   case expressio_ct_1:
     sentencia_1;
     break;
   case expressio_ct_2:
     sentencia_2;
     break;
   ...
   case expressio_ct_n:
     sentencia_n;
     break;
   [default:
     sentencia;]
```

"expressió" ha de ser un tipus enter (int, long) o caràcter (char)

Java==C

```
switch (a) {
   case 5:
    printf("aprovat");
    break;
   case 6:
    printf("bé");
    break;
   case 7:
   case 8:
    printf("notable");
    break;
   case 9:
    printf("excel·lent");
    break;
   case 10:
    printf("matrícula");
    break;
   default:
    printf("suspés");
}
```

Sentències de control de flux

- **Bucle "while"**

```
while (expressio)
    sentencia;
```

- **Bucle "do... while"**

```
do
    sentencia;
while (expressio);
```

- **Sentències "break", "continue"**
  - `break;`
    - termina el bucle
  - `continue;`
    - salta al fi del bucle

Java==C

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

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

Java==C

- **Bucle “for”**

```
for (inicializacio; expresio_de_control; actualizacio)
    sentencia;
```

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

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

Sentències de control de flux

- Practica amb exemples

sumaserie.c

```
#include <stdio.h>

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

sumaserie2.c

```
#include <stdio.h>

main()
{
  int N, suma, j;
  do
    {
      /* llegir el nombre N */
      printf("Introduiu N: ");
      scanf("%d", &N);
      suma = 0;
    for (j = 0; j <= N; j++)
     /* bucle anidat */
            suma = suma + j;
      printf("1 + 2 + … + N = %d\n",
      suma);
    } while (N > 0);  /* fi del bucle do */
}
```

- Introducció
- Procés de compilación i enllaçat
- Elements d'un programa C
- Sentències de control de flux
- **Tipus de dades derivats**
- Funcions
- Preprocessador i biblioteques

Tipus de dades derivats

- **Arrays** (llistes o vectors)
  - Definició

    ```
    tipus var[tam];
    ```

  - Accés

    ```
    var[enter];
    ```

  - L'índex comença en 0
- **Matrius**

  ```
  tipus var[tam1][tam2];
  ```

- **Còpia de vectors**
  - funció "**memcpy**"

    ```
    memcpy(v1,v2,tam)
    ```

    - Se còpia el vector v2 en v1

```
// arrays: definició i accés
int v[10];
int v2[10];
suma = 0;
int i;
v[1] = 5;
for (i=0; i< 10; i++)
{
  suma = suma + v[i];
}

// còpia de vectors
int m[10][5];
int i,j;
for (i=0; i < 10; i++)
{
    for (j=0; j < 5; j++)
    {  suma = suma + m[i][j];
    }
}
memcpy(v2,v, sizeof(v));
```

- **Punters**
  - Un punter és una variable que conté l'adreça d'altre objecte
  - Es defineix com:

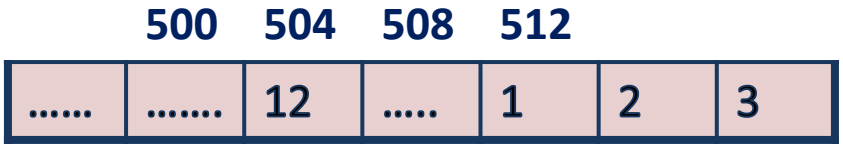
    ```
    tipus *nom;
    ```

  - L'**operador &** obté l'adreça d'una variable
  - Es denomina **indirecció** a tornar la dada apuntada pel punter

    operador *

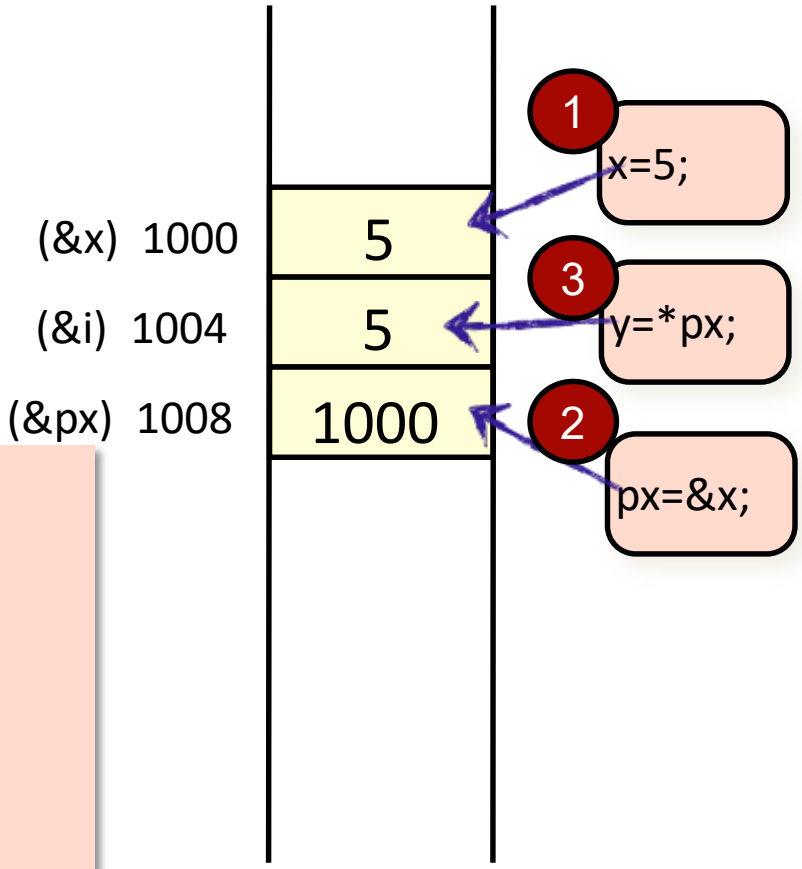
```
int b;
int x = 12;
int *p;
int N[3] = { 1, 2, 3 };
char *pc;// Punter a caràcter
p = &x; // p val 504 (apunta a x)
b = *p;  // Indirecció: b = 12
*p = 10; // Modifica contingut x
p = N;  // p apunta a N I val 512)
```

Suposem que el compilador assigna memòria a partir de l'adreça 500 i que els enters ocupen 4 bytes



Tipus de dades derivats

- **Punters**
- Practica amb exemples



punters.c

```
#include <stdio.h>
main()
{
  int x; /* variable entera */
     int y;  /* variable entera */
     int *px; /* punter a enter */
     x = 5;
     px = &x; /* px = adreça de x */
     y = *px; /* y = conté l'adreça
                 emmagatzemada en px */

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

- **Cadenes**
  - Array de caracters (**char**)
    - Es declara indicant longitut o amb punter

```
char cadena[10];
char* cadena;
```

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

  - Acaben amb un caràcter nul **“\0”**
  - Funció per copiar cadenes

strcpy(cad1, cad2)

Còpia la cadena cad2 en cad1

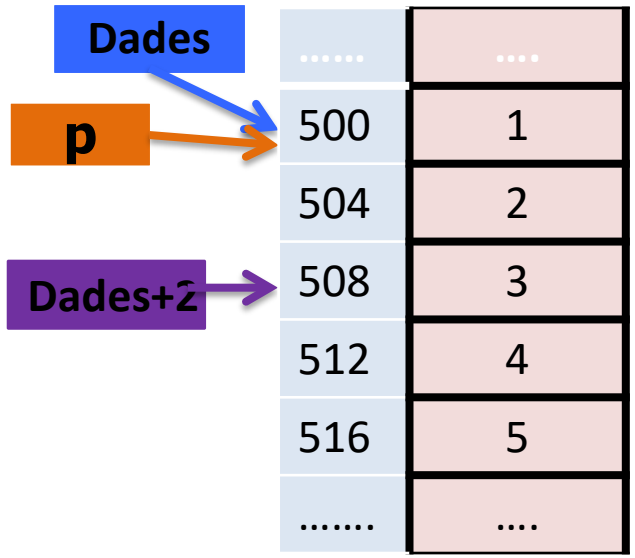
```
char ciutat[20] = "San Sebastián";
char nom[] = ”PEPITO PEREZ";
char nom[20];
strcpy(nom, nom2);
```

- Aritmètica de punters
  - C permet fer operacions amb variables de tipus punter
  - Operadors increment/decrement (++/--).
  - Suma i resta (desplaçament de la posició)
  - El desplaçament és sempre del tipus a què apunta la variable
  - Exemples:

```
int Dades[5] = {1,2,3,4,5};
int  *p;
int i;
int b;
p = Dades+2;  // p apunta al 3er element(508)

p = Dades;          // p apunta a Dades (500)

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



Tipus de dades derivats

- Altra classe de punters
  - **Punter genèric**: el seu tipus és **"void"** i poden apuntar a qualsevol tipus de dada
  - **Punter nul**: és una variable punter que val 0 o **NULL**
    - El valor NULL s'utilitza per a indicar que hi ha hagut algún error o que el punter no apunta a cap dada

```
void *v;
int i[10];
int a;
v = i;
a = (int)v;      // Cal fer casting
v = malloc(100000000); // Hi haurà error?
if (v == NULL) exit(-1);
```

Tipus de Dades Derivades

- **Estructures**
  - Definició

```
struct nom_estructura
{ tipusDada1 membre_1;
  tipusDada2 membre_2;
    ...
  tipusDadaN membre_N;
};
```

  - Declarar una variable

```
struct nom_estruc v;
```

  - Accés als membres

```
v.membre
v->membre
```

```
//definició d'una estructura
struct CD
{
   char titol[100];
   char artista[50];
   int nombre_cancons;
   int any;
   float preu;
};
//declaració d'una variable
struct CD cd1;
//accedint a l'estructura
strcpi(cd1.titol, "La Boheme");
strcpi(cd1.artista, "Puccini");
cd1.nombre_cancons = 2;
cd1.any = 2006;

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

Tipus de dades derivats

- **Vector d'estructures**
  - declarar com a vector

    ```
    struct nombre vector[llarg];
    ```

  - accés als membres

    ```
    v[i].membre
    v->membre
    ```

- Pas d'estructures a funcions
  - Per valor -> molt costós
  - Millor per referència (el punter)

```
//definició d'una estructura
struct CD
{
  char titol[100];
    char artista[50];
    int nombre_cancons;
    int any;
    float preu;
};
struct CD col[100];

for (i = 1; i< 100; i++)
  total = col[i].preu;
pcd = &col[10];
//pcd apunta a l'onzé cd
pcd->preu = 16.5;
```

```
function imprimeix_cd (struct cd *pcd) {
    printf("Preu = %d\n", pcd->preu);
    // Es pot modificar
}

// Invocant la funció imprimeix
imprimeix_cd(&cd1);
imprimeix_cd(&col[10]);
```

- Practica amb exemples (I): Gestió d'un magatzem

magatzem.c

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

typedef struct {
  char peza[20];// Tipus de peça
  int quantitat;// Nombre de peces
  float preu_unitat;// Preu de peça
  char existeix;// Comprova si
          // existeix registre
} registre_peces;

main() {
     registre_peces caixes[NRECAIXES];
     int registre=0;
```

```
/* Llig les dades des de teclat */
do {
     /* Llig el nom de la peça. */
     printf("Nom de la peça => ");
     scanf("%s", caixes[registre].peza);

     /* Llig el nombre de peces. */
     printf("Nombre de peces => ");
     scanf("%d", &caixes[registre].quantitat);

     /* Llig el preu de cada peça. */
     printf("Preu de cada peça => ");
     scanf("%f",
      &caixes[registre].preu_unitat);

     /* Indica que el registro té dades, V */
     caixes[registre].existeix = 'V';
     registre ++;
} while (registre < NRECAIXES);
```

- Practica amb exemples(II): Gestión d'un magatzem

```
     /* Imprimir la informació. */
for(registre = 0; registre < NRECAIXES; registre++) {
    if(caixes[registre].existeix == 'V') {
      printf("La caixa %d conté:\n", registre + 1);
      printf("Peza => %s\n", cajas[registre].peza);
      printf("Quantitat => %d\n",caixes[registre].quantitat);
       printf("Preu => %f euros\n",caixes[registre].preu_unitat);
     }
     } /* Fi for. */
} /*fi main*/
```

- Introducció
- Procés de compilació i enllaçat
- Elements d'un programa C
- Sentències de control de flux
- Tipus de datos derivats
- **Funcions**
- Preprocessador i biblioteques

Funcions

- C és un llenguatge basat en **funcions**
  - ha d'haver sempre una funció main
- Java és un llenguatge basat en **objectes**….
- **Definició d'una funció en C**

```
tipus_retorn nom_funcio(tipus1 arg1,..., tipusN argN)
{
    [declaració de variables locals;]
    codi executable
    [return (expressió);]
}
```

- **Declaració d'una funció en C**
  - Es pot declarar una función per a poder usar-la abans de definir-la

```
tipus_retorn nom_funcio(tipus1 arg1,..., tipusN argN)
```

- **Crida a una funció**

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

- **Pas de paràmetres per valor**

main.c

```
#include <stdio.h>

// declaració

double valor_abs(double dada);

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

valor.c

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

- Pas de paràmetres **per referència**
  - en la definició de funció, **a l'argument li precedeix "*"**

```
tipus_retorn nom_funcio(tipus1 *arg1,..., tipusN argN)
```

  - en la crida, a l'argument li **precedeix "&"**

```
retorn = nom_funcion(&arg1,..., tipusN argN)
```

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

- **Àmbit de les variables**
  - **Globals**
    - es declaren fora de qualsevol funció
    - es poden accedir des de qualsevol funció del fitxer
  - **Locals**
    - es defineixen dins de les funcions
    - només accessibles dins de la funció
  - **Estàtiques**
    - són locals però guarden el seu valor

Global_local.c

```
#include <stdio.h>

void funcio1(void);

int a = 10; /* variable global */

main()
{
    int b = 2; /* variable local */
    a++;
    funcio1();
    a++;
    printf("a= %d, b= %d\n", a, b);
    a++;
    funcio1();
}

void funcio1(void)
{
    static int c = 4;/* variable estàtica */
    printf("a= %d, c= %d\n", a, c);
    c++;
    return;
}
```

- Practica amb exemples

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 llegir (float *a, float *b) {
    printf("Dóna'm valors a i b:\n");
    scanf("%f %f", *&a, *&b);
}

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

- **Paràmetres** d'entrada en la **línia d'ordres del shell**
  - En invocar un programa per la línia d'ordres, podem passar-li paràmetres
    - Exemple: **$suma 2 3**
  - Es defineix el main de la manera següent:
    - **int main (int argc, char *argv[])**

    on **"argc"** és el nombre d'arguments i **"argv"** un array amb els arguments. El primer argument **argv[0]** és el nom 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 és %d\n", sum1+sum2);
    }
    else {
    printf("Ús de l'ordre: %s arg1 arg2\n", argv[0]);
    }
}
```

- Introducció
- Procés de compilació i enllaçat
- Elements d'un programa C
- Sentències de control de flux
- Tipus de datos derivats
- Funcions
- **Preprocessador i biblioteques**

Preprocessador i biblioteques

- Preprocessador I
  - Abans de compilar, hi ha una fase anomenada preprocessar:
    - Substitueix macros (#define, #undef)
    - Compilacions condicionals (#if, #ifdef)
    - Incloure arxius (#include)
    - Ordres directes al compilador (#pragma i #error)
  - Ordre **#define**
    - Permet **definir macros** (constants)

```
#define E 2.7182
#define g 9.81
```

    - Funcions **inline**

```
#define SUMA(c,d) (c + d)
#define MAX(a, b) (((a) > (b)) ? (a) : (b))
if (MAX(altura1, altura2) == 5) {
```

  - Ordre **#undef MACRO** -> elimina la definició d'una macro

- Preprocessador II
  - Compilació condicional

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

    - Permet compilar un bloc de codi de forma opcional
  - Més opcions
    - `#ifndef`
    - `#elif`

```
#ifdef MODE_64BITS
   // Codi per a 64 bits
   int x = 5;
#else
   // Codi per a 32 bits
   long x = 5;
#end

#ifdef CPU_ARM7
   // Codi per a ARM
   xARM.b = 5;
#elif CPU_INTEL
   // Codi per a INTEL
   x.b = 5;
#else
   // Resta de CPUs
#end
```

Preprocessador i biblioteques

- Llibreries o biblioteques
  - Conjunt de funcions d'ús comú
    - matemàtiques, entrada/eixida, temps, etc.
  - Les funcions estan declaradas en un **fitxer ".h"** anomenat capçalera (header)
  - Per a usar una funció cal incloure el fitxer:

```
#include <nom_fich.h>  // Llibrería del sistema
#include "nom_fich.h"  // Directori local
```

  - Per exemple printf està en la llibreria "stdio" que està definida en la capçalera **"stdio.h"**

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

Preprocessador i biblioteques

- Biblioteca string (string.h)
  - **`char *strcat (char *cad1, char *cad2)`**
    - Concatena cad2 a cad1 tornant la dirección de cad1. Elimina el nul de terminació de cad1 inicial
  - **`char *strcpi (char *cad1, char *cad2)`**
    - Còpia la cadena cad2 en cad1, sobreescrivint-la. Torna l'adreça de cad1. La longitut de cad1 ha de ser suficient per albergar cad2
  - **`int strlen (char *cad)`**
    - Torna el nombre de caràcters que emmagatzema cad (sense comptar el nul final)
  - **`int strcmp (char *cad1, char *cad2)`**
    - Tornar
      - > 0 : cad1 > cad2
      - 0: cad1 == cad2
      - <0: cad1 < cad2

Preprocessador i biblioteques

- **Gestió de memòria** (stdlib.h o malloc.h)
  - `void *malloc(int bytes)`
    - Reserva n bytes de memòria. Torna un punter al principi de l'espai reservat.
  - `free(void *p)`
    - Allibera un bloc de memòria al què apunta p.
- i moltes, moltes més....