RESUM INTRODUCCIÓ A `C`

Estructura d'un programa en llenguatge C

```
/* nom del programa */
/* breu explicació del que fa el programa */

/* llista d'importacions o inclusions */
#include ...        /* s'ha de posar sempre #include <stdio.h> */


/* definició de constants */
#define ...

/* declaracions de variables globals i prototipus de funcions */
tipus funció_1 (tipus, tipus, ...);
⋮

int main(void) {
    /* declaracions de variables locals */
    /* instruccions */
}

tipus funció_1 (par_formal, par_formal, ...) {
    /* declaracions de variables locals */
    /* instruccions */
}
⋮
tipus funció_n (par_formal, par_formal, ...) {
    /* declaracions de variables locals */
    /* instruccions */
}
```

Breu descripció

- La **llista d'importacions o d'inclusions** s'utilitza per indicar fitxers especials que seran inclosos en el programa pel preprocessador abans de la compilació. Corresponen als `import` del llenguatge JAVA. Solen portar extensió **h** (de "header", capçalera). El fitxer `stdio.h` conté procediments que gestionen entrades i sortides.

- Les funcions són els mètodes del JAVA.

  En la programació de funcions cal tenir en compte tres coses: la declaració del **prototipus** de la funció, la **crida** a la funció i la **definició** del bloc d'instruccions que la formen.

  Els **prototipus de funcions** especifiquen les característiques d'una funció:

  El **tipus de funció** que és el tipus del valor que retorna (`int, char, float`, ...). Si no en retorna es posa `void`.

El **nom de la funció**.

El **tipus dels paràmetres** que rep (`int, char, float`, ...). Si no en rep cap es posa `void`.

La **crida** i la **definició** són anàlogues a les dels mètodes de JAVA.

El pas de paràmetres es fa **per valor** si es tracta de variables no indexades i **per adreça o referència** en el cas de vectors i matrius.

- **L'execució del programa** sempre comença per la funció `main` que després va "encadenant" les altres funcions. Aquesta funció sol posar-se al principi i habitualment no porta arguments.

Variables

Els tipus bàsics de variables són: `int`, `float`, `double` i `char`.

Cadascun d'aquests tipus de variable tindrà una ocupació de memòria de la qual dependrà el rang de valors de les constants que pot contenir. Per exemple, en una determinada implementació de C l'ocupació és la que es descriu a la següent taula:

Codi del tipus	Ocupació de memòria	Rang	Observacions
`int`	2 bytes	Des de -32768 a 32767	$2^{16} = 65536$ 65536/2=32768
`float`	4 bytes	Des de 10^{-38} a 10^{38} (7 dígits)	S'expressa en 7 o 8 dígits depenent de les conversions de base 2 a base 10.
`char`	1 byte	Des del 0 al 255 (2^8 caràcters)	Podrem referir-nos tant al caràcter com al seu codi ASCII segons el context en què aparegui.
`double`	8 bytes	Des de 10^{-308} a 10^{308} (15 dígits)	S'expressa en 15 o 16 dígits depenent de les conversions de base 2 a base 10

Alguns tipus de variables, per exemple `int`, poden ser modificats pels atributs `short` o `long` i `signed` o `unsigned`. Així tindríem, per exemple:

Tipus	Codi del tipus	Constants que emmagatzema	Ocupació de memòria	Rang	Observacions
Entera llarga	`long` o `long int`	Enteres en un rang ampli	4 bytes	Des de -2147483648 a 2147483647	$2^{32} = 2147483648 \times 2$
Entera sense signe	`unsigned int`	Enteres sempre positives	2 bytes	Des de 0 a 65535	$2^{16} = 65536$

Hi ha un tipus especial de "variables" que convé conèixer per entendre bé la lògica de la declaració, la crida i la definició de funcions. Es tracta del tipus `void`.

Tipus	Codi del tipus	Tipus de magnituds	Ocupació de memòria	Rang	Observacions
No dada. Té importància lògica en determinats contextos	`void`	No correspon a cap dada	0 bytes	Cap dada	Per declarar funcions que no tornen cap valor i/o sense paràmetres.

Observem que no hi ha variables específiques per contenir cadenes de caràcters. Les cadenes de caràcters es guardaran en **vectors de caràcters**.

Operadors

Llevat que l'existència de parèntesis ho modifiqui, la taula de precedència d'alguns dels operadors és:

Categoria d'operadors	**Operadors**	**Associativitat**
Operadors mònics	`! ++ -- + - * & sizeof (`*tipus*`)`	De dreta a esquerra
Operadors aritmètics	`* / %`	D'esquerra a dreta
	`+ -`	D'esquerra a dreta
Operadors relacionals	`< <= > >=`	D'esquerra a dreta
	`== !=`	D'esquerra a dreta
Operadors lògics	`&&`	D'esquerra a dreta
	`||`	D'esquerra a dreta
Operador condicional	`?:`	De dreta a esquerra
Operadors d'assignació	`= += -= *= /= %=`	De dreta a esquerra

Entrada i sortida de dades: funcions `printf` i `scanf`

- `printf`: és la funció estàndard per escriure dades. La seva sintaxi és:

  `printf("`*cadena de formats*`",` *expr1, expr2, ...*`);`

  on els arguments *expr1, expr2, ...* són optatius i representen expressions que s'avaluaran.

  Dins de la cadena de formats podran haver-hi: caràcters per ser impresos, seqüències d'escapament i especificadors de format (que indiquen el lloc i el format en què cal escriure una dada).

Format	**Significat**
`%c`	Un únic caràcter
`%s`	Una cadena de caràcters
`%d`	Un enter decimal
`%e`	Un punt flotant amb exponent
`%f`	Un punt flotant sense l'exponent
`%ld`	Un enter decimal llarg
`%le`	Un punt flotant llarg amb exponent
`%lf`	Un punt flotant llarg sense l'exponent
`\n`	Salt de línia, situant-se a l'inici de la línia següent

  En els especificadors de format pot incloure-s'hi el control d'amplituds i precisions. Per exemple,

pels nombres en punt flotant `%`*p*`.`*q*`e` escriu valors reals en format exponencial amb *q* dígits decimals després del punt en la mantissa (*q* indica la **precisió**) amb una amplitud mínima del camp indicada per *p*. *p* i *q* representen enters decimals sense signe.

- `scanf`: és la funció estàndard per entrar dades. La seva sintaxi és:

  `scanf("`*cadena de formats*`",` *arg1, arg2, ...*`);`

  on els arguments *arg1, arg2, ...* són adreces de variables (punters) que consisteixen en el nom de la variable precedit per `&` (l'operador adreça). Per a cada variable que volem llegir haurà d'haver-hi un especificador de format. Els formats són els mateixos que els de la funció `printf`.

Vectors i matrius

- Els vectors i matrius són col·leccions de variables del mateix tipus.
- La declaració respon a la següent sintaxi:

  tipus nom`[`*dimensió*`];` /* vector */
  tipus nom`[`*dimensió_1*`][` *dimensió_2*`] ...;` /* matriu*/

  on *dimensió_i* és una expressió constant. Les matrius són vectors de vectors.

- A l'igual que en JAVA, els índexos comencen en 0 i cal posar molta atenció en **evitar referències a elements fora dels límits**.

Cadenes de caràcters

- Les cadenes de caràcters són vectors que, a cada posició, tenen un caràcter. El final de la cadena (no necessàriament el final del vector) ve marcat pel caràcter nul: `\0`.
- La declaració serà: `char` *nom*`[`*dimensió*`];`.
- L'escriptura es pot fer mitjançant `printf("%s", ...)`. La lectura amb `scanf("%s", ...)` només llegeix fins a trobar el primer caràcter blanc.
- Les funcions relacionades amb cadenes de caràcters es troben a `string.h`.

Fitxers de text

- s'ha de declarar una variable `FILE*`, que servirà per treballar amb el fitxer;
- obertura: mitjançant la funció `fopen` de prototipus

  `FILE *fopen(char *`*nom-fitxer*`, char *`*mode*`)`

  que retorna `NULL` si hi ha hagut algun problema. *nom-fitxer* és el nom que té el fitxer al disc, *mode* pot ser:

`"r"`	llegirem (el fitxer ha d'existir)	`"r+"`	llegirem i escriurem (el fitxer ha d'existir)
`"w"`	escriurem	`"w+"`	escriurem i llegirem
`"a"`	afegirem informació	`"a+"`	afegirem i llegirem

- entrada i sortida: funcions `fprintf` i `fscanf` (declarades a `<stdio.h>`)

  ```
  int fscanf (FILE *fp, char *format, ...)  /* entrada*/
  int fprintf (FILE *fp, char *format, ...)  /* sortida */
  ```

  el punter *fp* és el que ha retornat la funció `fopen` en la crida que hem fet abans d'utilitzar una d'aquestes funcions i serveix per indicar quin fitxer estem utilitzant. Són anàlogues a les funcions `scanf` i `printf`.

- tancament de fitxers: es fa mitjançant la funció `fclose` de prototipus:

  ```
  int fclose(FILE *)
  ```

Punters

- Una **constant punter** és l'adreça d'una variable.

- Una **variable punter** és una variable que emmagatzema adreces d'altres variables. Llavors es diu que la variable punter punteja o apunta a l'altra variable.

- La declaració de variables punter és: *tipus* `*`*nom1,* `*`*nom2, ...* `;`

- Hi ha dos operadors mònics relacionats amb punters: `&` (operador adreça), `*` (operador d'indirecció).

- A part dels valors de les adreces de les variables, un punter pot tenir el valor `NULL` i es pot comparar amb ell.

- Operacions possibles amb punters són assignació d'una constant punter a una variable punter, sumar i restar enters a un punter, restar dos punters a elements d'un vector, comparar dos punters, pas com a arguments de funcions.

- Punters a vectors:

vector`[`*index*`]`	és equivalent a	`*(`*vector* `+`*index*`)`
`&`*vector*`[`*index*`]`	és equivalent a	*vector* `+`*index*.

- Les funcions d'**assignació dinàmica de memòria** (`stdlib.h`) responen als prototipus següents:

  ```
  void *malloc(size_t);
  ```

  ```
  void *calloc(size_t, size_t);
  ```

  ```
  void *realloc (void *, size_t);
  ```

  on el tipus `size_t` és el tipus d'enter sense signe que retorna l'operador `sizeof`.

  Retornen l'adreça de la primera posició de memòria reservada. Si no hi ha suficient memòria retornen el punter `NULL`.

- Per a **alliberar** memòria dinàmica:

  ```
  void free (void *);
  ```

Algunes funcions matemàtiques

Les següents funcions retornen un `double` i tenen paràmetres `double`. Cal incloure l'arxiu `math.h`. En les funcions trigonomètriques els angles estan representats en radians.

`sin(`x`)`	retorna el valor del sinus de x
`cos(`x`)`	retorna el valor del cosinus de x
`tan(`x`)`	retorna el valor de la tangent de x
`acos(`x`)`	retorna el valor de l'arccosinus de x en el rang [0, π]
`asin(`x`)`	retorna el valor de l'arcsinus de x en el rang [-π/2, π/2]
`atan(`x`)`	retorna el valor de l'arctangent de x en el rang [-π/2, π/2]
`atan2(`x`,`y`)`	retorna el valor de l'arctangent de x/y en el rang [-π, π]
`exp(`x`)`	funció exponencial
`log(`x`)`	logaritme natural, $x > 0$
`log10(`x`)`	logaritme en base 10, $x > 0$
`pow(`x`,`y`)`	potenciació x^y (dóna error si $x = 0$ i $y <= 0$ o si $x < 0$ i y no enter)
`sqrt(`x`)`	arrel quadrada
`ceil(`x`)`	retorna el menor enter més gran o igual que x
`floor(`x`)`	retorna el major enter més petit o igual que x
`fabs(`x`)`	valor absolut

Hi ha dues funcions més (els prototipus de les quals es troben a `stdlib.h`) que calculen el valor absolut, però actuen sobre enters senzills o llargs:

```
int abs(int)
long int labs(long int)
```

Els errors de representació interna de números reals fan aconsellable que la comparació entre valors reals es faci sempre amb un marge de tolerància, la petitesa del qual dependrà del tipus de variable real (senzilla o doble), de l'exactitud de les dades que es tracten, de les necessitats de precisió de l'algorisme que es programa, etc. En aquests casos usarem la funció `fabs` com en l'exemple que segueix:

`fabs ( a - b ) <` *tolerància*

Compilació, muntatge i execució d'un programa en C

- Editar el fitxer *arxiu*.**c** per escriure o modificar el programa.
- Compilar un fitxer:

  gcc -c *arxiu*.**c -ansi -pedantic -O -Wall**

  * **-ansi** compila segons l'estàndard C-ANSI
  * **-pedantic** fa que el compilador sols accepti programes font escrits en C-ANSI
  * **-O** permet detectar l'ús de variables no inicialitzades, reduir les dimensions del codi generat i millorar el temps d'execució
  * **-Wall** indica que volem que ens mostri també els missatges d'avís

  genera el fitxer *arxiu*.**o**

- Muntar un programa: **gcc** *arxiu1*.**o** *arxiu2*.**o** ... **-o** *programa.exe* **-lm**
  * **-o** *programa.exe* és optatiu, indica el nom que tindrà l'executable. Per defecte, el nom és `a.out`
  * **-lm** indica que volem linkar les biblioteques matemàtiques
- Executar un programa: **./***programa.exe*

Introducció al programa `gnuplot`

- **plot**: dibuixa funcions i dades en dues dimensions
  - si coneixem una expressió analítica de la funció *fun(x)*: `plot` *fun(x)*
    Per exemple: `plot sin(x)` o `plot x**4`
  - si volem fer la gràfica dels valors que tenim en un fitxer de nom `fit.res`: `plot 'fit.res'`
  - es poden fer dues gràfiques simultàniament:
    Per exemple: `plot sin(x), x*x`
    `plot sin(x), 'fit.res'`
- Si volem dibuixar punts gruixuts o fins, o unir els punts amb línies, s'ha d'afegir al final de cada funció a dibuixar `with` *estil* on *estil* serà: `points`, `dots` o `lines`. Per defecte, les funcions es dibuixen amb línies i els fitxers, amb punts. Per exemple: `plot 'fit.res' w l`
- Si el fitxer té més de dues columnes, hem d'indicar quines variables volem usar. Llavors s'afegeix `using` *xcol*:*ycol*. Per exemple: `plot 'fit.res' u 1:3 w l`
- Per canviar el rang de valors que volem dibuixar: `set xrange[`*xmin*:*xmax*`]` (per les abscisses) o `set yrange[`*ymin*:*ymax*`]` (per les ordenades).
- Si volem que ens dibuixi una quadrícula: `set grid`
- Si volem que ens dibuixi els eixos: `set zeroaxis`
- Per repetir l'últim `plot`: `replot`
- Sortir: `quit`
- Ayuda: `help`
- Si volem guardar el dibuix en un fitxer, per poder-lo imprimir posteriorment:
  `set term postscript`
  `set output '`*fitxer*`'`

  (Si volem la gràfica en color cal posar: `set term postscript color`)

  Per tancar el fitxer i tornar a fer dibuixos per pantalla
  `set output`
  `set term x11`