

Tema 3

Fonaments de C

- 1 Característiques bàsiques de C
- 2 Instal·lació de l'aplicació per a programar en C
- 3 El meu primer programa
- 4 Estructura d'un programa en C
- 5 Fonaments del lèxic en C
- 6 Tipus de dades
- 7 Declaració de variables
- 8 Àmbit i visibilitat
- 9 Modes d'emmagatzemament de variables
- 10 Constants
- 11 Operadors
- 12 Expressions
- 13 Entrada i eixida en C
- 14 Exercicis

1 Característiques bàsiques de C

Història de C

Va ser creat en 1972 per a crear el sistema operatiu **UNIX** (d'on ve el Linux). Al llarg del temps aparegueren distintes versions (Quick C de Microsoft, Turbo C de Borland...) En 1983 es creà el **C estàndard** o **ANSI C**, que inclou les característiques bàsiques que ha de complir un compilador de C.

Nivell

C és un llenguatge de nivell alt però també es pot considerar de nivell mitjà, ja que combina estructures de llenguatge d'alt nivell amb la funcionalitat del llenguatge ensamblador per a poder treballar amb bits, bytes i adreces de memòria.

Característiques

- ✓ Estructurat perquè utilitza les 3 estructures de control de flux:
seqüencial, alternativa i repetitiva.
- ✓ Modular perquè hi podem definir subprogrames
- ✓ De propòsit general
- ✓ Ràpid, eficaç, portable i compacte

2 Instal·lació de l'aplicació per a programar en C

El que necessitem per a fer un programa en C és un programa editor, com per exemple el bloc de notes.

Després, una volta escrit i guardat (normalment amb extensió *.c), hem de compilar-lo per a fer l'executable (*.exe).

Però no ho farem amb el bloc de notes sinó amb un IDE (Entorn Integrat de Desenvolupament). Un IDE és una aplicació per a escriure programes, compilar-los, depurar-los, executar-los, etc.

Per a Windows podem usar el DevC++, que podem abaixar de www.bloodshed.net

Per a Linux podem usar el Geany, el qual pot instal·lar Linux des del Synaptic.

Diferències:



- Dev-Cpp (Windows):
 - o No cal posar l'extensió als noms dels fonts (ja posa .cpp)
 - o Cal posar una instrucció al final del programa per a que no desaparega la pantalla de resultats. Per exemple un System("pause") i importar el <stdlib.h>
 - o Construir amb Ctrl+F9 i executar amb Ctrl+F10



- Geany (Linux):
 - o Cal posar l'extensió als noms dels fonts (.c)
 - o No cal posar cap instrucció al final del programa, ja que no desapareix la pantalla de resultats fins que no polsem <intro>
 - o Construir amb F9 i executar amb F5

O bé pots utilitzar algun altre IDE que cregues convenient. Per exemple, en Linux està el Anjuta, etc.

3 El meu primer programa en C

Després d'abaixar-te l'IDE, executa'l i crea un fitxer (on escriurem el programa):

Geany: Fitxer → Nou

Dev-Cpp: Fitxer → Nou → Fitxer font

Copia este programa en eixe fitxer font:

```
// programa hola.c
#include <stdio.h>

int main() {
    printf("Hola a tots!\n");
    printf("Este és el meu primer programa.\n");
    getchar();
}
```

Ara dóna-li un nom al fitxer (grava'l), compila'l i executa'l.

Fes el mateix per a este altre programa:

```
// programa suma.c
#include <stdio.h>

int main() {
    int a, b, c;
    a=5;
    b=7;
    c=a+b;
    printf("La suma de %d i %d és %d\n", a, b, c);
    getchar();
}
```

4 Estructura d'un programa en C

Qualsevol programa escrit en C té la següent estructura:

```
[ Instruccions del preprocessador. Inclusió d'arxius ]
[ Definició de constants i macros ]
[ Definició de tipus ]
[ Declaració de funcions o subprogrames (prototips) ]
[ Definició de variables globals ]
int main ()
{
    [ Definició de variables locals ]
    Sentències
}
[ Definició de funcions o subprogrames ]
```

Els claudàtors [] no es posen, simplement ens indiquen que eixos apartats són opcionals. Per tant, veiem que l'única part necessària en un programa és la funció principal: `main()`, però ara veurem cadascuna de les parts que pot aparèixer en un programa escrit en C.

4.1 Instruccions del preprocessador

No són instruccions per a l'execució del programa, sinó per a la fase prèvia a la compilació. És a dir, quan anem a compilar, abans que el compilador comprove els errors, executarà este tipus d'instruccions. 3 tipus d'instruccions del preprocessador:

- ✓ Inclusió d'arxius de capçalera
- ✓ Substitució de macros
- ✓ Compilació condicional

La més important és la inclusió d'arxius de capçalera (les altres les vorem més endavant). En esta inclusió d'arxius indiquem al compilador on estan guardats els fitxers que utilitza el nostre programa. En C estos fitxers tenen l'extensió `.h` (*header file*).

Estos fitxers també s'anomenen llibreries, ja que inclouen funcions que pot fer servir el programador quan vulga.

Exemples:

```
#include <stdio.h> // Funcions d'entrada i eixida (printf i scanf)
#include <math.h> // Funcions matemàtiques com sin o pow
#include <stdlib.h> // Funcions com system("pause") (parar fins Intro)
```

La sintaxi d'inclusió de llibreries és:

```
# include <nom_arxiu_capçalera.h>
```

També podem incloure les nostres pròpies llibreries:

```
# include "llibreria_meua.h"
```

Quina diferència n'hi ha entre <> i " " ?

- " " buscarà l'arxiu al directori actual. Si no el troba, el buscarà als directoris estàndards d'include (/usr/include o /include).
- < > només busca als directoris estàndards d'include.

4.2 Definició de constants

En este apartat es definixen les constants que anem a utilitzar al nostre programa. La sintaxi és la següent:

```
# define nom_de_la_constant valor_de_la_constant
```

Quan definim una constant no indiquem el tipus, sinó que la constant serà del mateix tipus del valor que s'assigna a la constant.

Exemples de definicions de constants de diferents tipus:

```
# define PI 3.1416
# define MAXIM 999
# define ULTIMA_LLETRA 'Z'
# define EMPRESA "LLONGANI, S.A."
```

Notes:

- ✓ Posarem les constants en majúscules per a diferenciar-les de les variables.
- ✓ Al final de cada assignació de constant no es posa el ';'.
- ✓ Una constant mai pot aparèixer a l'esquerra d'una assignació.

4.3 Definició de tipus

A més dels tipus existents en C (int, char, float...), podem definir-nos els nostres propis tipus de dades. Això ens donarà una major flexibilitat.

Podem definir un nou tipus de diverses maneres. Per exemple:

- ✓ Basant-nos en un tipus ja existent:

```
typedef char t_edat;           // Definisc el nou tipus t_edat
t_edat edat_meua, edat_teua;   // Declare 2 variables del nou tipus
```

- ✓ Basant-nos en l'enumeració dels possibles valors (enum):

```
typedef enum {ford, renault, audi, mercedes, bmw} t_marca_vehicle;
t_marca_vehicle marca;
marca = audi; // Si férem marca = xxx donaria rror pq no és un valor vàlid
printf("\n%d", marca); // escriu un 2 pq ford=0, renault=1, audi=2, ...

typedef enum { Primera='a', Segona, Tercera, Penultima='y', Ultima } t_lletra;
t_lletra lletra;
lletra = Tercera;
printf("\n%c", lletra); // Mostra el caràcter 'c' (2 més que la lletra 'a')
printf("\n%d", lletra); // Mostra el 99 (codi ASCII de la lletra 'c')
```

Exercicis

1. Definix un tipus enumerat `t_dia_setmana` amb els valors corresponents (dilluns, dimarts...). Definix una variable d'eixe tipus. Fes proves assignant a la variable valors vàlids i no vàlids, per comprovar si dóna error de compilació.
2. Quin valor té cada identificador d'este tipus enumerat?

```
typedef enum { a, e=2, i, o, u } t_vocal;
```

4.4 Declaració de prototips de subprogrames

S'estudiarà més endavant. Com ja sabem, un subprograma és un tros de codi que fa una acció determinada. Estes funcions es poden definir en qualsevol part del programa però:

- Cal declarar el prototipus de cada funció abans del main:
 - ✓ tipus_valor_retorn nom_de_la_funció paràmetres amb tipus
- Cal definir el cos de cada funció després del main:
 - ✓ tipus_valor_retorn nom_de_la_funció paràmetres amb tipus
 - ✓ instruccions que formen el cos de la funció

4.5 Declaració de variables globals

Hi ha 2 tipus de variables: globals i locals

- Les globals són aquelles que es definixen fora del main (i de qualsevol funció) i, per tant, són accessibles des de qualsevol punt del fitxer.
- Les locals són les que es definixen dins de la funció que va a utilitzar-les i, per tant, només són accessibles des de les instruccions de dins d'eixa funció.

4.6 Funció main

Un programa està format per una o moltes funcions. Però tots els programes han de tindre una funció principal anomenada main, que és per on comença el programa a executar-se.

No és una funció predefinida, sinó que el programador ha d'implementar-la en cada nou programa.

Si el programa té altres funcions declarades, estes seran invocades en algun moment des de l'interior de main.

4.7 Definició de subprogrames

Després del main, s'han de definir les funcions que prèviament s'hagen declarat, però també ho veurem més endavant.

5. Fonaments del lèxic en C

A continuació definirem uns conceptes fonamentals per a saber programar en C. És el que s'anomena el lèxic, és a dir, els símbols de puntuació o separadors d'instruccions, ús de claus i parèntesis, comentaris, etc.

5.1 Comentaris

Els comentaris s'utilitzen per a explicar (al programador, no a l'usuari final) alguna part del programa.

Per exemple, servixen per a explicar el significat d'alguna variable o bé indicar a principi del programa: l'autor del programa, data de creació, etc.

El compilador no analitza les línies comentades. Simplement, passa d'elles.

En C podem incloure comentaris als nostres programes de 2 formes diferents:

- ✓ Comentari d'una línia: A partir de la combinació de caràcters //

```
int import;    // guardarà els diners, en euros
char torn;     // guardarà una 'N' si és Nocturn o una 'D' si és Diürn
```

- ✓ Comentari de diverses línies: Per a comentar grans blocs. Es considera comentari tot allò que queda entre /* i */

```
/*
    Autor: Abdó Garcia
    Data: 15-10-2008
    Utilitat: Este programa només és per a explicar els comentaris
*/

void main() {
    ...
}
```

5.2 Identificadors

Els identificadors són els noms que tenen els objectes que anem a utilitzar en el programa. Bàsicament són els noms de les variables, constants i funcions.

Eixos noms han de tindre unes normes, que són les següents:

- ✓ Començar per lletra o caràcter subratllat '_' (no números ni altres signes).
- ✓ A partir del segon caràcter ja poden ser números.
- ✓ No hi ha espais en blanc ni símbols de puntuació.
- ✓ La longitud depèn del compilador però no cal fer-los massa llargs.
- ✓ C és case sensitive (són distints: Cadena, cadena, CADENA, caDeNa, ...)
- ✓ No podem usar paraules reservades ni noms de funcions de llibreries

5.3 Paraules reservades

Sempre van en minúscula. Cada compilador de C pot tindre una quantitat diferent de paraules reservades, però les més utilitzades són:

auto	default	goto	static
break	double	if	struct
case	else	int	switch
char	enum	long	typedef
const	float	return	void
continue	for	sizeof	while

5.4 Delimitadors

Són símbols especials que permeten al compilador separar i reconèixer les diferents unitats sintàctiques del llenguatge. Les principals són:

DELIMITADOR	NOM	UTILITAT
;	Finalitzador	Finalitzar una instrucció simple o una declaració d'objectes
,	Separador	Separar els elements d'una llista
()	Parèntesis	Agrupar operacions i per als paràmetres de les funcions
{ }	Claus	Delimitar l'inici i fi d'un bloc de codi
[]	Claudàtors	Per als vectors

6 Tipus de dades

6.1 Tipus elementals

TIPUS	SIGNIFICAT	BYTES	RANG DE VALORS
char	Caràcter	1	[0 .. 255]
int	Enter	2 o 4	$[-2^{15} \dots 2^{15}]$
float	Real simple precisió	4	$\pm[3,4^{38} \dots 3,4^{-38}]$
double	Real doble precisió	8	$\pm[1,7^{308} \dots 1,7^{-308}]$
void	(cap valor)	0	-

El tipus `void` servix per a representar l'absència de valor, com per exemple en funcions que no retornen cap valor.

Els valors màxims i mínims de cada tipus estan definits en les llibreries `limits.h` i `float.h`.

En C++ podem usar també el tipus `bool` (no en C), per a guardar valors lògics (`true` i `false`) però realment no cal, ja que podem utilitzar el tipus `int` amb els valors 0 (per al vertader) i distint de 0 (per al fals).

Exercici

3. Quin tipus de dades bàsic es deu fer servir per a definir variables que continguin els següents valors?
- a) La quantitat d'habitants de Sueca
 - b) La nota mitja dels alumnes de classe
 - c) La lletra més freqüent del *Tirant lo Blanc*.
 - d) La distància en metres de València a Barcelona.
 - e) L'edat en anys d'una persona.

6.2 Modificadors dels tipus elementals

Servixen per a alterar els rangs dels tipus elementals vistos anteriorment.

Existixen 2 tipus de modificadors:

- Modificadors de longitud
 - **short** per a enters (opció per defecte en alguns compiladors)
 - **long** per a enters i reals. Dobla el rang
- Modificadors de signe (per a enters)
 - **signed** amb signe (opció per defecte)
 - **unsigned** sense signe

Tipo	Modificadores		Not. Abrev.	Rango	Ocupa
char				-128 a 127	1 byte
		unsigned		0 a 255	1 byte
		signed		-128 a 127	1 byte
int		unsigned	unsigned	0 a 65.535	2 bytes
		signed	int	-32.768 a 32.767	2 bytes
	short	unsigned	unsigned short	0 a 65.535	2 bytes
		signed	short	-32.768 a 32.767	2 bytes
	long	unsigned	unsigned long	0 a 4.294.967.295	4 bytes
		signed	long	-2.147.483.648 a 2.147.483.647	4 bytes
float				$\pm 3,4 \cdot 10^{-38}$ a $\pm 3,4 \cdot 10^{38}$	4 bytes
double				$\pm 1,7 \cdot 10^{-308}$ a $\pm 1,7 \cdot 10^{308}$	8 bytes
	long			$\pm 3,4 \cdot 10^{-4932}$ a $\pm 1,1 \cdot 10^{4932}$	10 bytes
void					0 bytes

Nota: el rang de dades és orientatiu. Vindrà donat pel compilador.

7 Declaració de variables

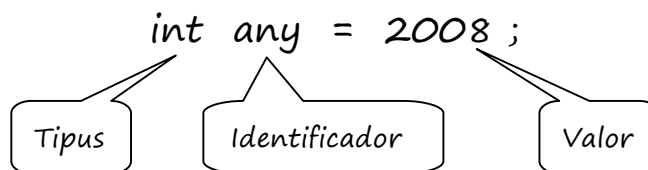
Una variable és una porció de memòria (RAM), representada per un nom (identificador) on es guardarà un valor que pot variar al llarg de l'execució d'un programa.

Totes les variables deuen ser declarades abans de ser utilitzades. Així aconseguim assignar a una posició de memòria un nom i un tipus. Eixa posició o adreça de memòria és assignada pel sistema operatiu. Amb el tipus aconseguim acotar el conjunt de valors que admet la variable (domini) i el conjunt d'operacions que podem fer sobre la variable.

Per tant, la informació bàsica que hem de saber d'una variable és:

Tipus	El tipus bàsic de la variable (char, float, ...)
Identificador	El nom amb el qual accedim a la variable.
Valor	La informació que guarda la variable en un moment donat de l'execució del programa.

Per exemple:



La declaració de variables en C es fa de la següent forma:

`[emmagatzemament] [modificador_de_tipus] tipus identificador [= valor] ;`

On els claudàtors [] indiquen que els camps corresponents no són obligatoris.

Notes:

- ✓ L'emmagatzemament determina el lloc i forma en què una variable es guarda en memòria. Ho vorem més endavant
- ✓ El modificador_de_tipus serà signed/unsigned i short/long
- ✓ Tipus és un tipus de dades bàsic o definit per l'usuari
- ✓ Podem inicialitzar o no la variable en el moment de la declaració. Si usem la variable sense inicialitzar-la, no podrem saber quin valor tindrà.
- ✓ També podem definir diverses variables del mateix tipus en una sola línia.

Exemples:

```
long int numero;  
unsigned short int comptador=0, x, y;  
char coord1='x', coord2='y', coord3='z';  
float radi, longitud;
```

Exercicis

4. Indica les definicions de variables que donen error i especifica per què.

- | | |
|-------------------------|-----------------------|
| a) char car=72; | i) #define G 9.8; |
| b) int num=3.75; | j) int a='a'; |
| c) unsigned quant=-75; | k) char nom= "pere"; |
| d) float llarg=0.67e10; | l) int i j=0; |
| e) #define PI = 3.1415 | m) char var = 27; |
| f) int i,j=12,1; | n) int ample = 2.5; |
| g) char c= '1'; | o) float mol= 1.5E-7; |
| h) long tr=-12; | p) int num= '3'; |

8. Àmbit i visibilitat

Les variables (i constants) poden declarar-se en qualsevol part del programa, però segons el lloc on siguin declarades, les podrem fer servir en tot el programa (globals) o només en alguna part (locals).

La visibilitat és la propietat que indica si es pot accedir o no a una variable en un punt determinat del programa.

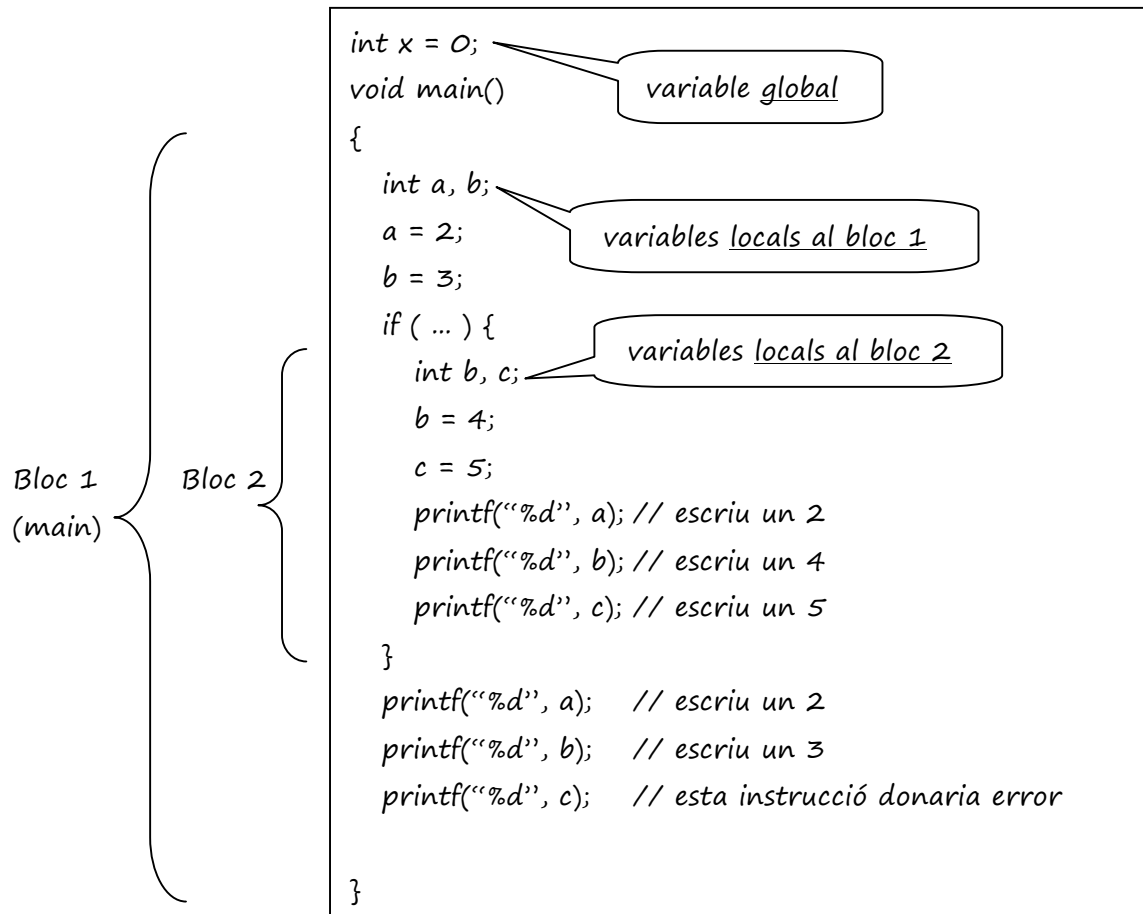
L'àmbit és la zona del programa on és visible una variable.

A partir d'estos conceptes podem diferenciar entre objectes locals i globals:

- Objectes locals: Variables (o constants) declarades dins d'un bloc o funció i, per tant, visibles només en l'àmbit d'eixe bloc i dins dels seus sub-blocs.
- Objectes globals: Variables (o constants) declarades al programa principal (fora del main) i, per tant, visibles des de qualsevol lloc del programa.

En C anomenem bloc a tot allò que està entre claus { }. Per tant, qualsevol funció, com veurem més endavant, és un bloc.

Cal tindre clar que si es declaren variables en un bloc (locals) oculten les variables amb el mateix nom globals.



Visibilitats dels blocs

- Bloc 1 veu les variables: - a i b del bloc 1
- Bloc 2 veu les variables: - a del bloc 1
 - b i c del bloc 2

Àmbits de les variables

- La variable a del bloc 1 té l'àmbit dels blocs 1 i 2.
- La variable b del bloc 1 només té l'àmbit del bloc 1.
- Les variables b i c del bloc 2 tenen només tenen l'àmbit del bloc 2.

Consell:

Ja hem dit que el llenguatge C ens permet declarar les variables en qualsevol lloc del programa, però com a bons programadors que som, les declararem al principi del `main`, al principi de cada funció o al principi de cada bloc, per a millorar la llegibilitat dels programes.

9 Modes d'emmagatzemament de variables

Quan declarem una variable, a més del tipus, nom i valor, podem indicar el **mode d'emmagatzemament**. Serveix per a determinar coses com el lloc on es guardarà (RAM o registres de CPU), el temps de vida del valor de la variable i l'accés des d'altres fitxers.

Mode auto

És l'opció per defecte. Naixen i moren al bloc on es definixen

Mode register

Igual que el mode auto però s'intenten guardar als registres de la CPU en compte de la memòria RAM. Solen utilitzar-se en bucles, per a accelerar els càlculs, ja que el processador no ha d'anar fins la memòria a llegir/escriure el valor de la variable.

Mode static

- ✓ Si una variable és declarada *static* dins d'una funció (o bloc), quan esta acabe, la variable no morirà, sinó que mantindrà el seu valor si es torna a cridar a la funció (o a passar pel bloc).
- ✓ El compilador inicialitza automàticament estes variables a 0.

Mode extern

Fa que les variables globals siguin encara més globals. És a dir, que puguin ser accedides des d'altres fitxers.

```
// fitxer1.c
int var_global = 50;
void main() {
    ...
}
```

```
// fitxer2.c
extern int var_global;
void funcio_x() {
    ...
}
```

La variable `var_global` és accessible des de qualsevol lloc dels 2 fitxers: en `fitxer1.c` per ser global, i en `fitxer2.c` per ser externa. La global ha de definir-se sempre al fitxer que té la funció `main`.

10 Constants

Una constant és un valor que no varia al llarg del programa.

Les constants en C poden ser de tipus enter, real, caràcter, de cadena o enumerat. Les de tipus enumerat les vorem més endavant. Les altres poden vindre expressades de 2 formes diferents:

- Pel seu valor (constants literals)
- Amb un nom o identificador (constants simbòliques)

10.1 Constants literals

TIPUS	DESCRIPCIÓ	EXEMPLES
Enters	Números sense decimals	18, -1000
Reals	Números amb el punt decimal	3.14, 166.386, 2.8E-2, 3.4e7
Caràcter	Símbols. Van entre cometes simples	'a', 'A', '4', '+', '?'
Cadena	Cadenes de símbols. Entre cometes dobles	"Pep García", "LLONGANIS, S. A."

Hi ha caràcters especials que podem fer servir a les constants de tipus caràcter com a les cadenes, com són:

\n Nova línia
\t Tabulador
\" Cometes dobles

Exercici

5. Programa que mostre les següents línies. Usa els caràcters especials vistos.

Un pollet se'n va a la mar.
Es revolca per l'arena,
"Xim", "pum", fora.

10.2 Simbòliques

Són constants a les quals assignem un nom. A l'apartat 3.2 ja hem vist que cal definir-les així:

```
# define nom_de_la_constant valor_de_la_constant
```

Compte! No es posa el punt i coma al final.

Exemples:

```
# define PI 3.1416
```

```
# define EMPRESA "LLONGANI, S.A."
```

Cal tindre en compte que les constants simbòliques no són guardades en memòria (com es fa a les variables) sinó que el preprocessador, quan es troba una constant simbòlica, la substitueix pel valor assignat en el *define*. És com fer un "busca i reemplaça" en un editor de text.

11 Operadors

Per a poder maniobrar amb les dades, necessitem els operadors. Depenent del tipus d'estes dades, tindrem uns tipus determinats d'operadors:

11.1 Operadors aritmètics

OPERADOR	SIGNIFICAT
*	Producte
/	Divisió
%	Residu de divisió entera
+	Suma
-	Resta o signe

Notes:

- ✓ Si l'operador resta (-) només actua sobre un operand, fa la funció de signe.
- ✓ L'operador residu (%) opera només amb enters (i retorna un enter, clar)
- ✓ L'operador divisió (/) funciona segons el tipus dels operands. Si són enters, fa la divisió entera. Però si algun és real, fa la divisió real.

<u>Expressió</u>		<u>Valor</u>
10/4	→	2
10/4.0	→	2.5
10.0/4.0	→	2.5
10.0/5	→	2.0

Exercici

6. Completa el següent programa amb %f (real) o %d (enter). Què traurà per pantalla? Executa'l i compara'n els resultats. Hi ha algun error? Per què?

```
#include <stdio.h>
int main(void){
float x,y;
int a,b;
x=10.0;
y=2.0;
a=10;
b=2;
printf("L'operació x/y val __ \n", x/y);
printf("L'operació x/b val __ \n", x/b);
printf("L'operació a/b val __ \n", a/b);
printf("L'operació a%b val __ \n", a%b);
printf("L'operació x%b val __ \n", x%b);
getchar();
}
```

11.2 Operadors relacionals

Servixen per a comparar 2 expressions, retornant un valor lògic: vertader o fals.

OPERADOR	SIGNIFICAT
<	Menor
>	Major
==	Igual
!=	Distint
<=	Menor o igual
>=	Major o igual

Notes importants:

- ✓ Alerta! En C++ sí que existix el tipus *bool* (lògic) però no en C. En C es representa el valor lògic false com a 0 i el true com a distint de 0. Així, si una expressió relacional és certa, s'avalua a 1; i si és falsa, a 0. És a dir, realment les expressions relacionals no donen com a resultat un valor lògic sinó un enter (0 o 1).
- ✓ Cal anar en compte en les comparacions: "if (a == b)..." és una comparació, però "if (a = b)..." és una assignació.

11.3 Operadors lògics

Els operadors lògics són els mateixos que veïrem en el FreeDFD però expressats de forma diferent:

FreeDFD	C
OR	
AND	&&
NOT	!

Curtcircuit d'expressions

Si recordem les taules de veritat, podem afirmar que:

false AND ... → false

true OR ... → true

Per tant, com les expressions s'avaluen d'esquerra a dreta, en el moment en què el compilador puga assegurar el valor final de l'expressió lògica (true o false), parerà d'avaluar-la. Esta manera de treballar s'anomena *curtcircuit d'expressions*. Això ens dóna un benefici pel que fa a control d'errors i a velocitat d'execució.

Exemples

<pre>if ((descompte_1>0) (descompte_2>0) (descompte_3>0)) printf("S'ha aplicat algun descompte");</pre>	Si el <code>descompte_1</code> és major que 0, ja no es comproven les altres 2 expressions i passa a executar-se directament el <code>printf</code> .
<pre>(x<0) && printf("El valor de la variable x és negatiu")</pre>	Només es farà el <code>printf</code> si el valor de <code>x</code> és negatiu.

Exercici

7. Pensa què passaria en cada cas sense i amb *curtcircuit d'expressions*.

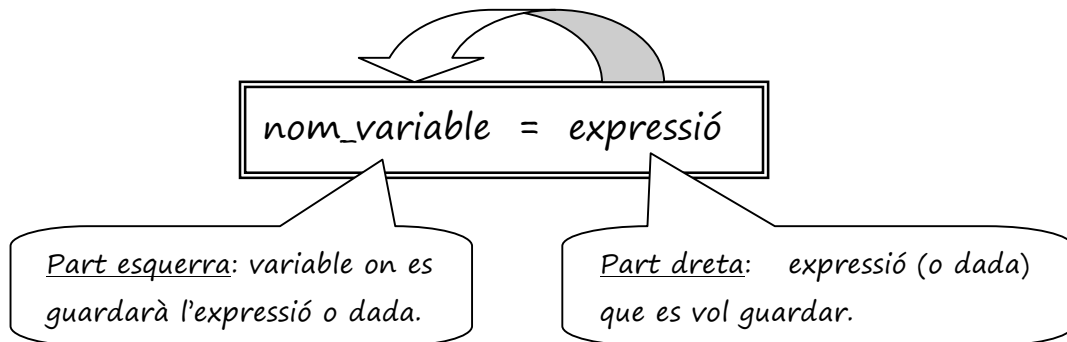
<pre>x=10; y=0; if ((x/y)>2 && (y!=0)) {...}</pre>	<pre>x=10; y=0; if ((y!=0) && (x/y)>2) {...}</pre>
---	---

Ara bé, si no volguérem que *C* faci el *curtcircuit d'expressions*, en compte dels operadors `||` i `&&` usàriem simplement `|` i `&`.

11.4 Operador d'assignació

En C l'operador d'assignació és el símbol igual (=).

A l'apartat 3.3.1 ja veiérem que esta operació consistix en assignar a una variable el valor d'una expressió. La variable sempre està a la part esquerra del = i l'expressió a la dreta:



Funciona avaluant primer la part de la dreta, obtenint el seu resultat i assignant-lo a la variable de l'esquerra.

Per exemple:

```
x = 10;  
y = 20;  
x = x + 1;  
y = x + y;
```

Per descomptat, l'assignació és destructiva, ja que elimina el valor antic de la variable. És a dir, només pot guardar una dada en un moment determinat.

C també permet assignar un valor a diverses variables. Es fa amb esta expressió:

```
a = b = c = d = 2;
```

Diagrama de la expressió anterior mostrant la associació dreta a l'esquerra:

$\underbrace{\underbrace{\underbrace{a = b = c = d}_{= 2}}_{= 2}}_{= 2}$

És a dir, cada expressió d'assignació s'avalua al valor de l'assignació. Per tant, l'expressió anterior és equivalent a fer les següents assignacions (i en eixe ordre):

```
d = 2;  
c = d;  
b = d;  
a = d;
```

11.5 Operadors aritmètics reduïts (Operació-assignació)

Existixen formes compactes d'escriure algunes operacions aritmètiques molt freqüents. Són els **operadors reduïts**:

- Operadors d'actualització: $+=$, $-=$, $*=$, $/=$ i $\%=$
- Operadors autoincrement i autodecrement: $++$, $--$

Estos operadors fan 2 coses: una operació i una assignació.

Operadors d'actualització

En C, a banda del clàssic $=$, existixen 5 operadors més d'assignació, que estan formats amb els operadors aritmètics i l'operador $=$.

Assignació reduïda	Assignació equivalent
$x += y ;$	$x = x + y ;$
$x -= y ;$	$x = x - y ;$
$x *= y ;$	$x = x * y ;$
$x /= y ;$	$x = x / y ;$
$x %= y ;$	$x = x \% y ;$

On 'x' és una variable i 'y' una expressió, constant o variable.

Exemple:

```
int x, y;
y = 1;
x = 4;
x += y;      // x = x + y      → x = 5
x *= 2;      // x = x * 2      → x = 10
x -= 3 - y;  // x = x - (3 - y) → x = 10 - (2) → x = 8
```

Operadors autoincrement i autodecrement

Els operadors incrementals ++ i -- són operadors unaris que incrementen o decrementen en 1 unitat la variable a qui acompanyen.

- ✓ Si l'operador va davant (++x), primer s'incrementa la variable i després s'utilitza el valor de la variable en l'expressió on apareix.
- ✓ Si l'operador va després (x++), primer s'utilitza el valor de la variable en l'expressió i després s'incrementa la variable.

	Expressió amb autoincrement	Expressió equivalent	
		1a operació	2a operació
Pre-Increment	++x ;	x = x + 1 ;	Ús de x
	--x ;	x = x - 1 ;	Ús de x
Post-increment	x++ ;	Ús de x	x = x + 1 ;
	x-- ;	Ús de x	x = x - 1 ;

Exemples:

Nom	Instrucció	Funcionament	Resultat final
Pre-increment	X = 10;		
	Y = ++X;	1r) s'incrementa X 2n) S'assigna el valor de X a Y	X ← 11 Y ← 11
Post-increment	X = 10;		
	Y = X++	1r) S'assigna el valor de X a Y 2n) S'incrementa X	Y ← 10 X ← 11

Instrucció amb operadors incrementals	Instruccions equivalents (s'executarien en eixe ordre)
$x = (y--)*(++z)/10;$	$z = z + 1;$ $x = (z * y) / 10;$ $y = y - 1;$

Exercici

8. Donades les variables a=4 i b=7, escriu la instrucció $a += 10 + ++b$ de forma equivalent però sense usar les formes reduïdes. Què valdran finalment a i b ?

11.6 Altres operadors

Operador condicional ? :

Sintaxi:

```
condició ? expressió_sí : expressió_no ;
```

Exemples d'ús:

a) Assignar un valor diferent a una variable segons una condició:

```
major = a > b ? a : b ;
```

```
distancia = a > b ? a - b ; b - a;
```

```
num_positiu = num > 0 ? num : -num ;
```

b) Mostrar un valor diferent segons una condició:

```
printf( edat>18 ? "Adult" : "Infantil" );
```

c) Executar una instrucció diferent segons una condició:

```
edat>18? printf("Adult") : printf("Infantil");
```

Nota: també es poden posar operadors condicionals dins d'altres.

Exercicis

9. Fes un programa que, donats 2 variables enteres, mostre quin és el número més gran i quin el més menut.
10. Fes un programa que, donats 3 variables enteres, mostre quin és el número més gran.

Operador sizeof

La quantitat de bytes que s'utilitza per a guardar un determinat tipus de dades depèn de la versió del compilador i del tipus de processador que s'utilitzi.

Per tant, perquè el nostre programa pugui ser portable, de vegades és necessari saber quants bytes ocupen les variables amb les quals treballarem.

Per a això, C disposa de l'operador `sizeof()`, qui calcula la quantitat de bytes que ocupa la variable o tipus que li posem dins dels parèntesis.

`x = sizeof(variable o tipus)`

Exercicis

11. Fes un programa que mostri la grandària en bytes dels tipus bàsics de C.
12. Programa que mostri quantes lletres es podrien guardar en 1 MB de RAM.

Operador coma (,)

Els operands d'este operador són expressions, i té la forma següent:

`variable = expressió_1 , expressió_2 , ... , expressió_n`

Primer s'avalua l'expressió_1, després la 2, i així successivament fins la n. Finalment, el valor d'esta última expressió s'assigna a la variable.


Exemple:

```
int x, y, z;
x = 7;
y = 3;
z = (x++ , y);           // z = 3; x = 8 ;
z = (x=x+2 , y=x+3 , y++); // x = 10 ; y = 13 ; z = 13 ; y = 14
```

Operadors adreça (&) i indirecció (*)

Nota: Este apartat el vorem per damunt, damunt, ja que és molt complex i no s'utilitza en llenguatges de més alt nivell com Java, que és el que vorem més endavant.

Malgrat que C és un llenguatge d'alt nivell, ja diguérem que C es pot considerar també de nivell mig, ja que pot treballar directament amb les adreces de memòria on estan guardades les dades.

- L'operador adreça (&) serveix per a obtenir l'adreça d'una variable.
- L'operador indirecció (*) serveix per a obtenir el valor que hi ha en una adreça. Les variables que emmagatzemen adreces d'altres variables es denominen **punter**  (o apuntadors) han de ser declarades com a tals, i tenen la seua pròpia forma de funcionar.

Exemples:

```
int quant, *pquant; // pquant és un punter a un enter
quant = 15;
pquant = &quant; // assignem a pquant l'adreça de la variable quant
printf("L'adreça de memòria de la variable quant és %p", pquant);
printf("El valor de la variable quant és %d", quant);
printf("A eixe valor també podem accedir pel punter: %d", *pquant);
```

Una de les utilitats dels punters és l'ús d'estructures de dades dinàmiques. És a dir, durant l'execució d'un programa podem reservar zones de memòria per a poder guardar llistes de dades, etc.

Per a reservar memòria tenim la funció `malloc` i, per a alliberar-la, `free`. Estes funcions estan dins la llibreria `<stdlib.h>`.

12 Expressions

Ara que ja hem vist els tipus d'operands i els operadors, ja podem formar expressions. Què pot ser una expressió?

- ✓ Un valor (constant literal)
- ✓ Una constant amb nom o una variable
- ✓ Una funció
- ✓ Una combinació dels elements anteriors amb operadors, els quals han de complir certes regles de construcció.
- ✓ Res més és una expressió

12.1 Conversions de tipus

Dins d'una expressió poden aparèixer dades de distints tipus. Aleshores, el compilador ha de prendre una decisió de com fer el càlcul i no donar error.

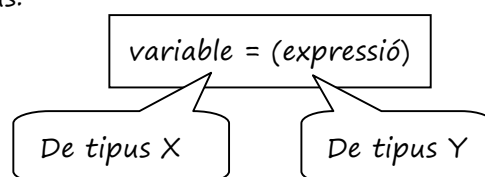
Promoció interna

Si en una expressió hi ha dades de diferent tipus, el compilador transforma els tipus d'una grandària inferior al tipus de la dada de grandària superior.

char → int → unsigned → long → unsigned long → float → double → long double

Promoció per assignació

Esta conversió la fa el compilador quan s'intenta assignar a una variable una expressió de diferent tipus.



- | | | |
|----------------------|---|---------------------------------------|
| ✓ (int) = (char) | → | no hi ha problema. |
| ✓ (char) = (int) | → | s'elimina el byte menys significatiu. |
| ✓ (int) = (float) | → | es trunca la part fraccionària. |
| ✓ (float) = (double) | → | s'arredonix el número |

En els casos on es perd certa informació ens pot portar a resultats inesperats. Per tant, en eixos casos és recomana fer **conversions forçades**.

Promoció forçada

En este cas no esperem que siga el compilador qui faça la promoció (conversió de tipus), sinó que serà el propi programador qui indique quina dada ha de canviar de tipus i a quin.

S'especifica posant el tipus entre parèntesi just abans de l'expressió:

`(tipus) expressió`

Aleshores, després d'avaluar-se l'expressió, es convertirà al tipus indicat entre parèntesi. En C++ no cal posar els parèntesis.

Exemples:

```
int n = 10; m = 3;
float x, y;
x = (float)(n/m) ;    // (float) 3        → 3.0
x = (float)n/m;       // (float)(n) / m    → 10.0 / 3 → 3.333...
```

Exercici

13. Quin és el resultat de les següents expressions? Indica també si hi ha pèrdua o promoció d'informació.

- a) `int a = 3 / 5 + 0.75 / 2;`
- b) `float b = 50 / 2 + 7 % 3;`
- c) `int c = 1.75E9;`
- d) `char d = 160 / 2 + 5;`
- e) `char a = 600 / 2 * 5 + 1;`
- f) `int b = 5 / 2 + 6.5 / 2;`
- g) `float c = 2.5E-125;`
- h) `int d = 0.5E10;`
- i) `long e = -23;`

12.2 Precedència i associativitat d'operadors

Ací tenim una relació dels operadors ordenats per precedència de major a menor. En cas d'igualtat de precedència de diversos operadors en una expressió, l'associativitat ens diu per on es comença a avaluar.

CATEGORIA DE L'OPERADOR	OPERADOR	ASSOCIATIVITAT
Parèntesis, vectors, punters	() [] -> .	ESQUERRA
Operadors unaris	++ -- ! sizeof + - * (indirecció) & (adreça)	DRETA
Multiplicació, divisió i residu	* / %	ESQUERRA
Suma i resta	+ -	ESQUERRA
Desplaçament bit a bit	<< >>	ESQUERRA
Operadors relacionals	< <= > >=	ESQUERRA
Operadors d'igualtat	== !=	ESQUERRA
AND bit a bit	&	ESQUERRA
OR bit a bit		ESQUERRA
AND lògic	&&	ESQUERRA
OR lògic		ESQUERRA
Condició	? :	DRETA
Assignacions	= += -= *= /=	DRETA
Operador coma	,	ESQUERRA

13 Entrada i eixida en C

Les instruccions d'entrada i eixida (a partir d'ara E/S) permeten la construcció de programes interactius. És a dir, introduir dades des de teclat i visualitzar en pantalla els resultats. C no incorpora instruccions d'E/S, però gràcies a la modularitat de C, mitjançant la inclusió d'arxius de capçalera, podrem utilitzar unes funcions predefinides per a poder interactuar amb teclat i monitor.

L'arxiu que hem de posar a la capçalera és `<stdio.h>` (STanDard Input Output). Per tant, la primera línia dels nostres programes deuria ser:

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

Les funcions d'E/S es dividixen en 2 grups:

- ✓ Amb format (*printf* i *scanf*)
- ✓ Sense format

Abans de vore en detall com funcionen les diferents funcions d'E/S de C, vorem amb un exemple les funcions més utilitzades: *printf* i *scanf*.

El que abans fèiem així per al pseudocodi:

```
...  
escriu("Dis-me el radi d'una circumferència: ")  
lleg(radi)  
escriu("El perímetre és ", 2*PI*radi, " i l'àrea és ", PI*radi*radi)  
...
```

Ara ho farem així en llenguatge C:

```
...  
printf("Dis-me el radi d'una circumferència: ");  
scanf("%d", &radi)  
printf("El perímetre és %d i l'àrea és %d", 2*PI*radi, PI*radi*radi);  
...
```

Exercicis

14. Fes un programa que pregunte quants anys té algú i que mostre per pantalla la quantitat d'anys que falten per a la majoria d'edat i per a jubilar-se.
15. Programa que pregunte per la base i l'altura d'un triangle i mostre per pantalla l'àrea d'eixe triangle.

13.1 Eixida estàndard: *printf*

La funció *printf* escriu dades en pantalla. La seua sintaxi és:

```
printf("cadena de control", arg1, arg2, ..., argn)
```

Encara que a priori no ens servisca de molt, esta funció retorna la quantitat de bytes escrits en pantalla (quantitat de caràcters).

Els arguments *arg1*, *arg2*, ..., *argn* poden ser constants, variables, expressions i/o crides a altres funcions.

La cadena de control és una cadena de caràcters tancada entre cometes dobles, on poden aparèixer 3 tipus d'elements:

1) Text. És el text literal que apareixerà tal i com l'escrivim.

2) Caràcters de control (\). També anomenats seqüències d'escapament o de barra invertida. S'utilitzen per a posar bots de línia, tabuladors, etc. Els més usuals són:

Caràcter de control	Significat
\n	Nova línia
\b	Retrocés
\f	Nova pàgina
\r	Retorn de carro
\t	Tabulador horitzontal
\v	Tabulador vertical
\"	Cometes dobles
\'	Cometa simple (apòstrof)
\?	Interrogant tancat
\\	Barra invertida

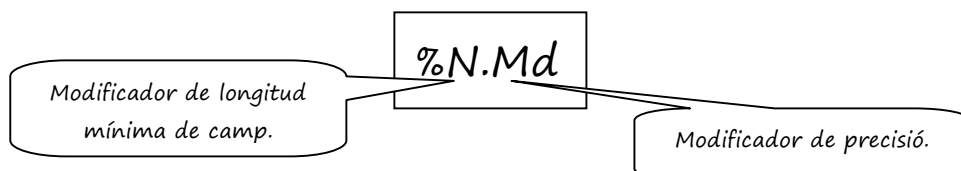
3) Especificadors de format (%). Indiquen la forma en què es mostren els arguments. Deu haver la mateixa quantitat d'especificadors de format que d'arguments, i han de coincidir amb el seu ordre d'aparició d'esquerra a dreta.

Especificadors de format	Tipus
%c	Caràcter
%s	Cadena de caràcters
%d o %i	Enter amb signe
%ld	Enter llarg amb signe
%u	Enter sense signe
%e	Real amb notació científica (e minúscula)
%E	Real amb notació científica (E majúscula)
%f	Float
%lf	Double
%g o %G	El més curt entre %e (o %E) i %f. Si l'exponent és ≥ 6 o < -4 aleshores %e o %E.
%p	Punter
%%	Símbol % (no especifica el format de cap argument)

Compte! Si posem un especificador de format que no es correspon al tipus de l'argument corresponent, pot mostrar per pantalla un valor inesperat.

Modificadors de format

Alguns especificadors de format poden acceptar modificadors que canvien un poc el seu significat. Es posen entre el signe % i la lletra de format:



- ✓ Modificador de longitud mínima de camp: N (número enter).

Si la cadena o el número a mostrar tenen una longitud menor que N, s'omplir d'espais els buits. Si no, es mostra tot.

```
printf("#%6d#", 24); // # 24# (quatre blancs i 24)
```

```
printf("#%1d#", 24); // #24# (no retalla el número)
```

- Si volem omplir amb zeros per l'esquerra, cal posar un zero abans del número N.

```
printf("#%06d#", 24); // #000024# (quatre zeros i 24)
```

- Els números i cadenes s'ajusten a la dreta. Si volem que s'ajusten a l'esquerra, cal posar el signe - davant del número N.

```
printf("#%-6d#", 24); // #24 # (24 i quatre blancs)
```

- ✓ Modificador de precisió: .M (número enter, es posa entre el N i la lletra)
 - Si és un real, M és la quantitat exacta de decimals a mostrar.

```
printf("#%9.5f#", 10/3.0); // # 3.33333# (total:9 ; decimals:5)
```
 - Si es una cadena, la M és la quantitat màxima de caràcters a mostrar. Si la cadena és més llarga que M, es trunquen els caràcters finals.

```
printf("#%7.5s#", "hola"); // # hola#  
printf("#%7.3s#", "hola"); // # hol# (retalla la paraula)
```
 - Si és un enter, la M és la quantitat mínima de xifres.

```
printf("#%7.5d#", 123); // # 00123# (total:7 ; xifres:5)
```

Estos modificadors de longitud i precisió també es poden passar com a arguments posant els símbols `.*` en compte dels modificadors.

Per exemple, la instrucció següent:

```
printf("%*.*f", 10, 4, 1234.34);
```

... és equivalent a esta altra:

```
printf("%10.4f", 1234.34);
```

La utilitat de `.*` és poder posar la longitud i precisió en variables.

Exemples:

```
printf("%f", 1.24); // 1.240000
printf("%g", 1.24); // 1.24
printf("%e-%f", 2150, 2150); // 2.15e3-2150.000000
printf("%-5d%c", 24, 'A'); // 24_ _A
printf("%-5d%c", 123456, 'A'); // 123456A
printf("%-5.3d%c", 24, 'A'); // 024_ _A
printf("%4.2d%c%-5.2f%5.3s", 1, 'A', 1.127, "HOLA"); // _01A1.13_ _ _HOL
```

13.2 Entrada estàndard: *scanf*

La funció *scanf* llig dades de teclat i les assigna a les variables que indiquem.

La seua sintaxi és:

```
scanf("cadena de control", &arg1, &arg2, ..., &argn)
```

Encara que a priori no ens servisca de molt, esta funció retorna la quantitat de variables llegides (zero si no n'ha llegit cap).

La cadena de control és una cadena de caràcters tancada entre cometes dobles, on poden aparèixer 2 tipus d'elements:

- 1) Especificadors de format (%), semblant als del *printf*.
- 2) Altres caràcters que servixen de separador d'arguments (com la coma, l'espai, etc):

```
scanf("%d,%d", &x, &y); // Llig enter, espera coma i llig altre enter.  
// Si posem enters sense coma, donarà error.
```

Els arguments (&arg1, &arg2, ..., &argn):

- ✓ Han de ser obligatòriament variables (arg1, arg2, ..., argn).
- ✓ Davant de cada variable anteposarem el símbol & (adreça), a excepció de les variables de tipus cadena. De moment ens ho creiem, però ja veurem més endavant que amb el símbol & estem indicant l'adreça en memòria de la variable on guardar les dades.

La funció *scanf* acaba quan:

- ✓ S'introdueix un caràcter no vàlid.
- ✓ S'han llegit els valors esperats i es polsa Intro. Tot el que hi ha entre l'últim valor esperat i Intro, s'ignora.

Nota: Quan introduïm un valor per teclat, després polsem intro ("n") per a acceptar. Tant el valor com l'intro es guarden al buffer d'entrada, però l'*scanf* NO arreplega el caràcter intro. Per tant, si després férem un *scanf* d'un caràcter, agafaria l'intro del buffer i no arreplegaria res de teclat. Per a evitar això, convé buidar el buffer després de cada lectura de teclat. Això ho podem fer amb la següent instrucció: `while (getchar()!='n');`

14 Exercicis

16. Suposades les següents declaracions i inicialitzacions de variables, calcula el valor de les següents expressions, tenint en compte que en cada cas partim dels valors inicials de les variables i sabent que $\text{ASCII}('a') = 97$.

`long int i=6, j=4;`

`double VAR1=0.5, VAR2=-0.01;`

`char CAR='c', Lletra='d';`

a) $(2 * i - 3 * j) \% (2 * \text{Lletra} - \text{CAR}) + 2$	j) $(i > 0) \parallel (j < 6)$
b) $-(i + j) / 2.0$	k) $(\text{VAR1} > \text{VAR2}) \&\& (i > 0) \parallel (j < 5)$
c) <code>++i</code>	l) $(\text{VAR1} > \text{VAR2}) \parallel (i > 0) \&\& (j < 5)$
d) <code>i--</code>	m) $(i \geq 1) \&\& (i < 11)$
e) $\text{CAR} > \text{Lletra}$	n) $(i < 1) \parallel (i \geq 11)$
f) <code>j!=6</code>	o) <code>i/j</code>
g) $\text{CAR} == 99$	p) $(\text{float})(i/j)$
h) $2 * \text{VAR1} + \text{VAR2} == 0$	q) $(\text{float})i/j$
i) $(i > 0) \&\& (j < 6)$	r) $(\text{float})(i)/j$

17. Escriu el resultat de la següent expressió:

$((5+3)/2*3)/2 - (\text{int})28.75/4 + 29\%3*4$

18. Escriu una expressió on s'especifiqui que una variable numèrica de nom *quant* siga menor o igual que 500 i múltiple de 5 però distinta de 100.

19. Escriu el resultat de la següent expressió:

$35 > 47 \&\& 9 == 9 \parallel 35 != 3 + 2 \&\& 3 >= 3$

20. Quin serà el valor de la variable definida com *int result* en la següent expressió d'assignació?

`result = 'C' - (float) 5 / 2 + 3.5 + 0.4`

21. Tenint en compte la definició $\text{int } a=14, b=7$, troba els valors que pren la variable (a) en cadascuna de les següents assignacions:

- a) $a = -b + 6$;
- b) $a /= b - 3$;
- c) $a = (++b, b + 3)$;
- d) $a = (\text{float}) b / 3$;

22. Tenint en compte la definició $\text{int } y=10, x=5$, troba el valor de y en les següents expressions:

- a) $y *= x + 1$;
- b) $y /= (5 + x)$;
- c) $y = ++x - 2$;
- d) $y = (-x, x + 2)$;
- e) $y = (\text{float}) x/2$;

23. Escriu el resultat de les següents expressions:

- a) $5 / 2 + 17 \% 3$
- b) $21.6 / 3 + 8.5$
- c) $3 * 6 / 2 + 18 / 3 * 2$;
- d) $(47 - 5) * 2 / 3 / (5 + 2)$
- e) $9 == 15 \parallel 8 != 5 \&\& 7 > 4$
- f) $(8 * 2 < 5 \parallel 7 + 2 > 9) \&\& 8 - 5 < 18$

24. Escriu el resultat de les següents expressions:

- a) $24 \% 5$
- b) $7 / 2 + 2.5$
- c) $10.8 / 2 + 2$
- d) $16 / 3 * 2 - 4 * 5 / 2$
- e) $(4 + 6) * 3 + 2 * (5.5 - 1)$
- f) $7 >= 5 \&\& 27 == 8$
- g) $45 <= 7 \parallel !(5 >= 7)$
- h) $8 > 8 \parallel 7 = 7 \&\& !(5 < 5)$
- i) $4 + 2 < 8 \&\& 24 + 1 == 25 \parallel 0$
- j) $(2 * 7 > 5 \parallel 7 / 2 == 3) \&\& (7 > 25 \parallel !1) \&\& 27$

25. Troba els errors en el següent programa que calcula l'àrea d'un cercle a partir del radi. Després edita'l, compila'l i executa'l.

```
#include <stdio.h>
define PI 3,14
void main (void)
{
float radio, area
printf ("\nPrograma de càlcul de l'àrea d'un cercle")
printf ("\n\nEscriu el radi: ");
/* Llegir radi de teclat */
scan ("%d", &radi);
/* Calcular i imprimir l'àrea
area = PI * radio^2;
printf ("\n\nL'àrea del cercle és: %f\n", area;
}
```

26. Troba els errors en el següent programa que calcula l'àrea d'un rectangle. Després edita'l, compila'l i executa'l.

```
include <stdio.h>
void main()
{
int base; altura;
printf("\nPrograma que calcula l'àrea d'un rectangle");
printf("\n\nDóna'm base i altura:");
/* Llegir base i altura de teclat*/
scanf("%c%f, &base, altura);
area=base*altura;
printf("\n\nL'àrea del rectangle és: %d\n", aera);
}
```

27. Quins seran els valors finals de cadascuna de les variables?

```
int a=10, b=3, c, d, e;
float x, y;
x = a/b;
c = a<b && 25;
d = a+ b++;
e = ++a + b;
y = (float) a/b;
```

28. Contesta les següents qüestions tipus test:

a) Quina de les següents expressions es correspon amb una seqüència d'escapament?

- 1) ESC
- 2) \n
- 3) 0x07
- 4) n

b) Els tipus primitius en C són:

- 1) int, float, void
- 2) bool, char, short, int, long, float, double
- 3) char, short, int, long, enum, float, double, long double
- 4) caràcters, variables i constants

c) C assumix que un tipus enumerat és:

- 1) un tipus enter
- 2) un tipus real
- 3) un tipus nou
- 4) una constant

d) 01234 és un literal...

- 1) decimal
- 2) octal
- 3) hexadecimal
- 4) no és un literal

e) 17.25 és un literal de tipus:

- 1) char
- 2) int
- 3) float
- 4) double

f) L'expressió 's' és:

- 1) una variable
- 2) una cadena de caràcters
- 3) un enter
- 4) un valor de tipus long

g) Es definix $a=5$ i $b=2$ de tipus int. El resultat de a/b és:

- 1) 2 de tipus int
- 2) 2.5 de tipus double
- 3) 2 de tipus float
- 4) 2.5 de tipus float

h) Es definix $a=5$, $b=2$ i $c=0$ de tipus int. El resultat de $c=a>b$ és:

- 1) 3
- 2) 2
- 3) 1
- 4) 0

i) Es definix $a=5$ i $b=0$ de tipus int. El resultat de $b=a++$ és:

- 1) 4
- 2) 0
- 3) 6
- 4) 5

j) Es definix $a=5$ de tipus int. El resultat de $a/(double) 2$ és:

- 1) 2
- 2) 2.5
- 3) 3
- 4) 0

29. Troba els errors d'este programa en C:

```
#include <stdio.h>
int a, b;
scanf ("%d", &a); scanf ("%d", &b)
while (a <= b):
    scanf ("%d", &a)
    scanf ("%d", &b)
printf ("%d %d\n", a, b);
```


30. Tabula correctament este programa en C:

```
#include <stdio.h>
int main(void)
{int a, b;
scanf ("%d", &a);
if (a>0) {
scanf ("%d", &b);
while(a > b) {
scanf ("%d", &a);
if (a>0) {
scanf ("%d", &b);}}
printf ("%d %d\n", a, b);
return 0;}
}
```

31. Fent proves en la creació d'un programa, hem decidit comentar una línia per a que no siga compilada, quedant així:

```
1  #include <stdio.h>
2  int main(void)
3  {
4  int a, b, i, j;
5  scanf ("%d", &a);
6  scanf ("%d", &b);
7  i = a;
8  j = 1;
9
10 while (i <= b) {
11     /* printf ("%d %d\n", i, j); */
12     j *= 2;
13     i += 1;
14 }
15
16 printf ("%d\n", j);
17 }
```

Compilem el programa i el compilador no detecta cap error. Ara decidim comentar el bucle *while* complet. Per tant, afegim una marca */** a la línia 9 i una marca **/* a la línia 15. Tornem a compilar el programa i dóna error. Per què?

32. Donaria error si la línia 11 haguera estat comentada amb només *//* ?

33. Com s'interpreta la següent sentència?

$i = x // *y* / z++;$

34. Donades 3 variables enteres $a=0$, $b=1$ i $c=2$, quin valor té cada variable després de cada seqüència d'assignacions?

- a) $a = b++ - c--;$
- b) $a += --b;$
- c) $c *= a + b;$
- d) $b = (a > 0) ? ++a : ++c;$
- e) $a += a = b + c;$

35. Quin valor s'emmagatzema a les variables *int i*, *float x* en estes sentències?

- a) $i = 2;$
- b) $i = 1 / 2;$
- c) $i = 2 / 4;$
- d) $i = 2.0 / 4;$
- e) $x = 2.0 / 4.0;$
- f) $x = 2.0 / 4;$
- g) $x = 2 / 4;$
- h) $x = 1 / 2;$

36. Quin valor s'emmagatzema a les variables *int i*, *float x* en estes sentències?

- a) $i = (\text{float}) 2;$
- b) $i = 1 / (\text{float}) 2;$
- c) $i = (\text{int}) (2 / 4);$
- d) $i = (\text{int}) 2.0 / (\text{float}) 4;$
- e) $x = 2.0 / (\text{int}) 4.0;$
- f) $x = (\text{int}) 2.0 / 4;$
- g) $x = (\text{int}) (2.0 / 4);$
- h) $x = 2 / (\text{float}) 4;$
- i) $x = (\text{float}) (1 / 2)$
- j) $x = 1 / (\text{float}) 2;$

37. Per a veure com actuen les funcions d'E/S que hem vist fes un programeta per a provar-les amb valors de diferents tipus.