Taller de simulació medi–ambiental

Eines informàtiques per a la modelització per ordinador

Juan Carlos Cañadas* i Jordi Sellarès†

12 de febrer de 2002

Juan Carlos Cañadas
Barcelona (1963)
Doctor en Física
Dept. de Física i Eng. Nuclear (ETSEIT)
Universitat Politècnica de Catalunya

Jordi Sellarès
Barcelona (1969)
Doctor en Física
Dept. de Física i Eng. Nuclear (EUETIT)
Universitat Politècnica de Catalunya

*juan.carlos.canadas@upc.es
†jordi.sellares@upc.es

Índex

Presentació

El motiu principal d'aquesta unitat és presentar totes les eines informàtiques que s'utilitzaran per desenvolupar l'assignatura. Inicialment s'explica com procedir a la instal·lació del compilador *gcc* pel llenguatge de programació C. A continuació, s'explica la instal·lació del programa *gnuplot* que s'utilitzarà per a la representació gràfica dels resultats obtinguts a les diferents pràctiques.

En aquesta unitat repasarem conceptes fonamentals del llenguatge de programació C. Sense arribar a ser un curs de programació, s'introdueixen tots els conceptes considerats importants, per poder llegir i entendre els programes de simulació mediambiental que es donen a l'assignatura. Aquest repàs servirà per efectuar, en algunes unitats, modificacions en els programes, tornar a compilar-los i veure així nous resultats.

Objectius

- Introduir els conceptes necessaris per poder fer anar totes les eines informàtiques imprescindibles per a l'assignatura.
- Entendre el funcionament del compilador *gcc* per al llenguatge C.
- Entendre el funcionament del programa *gnuplot* per a la representació gràfica dels resultats.
- Conèixer els conceptes fonamentals de programació del llenguatge C.

Esquema

1. Ús bàsic del *gcc* i del *gnuplot*
2. Conceptes fonamentals de programació de C
   (a) Estructura bàsica d'un programa
      i. `main()`
      ii. `#include`
      iii. `#define`
      iv. operadors
   (b) Instruccions bàsiques
      i. `printf()`
      ii. `scanf()`
   (c) Estructures de control de flux.
      i. `while`
      ii. `do...while`
      iii. `for`
      iv. `if`
      v. `case`
3. Programació estructurada
   (a) Arrays
   (b) Strings
   (c) Funcions
4. Entrada i sortida de fitxers
5. Interacció dels programes amb altres aplicacions

1 Ús bàsic del *gcc* i del *gnuplot*

1.1 Instal·lació del compilador *gcc*

El compilador *gcc* (veure glossari) que recomanem en aquesta assignatura va ser iniciat per la *Free Software Foundation*, que desenvolupa software d'ús i distribució lliure. Actualment l'empresa *Cygnus* s'encarrega de la seva actualització. A continuació, es descriu com fer la instal·lació d'aquest programa en un PC amb *Windows 95* o superior. També existeixen versions per a altres plataformes i/o sistemes operatius.

- Requeriments
  - PC 486 o superior
  - mínim 4 MB de RAM
  - Sistema operatiu *Windows* des de la versió 95
- Arxius que han de descarregar-se: `full.exe`
- Instal·lació
  1. Descarregar i guardar l'arxiu `full.exe` a un directori temporal, per exemple `c:\tmp`
  2. Seleccionar el fitxer `full.exe` des de l'explorador de *Windows* i fer un doble clik.
  3. Al començament de la instal·lació, el programa ens dóna la benvinguda, farem clik a NEXT.
  4. Sortirà informació sobre la llicència del software, farem clik a YES.
  5. Sortirà informació general, farem clik a NEXT.
  6. Ens aconsella en quin directori instal.lar els fitxers (`c:\cygnus`), farem clik a NEXT.
  7. Ens aconsella on localitzar l'icona de *cygwinB20*, farem clik a NEXT.
  8. Fi de la instal·lació, farem clik a FINISH.
  9. Des del menú d'inici del *Windows* seleccionar el menú de programes i obrir una finestra de *MS-DOS*. En aquesta finestra sortirà `c:\windows`. Heu de posar `cd ..` i sortirà el directori principal (`c:\`). Feu `edit autoexec.bat` i afegiu la següent línia al final

```
SET PATH=C:\CYGNUS\CYGWIN-B20\H-I586-CYGWIN32\BIN;%PATH%
```

10. Sortir de l'*edit*, guardant el fitxer `autoexec.bat`.
11. Reiniciar el PC.

1.2 Editar, compilar i executar programes

L'edició dels programes del llenguatge C es pot realitzar mitjançant qualsevol editor de text convencional. Es pot fer servir l'editor (*edit*) del *MS-DOS* que és suficient per a l'edició de programes no molt llargs. Per evitar problemes de compilació al finalitzar el programa és important guardar-lo en mode text amb l'extensió `.c`.

> **Atenció:** Si es vol, també es pot utilitzar el processador de textes *Word* però és important no guardar el fitxer com a document `.doc` sinó com a text amb la extenxió `.c`

Un cop acabada l'edició del programa s'ha d'efectuar la compilació. La forma més senzilla de compilar programes en C és utilitzant un compilador de línia. Aquest tipus de compiladors existeixen per a tots els sistemes operatius.

La sintaxi de compilació en línia emprant el compilador *gcc* és:

```
gcc [opció | nom\_fitxer] ...
```

on `nom_fitxer` indica el nom del fitxer a compilar.

El compilador interpreta per defecte quins codis font ha d'incloure a la compilació. El tipus d'aquests codis es pot conèixer per l'extensió del fitxer. Les extensions més importants són

- `.c`: Codi font C
- `.h`: Fitxer de capçalera
- `.cpp`: Codi font C++

Existeixen opcions per indicar al compilador quines etapes volem que es realitzin en el procés de compilació o com volem que es dugui a terme aquesta compilació. Abans de cadascuna d'aquestes opcions s'ha de posar el signe `-`. Les opcions més freqüents són

- `-c` per realitzar solament el preprocesament i la compilació dels fitxers font. No fa l'enllaçament de fitxers.
- `-E` per realitzar solament el preprocesament dels fitxers font. No fa ni la compilació ni l'enllaçament de fitxers.
- `-o nom_fitxer` fitxer de sortida que emmagatzema el resultat de la compilació en el fitxer que es vulgui. Si no es posa l'opció `-o`, el compilador crearà un fitxer executable en `a.exe`.

- `-I nom_directori` especifica el directori (carpeta) on es troben els fitxers de capçalera (`.h`).
- `-L nom_directori` especifica el directori (carpeta) on es troben els fitxers de la biblioteca.
- `-Wall` Mostra tots els missatges d'advertència del compilador.
- `-lm` Opció per incloure funcions matemàtiques.

Per exemple, si es vol compilar el programa `prova.c`, que el fitxer de sortida executable sigui `prova.exe` i que quedi emmagatzemat al directori `c:\tsma`, es fa

```
gcc -o c:/tsma/prova.exe prova.c
```

El compilador crea l'arxiu `prova.exe` i per executar-lo tenim prou amb escriure `prova` i pressionar ENTER.

1.3 Instal·lació del *gnuplot*

El *gnuplot* (veure glossari) serveix per realitzar representacions de corbes. El fitxer comprimit des del que es pot extreure tota la sèrie de fitxers necessaris per a la seva instal.lació s'anomena `gp371w32.zip`.

A continuació es descriu com fer la instal·lació d'aquest programa a un PC amb *Windows 95* o superior.

1. Descarregar i guardar l'arxiu `gp371w32.zip` a un directori temporal, per exemple `c:\tmp`
2. A continuació cal seleccionar el fitxer `gp371w32.zip` des de l'explorador de *Windows* i fer un doble clik. Per descomprimir l'arxiu heu de tenir instal·lat el programa *winzip* (veure glossari).
3. Guardar els arxius descomprimits en el directori que vulgueu (per exemple, `c:\gnuplot`
4. Al directori on siguin els arxius, seleccionar el fitxer `gnupl32.exe` i fer un doble clik. Sortirà la finestra del *gnuplot* i apareixerà el prompt `gnuplot>`.

El programa presenta diverses possibilitats com ara

- Dibuixar gràfiques des de les dades dels fitxers
  - `plot nom_fitxer` dibuixa la gràfica mitjançant punts
  - `plot nom_fitxer with lines` dibuixa la gràfica mitjançant línies.
  - `plot nom_fitxer using 1:3` dibuixa la gràfica prenent la primera columna com a x i la tercera com a y.

```
#include<>  /* importació dels fitxers de definicions
               generals necessaris per al programa, són
               anomenats 'headers' */

#define  /* són macros, definits usualment al
            començament del programa */

        main ()  /* inici del programa */
        {
        instrucció 0  /* declaració de variables */
        instrucció 1  /* 1a instrucció del cos principal */
        ...
        instrucció n  /* última instrucció del cos principal */
        }

/* fi del programa */
```

Taula 1: Estructura bàsica d'un programa en C

- Dibuixar funcions: Per exemple, `plot 4*x**3-3*x**2+7`
- Es poden dibuixar diverses gràfiques alhora. Per exemple, `plot 3*x**2-8,nom_fitxer using 2:3 with lines`
- Fixar rangs. Per exemple,
  - `set xrange [-4:10]` limita les x a valors entre -4 i 10
  - `set yrange [0:100]` limita les y a valors entre 0 i 100
  - `set auto` fa que gnuplot agafi els rangs com vulgui
- Imprimir gràfiques. Primer es fa `set term postscript`, després `set output nom.ps` i finalment `plot el_que_sigui`. El resultat és el fitxer `nom.ps`, un fitxer *postscript* (veure glossari) a on hi ha la gràfica.
- Ajuda. La instrució `help` dóna informació sobre les comandes i subcomandes disponibles.
- Altres opcions permeten modificar colors, estils, títols, ...
- Sortir. Cal fer `exit`

2 Conceptes fonamentals de programació de C

2.1 Estructura bàsica d'un programa

L'estructura bàsica d'un programa en C és pot veure a la taula 1.

En aquest llenguatge de programació la funció `main` inicia l'execució del programa. A dins d'ella es defineix el cos principal del programa, les variables locals associades al cos principal i les crides a les funcions més importants del programa. El nom `main` és fix, al contrari de la resta de funcions. El parèntesi `( )` és obligatori després del nom de la funció. Les instruccions del programa sempre han d'ésser incloses entre els símbols `{}` que indiquen l'inici i la fi del programa.

2.1.1 Comentaris

Els comentaris personals, que no formen part del programa, són importants a l'hora de conèixer el contingut. Poden anar col·locats a qualsevol lloc i es troben entre els símbols `/*` i `*/`. Aquests comentaris poden ser de vàries línies, però cal anar amb compte de tancar-los, perquè sinó poden englobar tot el text que ve al darrera, és a dir, les sentències de C que apareixen abans del següent fi de comentari.

2.1.2 directives de compilació

A l'inici del programa es troben diverses directives de compilació, indicades per `#include`. Aquestes directives serveixen per incloure els fitxers de capçalera (*headers*) de les llibreries de C. Aquests fitxers contenen informació que li cal al compilador a l'hora de fer referència a variables, tipus, constants i funcions externes que són proporcionades per les llibreries estàndar. Els fitxers *headers* es poden editar i, per aquest motiu, es poden analitzar les definicions de funcions que contenen.

> **Atenció:** Per conveni internacional, els noms dels fitxers Header tenen l'extensió `.h`

A la taula 2 s'ofereix una descripció d'alguns dels fitxers `*.h` considerats més importants.

Les funcions estàndar, contingudes en aquests fitxers, estan disponibles en tots els entorns on es pugui programar en C. Per tant, es poden transportar els programes que únicament fan servir aquestes llibreries entre diferents compiladors de llenguatge C, així com entre diferents sistemes operatius.

Els `#define` són macros que usualment es posen al començament del programa. La definició comença per `#define`, seguida del nom simbòlic i del valor a substituir durant la compilació. Es fa servir per definir valors constants que s'utilitzaran al llarg del programa, per exemple:

```
#define PI 3.141596  /* macro que defineix la constant pi*/.
```

És usual escriure el nom simbòlic en majúscules per distingir-los de les variables, que es posen en minúscula.

complex.h	Utilització de números complexes
conio.h	Control de sortida a la pantalla
dir.h	Gestió de directoris
dos.h	Funcions diverses per al control del sistema
float.h	Paràmetres per rutines de coma flotant
io.h	Rutines d'entrada i sortida de baix nivell
math.h	Funcions matemàtiques
mem.h	Gestió de memòria i cadenes de caràcters
process.h	Control de programes externs
search.h	Funcions de cercar i substituir
share.h	Fitxers compartits
stdio.h	Gestió de sortida a la pantalla
stdlib.h	Funcions d'us més freqüent
string.h	Manipulació de cadenes de caràcters
time.h	Rutines per conversió de formats de temps

Taula 2: Principals fitxers de capçalera

NOM	TIPUS	RANG
char	Caràcter	-128 a 127
unsigned char	Caràcter o número positiu	0 a 255
int	Enter	-32768 a 32767
unsigned int	Enter positiu	0 a 65535
long	Enter llarg	-214783648 a 214783647
unsigned long	Enter llarg positiu	0 a 4294967295
short	Enter curt	-32768 a 32767
float	Real	3.4E-38 a 3.4E38
double	Real doble precisió	1.7E-308 a 1.7E308
long double	Real doble precisió llarg	3.4E-4932 a 3.4E4932
void (no res)		

Taula 3: Tipus de variable en C

2.1.3 Tipus de dades i declaració de variables

A la taula 3 estan sumaritzats els diferents tipus de variable disponibles en C.

`void` és el tipus de dades buit, i es fa servir per expressar que una funció no retorna res, o no rep cap paràmetre.

Hi ha la possibilitat de definir `signed` o `unsigned`. `unsigned` especifica que només s'admetran valors positius i, per tant, permet treballar amb el rang desplaçat (per exemple, `unsigned char` tindrà un rang de 0 a 255 mentre que `signed char` el té de -128 a 127).

Les variables del tipus `char` són un caràcter però alhora es poden considerar com un número. Per exemple,

```
c = 'A';  /* c pren per valor el número ASCII de la lletra "A" */
```

Les dades numèriques reals usualment són donades amb 7 xifres significatives (`float`). Existeixen dues formes de representar les constants reals

- notació decimal (coma fixa). Per exemple, `0.0327`
- notació científica (coma flotant). Per exemple, `3.27E-2`

Per definir una variable primer es posa el tipus i després el nom de les variables d'aquell tipus. Per exemple,

```
int i, j;  /* i, j són números enters */
char car;  /* car és una variable de caràcter */
float x,y;  /* x, y són números reals */
```

2.1.4 Operadors

- Operadors aritmètics
  - Suma `+`
  - Resta i signe menys `-`
  - Multiplicació `*`
  - Divisió `/`
  - Mòdul (resta divisió entera) `%`
  - Autoincrement (augmenta una variable en 1) `++`
  - Autodecrement (disminueix una variable en 1) `--`
- Operadors de relació
  - Igual `==` (és diferent del signe igual d'assignació)
  - Diferent `!=`

  - Menor o igual <=
  - Major o igual >=
  - Major >
  - Menor <

- Operadors lògics

  - I lògic (AND) &&
  - O lògic (OR) ||
  - No lògic (NOT) !

- Operadors d'assignació

  - Assignació simple =
  - Multiplicar i assignar *=
  - Dividir i assignar /=
  - Mòdul i assignar %=
  - Restar i assignar -=
  - Sumar i assignar +=

- Operadors de direcció (tractament de bits)

  - Operació AND & (també representa la direcció d'una variable) valors)
  - Operació OR |
  - Operació XOR ^
  - Operació NOT ~ (NOT bit a bit)
  - Desplaçament a la dreta >> (desplaçar un cert número de bits)
  - Desplaçament a l'esquerra <<

- Operador condicional

  Podem efectuar una operació condicional del tipus "si <condició> llavors retorna <A> sinó retorna <B>" mitjançant

  ```
  <condició>?<A>:<B>
  ```

  Per exemple, si volem $x = a$ quan $a \geq b$ i que $x = 5$ en cas contrari,

  ```
  x = (a>=b) ? a : 5;
  ```

  Un altre exemple,

  ```
  (x>5) ? printf("a"):printf("b");
  ```

```
char c;
int i;
float real;

c = 'A';  /* tipus char, c pren per valor el
             número ASCII de la lletra "A" */
i = c+1;  /* c es converteix a enter, perquè 1 és un
             enter, se sumen enters, i el resultat
             és enter */
c = i;  /* el valor numèric de i es converteix en
           un caràcter */
real = 3*c;  /* c es converteix a enter i el resultat
                de la multiplicació es converteix a real */
c= real/12.0;  /* es divideixen dos reals i el resultat
                  es converteix en un caràcter */
```

Taula 4: Exemples de conversions

En aquest cas si $x > 5$ llavors surt a i en cas contrari surt b.

Aquest operador condicional es tornarà a tractar posteriorment amb la forma alternativa `if...else`.

- Operadors interns

  Quan una expressió d'assignació està entre parèntesi es converteix en un operand per a l'assignació d'una segona expressió, el valor del qual és el resultat de la primera expressió. Per exemple,

  ```
  a=(b+=10); /* primer b=b+10 resultat a=b */
  ```

Per acabar aquesta breu descripció d'operadors i tipus de dades, és necessari assenyalar que en expressions on estiguin involucrades variables i constants de diferents tipus, el C realitza una conversió automàtica de totes les variables de l'expressió al tipus més gran, i després opera. El resultat d'aquestes operacions es converteix posteriorment al tipus de la variable resultat. A la taula 4 hi podem veure un exemple.

2.1.5 Tipus d'instruccions

El llenguatge C, com qualsevol altre llenguatge de programació, disposa de paraules reservades, d'identificadors i d'operadors. Les paraules reservades són les que pertanyen al vocabulari propi del llenguatge (`for`, `else`, `while`, ...). Els identificadors poden ser estàndar, que són com les paraules reservades; o no estàndar, que són aquells que crea el programador utilitzant caràcters alfabètics i numèrics.

Els operadors poden ser de tres classes: aritmètics (+,-,*), de relació (¡=,¿=,=) i lògics (`and`, `or`, `not`).

Les instruccions es poden classificar en:

```
#include <stdio.h>  /* importació del fitxer de
                        definicions generals */
main()  /* inici del programa */
{
printf("programa");  /* imprimeix en pantalla:
                             programa */
}  /* fi del programa */
```

Taula 5: Exemple d'un programa senzill

- Instruccions sense paràmetres: s'utilitzen soles seguides de parèntesi i ";". Per exemple, `clrscr ();` que esborra la pantalla.
- Instruccions amb paràmetres: són instruccions que per la seva execució necessiten dades d'un tipus determinat. Per exemple, `printf ();` requereix paraules entre cometes, més les variables involucrades.
- Instruccions no executables de forma immediata: no produeixen cap efecte fins que s'envia una instrucció per escriure en pantalla.

2.2 Instruccions bàsiques

2.2.1 printf()

La funció estàndard de sortida `printf` serveix per imprimir a la pantalla dades de qualsevol tipus elemental (caràcters, enters, reals, ...) en varis formats (decimal, hexadecimal, punt fix, punt flotant, ...). La forma més simple d'usar `printf` és

```
printf("la frase apareix a la pantalla");
```

També es poden incloure variables a dintre de la cadena indicant amb el signe `%` on sortirà. Per exemple, l'expressió

```
printf("el caràcter %c apareix %d vegades en el text",c,nc);
```

indica que per pantalla ha de sortir la frase esmentada, però substituint (en el lloc on apareixen) els codis `%c` i `%d` pel valor de les dues variables `c` (char) i `nc` (decimal), les variables s'assignen als codis per ordre d'aparició. Han d'usar-se tants signes `%` com variables. Les lletres que hi ha després del signe `%` ens indiquen el tipus de variable de sortida a la pantalla, en aquest cas, `%c` indica que `c` s'ha de visualitzar com a caràcter, i `%d` indica que `nc` s'ha de fer com a decimal. Els codis més usuals són

- `%c` - caràcter
- `%d` - decimal

- `%f` - real punt fix
- `%s` - string
- `%o` - octal
- `%e` - real punt flotant
- `%x` - hexadecimal

Al final de la cadena de sortida, també denominada cadena de format, apareixen uns codis de caràcters de format. Aquests codis són necessaris per especificar instruccions que poden provocar problemes a l'edició del programa. Per exemple, si es vol imprimir el signe `%` i es posa com a tal, el programa interpretarà que el signe és per substituir una variable i no per imprimir el caràcter `%` directament. Per solucionar aquests problemes existeixen seqüències de escapament. Algunes d'aquestes són

- `\n` nova línia (Return)
- `\r` principi línia
- `\f` nova pàgina
- `\v` tabulador vertical
- `\t` tabulador horitzontal
- `\b` retrocés
- `\"` cometes
- `\\` barra invertida
- `\'` apòstrof
- `\%` tant per cent
- `\0` codi nul (ASCII 0)
- `\xhhhh` caràcter ASCII número hexadecimal

2.2.2 `scanf()`

És un mecanisme de lectura d'una dada des del teclat. Pot llegir-se una dada amb `scanf()`, que és una funció general que serveix per introduir qualsevol tipus elemental de dades (caràcters, enters, reals, strings, ...) i donar-li el format adequat mitjançant la cadena de format. Els arguments i codis associats amb `scanf()` són anàlegs als utilitzats amb `printf()`.

Existeixen moltes més instruccions estàndars importants, com `getchar()` (veure glossari), `getch()` (veure glossari) i `strncmp()` (veure glossari).

```
...

while (condició de mentre)
 {
    ...  /* cos del bucle */
    ...  /* increments */
 }  /* fi del bucle */

... /* continuació de la resta del programa */
```

Taula 6: Sintaxi de l'estructura `while`

```
#include <stdio.h>
void main(void)
{
    const x = 20;  /* declaració de constants */
    int y;  /* declaració de variables */

y=1;  /* inicialització de variables */
while (y<=25)  /*inici del bucle */
  {
  printf ("programa\n");  /*imprimeix a la
                              pantalla programa */
  y++;  /* increment y */
  }  /* fi bucle */
}  /* fi programa */
```

Taula 7: Exemple d'utilització de l'estructura `while`

2.3 Estructures de control de flux

2.3.1 Estructura `while`

La sintaxi d'aquesta estructura es pot veure a la taula 6.

És una instrucció repetitiva que s'utilitza quan hom desconeix les vegades que es repetirà el cos del bucle. En aquesta estructura s'avalua la condició. Si aquesta no es compleix, el bucle no s'executa. El cos del bucle solament s'executa si la condició és certa. A la la taula 7 en podem veure un exemple.

2.3.2 Estructura `do...while`

La seva sintaxi ve donada a la taula 8.

En aquesta estructura, la condició de sortida del bucle està al final. És l'expressió que segueix a `while` entre parèntesi. El valor de l'expressió pot ser CERT o FALS. El bucle es continuarà executant mentre la condició booleana de repetir sigui CERTA. És equivalent a l'estructura `while`, però amb la diferència que l'avaluació de la condició es realitza després de l'execució del bucle (sempre s'executarà el bucle almenys una

```
...

do
  {  /*inici del bucle */
    ...  /* cos del bucle */
    ...  /* increments */
  } while (cond);  /* condició de repetir  i fi del bucle */

...  /*continuació del programa */
```

Taula 8: Sintaxi de l'estructura `do...while`

```
...

for (inicialització; condició de mentre; increment)
  {  /*inici del bucle */
    ...  /*cos del bucle */
  }  /* fi del bucle */

...  /*continuació del programa */
```

Taula 9: Sintaxi del bucle `for`

vegada). Una altra diferència entre aquesta estructura i la del bucle d'estructura `while` és que aquí sempre hi ha un punt i coma després de `while`.

S'ha de tenir en compte que un dels errors més freqüents que es poden arribar a fer en aquests bucles és l'anomenat bucle infinit. El programa "es penja" per no incrementar la variable, per no inicializar-la o per utilitzar una condició errònia.

2.3.3 Bucle `for`

A la taula 9 podem veure la seva sintaxi.

L'estructura `for` és una sentència de repetició equivalent a l'estructura `while`. Consta d'una inicialització de variables, d'una condició de fi, i d'una sentència d'increment del comptador del bucle. Un cop s'han iniciat les variables es comprova si la condició és CERTA. Si inicialment és FALSA, el bucle no s'executa. Si l'expressió és CERTA, s'executen les instruccions contingudes a l'interior del bucle i, a continuació, s'efectua l'increment del comptador. Es repeteix el procés mentre es compleixin les condicions. L'única diferència d'aquesta estructura `for` respecte la `while` és conceptual, si el número d'iteracions és conegut s'utilitza la sentència `for`, si no és conegut i depèn del context del programa, llavors es posarà un `while`.

A la taula 10 hi podem veure un exemple.

2.3.4 Estructura condicional `if`

La sintaxi d'aquesta estructura és la que es pot veure a la taula 11.

```
/*programa que mostra a la pantalla la taula de multiplicar
del 1 al 5 */

#include <conio.h>
#include <stdio.h>

void main()
{
int i,y,numero,factor,producte;
for (numero=1; numero<=5; numero++)
{
y=2
for (factor=1; factor <=10; factor ++)
{
producte=numero*factor;
printf("%2i", numero);
printf("x");
printf("%2i", factor);
printf("=");
printf("%3i\n",producte);
y++;
}  /* fi del for 2 */
printf("\n");
}  /* fi del for 1 */
}  /* fi del programa */
```

Taula 10: Exemple d'utilització del bucle `do`

```
...

if (expressió booleana)
 {
  ...  /* instruccions del cos del LLAVORS */
 }
else  /* optatiu */
 {
   ...  /*instruccions del cos del SINÓ */
 }
...  /*continuació del programa */
```

Taula 11: Sintaxi de l'estructura condicional `if`

Existeixen tres tipus d'estructures condicionals

- condicional simple

  ```
  if (expressió booleana) (instrucció del LLAVORS);
  ```

  La instrucció o instruccions controlades per una expressió condicional simple s'executa tan sols si l'expressió és CERTA. Si només hi ha una instrucció, no són necessàries les claus. L'expressió booleana sempre ha d'anar entre parèntesi.

- condicional doble

  ```
  if (expressió booleana) (expressió del LLAVORS);
  else (expressió del SINÓ);
  ```

  Tan sols s'executa una de les dues instruccions, o blocs d'instruccions, alternatives, en funció de ser CERTA o FALSA l'expressió del control.

- condicional múltiple

  És tornar a repetir els bucles `if ...else`, `if ...else`, ..., i les distintes opcions s'executen o no segon els valors de les variables de control. En comptes d'utilitzar aquesta opció, usualment el condicional múltiple es fa amb l'estructura `switch()` que es comenta a continuació.

2.3.5 Estructura `switch...case`

La seva sintaxi és a la taula 12.

Si no hi ha `break` al final de les sentències de cada cas, s'executaran les sentències del cas següent, ja que el codi generat per a tots els casos està situat consecutivament a la memòria, i les paraules reservades (`case`) fan de simples etiquetes del punt a on es salta (quan s'executa el `switch`) si es compleix la igualtat de l'expressió amb el valor de cada cas.

Només es pot estalviar el `break` del final de les sentències en l'última línia (sigui un `default` o no).

Abans de continuar mira de fer les activitats 1 i 2.

3 Programació estructurada

3.1 Arrays

Un array és un tipus de variable estructurada que conté un conjunt d'elements o dades, del mateix tipus, ordenats mitjançant un o varis índex. Són matrius d'elements que poden tenir des d'una fins a n dimensions. Per definir un array s'ha d'escriure el tipus de dades i el nom de la matriu. Després del nom s'han d'escriure tants claudàtors `[ ]` com dimensions tingui la matriu, indicant el número d'elements per a cada dimensió. Per exemple, la declaració d'un array de dues dimensions de dades reals i format 3x4 és

```
...

switch (expressió)  /* una expressió qualsevol (en
                        particular, una variable) */
 {
    case valor1 :  ...  /* cas (expressió==valor1) */
      break;
    case valor2 :  ...
      break;  /* break salta al final
                 de l'estructura switch */
    ...
    case valorn :  ...
      break;
    default : ...  /* Altrament de tots els altres
                      casos (opcional) */
 }
...  /* continuació del programa */
```

Taula 12: Sintaxi de l'estructura `switch...case`

```
matriu 3x4:

m[0][0], m[0][1], m[0][2], m[0][3]
m[1][0], m[1][1], m[1][2], m[1][3]
m[2][0], m[2][1], m[2][2], m[2][3]
```

Taula 13: Representació d'una matriu

```
float     m[3][4];
```

on `m` és el nom de l'array. Normalment s'adopta el conveni segons el qual els arrays de dues dimensions s'emmagatzemen per files. Segons això, el primer índex fa referència a la fila i el segon a la columna. En qualsevol cas, el primer índex és el més significatiu i l'últim índex és el menys significatiu. Per aquesta raó, els elements es van emmagatzemant per variació dels índexs de més a la dreta cap als índexs de més a l'esquerra, com es pot veure a la taula 13.

Les operacions amb arrays poden ser globals o per elements.

- Operacions globals:
  - Assignar: `array1=array2`
  - Comparar: `if (array1=array2) (instrucció)`
  - Lectura/escriptura a un fitxer
- Operacions per elements: Es fa sobre elements individuals. Cada element de la matriu es comporta com una variable del tipus que s'ha definit l'array. Les operacions més simples són

```
for (i=0; i<3; i++)
for (j=0;j i<4; j++)
printf(array1[i][j]);
```

Taula 14: Exemple de bucles niats

```
#include <stdio.h>
void main(void)
{
    char sal[5];  /* vector de 5 caràcters */

sal[0]='h';  /* el primer element del vector és
                 sal[0] */
sal[1]='o';
sal[2]='l';
sal[3]='a';
sal[4]='\0';
printf("%s\n",sal);  /* %s indica que s'ha d'imprimir
                        un string, imprimeix tots els
                        elements del string */
}
```

Taula 15: Aquest programa defineix un vector de caràcters i l'omple, un per un, amb una cadena més el corresponent fi d'string

- Assignar: `array1[5]= 100;`
- Llegir: `scanf(&array1[i]);`
- Escriure: `printf(array1[i]);`
- Recorrer tot el array. Es fa mitjançant bucles niats (veure taula 14)

3.2 Strings

Els strings són arrays de caràcters (`char`). L'única diferència és que, per conveni, l'últim element d'un string sempre acaba amb `\0`. Per tant, a l'hora de definir una cadena de caràcters, cal tenir en compte la longitud màxima que es preveu per a la cadena, més un element addicional per guardar un `\0` de fi de string (cal que existeixi sempre). Les cadenes de caràcters es declaren com un array

```
char nomstring [núm_màx]
```

a on `núm_màx` ens indica el número màxim de caràcters que pot tenir la cadena. A la taula 15 hi podem veure un exemple.

Com als arrays, amb un string es poden realitzar operacions element a element o operacions globals.

Ara ja pots fer l'activitat 3.

```
tipus nomfun(tipus de paràmetres)  /* defineix nom i tipus de funció */
{
...  /* declaració de variables locals */
...  /* instruccions */
...  /* crides  a altres funcions */

return(paràmetres retornats)  /* retorn dels paràmetres */
}

main ()  /* comença la funció principal */
```

Taula 16: Opció 1 per a la sintaxi d'una funció

3.3 Funcions

Una funció es defineix com una sèrie d'instruccions d'un programa que s'executen cada cop que es crida a funció. La funció s'ha de declarar abans de cridar-la i abans de la funció principal `main()`. Tanmateix, el seu cos principal pot situar-se a continuació de la declaració de la funció o després de les instruccions de la funció principal `main()`.

Per definir-la, primer cal escriure el tipus de valor que retorna aquella funció, el nom de la funció, i la llista dels paràmetres entre parèntesi, especificant el tipus i el nom de cada paràmetre. Si aquests paràmetres són del tipus array, es posen els claudàtors sense número d'elements. Aquest número serà el de l'array que es passi per paràmetre quan es cridi la funció.

Si es posa el tipus `void` és per indicar que la funció no retorna cap dada després de la seva execució. Si tampoc té paràmetres, és obligatori per C++ posar void dins del parèntesi de paràmetres. Per exemple: `void funció (void)`. Si es treballa en C i no s'especifica res, s'identifica com funció tipus `int` (enter).

Hi ha dues opcions per a la sintaxi d'una funció, que es poden veure a les taules 16 i 17.

4 Entrada i sortida de fitxers

Els exemples que s'han tractat fins ara han demanat informació que s'ha introduït mitjançant el teclat i han mostrat per la pantalla els resultats. Aquestes dades, però, desapareixen en finalitzar el programa, ja que al programa no hi havia instruccions per guardar-les. De vegades, la informació per introduir o per visualitzar és massa extensa com per teclejar-la o per veure-la en pantalla. En aquests casos és interessant poder llegir o gravar la informació en fitxers registrats al disc dur, disquet o altres medis materials.

Quan al programa s'especifica que els canals d'entrada o de sortida d'informació són fitxers, i no el teclat o la pantalla, el programa llegeix o escriu informació als fitxers. El S.O. s'encarregarà d'obrir els fitxers i de transferir la informació entre aquests i el programa.

```
tipus nomfun(tipus de paràmetres);  /* prototipus de la funció */

...  /*definició de la resta de variables i
       funcions */

main()  /* comença la funció principal */
{
...  /*instrucions funció principal */
}
tipus nomfun(tipus de paràmetres);  /* defineix nom i tipus de funció */
{
...  /* declaració de variables locals */
...  /* instruccions */
...  /* crides a altres funcions */

return(paràmetres retornats)  /* retorn dels paràmetres */
}
```

Taula 17: Opció 2 per a la sintaxi d'una funció

Per accedir als fitxers directament des del programa disposem d'una àmplia sèrie de funcions estàndar, però les funcions d'ús més freqüent és més reduïda i segueixen una seqüència típica

1. Declaració de variable punter a `FILE (FILE *)`. Aquesta declaració és obligatòria i es pot fer com a variable local o global. Per exemple `FILE *fitxer1, *fitxer2;`.
2. Obrir fitxer (`fopen`). Per obrir un fitxer s'utilitza normalment la funció `fopen()`, que requereix dos paràmetres de tipus string (cadena de caràcters).

   La seva sintaxi és `fopen(nomfitxer, mode d'accés)`. El primer paràmetre és el nom del fitxer que es vol obrir (nom físic) i el segon paràmetre és el mode d'accés al fitxer (`r` llegir, `w` escriure, ...). Si l'operació es realitza amb èxit, `fopen` torna un punter (adreça de la variable) a `FILE`, que serà emmagatzemada sobre una variable del programa. Aquesta variable serà la que es farà servir per identificar el fitxer en totes les altres funcions.
3. Escriure o llegir al fitxer. Existeixen moltes funcions per a la lectura i escriptura dels fitxers, algunes de les més importants són

   - `fread()` funció de lectura de dades d'un fitxer binari
   - `fwrite()` funció d'escriptura de dades d'un fitxer binari
   - `fprintf()` funció d'escriptura de dades amb format. S'utilitza com `printf()`
   - `fscanf()` funció de lectura de dades amb format. S'utilitza com `scanf()`
   - `fgetc()` funció de lectura d'un caràcter
   - `fputc()` funció d'escriptura d'un caràcter

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

void main(void)
{
 char lin[80];
 FILE *fit;

 printf("Nom d'un fitxer : "); scanf("%s",lin);

 fit = fopen(lin,"r");
 if(fit==NULL) printf("Error obrint fitxer\n");
 else
 {
    while (!feof(fit));
     {
        fgets(lin,80,fit);
        if(strncmp(lin,"#define",7)==0)
printf("%s",lin);
     }
    fclose(fit);
 }
}
```

Taula 18: El programa imprimeix les línies d'un fitxer

4. Tancar el fitxer (`fclose`). És important no oblidar-nos de tancar tots els fitxers abans de sortir del programa, per no perdre informació. La funció que tanca el fitxer és `fclose (FILE*fitxer)`. porta solament un paràmetre que és el punter a `fitxer FILE*fitxer`. A la taula 18 hi podem veure un exemple.

Utilitzant aquestes idees, fes les activitats 4 i 5.

5 Interacció dels programes amb altres aplicacions

Les gràfiques realitzades amb el *gnuplot* es poden importar, en windows, amb el sistema tallar/enganxar. Primer heu de fer la representació al *gnuplot* i fer clik amb el botó dret del ratolí. Llavors sortiran totes les opcions possibles. Seleccioneu l'opció *copy to clipboard*, torneu al vostre processador de text (per exemple *Word*) i al menú d'edició seleccioneu *enganxar*. D'aquesta forma ja disposeu de les gràfiques al document que estigueu editant.

Resum

Hem explicat com instal·lar el compilador *gcc* (GNU C Compiler) i el programa de representacions gràfiques *gnuplot*. S'ha explicat esquemàticament com editar i compilar en línia els programes C. A més, s'han descrit instruccions bàsiques i suficients per dibuixar funcions i dades experimentals mitjançant el programa *gnuplot*.

En els conceptes fonamentals de programació de C s'ha tractat l'estructura bàsica d'un programa amb la descripció de les funcions i arxius: `main`, `#include`, `#define` així com els operadors útils per a la programació. S'han analitzat diferents formes d'estructures de control de flux amb bucles `while`, `do...while`, `for`, `if`.

En la programació estructurada hem vist el concepte d'array i les seves aplicacions, així com també els strings, que són arrays de caràcters. Una particularitat important que hem destacat del string és que, l'últim element d'un string sempre s'acaba amb `\0`. A continuació s'ha definit les funcions com una sèrie d'instruccions d'un programa que s'executen cada cop que es crida a la funció.

Per acabar aquesta introducció a la programacio hem tractat els fitxers com una forma útil d'emmagatzemar dades i hem repassat les funcions d'ús més freqüent per obrir, escriure, llegir i tancar els fitxer de dades.

Glossari

gnuplot És una eina interactiva per dibuixar gràfiques. El fitxer comprimit `gp371-w32.zip` correspon a la versió 3.7 per a *Windows* de 32 bits. Aquest fitxer es troba a l'adreça http://www.ucc.ie/gnuplot/gnuplot-faq.html.

winzip Programa que serveix per descomprimir arxius. Si no el teniu instal·lat, el podeu descarregar de la pàgina web http://www.winzip.com.

postscript És un llenguatge de programació òptim per a impressores. És una marca registrada per *Adobe Systems Incorporated* (1985). Si disposeu d'impressora tipus *postscript*, no hi ha cap problema. Però en el cas de que no tingueu accés a aquesta mena d'impressores, necessitareu programes per interpretar el llenguatge *postscript* i convertir els fitxers *postscript* a altres formats. El programa més aconsellable és el *gsview/ghostscript*.

gsview/ghostscript Són els programes de distribució lliure més populars que serveixen per imprimir els fitxers *postscript* en qualsevol impressora. *ghostscript* és el nom d'un conjunt de software que inclou: un intèrpret pel llenguatge *postscript*, amb la capacitat de convertir fitxers *postscript* a diversos formats, mostrar-los per pantalla i imprimir-los en qualsevol impressora, un intèrpret per fitxers `*.pdf`(*Portable Document Format*), amb les mateixes capacitats. Per a OS/2, Win16 i Win32 hi ha disponible *gsview*. *gsview* és un interfície gràfica per a *ghostscript* que fa molt fàcil el maneig i, per tant, el tractament de documents *postscript*. Els programes *gsview/ghostscript* es poden descarregar des de l'adreça http://www.cs.wisc.edu/∼ghost/.

gcc GNU C Compliler. Els programes en C d'aquesta assignatura estan pensats per correr en un compilador ANSI estàndar. Un compilador d'aquestes característiques, per a *Windows 9x/NT/2000*, és el *cygwin* que es pot trobar a ftp://sunsite.cnlab-switch.ch/mirror/cygwin/.

getchar() Retorna el codi ASCII del caràcter llegit pel teclat.

getch() Atura l'execució del programa fins que es polsi una tecla qualsevol.

strncmp() Compara dues cadenes.

Referències addicionals

- **Enrique Alonso Gutiérrez:** *Tutorial para turbo C y C++*. Ediciones Octaedro, (1997)..

  Llibre sobre el llenguatge C que tracta de forma "tutorial" el seu aprenentatge. Els seus continguts segueixen un criteri pedagògic amb molts exemples pràctics.

- **Brian W. Kernighan, Dennis M. Ritchie:** *El lenguaje de programación C*. Editorial Prentice Hall, Upper Saddle River (1988).

  Llibre bàsic per aprendre els fonaments del llenguatge de programació C. El seu contingut és molt ample, des de conceptes inicials simples fins a conceptes més avançats.

Activitats

1. Fer un programa que imprimeixi la taula dels codis ASCII del 1 al 32, mostrant codi ASCII i caràcter.

2. Fer un programa que pregunti un número i imprimeixi el seu factorial

3. Fer un programa que compti les lletres d'una paraula d' entrada.

4. Fer un programa que calculi 100 punts de la funció

   $$y = 3x^3 + 2x^2 + 1$$

   des de 0 fins a 100 i els resultats quedin guardats a un fitxer.

5. Fer, mitjançant el gnuplot, la representació gràfica de l'exercici anterior, primer per lectura directe del fitxer, i després mitjantçant la instrucció de dibuixar funcions del gnuplot. Comparar els resultats.

6. Realitza les activitats descrites al guió de pràctiques corresponent a aquesta unitat.

Exercicis d'autocomprovació

1. Quin dels següents bucles for no s'executarà cap vegada?

   (a) `for (s=0, i=1; i<=max; s += i, i++);`

   (b) `for (s=0, i=1; i<=max; i++);`

   (c) `for (s=0, i=1; i=max; s *= i,i++);`

   (d) `for (s=0, i=1; i==max; i++, s += i);`

2. De les següents afirmacions, sobre els resultats generats per operadors, quina és falsa?

   (a) `a=(b+=5); - b=b+5; a=b;`

   (b) `a=(c++)*2; - a=c*2; c=c+1;`

   (c) `a=(++c)*2; - c=c+1; a=c;`

   (d) `a=3*(b=5+c*2)-1 - b=5+c*2; a=3*b-1;`

3. Quin dels següents bucles i/o afirmacions del condicional if és incorrecte?

   (a) `if (a>=b) c='X'; else c='Y';`
   és equivalent a:
   `c=((a>=b)?'X':'Y');`

   (b)
   ```
   if ((a==b) && (!(b>c) || (a!=6)))
   {
   x*=a+b;
   y=x/(a-b) % 4;
   }
   ```

   (c) `if (a==(b=c)) a>>=2;`
   és equivalent a:
   `b=c; if (a==b) a>>=2;`

   (d)
   ```
   if ((a>=b) && ||c=3)
   {
   x*=a+b;
   }
   ```

Solucions dels exercicis d'autocomprovació

1. (d) 2. (c) 3. (d)