Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Part 1: Introduction

SUT01: C language

fSO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Goals and bibliography

- **Goals**
  - To introduce the **C programming language** highlighting its differences with Java
  - To understand **compiling and linking** processes
  - To introduce the **pointer** concept in C
  - To practice with **funtion calls** and **parameter passing**, both by value and by reference

- **Bibliography**
  - C reference card
  - “The C Programming Language”, B.W. Kernighan y D.M. Ritchie, 2nd Ed. Prentice-Hall
  - “C++ Estándar”, Enrique Hernández et al. Paraninfo, 2002.

- **Introduction**
- Compiling and linking
- Elements of a C program
- Flow control sentences
- Derived data types
- Functions
- Preprocessor and libraries

Linking required in C but not done in JAVA

The same in C and JAVA

C pointers

C is a function based lenguage

- **Characteristics of the C language**
  - **General-purpose programming language**, very suitable for system programming and embedded systems (UNIX was written in C).
  - **Relatively small**: only offers controls senteces and funtions
  - **Very portable**: there are compilers for all platforms
    - Java is portable at binary level
  - **Very efficient** generated code : both in speed and size. It is a compiled language
  - **Compiling and linking**: from source code (text) a complete binary executable is generated
    - Java is interpreted: after compiling the byte codes contained in .class file are interpreted to be executed in the Java virtual machine

Introduction

- History of the C language
  - C was originally designed in 1972 to build the first Unix OS on the DEC PDP-11 by Dennis Ritchie at Bell Laboratories
  - In the 80's the most programming was done in C
  - In 1983 C + + appears (object oriented programming)
  - C is standardized
    - ANSI C, ISO C.
- **Java and C**
  - Java is based on C / C + +
  - The control structures are the same as in C
  - Class syntax is similar to C ++
  - **Java eliminates direct memory management (explicit pointer handling)**

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)

- Introduction
- **Compiling and linking**
- Elements of a C program
- Flow control sentences
- Derived data types
- Functions
- Preprocessor and Libreries

Linking required in C but not done in JAVA

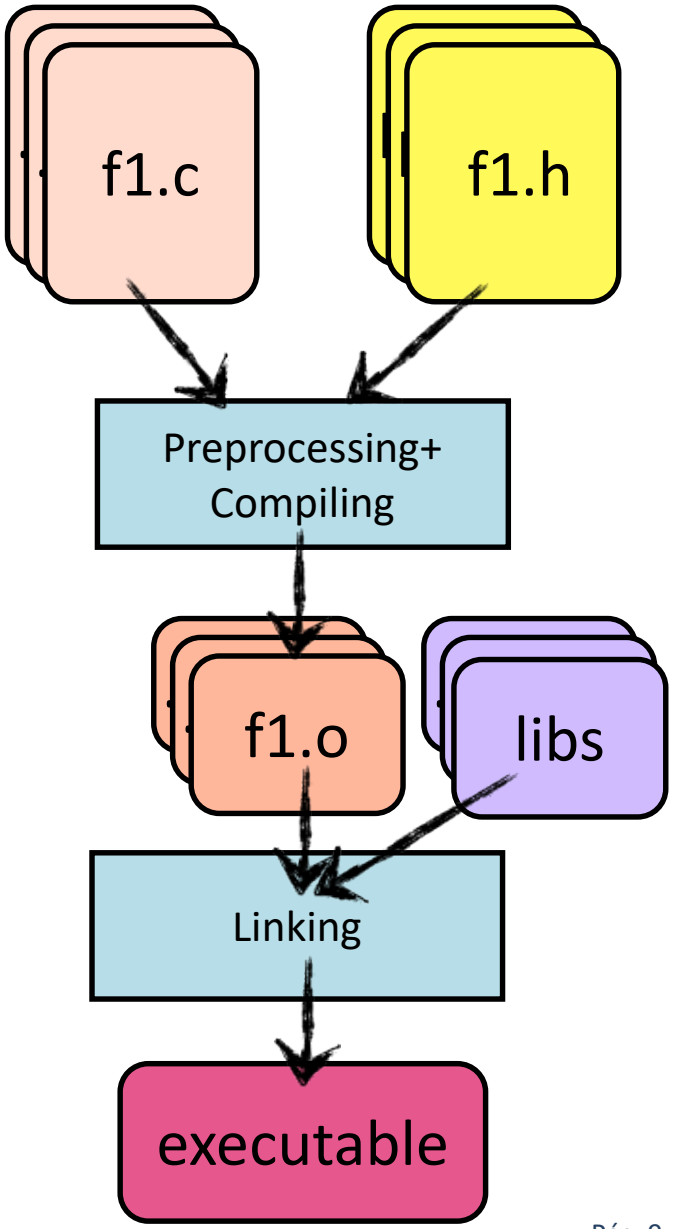
Compiling and linking

- **Source files:** contain source code, **".c "** files

```
/*  source.c */
#include   <stdio.h>
main()
{
 printf ("Hello world!\n");
}
```

- **Header files:** with data declaration and funtion prototypes, **".h"** files

```
/*  header.h */
#define My_STRING  "Hello World"
#define PI     3.1415925
#define MAX(A,B)  ((A)>(B)?(A):(B))
```

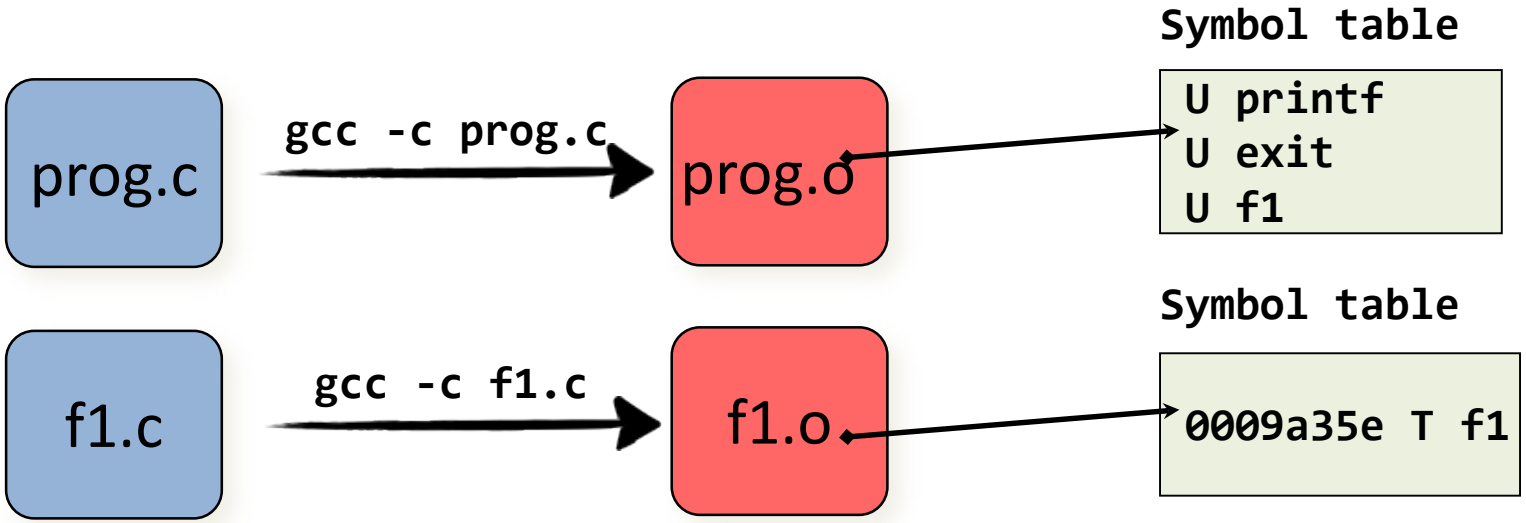


- It is required to compile and link in order to generate an executable file

  $ gcc helloworld.c -o example

- There are development environments (IDE) that include the editor, compiler, debugger, etc..
  - Dev C++, Eclipse, etc

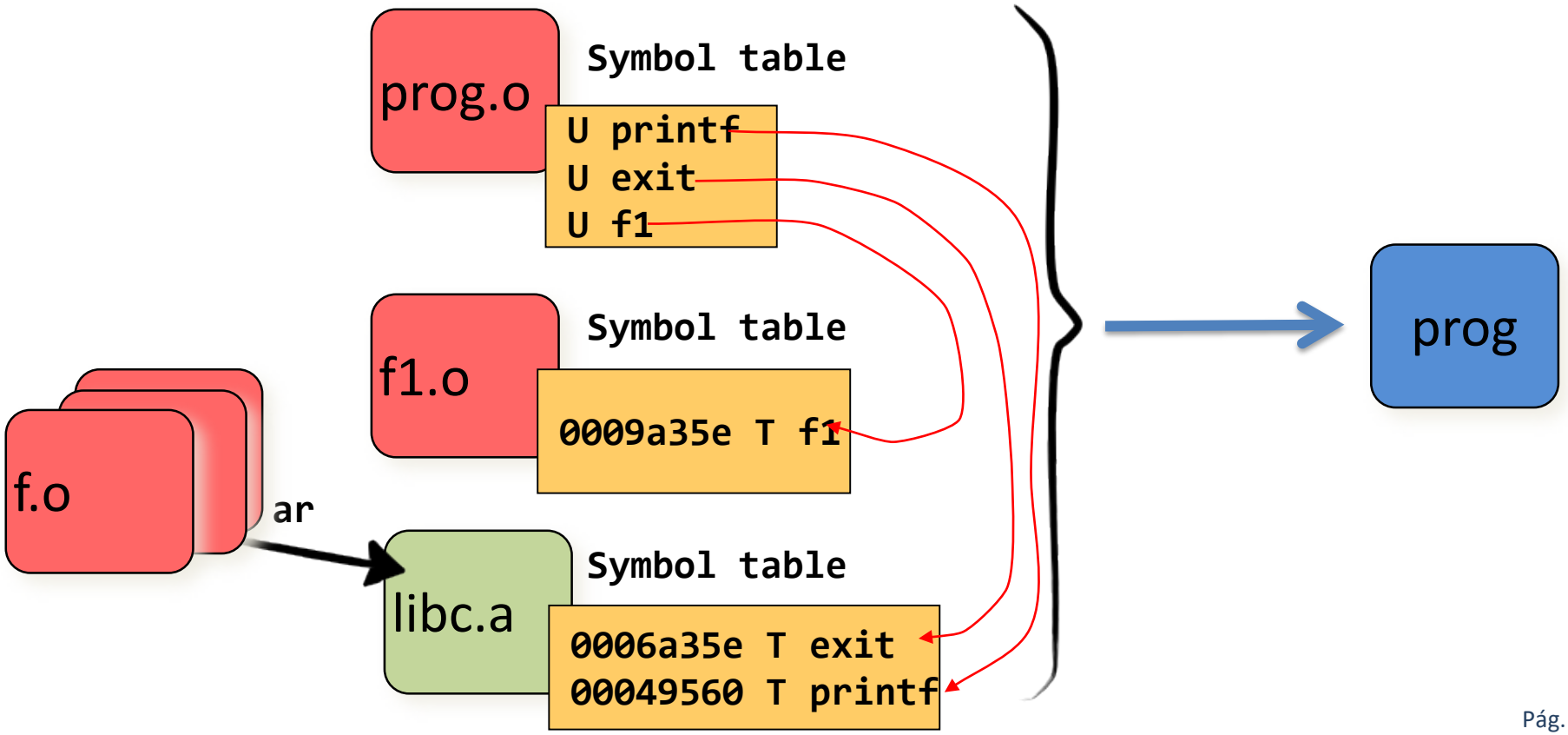
- **Compilation**
  - Generation of **relocatable machine code**
  - Symbol table generation with other modules dependencies



Compiling and linking

- Linking process
  - **Crossing references resolution**: not solved symbol linking with other modules and libraries
  - Executable file generation

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



Content

- Introduction
- Compiling and linking
- **Elements of a C program**
- Flow control sentences
- Derived data types
- Functions
- Preprocessor and libraries

Elements of a C program

- Basically C is composed by the following elements:
  - Comments
  - Identifiers
  - Reserved words
  - Variables and data types
  - Constants
  - Operators
  - Expressions
  - Sentences

Elements of a C program

- **Comments**
  - Two types / * and * / and / / to end of line
- **Identifiers**
  - Names assigned to the parts of a program (variables, functions)
    - Can not begin with a number
    - Can not contain the dash (-) character
    - Reserved characters ('{', '} ', '(', ') ', etc) and words can not be used
- **Reserved words**
  - All included in the language definition: *if, else, float, ….*
- **C (as Java) distinguishes upper and lower case**

Java==C

```
/* Several lines comment
   this line too
 */

// One line only comment

// Valid identifiers
a
b12
la_variable_es_larga
// Non valid identifiers
3b
b-s
```

Elements of a C program

- **Literals**
  - **Integers**
    - Hexadecimal: 0x2f, 0xFFFF, 0x123
    - Octal: 027, 035
  - **Real numbers** (exponential notation)
    - 1.04E-12, -1.2e5
  - **Characters**
    - 'c', 'b', '\n','\t','\0'
  - **Strings** "Juan"
- **Constans**
  - Declared by **"#define"**

```
#define PI 3.141593
#define TRUE 1
#define FALSE 0
#define FRIEND "Marta"
```

Same as Java

Elements of a C program

- **Variables**
  - They are declared with a data type and optional type modifier

    `type var[=value];`
- **Types**
  - Character **char** [=1 byte] (in Java type **byte**, **char** in Java is 2 bytes UNICODE)
  - Integer
    - **int** [=2/4 bytes]
    - **long** [=4/8 bytes]
  - Real (IEEE 754)
    - **float** [= 4 bytes]
    - **double** [= 8 bytes]
  - **Type casting**
    - `var_type_a=(type_a)var_type_b`
- **Modifiers (not defined in Java)**
  - **signed** (by default)
  - **unsigned**
- **Custom type definition**
  - `typedef type type_name;`

Java~=C

```
char c;
unsigned char b; // one byte
int b;  // signed integer
unsigned int c; // unsigned
integer
long l; // signed long integer
signed long l;  // the same
unsigned long l2;
float f1;
double s2;
int d= 5; // initial value 5

b = (int)c; // b becomes
integer
f1 = (float)c; // c becomes
float


// type definition
typedef float kg;
typedef int altura;
```

Elements of a C program

- **Operators**
  - Assignation =
  - Inc/Decremental ++,--
  - Arithmetic +,-,*,/,% (module)
  - Relationals ==,<,>,<=,>=,!=
  - Logic ||(or), && (and)
  - Unary: -,+,!
    - sizeof: variable memory size in bytes
    - Address (&) and indirection (*)
- **Expressions**
  - logic: give a logic value
  - arithmetic: give a numeric value

Java==C

```
int a;
a=5; //assignation

a++; // a is 6
a--; // a is back to 5
b=5%2; // gives 1

(2==1) // result=0 (false)
(3<=3) // result=1 (true)
(1!=1)  // result=0 (false)
(2==1) || (-1==-1) // result=1
(true)
((2==2) && (3==3)) || (4==0) //
result=1(true)

sizeof(a); // gives 4

a =
((b>c)&&(c>d))||((c==e)||(e==b));
// logic expression

x=(-b+sqrt((b*b)-(4*a*c)))/(2*a);
// arithmetic expression
```

Java==C

- **Sentences**
  - Simple: line ended by ;

```
float real;
space = initial_space + speed * time;
```

  - Empty

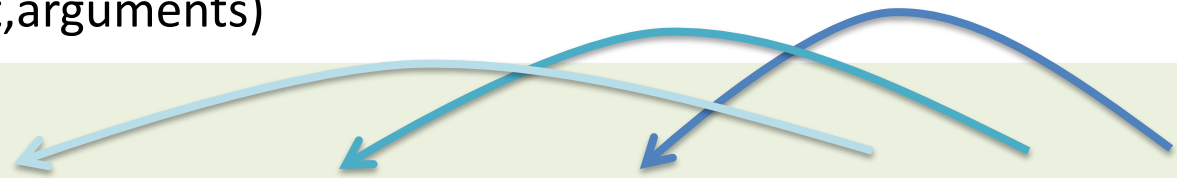
```
;
```

  - Block: delimited by { and }

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

Elements of a C program

- **Input and output**
  - scanf (format,arguments)
  - printf (format,arguments)



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

OUTS ON THE CONSOLE
Text: red, Integer: 5, Float: 3.14);
```

  - EXAMPLES

```
printf("Hello world \n");
printf("Number 28 is %d\n", 28);
printf("Print %c %d %f\n", 'a', 28, 3.0e+8);

scanf("%f", &number);
scanf("%c\n", &character);
scanf("%f %d %c", &real, &integer, &character);
scanf("%ld", &long_integer);
scanf("%s", string);
```

- Hands on
  - Compute the square of a number

square.c

```
#include <stdio.h>
main()
{
   int number;
   int square;
   printf("Please, write a number: ");
   scanf("%d", &number);
   square = number * number;
   printf("The square of %d is %d\n", number, square);
}
```

Generate the executable file

```
$gcc -o square square.c
```

Run it

```
$./square
```

- Introduction
- Compiling and linking
- Elements of a C program
- **Flow control sentences**
- Derived data types
- Functions
- Preprocessor and libraries

The same in C and JAVA

Flow control sentences

- **if**

```
if (expresion)
    sentence;
```

- **if … else**

```
if (expresion)
    sentence1;
else
    sentence2;
```

- Multiple **if … else**

```
if (expression1)
   sentence1;
else if (expression2)
   sentence2;
[else
   sentence3;]
```

Java==C

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

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

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

// multiple if … else
if (IMC < 20)
    printf(“thin");
else if (IMC <= 25)
    printf("normal");
else
    printf(“fat");
```

Flow control sentences

- **Sentence "switch"**

```
switch (expression) {
   case expression_cte_1:
     sentence_1;
     break;
   case expression_cte_2:
     sentence_2;
     break;
   ...
   case expression_cte_n:
     sentence_n;
     break;
   [default:
     sentence;]
```

"expression" must be an integer type (int, long) or character (char)

Java==C

```
switch (a) {
    case 5:
   printf("passed");
   break;
    case 6:
   printf("acceptable");
   break;
    case 7:
    case 8:
   printf("quite good");
   break;
    case 9:
   printf("very good");
   break;
    case 10:
   printf("Excelent");
   break;
    default:
   printf("non passed");
}
```

Flow control sentences

- **Loop "while"**

```
while (expression)
    sentence;
```

- **Loop "do... While"**

```
do
    sentence;
while (expression);
```

- **Sentences "break", "continue"**
  - `break;`
    - Ends the loop
  - `continue;`
    - Jumps to the loop end

Java==C

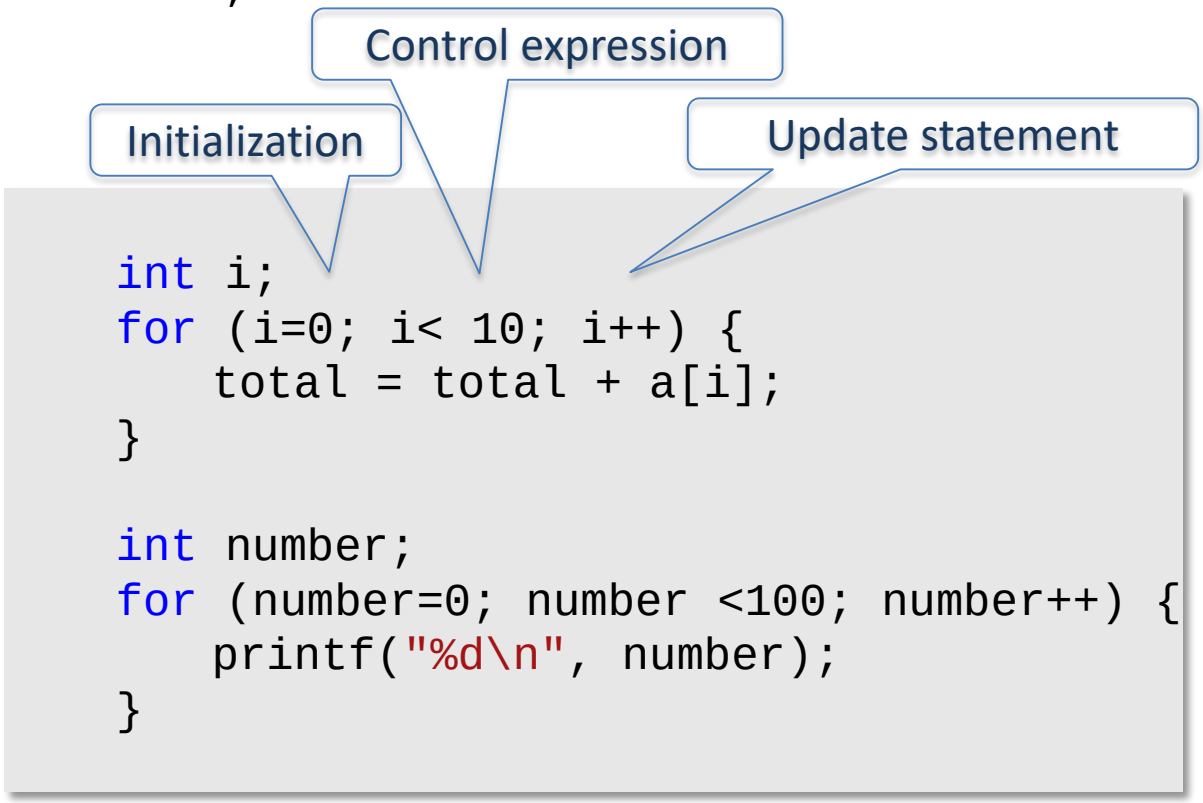
```
x = 1;
while (x < 10)
x++;
x = 1; z = 2
while (x < 10 && z != 0)
{
   b = x/z;
   x++;
   z--;
}
// Another loop
x = 1;
do {
   x++;
} while (x < 20)

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

Java==C

- **Loop "for"**

```
for (initialization; control_expression; update)
    sentence;
```



Flow control sentences

- Hands on

addseries2.c

```c
#include <stdio.h>

main()
{
  int N, add, j;
  do
  {
    /* Read N */
    printf("Introduce N: ");
    scanf("%d", &N);
    add = 0;
    for (j = 0; j <= N; j++)/*nested loop*/
      add = add + j;
    printf("1 + 2 + ... + N = %d\n", add);
  } while (N > 0);/* loop end */
}
```

Generate the executable file

```
$gcc -o addserie2 addserie2.c
```

Run it

```
$./addserie2
```

addseries.c

```c
#include <stdio.h>

main()
{
  int N;
  int add = 0; /* Read N */
  printf("N: ");
  scanf("%d", &N);
  while (N > 0) {
    add = add + N;
    N = N - 1; /* same as N-- */
  }
  printf("1 + 2 +...+ N = d\n“, add);
}
```

Generate the executable file

```
$gcc –o addserie addserie.c
```

Run it

```
$./addserie
```

Content

- Introduction
- Compiling and linking
- Elements of a C program
- Flow control sentences
- **Derived data types**
- Functions
- Preprocessor and libraries

C pointers

Derived data types

- **Arrays** (vectors)
  - Declaration

    `type var[tam];`
  - Access

    `var[entero];`
  - Indexed from 0
- **Matrices**

  `type var[size1][size2];`
- **Vector copy**
  - Function "**memcpy**"

    `memcpy(v1,v2,size)`
    - Copies vector v2 in v1

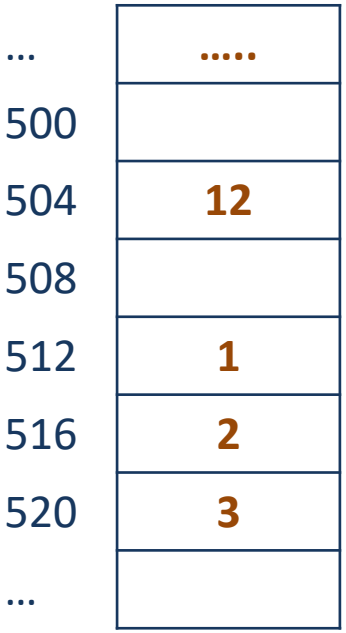
```
// arrays declaration and
access
int v[10];
int v2[10];
add = 0;
int i;
v[1] = 5;
for (i=0; i< 10; i++)
{
   add = add + v[i];
}

// vector copy
int m[10][5];
int i,j;
for (i=0; i < 10; i++)
{
    for (j=0; j < 5; j++) {
      add = add + m[i][j];
    }
}
memcpy(v2,v, sizeof(v));
```

Derived data types

- **Pointers**
  - A pointer is a variable which value is the memory address of another object
  - It is defined as: **`type *name;`**
    - **Operator &** : gets the address of a variable
    - **Operator *** : (indirection) returns the data pointed by a pointer

We assume that the compiler allocates memory from address 500 and integers occupy 4 bytes




```
int b;
int x = 12;
int *p;
int N[3] = { 1, 2, 3 };
char *pc; // Pointer to character
p = &x; // p is 504 (points to x)
b = *p; // b = 12
*p = 10; // Modifies x value
p = N; // p points to N[0] so p=512
```

- **Pointers** hands on

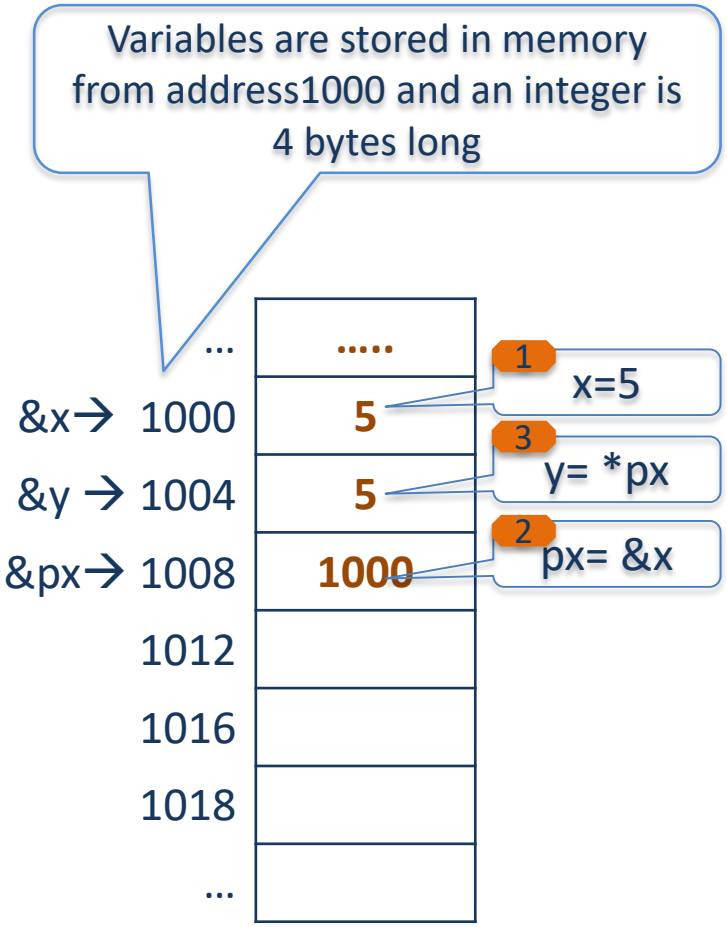
pointers.c

```
#include <stdio.h>

main()
{     int x; /* integer variable */
   int y;  /* integer variable */
   int *px; /* pointer to integer */
1  x = 5;
2  px = &x; /* px = address of x */
3  y = *px; /* y = x */

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

```
$gcc -o pointers pointers.c
$./pointers
```



Derived data types

- **Strings or array of char**
  - Se declara indicando tamaño o con puntero

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

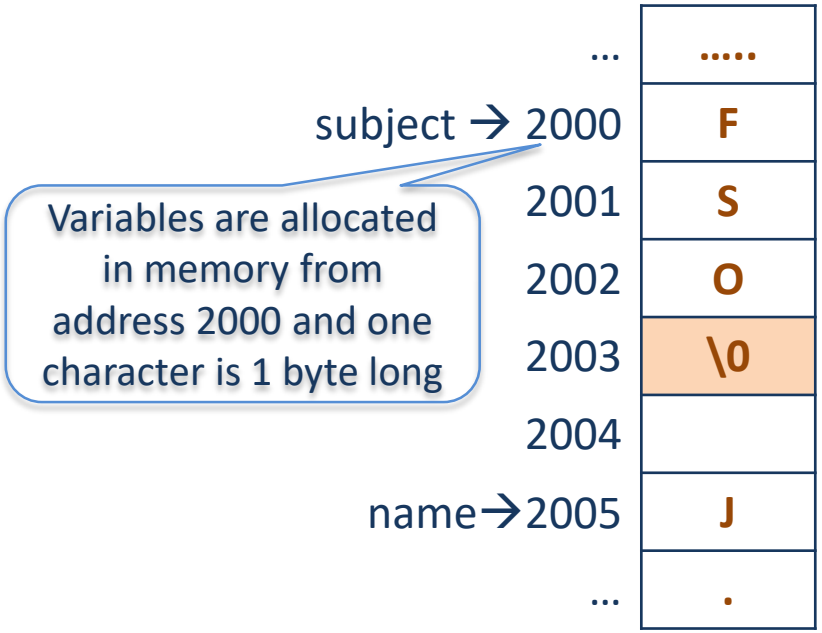
  - They end with a null character '**\0**'
  - Function to copy strings

```
strcpy (str1, str2)
```

-> It copies string str2 into str1

```
#include <stdio.h>
#define STRING_SIZE 80
main() {
  char string[STRING_SIZE];
  printf(" Enter a string: ");
  scanf("%s", string); // & is not
needed
  printf(" The string is %s\n",
string);
}
```

```
char subject[5] = "FSO";
char name[] = "J. Perez";
char name2[20];
strcpy(name2, name);
```

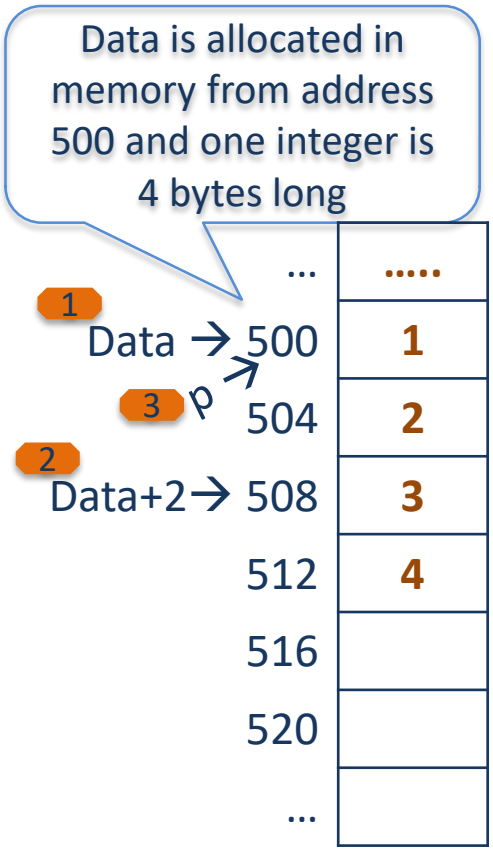


Derived data types

- Pointer arithmetic
  - C can perform various operations with pointer variables
  - Increment/Decrement operators (++/--).
  - Addition and subtraction (position shift)
  - Shift size corresponds to the data type that types the variable

aritpointers.c

```
#include <stdio.h>
main(){
1 int Data[5] = {1,2,3,4,5};
  int  *p;
  int i;
  int b;
2 p = Data+2;// p points to the 3rd
             // element at address 508
3 p = Data; // p points to Data (500)
  for (i = 0; i < 5; i++) {
       printf("Data[%u]=%u \n", i, Data[i]);
       printf("Data[%u]=%u \n", i, *p++);
  }
}
```




```
$gcc -o aritpoiters aritpointers.c
$./aritpointers
```

Derived data types

- Another class of pointers
  - **Generic pointer:** its type is "void" and can point to any data
  - **Null pointer:** is a pointer variable whose value is 0, the value is represented by NULL
    - The value NULL is used to indicate that an error has happened
    - The value NULL is used to indicate that the pointer does not point to any data

```
void *v; // generic pointer
int i[10];
int a;
v = i;
a = *(int*)v;    // casting should be done
v = malloc(100000000); // big memory allocation
if (v == NULL) exit(-1); // checkig error
```

Derived data types

- **Structures**
  - Definition

```
struct struct_name
{ type1 member_1;
  type2 member_2;
   ...
  typeN member_N;
};
```

  - Variable declaration

```
struct struc_name v, *pv;
```

  - Member access

```
v.member
pv->member
```

```
// structure definition
struct CD
{
  char title 100];
  char artist[50];
  int num_songs;
  int year;
  float price;
};
// Variable declaration
struct CD cd1;
// Accessing the structure
strcpy(cd1.title, "La
Boheme");
strcpy(cd1.artist,
"Puccini");
cd1.num_songs = 2;
c1.year = 2006;

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

Derived data types

- **Structure vectors**
  - Declaration

    `struct name v[size], *pv;`
  - Member access

    `v[i].member`

    `pv->member`
- Passing structures to functions
  - By value -> very expensive
  - Better by reference (pointer)

```
//definition of a structure
struct CD
{
  char title[100];
  char artist[50];
  int num_songs;
  int year;
  float price;
};
struct CD col[100];
struct CD *pcd;

for (i = 1; i< 100; i++)
    col[i].price = 12.5;
pcd = &col[10];
//pcd points to the eleventh
cd
pcd->price = 16.5;
```

```
function print_cd (struct CD *pcd) {
    printf("Price = %d\n", pcd->price);
    // can be modified
}

// Invoking the function print
print_cd(&cd1);
print_cd(&col[10]);
```

Derived data types

- Hands on

warehouse.c

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

typedef struct {
   char part[20];// part type
   int quantity;  // part number
   float unit_price; // part price
   char available; // there are
part units
} parts_record;
```

```
main() {
   parts_record boxes[NUMBOXES];
   int record=0;
   int i;
   /* Read data from the keyboard */
do {
   /* Read the part name */
   printf("Part name => ");
   scanf("%s", boxes[record].part);
   /* Read the number of parts */
   printf(" Number of parts => ");
   scanf("%d", &boxes[record].quantity);
   /* Read the price of each part */
   printf(" Price of each part => ");
   scanf("%f", &boxes[record].unit_price);
   /* Indicate the record has data (V) */
   boxes[record].available = 'V';
   record ++;
} while (record < NUMBOXES);
```

```
/* Print the information */
for (record = 0; record < NUMBOXES; record++) {
   if (boxes[record].available == 'V') {
   printf ("Box %d contains:\n", record + 1);
   printf ("Part => %s\n", boxes[record].part);
   printf ("Quantity => %d\n", boxes[record].quantity);
   printf ("Price => %f euros\n", boxes[record].unit_price);
   }
   } /* end for */
} /* end main */
```

Content

- Introduction
- Compiling and linking
- Elements of a C program
- Flow control sentences
- Derived data types
- **Functions**
- Preprocessor and libraries

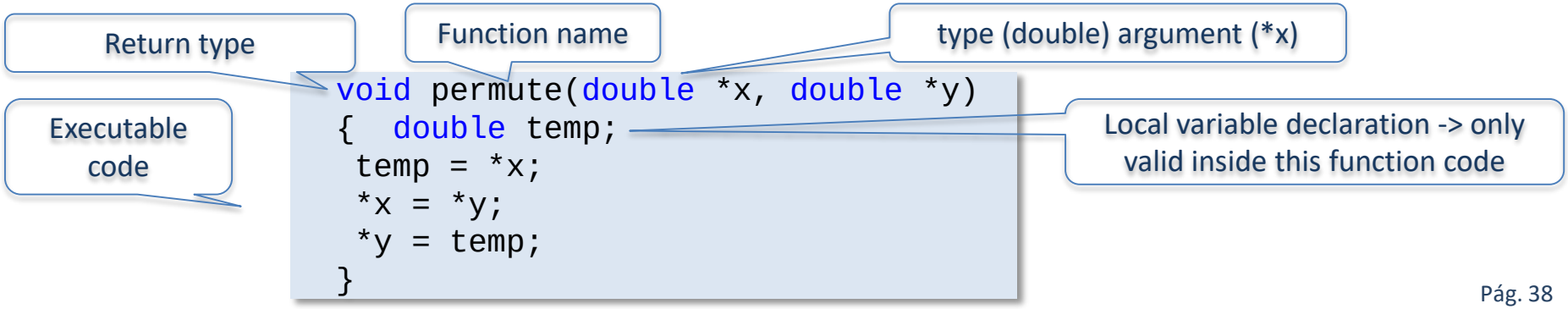
C is a function based lenguage

- C is a language based on **functions**
  - There must always be a **main** function
  - **Defining a function in C**

```
return_type function_name(type1 arg1,..., typeN argN)
{
    [local variable declarations;]
    executable code
    [return (expression);]
}
```

  - **Declaring a function in C**
    - You have to declare a function template if you use it before its implementation

```
return_type function_name(type1 arg1,..., typeN argN);
```



Functions

- **Function call**

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

- **Passing parameters by value**

twofunctions.c

```c
#include <stdio.h>
// declaration
double abs_value(double x);

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

// definition
double abs_value(double x)
{
    if (x < 0.0)
      return -x;
    else
      return x;
}
```

Calling function abs_value

Function sbs_value, retorns a double typed value

Parameter passed by value: The function receives a copy of the parameter and it works with it

- Passing parameters by reference
  - In the function definition, the argument **is preceded by "*"**

```
return_type function_name(type1 *arg1,..., typeN argN)
```

  - When calling the function, the argument **is preceded by "&"**

```
return = function_name(&arg1,..., argN)
```

```c
#include <stdio.h>
// Function definition
void permute(double *x, double *y){
void main(void) {
   double a=1.0, b=2.0;
   printf("a = %f, b = %f\n");
   permute(&a, &b);
   printf("a = %f, b = %f\n");
}
void permute(double *x, double *y){
   double temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

WARNING What it is passed to the function by reference is the **variable memory address** (not a copy of its value), the function works directly with the variable memory address, so it can change its content (value).

Function call passing parameters by reference

Declaration of reference parameter passiing

```
$gcc -o rparam rparam.c
$./rparam
```

- **Variable scope**
  - **Global**
    - They are declared outside of any function
    - They can be accessed from any function in the source file
  - **Local**
    - They are defined within functions
    - They are only accessible within the function
  - **Static**
    - They are local but keep its value between function calls

global_local.c

```
#include <stdio.h>

void funcion1(void);

int a = 10; // global variable

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

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

- Hands on

hypotenuse.c

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

void hypotenuse(float a, float b, float *h)
{
   *h = sqrt(pow(a,2) + pow(b, 2));
}
void read_catheti (float *a, float *b) {
   printf("Please enter catheti values a and b :\n");
   scanf("%f %f", a, b);
}

main() {
   float a, b, h;
   read_catheti (&a,&b);
   hypotenuse(a,b,&h);
   printf("The hypotenuse value is %f\n", h);
}
```

```
$gcc hypotenuse.c –lm -o hypotenuse
$./hypotenuse
```

- **Input parameters on the command line**
  - When invoking a C program from the command line, you can pass parameters to it as follows:
    - Example: `$addition 2 3`
  - The main function is defined as follows:

    `int main (int argc, char *argv[])`
    - **argc** is the number of arguments
    - **argv** an array that inludes the arguments values. The first argument argv [0] is the program name

add.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("Add = %d\n", sum1+sum2);
   }
   else {
   printf("Command use: %s arg1 arg2\n", argv[0]);
   }
}
```

Execution parameters

Convert the parameter values from ASCII to integer

```
$gcc -o suma suma.c
$./suma 4 8
```

Content

- Introduction
- Compiling and linking
- Elements of a C program
- Flow control sentences
- Derived data types
- Functions
- **Preprocessor and libraries**

Preprocessor and libraries

- Preprocessor I
  - Before compiling, there is a phase called preprocessing that deals with:
    - Macros (#define, #undef)
    - Conditional compilation (#if, #ifdef)
    - File inclusion (#include)
    - Direct compiler commands (#pragma y #error)
  - Command **#define** allows:
    - Defining macros (constants)

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

    - **Inline** function declaration

```
#define ADD(c,d) (c + d)
#define MAX(a, b) (((a) > (b)) ? (a) : (b))
if (MAX(height1, height2) == 5) {…}
```

  - Command **#undef MACRO** removes a macro definition

Preprocessor and libraries

- Preprocessor II
  - Conditional compilation

```
#ifdef MACRO
 // Compile if MACRO is defined
#else
 // Compile if not defined
#endif
```

    - It is possible to compile a code block as an option
  - More options
    - `#ifndef`
    - `#elif`

```
#ifdef 64BITS_MODE
    // 64bits code
    int x = 5;
#else
    // 32bits code
    long x = 5;
#end

#ifdef ARM7 _ CPU
    // ARM code
    xARM.b = 5;
#elif INTEL_CPU
    // INTEL code
    x.b = 5;
#else
    // Other CPU
#end
```

Preprocessor and libraries

- Libraries
  - Set of commonly used functions
    - math, input / output, time, etc.
  - The functions are declared in a ".h" file called header
  - To use a function the file must be included:

```
#include <name_file.h>  // System library
#include "name_file.h"  // Local Directory
```

  - For instance, `printf` is defined in the library "stdio" which is declared in the header "stdio.h"

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

Preprocessor and libraries

- **Library string (string.h)** Functions sample
  - **`char *strcat (char *str1, char *str2)`**
    - It concatenates str1 str2 in str1 returning the address of str1. It removes the initial null termination in str1
  - **`char *strcpy (char *str1, char *str2)`**
    - It copies the string str2 in str1, overwriting str1. It returns the address of str1. The size of str1 should be sufficient to accommodate str2
  - **`int strlen (char *str)`**
    - It returns the number of characters stored in cad (the termination null character is ignored)
  - **`int strcmp (char *str1, char *str2)`**
    - It returns:

```
Value > 0 if str1 > str2
Value = 0 if str1 == str2
Value < 0 if str1 < str2
```

Preprocessor and libraries

- **Memory management** (stdlib.h o malloc.h)
  - `void *malloc(int bytes)`
    - It reserves n bytes of memory and returns a pointer to the beginning of the reserved space
  - `free(void *p)`
    - It frees a previously reserved memory block pointed by **p**
- and many, many more …