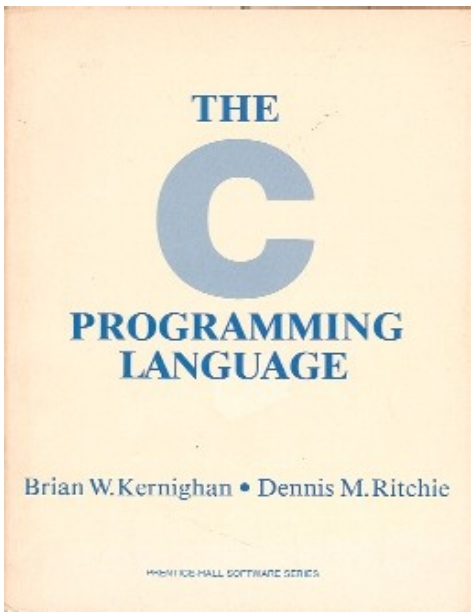
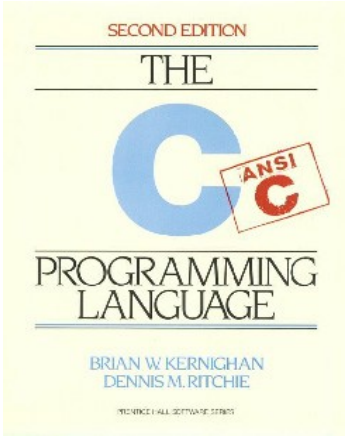


FEM UN PROGRAMA EN LLENGUATGE C



Jairo Vadillo Martín
2n de batxillerat A
Tutora: Imma Compte
Curs: 2007/2008
Institut: IES Frederic Martí Carreras
Data de presentació: 16/01/08

"Si compila esta bien, si arranca es perfecto."
Linus Torvalds

ÍNDEX

Introducció

Una mica d'història

Des de la creació de les primeres computadores, aquestes sempre han necessitat un software per tal de poder funcionar. El processador només entén el codi màquina o binari i com que llegir i aprendre aquest llenguatge per adaptar-lo a cada processador és molt difícil es van inventar els llenguatges de programació que són més fàcils d'entendre i els quals mitjançant uns intermediaris es transformen en codi màquina. Els primers llenguatges en aparèixer van ser llenguatges d'alt nivell (explicat després), com per exemple el Fortran i el COBOL.

El llenguatge C va sorgir entre 1969 i 1973 als Laboratoris Bell de AT&T, aquest llenguatge va ser una evolució del llenguatge B que a la vegada estava basat en BCPL. El llenguatge C es fa servir per crear sistemes operatius i també aplicacions, encara que va ser dissenyat per desenvolupar sistemes operatius.

El llenguatge C va ser el resultat del treball de dos programadors, Brian Kerninghan i Ken Thompson, creador del llenguatge B. Aquests volien portar el sistema operatiu UNIX d'una microcomputadora PDP-7 a una PDP-11. El problema era que aquest sistema operatiu estava escrit en ensamblador. Ensamblador és un sistema que fa que el codi màquina sigui una mica llegible mitjançant uns mnemònics, és a dir, transforma un 0010001010001 en el mnemònic MOV, provinent de "move"(moure), per tal de no haver-se d'aprendre tots els zeros i uns del codi màquina. Això feia que fos costós transportar el sistema operatiu degut a que s'havia de canviar totalment per adaptar-lo al processador de la computadora que l'anava a acollir.
Aquest fet va portar a B. Kerninghan i K. Thompson a voler traduir el sistema operatiu al llenguatge B però al veure que al B li mancaven certes funcions van decidir crear el llenguatge C que es va convertir en el llenguatge més utilitzat per fer sistemes operatius, tant que es va reescriure el nucli (kernel) de UNIX al llenguatge C i al poc temps el 70% dels sistemes operatius estaven fets en llenguatge C, degut a que el llenguatge C és molt portàtil, flexible i potent.
Al 1978 dos programadors, B. Kerninghan en companyia de D. Ritchie van introduir millores al llenguatge C i van crear el primer llibre que explicava el llenguatge, el "The C Programing Language". Al llenguatge que apareix en aquest llibre se li va donar el nom vulgar de "K&R C", en honor als dos programadors.
Entre 1983 i 1989 la *American National Standards Institute* (ANSI) va formar un comitè per normalitzar el llenguatge C. El llenguatge que va sorgir d'aquesta estandardització se li va donar el

nom de ANSI C. Al cap d'un any va sorgir l'estandardització ISO (International Organization for Standardization) que encara ara és l'internacional. Tot i així, l'estandardització més usada i compatible és l'ANSI.

Motius de la tria

El motiu principal pel qual he triat aquest tema és perquè m'agrada molt la programació i trobo que és un tema força entretingut. En aquest treball vull aconseguir, a més de poder desenvolupar un programa en llenguatge C, trobar l'explicació del funcionament de certes parts dels programes i de tots els elements que calen per programar, ja que cada dia utilitzem programes però poca gent sap quines són les bases del que està utilitzant.

He triat la programació en llenguatge C perquè crec que per programar és bàsic saber fer-ho en C perquè molts dels llenguatges importants estan molt influenciats pel llenguatge C, com per exemple el C++ i el Java, també perquè el llenguatge C com he dit abans és molt portable, ja que es pot passar de Windows a UNIX sense quasibé cap canvi, flexible i potent.

A més, avui en dia, la majoria de coses estan controlades per ordinadors que cada cop necessiten un software més sofisticat i penso que aprendre a programar i finalment saber programar té moltes sortides professionals.

Un altre dels motius que m'ha impulsat a fer aquest treball és l'afició que li tinc a la informàtica des de fa molts anys i ja que l'any que ve vull cursar la carrera d'informàtica penso que el treball em permetrà iniciar-me en aquest camp. A més, en aquest treball podré realitzar una part pràctica en la qual desenvoluparé un programa.

Metodologia

El treball estarà dividit en dues parts una clarament teòrica, amb anotacions cap a la part pràctica i una altra part totalment pràctica en la que es veurà el desenvolupament pas a pas del projecte.

A la part teòrica, s'introduiran conceptes bàsics abans d'introduir el llenguatge C, conceptes com el de programa, llenguatge de programació etc. En aquesta part es donarà tota la informació necessària per saber fer sense problemes un programa en llenguatge C mínimament elaborat, per tant, com introduir-hi els tipus de constants i variables, les funcions etc.

A la part pràctica, s'intentarà agafar tota la informació de la part teòrica i fer un programa que tingui alguna funció més que la d'un programa de pràctica, és a dir, que solucioni algun tipus de problema o serveixi per a alguna cosa concreta.

Objectius

L'objectiu principal d'aquest treball és desenvolupar un programa amb les instruccions del llenguatge C, en un cas concret i a la vegada fer que qualsevol persona que llegeixi el treball pugui fer programes iguals amb l'ajuda de les instruccions donades en el treball.

A la part pràctica, es descriurà tot el procés de creació des de la idea original fins a la prova del programa.

Per tant, a part de la teòrica intentaré desenvolupar un manual de C ràpid i fàcil d'entendre, ja que els manuals que hi ha per Internet són llargs i costosos de llegir. Aquesta apropiació a manual permet crear els teus propis programes en pocs dies per tal d'introduir-te en la programació en C, clar que si es vol aprendre més sobre el llenguatge C inclouré un annex amb diverses pàgines i llibres d'interès.

Un altre objectiu és demostrar la potència, la flexibilitat, la portabilitat i la funcionalitat del C envers altres llenguatges de programació.

1 El llenguatge C

1.1 Què és un llenguatge?

La primera computadora que va existir es feia funcionar amb interruptors, un interruptor encès era un 1 i un de tancat un 0, i fent una seqüència del tipus 0011101010 es creaven els programes. A aquest codi d'uns i zeros se'l va denominar codi màquina.

Per tal de facilitar el desenvolupament de programes es va crear el llenguatge ensamblador, que no és un llenguatge en si però ajuda a escriure programes substituint les funcions del binari per mnemònics (ADD és sumar, SUB és restar etc.).

Finalment es van desenvolupar els llenguatges de programació que reben aquest nom per la seva semblança sintàctica amb els llenguatges orals i escrits pels humans, per tant, podem definir llenguatge de programació com un escrit que pot controlar el comportament d'una computadora i que té coherència sintàctica.

Els llenguatges de programació poden acostar-se més o menys al llenguatge humà, i es poden classificar, segons el grau d'abstracció. Hi han aquests tipus:

- Els llenguatges de baix nivell: només n'hi ha un, el llenguatge ensamblador, i tot i així aquest no és un llenguatge en si. Els llenguatges de baix nivell es caracteritzen pel fet que s'allunyen del llenguatge humà i s'apropen molt més al codi màquina. Cada codi escrit en ensamblador s'ha d'adaptar al processador de la computadora, ja que es desenvolupa directament sobre aquest.

Exemple de codi ensamblador per un processador intel:

```
mov ax,cs
mov ds,ax
mov ah,9
mov dx, offset Hola
int 21h
xor ax,ax
int 21h

Hola:
  db "Hola món!",13,10,"$"
```

- Els llenguatges de mig nivell: els llenguatges de mig nivell es caracteritzen pel fet que tenen molta funcionalitat, com els llenguatges de baix nivell, i alhora utilitzen una sintaxis semblant a la dels llenguatges humans. L'avantatge d'aquests llenguatges és que s'escriuen independentment del processador de la computadora, per tant, a diferència dels llenguatges de baix nivell, es poden passar d'una computadora a una altra sense canvis en el codi font. L'exemple més clar és el llenguatge C.

Exemple del llenguatge C:

```
#include <stdio.h>
main()
{
printf("Hola món!\n");
}
```

- Els llenguatges d'alt nivell: són aquells que s'apropen molt més als llenguatges humans que al codi màquina. Els llenguatges d'alt nivell tenen la mateixa avantatge que els de mig nivell, que no s'han d'escriure adaptant-los al processador. El problema d'aquests és que generen un binari (arxiu executable) molt gran i lent. La majoria de llenguatges més moderns són d'aquest tipus, com per exemple el C++ (anomenat c sharp), el *Java*, el *perl*, el *COBOL*, el *bàsic* i molts més.

Exemple de C++:

```
#include <iostream.h>

main()
{
   cout << "Hola mon!" << endl;
   return 0;
}
```

Exemple de basic:

```
10 REM Hola mon in BASIC
20 PRINT "Hola món!"
```

Exemple de Java:

```
class HolaMon {
  static public void main( String args[] ) {
    System.out.println( "Hola món!" );
  }
}
```

Encara que els llenguatges es solen classificar per grau d'abstracció també els podem organitzar per altres mètodes, com per exemple pel paradigma de programació i pel tipus d'execució.

Els llenguatges també es poden classificar pel paradigma o orientació que té el llenguatge. Entre els paradigmes distingim entre el funcional, l'imperatiu (el que fa servir C), l'orientat a objectes i el lògic. Com que aquest tipus de classificació no s'utilitza molt i resulta llarg d'explicar en aquest treball no s'explicarà aquest tipus de classificació.

1.2 Què és i com es desenvolupa un programa?

Un programa està format per un conjunt d'instruccions ordenades, finites i no ambigües (algoritmes) molt detallades que li diuen a la computadora què ha de fer pas a pas per realitzar una tasca determinada.

Els codis dels programes es divideixen en mòduls per tal que siguin més senzills de desenvolupar i d'entendre al llegir-los, ja que es divideix un algoritme que podria ser molt complicat en blocs d'algoritme de menor complexitat.

Per crear un programa el primer que es fa és escriure un codi font en algun llenguatge de programació o directament en ensamblador o codi màquina. El programa es pot escriure en un bloc de notes normal o utilitzant algun programa que t'ajuda en alguns aspectes, encara que no deixa de tenir integrat un bloc de notes normal i corrent. El processador de text més utilitzat per escriure codis és el “vim” (a UNIX) o “vi” (a Windows), ja que és un editor simple, potent, ràpid i permet el colorejat de sintaxi. El “vim” s'executa des de la consola, terminal o línia de comandament del sistema operatiu.

Dintre de codi font es poden incloure comentaris que ajudin a la lectura d'aquest, els comentaris no influeixen en el funcionament del programa i només podem saber si hi són si tenim el codi font.

Un exemple de comentari en llenguatge C és:

```
funció(arguments a)          //aquesta funció té els arguments a
                                //s'han de posar les barres a cada línia
   funció(arguments b)       /*el mateix que abans però el comentari
                           s'acaba quan es tanqui amb */
```

Ara bé hem creat un codi però no està en un llenguatge que pugui llegir el processador de la computadora (el binari), per tant, un cop creat el codi necessitem un intermediari que el faci llegible pel processador. Segons el tipus d'intermediari o d'execució dels codis font podem classificar els llenguatges en tres grups:

- Els llenguatges compilats: en aquests cas necessitem un compilador que consta de dues fases: la primera és traduir el codi font al codi màquina o objecte i la segona és enllaçar amb les llibreries automàticament, quan cal, o amb alguna anotació per tal d'incloure les funcions necessàries i formar així l'executable. Un exemple de compilació d'un programa en llenguatge C des del terminal utilitzant el compilador *gcc (GNU Compiler Collection)* seria:

~$gcc -lm -o arxiu [executable] arxiu.c [codi font amb funcions matemàtiques]

gcc – executa el compilador.
-lm – anotació per fer que el compilador busqui la llibreria dinàmica amb les funcions matemàtiques.
-o – s'utilitza per donar-li nom a l'executable.

- Els llenguatges interpretats: en aquests casos no cal un compilador sinó que l'intèrpret informàtic s'encarrega de generar directament l'arxiu executable mentre s'està executant. És a dir, el codi font no es tradueix si no que l'intèrpret va llegint les instruccions una a una i les va executant, això fa que el programa resultant sigui més lent que un fet amb els llenguatges compilats. L'intèrpret es fa servir en llenguatges de programació com per exemple Perl, Javascript i Python.

Per executar aquests programes només cal introduir, en el cas de Perl:

~$./arxiu.pl
o bé:
~$perl arxiu.pl

- Els llenguatges *Framework:* són llenguatges que combinen la manera de procedir dels dos anteriors, és a dir, primer cal traduir el codi font a un bytecode i aquest executar-lo des d'una màquina virtual, la qual en aquest cas actua d'intermediari. Són exemples de *Framework* el Java i el .net.

Per compilar i executar en Java des del terminal fem:

```
~$javac arxiu.java
~$java arxiu [l'arxiu resultant de la compilació]
```

1.3 Un programa part per part en llenguatge C

1.3.1 Presentació

Els exemples que sortiran al llarg d'aquesta part del treball estan en ANSI C i fets per una computadora basada en UNIX, això no evita que puguem fer servir el programa en Windows ja que el C és molt portable d'un sistema a l'altre.

El llenguatge C és un llenguatge compilat per tant, necessitem un bloc de notes per escriure el codi font i un compilador. El primer que hem de fer és obrir un document en blanc amb el nom que vulguem però acabat en .c, que és la terminació que tenen els codis escrits en C. El nom que li donem a l'arxiu pot no ser el que tindrà al final el programa, ja que el nom final li donarem amb el compilador (Exemple al punt "Compilem el codi amb *gcc"* a la part pràctica del treball).

Ara tenint obert el document començarem a veure per ordre tot el que pot contenir el programa.

1.3.2 La capçalera del programa

A la capçalera s'han de posar les instruccions de preprocessador, aquest actua abans que ho faci el compilador, és a dir, fa tots els canvis que li indiquem a la capçalera del programa abans de compilar-lo.

Sense el preprocessador un programa escrit no podria funcionar, per exemple:

```
#include<stdio.h>
#define DOS 2

main( )
{
printf("%d", DOS);
}
```

Sense preprocessador el compilador no podria compilar aquest codi perquè no pot funcionar sense l'arxiu stdio.h, ja que declara les funcions printf i scanf, ni sense el #define ja que això fa que DOS passi a ser 2, que és el que volem.

Hi ha diverses instruccions, la llargada de l'explicació de les quals serà proporcional a la importància que tenen:

- La instrucció #include, aquesta instrucció inclou arxius .h, és a dir, per tal de no haver d'escriure totes les declaracions de funcions s'inclouen arxius com stdio.h, math.h... que

contenen aquestes declaracions. Veient que els arxius .h són de text el que podem fer és un document que inclogui diversos .h si és que sempre fem servir els mateixos, per exemple.

```
programa.c:

#include “recopilatori.h”

[cos del programa]
```

```
recopilatori.h:

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

La diferència principal entre els documents .h prototip i els que dissenyem nosaltres és que per incloure els prototips incloem els símbols '<' i '>', i en el personal incloem cometes.

- Una altra instrucció bastant important i útil és la #define, aquesta defineix la paraula que posem (macro) seguida de la cadena de caràcters de després (caràcters de repost).

```
#define TAULA “això és una taula”
```

Ara cada cop que posem TAULA al codi el preprocessador ho substituirà per “això és una taula”. Aquesta instrucció la podríem posar a l'arxiu recopilatori.h i funcionaria totalment igual.

També podem definir amb arguments, és a dir:

```
#define QUADRAT(x)  x*x      //es defineix QUADRAT d'x com x*x
int x = 2;
z=QUADRAT(x)                 //z serà igual a 4 (2*2)
y=QUADRAT(4)                 //y serà igual a 16 (4*4)
```

- Altres expressions: no és que tinguin menys importància però són fàcils si coneixes les dues anteriors, són *#ifdef, #else, #if, #undef, #endif i #ifndef.*

Un parell d'exemples d'aquestes últimes expressions:

```
#ifdef TREBALL 5
[acció a fer si està definit treball]
#else
[acció si no està definit]
#endif         //acaba la bifurcació
```

```
#define TREBALL 9          //TREBALL = 9
#undef TREBALL             //TREBALL ja no està definit
```

L'expressió #if la veurem més tard en l'apartat de bifurcacions.

Cadascú es pot fer el seu arxiu de capçalera adaptat a les seves necessitats i segons els tipus de programes que vulgui desenvolupar.

1.3.3 Les variables. Què són? Quins tipus hi han?

Les variables són estructures de dades que poden emmagatzemar al llarg del programa diverses dades, és a dir, a la variable se li ha de designar un tamany en *bytes* i quan es declara la variable el que es fa és reservar-li un espai de memòria a la computadora. Al llenguatge C s'han de declarar totes les variables que s'hagin d'utilitzar, això en principi pot resultar un inconvenient, a continuació es donen algunes de les raons per les quals declarar les variables no és cap inconvenient:

- La primera raó és que en C si no es declaren les variables el programa no funcionarà.
- La segona és que si declarem les variables, com que s'ha de fer al principi del programa, ja pressuposa una certa premeditació i també un cert ordre.
- I la més important és que declarar variables ens pot evitar errors com la confusió d'una variable, vegem l'exemple:

```
No declarem la variable en un altre llenguatge:
vot = 100
després intentem operar amb ella i ens confonem:
electors = bot + votnul
```

En aquest cas es crearia una variable "bot" i electors prendria un valor equivocat que potser trigaríem mesos en descobrir si es tracta d'un codi llarg.

Si declarem la variable "vot" en llenguatge C a l'utilitzar "bot" el compilador ens donaria l'error de "variable no declarada", error que podríem solucionar en escassos segons.

Ara bé, com es declara una variable? Molt senzill només s'ha de posar el tipus de variable (char, int, float... ara els veurem), el nom d'aquesta i la podem iniciar amb un assignador (=) i el valor que li volem donar.

- Definició *auto o static: l*es variables es poden declarar (com ja fa el compilador automàticament) per mòduls, és a dir, que només tinguin valor a un cert mòdul.
- Definició extern: com diu el nom les variables es declaren abans del main() i per tant tenen validesa a tot el programa.

Les variables es classifiquen segons si són reals o senceres i segons el tamany que tenen. Les principals (per ordre de tamany) són:

```
char  < short < int < long < float < double
```

A aquestes variables se'ls hi pot posar *short, long* o *unsigned* per tal de poder ficar menys o més caràcters dins d'una o per tal de fer que no tinguin signe.

Els tipus de variables són les següents:

- *Char* es fa servir per a emmagatzemar variables de caràcter (a, c, casa etc.) i el seu mètode de declaració és:

```
...
char nom[10] = "joan";          //observem que el nom va entre cometes
printf("%c", nom);              /*es fa servir el conjunt %c per expressar la
...                                variable char*/
```

La sortida a la pantalla seria:

joan

La variable char és la que reserva menys memòria per això li hem d'indicar quants caràcters ha de reservar perquè no li falti espai, en aquest cas serien 10 ja que un nom té unes 10 lletres. Si la variable a introduir tingués només un caràcter es posaria entre apòstrofs, i si la variable contingués més d'una paraula s'expressaria amb "%s" (de l'anglès *string)*.

- Short fa la mateixa funció que int però amb menys espai reservat.
- Int, aquesta variable es fa servir per designar nombres enters i no molt grans, el mètode de declaració és l'estàndard:

```
...
int num = 3;                //es declara i s'inicia a 3.
printf("%d", num);          //el conjunt especial per escriure la variable és %d.
...
```

La sortida a la pantalla seria:

3

- Long fa la mateixa funció que int però long disposa de més memòria per acumular més caràcters.
- Float o variable de punt flotant es fa servir per designar números reals, que tenen una part entera i una de decimal, i bastant grans, aquestes variables es declaren i es poden iniciar igual que les variables de tipus int.

```
...
float num=1/3;
printf("%1.3f", num);          //l'1.3 expressa que s'ha de reservar un espai per la part sencera i tres per la
                               //part decimal, també es pot posar només %f.
...
```

La sortida a la pantalla seria:

0.333

- Double és com la variable float només que double reserva moltíssima memòria com per exemple per a números exponencials.

```
...
double num = 4e9;                    //quatre elevat a la nou.
printf("%lf", num);                  //expressem la variable amb %lf (long float).
...
```

La sortida a la pantalla seria:
262144

1.3.4 La funció main()

Tots els programes necessiten començar per la funció main per tal d'arrancar. El programa pot contenir moltes altres funcions però totes estaran dins de la funció main.

Main() o void main(void) (on void vol dir que no té arguments) ha d'anar seguida d'un "{" per tant el programa ha d'acabar amb un "}".

```
[capçalera]
[declaració de variables externes]
main()
[declaració de variables auto o static]
{
[cos]
}
```

De tot el que hi ha dins dels claudàtors se'n diu el cos del programa, encara que, com hem vist abans, les variables es poden declarar abans del main() per tal que funcionin durant tot el programa i no només durant el mòdul.

1.3.4.1 Main() amb arguments

A vegades es fan programes en els que cal alguna indicació abans d'iniciar-lo. Quan es vol executar un programa simplement s'escriu el nom de l'executable a la terminal, però, per exemple, si es vol executar un reproductor de música caldrà dir-li que iniciï una o una altra cançó, per exemple:

```
~$mplayer -c SoLonely.mp3
```

Teclejant això iniciarem el programa "mplayer" (que segons el manual si poses -c i la cançó la reprodueix) amb el qual es farà sonar la cançó SoLonely.mp3 (del grup The police).

Perquè un programa accepti arguments simplement posem aquestes instruccions dins el main():

```
main(int argc, char *argv[ ]){}
```

Argc és una variable que conté el número de caràcters que es teclegen a continuació del programa. El *argv és un vector apuntador, ja es veurà que és més endavant, caràcter que conté la direcció de la primera lletra o caràcter de les paraules que teclegem.

1.3.5 Pràctica de variables i capçalera. Programa de preguntes i respostes

Ara que s'han vist conceptes com les variables i la forma dels codis, és un bon moment per fer el primer:

preguntes.c

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

main()
{
	int anys;
	float sou;
	char nom[12];

	system("clear");
	printf("Com et dius?\n");  //S'escriu el que hi ha entre cometes, el \n serveix per canviar de línia.
	scanf("%s", &nom);                    //Es recull com string el nom que s'ha teclejat.

	printf("Quants anys tens?\n");       //Repetim les accions per seguir recollint caràcters.
	scanf("%d", &anys);
	printf("Quant cobres al mes?\n");
	scanf("%f", &sou);

	printf("Hola %s, tens %d anys i guanyes %5.2f al mes\n", nom, anys, sou);
}
```

A la capçalera incloem stdio.h: aquest arxiu de capçalera conté les funcions d'entrada i sortida, printf(), que imprimeix a la pantalla i scanf(), que recull un valor teclejat, entre d'altres.

Stdlib() és un arxiu que inclou la funció system() entre d'altres. El que posem dins de system() serà com escriure-ho al terminal, en aquest cas el fem servir per netejar la pantalla posant *clear* (en Linux, *cls* en Windows). En l'última línia del codi imprimim les variables posant el conjunt adequat dins el printf() (%s, %d...) i després el nom de la variable. Com es pot observar la variable *float* reserva cinc llocs per els nombres reals i dos per els decimals. La resta de codi està comentat al codi mateix.

1.3.6 Operadors, sentències i expressions

1.3.6.1 Operadors

En el món de la programació igual que en el de la matemàtica calen diversos operadors per tal de fer càlculs o executar operacions, igual que per fer una suma necessitem un '+'.

Els operadors es divideixen en:

- Aritmètics: la majoria d'aquests són iguals que els de les matemàtiques (+ - % / *) menys els operadors ++ i --, que expressen increment o reducció d'una unitat a una variable, per exemple:

```
...
int a = 1;
num = a++   //num=1 ja que num agafa el valor d'a i DESPRÉS a a se li suma 1
num = ++a   //num=2 ja que primer es suma 1 i després el valor d'a.
num = a--    //num=1 pel mateix motiu que el a++
num = --a   //num=0 primer es resta 1 i després es suma el valor d'a
...
```

- D'assignació: en la programació en C el símbol '=' no té el valor d'igualar sinó d'assignar, per tant "num = 1" dóna a num el valor 1. Disposem de diversos assignadors (+=, -=, /=, *=) que actuen tots igual de la següent manera:

```
i (+, -, /, *)= 1 actua com i = i (+, -, /, *) 1
```

Com que '=' és un assignador per fer una igualació fem servir '==' i per dir que no és igual fem servir !=. Existeixen uns altres assignadors de relació que ja coneixem de matemàtiques com el '<', '>', '<=' i '>='.

- Lògics: només en disposem de tres i són més propis de la lògica filosòfica clàssica. El '&&' que actua com un 'i': num && lletra, si num i lletra existeixen valor 1 si no valor 0. El || que fa de 'o': num || lletra, només cal que un dels dos existeixi perquè sigui 1. El ! és una negació: !num: sinó hi ha num serà 1 i si hi ha serà 0 el valor obtingut.

1.3.6.2 Expressions

Les expressions són simplement la combinació correcta i lògica dels operadors per tal d'aconseguir alguna cosa. Per exemple, si volem que funcioni la bifurcació *if* que veurem més tard necessitem que es compleixi una expressió:

```
((a<=3) || (!b && (a==c)))
```

En aquest cas si 'a' és menor o igual que 3 o no hi ha 'b' i 'a' és igual a 'c', tindríem una expressió vàlida (representada amb un 1).

1.3.6.3 Sentències

Les sentències no són res més que les agrupacions d'operadors en línies del codi font. Les sentències s'anomenen simples si ocupen una sola línia;

```
a = 2*2+5
```

i compostes o blocs (mòduls) si ocupen diverses línies del codi:

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

1.3.7 Triant el camí correcte, bifurcacions

Podem dirigir l'execució de sentències utilitzant unes altres sentències, disposem de *l'if*, *l'else*, el *swich*, el *break*, el *case* i el *default*.

1.3.7.1 La sentència if.

La sentència *if* controla cap a on s'ha de dirigir el programa d'una manera molt simple, si es compleix l'expressió a continuació de l'*if* entra en el bloc, si no es compleix segueix endavant:

```
...
if(expressió1){
      sentència_1;
}
...
```

1.3.7.2 Else i else if.

La sentència else, aquesta sentència no pot anar separada de l'*if* ja que actua com a complement, és a dir, si l'*if* no es compleix en comptes de seguir amb el programa salta a l'*else*. També podem fer servir *else if*(*expressió*) per si no es compleix l'*if* el programa passi a fer l'*else* amb una condició *if*, és a dir, l'*else* actua si l'*if* no es compleix, en canvi l'*else* if actua si l'*if* no es compleix i es compleix l'expressió de l'*else if*.

```
...
if(expressió1){                  /*si es compleix l'expressió1 es fa la sentència1, si no es fa la sentència2.
      sentència1;
}
else{
      sentència2;
}
...
```

1.3.7.3 Pràctica amb if, else i else if. Solucionant equacions de segon grau

Les equacions de segon grau no són difícils de resoldre però si volen un cert temps de dedicació, fent aquest programa senzill podem aconseguir que el càlcul de les equacions de segon grau sigui ràpid i automatitzat.

equacions.c

```
#include <stdio.h>
#include <math.h>                  //Ens trobem amb una llibreria nova math.h que conté la funció sqrt(),
#include <stdlib.h>                //que ens permet fer arrels quadrades.
main ()
{
        float a, b, c, x1, x2, disc, pr, pi;          //Variables float ja que els números poden ser decimals.
        system("clear");                              //Netegem la pantalla.
        printf("\t\nAx*x+Bx+C\n Introdueix els valors d' A, B y C\n");      //Sol·licitem els valors d'A, B i C.
        scanf("%f %f %f", &a, &b, &c);                //Recollim els valors d'A, B i C.
        disc=(b*b)-(4*a*c);                           //Calculem el discriminant (el que es troba dins l'arrel.

        if(disc==0)                   //Si el discriminant és igual a zero les dues solucions seran iguals,
        {                             //per tant:
                x1=-b/2*a;
                printf("\n Les dues solucions són iguals, x1 y x2 = %6.3f\n", x1);
        }                     //%6.3f: es reserven sis espais per la part sencera i tres per la decimal.
        else if(disc<0)       //Si el discriminant és menor que zero...
        {
                pr = -b/2*a;                  //...entren en joc els números complexos.
                pi = (sqrt(-disc))/2*a;       //Calculem la part imaginària (pi) i la real (pr) per separat.
                printf("\n Les solucions són números complexos, %6.3f +%6.3fi y %6.3f -%6.3fi\n", pr, pi, pr,
pi);                                  //Aquí donem les dues solucions possibles (part real + part imaginària.
        }

        else          //Si el discriminant no és zero ni menor serà major que zero per eliminació per tant,
        {           //fem només else i així no cal entrar una altra condició, es farà la possibilitat que quedi.
                x1=(-b+sqrt(disc))/2*a;           //Calculem les dues solucions,
                x2=(-b-sqrt(disc))/2*a;
                printf("\n Solucions:\n\tx1=%6.3f\n\tx2=%6.3f\n",x1, x2);          //i les imprimim a la pantalla.
        }                     /*El \t del printf és com posar una tabulació*/
}
```

1.3.7.4 Switch

Les sentències swich, case, break i default actuen com a conjunt. El *swich* activa la bifurcació amb una condició (*swich*(expressió)), si l'expressió és 'a' es passa al *case* 'a' i si no és cap de les escrites el default fa que es compleixi la seva sentència. El *break* serveix per finalitzar els *case*. Vegem un exemple:

```
...
switch (expressió) {
    case expressió1:
        sentència1;
    break;
    case expressió2:
        sentència2;
    break;
    ...

    default:
        break;
}
...
```

L'exemple de bifurcació del tipus switch la veurem al desenvolupament de la pràctica.

1.3.8 Bucles

El bucles permeten executar una sentència sempre que es compleixin unes condicions. És difícil explicar teòricament que és un bucle per tant posaré exemples dels tres tipus de bucles que hi ha.

1.3.8.1 El bucle while

El bucle *while* té la següent estructura:

```
...
while(expressió)
{                          //mentre l'expressió sigui 1 s'anirà fent la sentència.
sentència;
}
...
```

acabar.c

```
#include <stdio.h>
main()
{
        char acabar = 'n';
        while(acabar == 'n')
        {
                printf("El programa funciona, vols acabar?");
                acabar = getchar();
         }
}
```

El bucle while es fa servir sobretot per controlar l'encesa i l'apagada de programes. En aquest cas hem creat una variable del tipus char que s'inicia en 'n', aleshores s'inicia el programa. El programa fa una impressió a la pantalla i ens pregunta si volem acabar, aquí es quan trobem una funció nova molt útil per recollir un caràcter, getchar(), i li donem el valor del que s'ha recollit a acabar, per tant el programa només continuarà si pitgem 'n', ja que qualsevol altra tecla faria que el while no s'iniciés.

1.3.8.2 El bucle for

El bucle *for* agrupa tres accions: inicialització, expressió de control i actualització. Vegem la forma bàsica dels bucles for:

```
...
int i;
for(i= [valor inicial]; i (< o >) [valor limit]; (++ o --) i (++ o  --))
{
sentencia;
}
...
```

La sentència dins del bucle for es repeteix tantes vegades com s'indica, és a dir, començant pel valor d'inicialització fins arribar al de control, incrementant o reduint la variable en cada repetició segons l'actualització.

1.3.8.2.1 Pràctica amb el bucle for. Les taules de multiplicar

Aquí exposo un petit exemple de bucle for per calcular taules de multiplicar senceres, des de l'u fins al deu:

taules.c

```
/*Exemple d'un programa amb bucles for.
                                        Les taules de multiplicar.*/
#include<stdio.h>
#include<stdlib.h>

main()
{
      int i, num;                          //Introduïm dues variables del tipus int.
      system("clear");                     //Netegem la pantalla.
      printf("De quin número vols la taula de multiplicar?\n");          //Preguntem la taula que es vol.
      scanf("%d", &num);                   //Recollim el valor que s'ha premut.
      for(i=1; i<11 ; i++)                 //Fem un bucle que creï valors del 1 al 10
            {
            printf("%d x %d = %d", num, i, i*num);          //i repetim aquesta acció el número de
            printf("\n");                                     //que indica el bucle, això escriurà la taula a la
            }                                                 //pantalla.
}
```

El programa és bastant senzill, el que sortirà a la pantalla serà la pregunta sobre quina taula volem, teclejarem una xifra i el bucle *for* generarà els números de l'1 al 10 que s'aniran multiplicant per la xifra inicial que li hem donat. En aquest codi podem observar com es poden multiplicar dos variables durant el printf, ja que l'últim *%d* es substituirà per *i*num*. De manera que, en el cas de teclejar un sis, el programa farà una sortida a pantalla amb:

```
6x1=6
6x2=12
...(ja sabem com continua).
```

1.3.8.3 Do...while

Aquest bucle actua de manera similar al bucle while:

```
...
do
{
    sentència;
}while(expressió)
...
```

Com en el while la sentència o sentències dins del bucle s'executaran sempre i quan l'expressió que conté el while es compleixi.

1.3.8.3.1 Pràctica amb el bucle do...while. Esbrina el número

Ja que el do...while és molt semblant al while provem la seva eficàcia dins un codi complert que faci alguna cosa útil, en aquest cas un "joc" d'esbrinar el número:

dowhile.c

```
/*Exemple d'un programa amb bifurcacions del tipus if i else, i amb un bucle do...while:
                Esbrina el numero*/

#include <stdio.h>
#define NUM 7                 //i printf() i definim que el valor de NUM sigui set, el set és el numero a esbrinar
                              //que podem canviar si volem.
main()
{
      int y, intents = 3;             //Declarem la variable 'y' i "intents", aquesta última la iniciem a tres.
/*El programa només iniciar-se escriurà a la pantalla el que hi ha entre cometes, el /n com ja sabem serveix
per passar a la següent línia*/
      system("clear");            //Com ja sabem amb aquesta funció netegem la pantalla.
      printf("Esbrina el numero sencer de l'u al deu\n");
/*Comencem el bucle do...while*/
      do{

      intents--;                        //Cada vegada k comenci el bucle traurà un intent dels tres que tenim.
      scanf("%d", &y);                  //Recollim el número que s'ha premut
      if(NUM == y)                      //i si el número recollit és igual al que hem d'esbrinar s'imprimeix a la
                                        //pantalla el següent, com ja sabem %d és la referència d'impressió de la
            {                           //variable y.
            printf("Has esbrinat el número! si si... és el %d\n", y);
            intents = 0;
            }
      else                              //si no esbrinem el número a la pantalla s'imprimeix el següent:
            {
            printf("Mmmh aquest no és, torna a provar:\n");
            printf("Tens %d intents més\n", intents);       //això ens dirà el número d'intents que ens queden.
            }
      }while(intents > 0);              /*el joc durarà fins que s'acabin els intents, quan s'acabin s'imprimirà el
                                        següent missatge a la pantalla*/
      printf("GAME OVER\n");
}
```

El que obtindríem d'aquest codi seria una frase inicial que ens demana que premem un número de l'u al deu, en aquest moment el programa esperarà a rebre una tecla, en aquest cas un número i el processarà de manera que si concorda amb el número correcte sortira la frase d'èxit i automàticament es donarà el valor de zero a intents per tal que acabi el bucle, i si fallem tindrem un intent menys i sortirà la frase de torna a provar fins que esgotem els intents i el programa acabi ja que no es complirà el do...while.

1.3.9 Dades derivades

Les dades derivades reben aquest nom perquè són dades importants que provenen de les variables.

1.3.9.1 Apuntadors

Els valors de les variables es guarden en una certa direcció de la memòria de l'ordinador, el llenguatge C disposa de l'operador & per buscar aquesta direcció i les variables anomenades apuntadors destinats a contenir les direccions on estan guardats els valors de les variables.

Per declarar un apuntador fem:

*(int, long, char, float, double) *apuntador;*

Alguns exemples:

```
int i, j, *p;          //p es un apuntador a int
p = &i;                //p pren la direcció d'i
*p = 10;               //i pren el valor 10
p = &j;                //p pren ara la direcció de j
*p = -2;               //j pren el valor -2
```

Per imprimir un apuntador amb printf( ) fem servir %u i %p.

Es poden fer operacions amb apuntadors però només la suma, podem fer que l'apuntador passi a apuntar a la següent direcció si fem: *p = p+1*.

1.3.10 Matrius i vectors

Les matrius línies i vectors, també anomenats *arrays* tenen la propietat de poder emmagatzemar, com si fos un arxivador diverses variables. La manera d'expressar un array és:

int vector[nombre de llocs];

Aquests llocs es numeraran des del 0 fins al nombre total menys u, per tant podem anar emplenant l'array d'aquesta manera:

```
int vector[5];
vector[0] = 3;
vector[4] = 8;
vector[n] = ...;
...
```

Encara que s'han posat dos exemples amb int es pot utilitzar qualsevol tipus de variable (double, float...). A més, l'array es pot emplenar com s'ha fet a l'exemple o amb un bucle for com es mostra a continuació:

```
...
int i;
for(i=0; i<=5;i++)
{
      scanf("%d", &vector[ i ]);
}
...
```

1.3.11 Estructures

Les estructures permeten agrupar diversos tipus de dades sota un mateix nom. Per fer-ho hem de fer servir la paraula clau *struct* seguida d'una variable, després obrim '{' i escrivim totes les variables dintre de l'estructura, finalment tanquem amb un altre '}', un exemple senzill utilitzant un vector o array:

```
...
struct alumne[30] {
    char nom[20];
    char direcció[21];
    unsigned int telèfon;
}
...
```

Dintre del vector alumne hi ha trenta llocs disponibles, per accedir al telèfon de l'alumne 18 faríem:

```
alumne[18].telèfon
```

Utilitzant aquesta variable i les funcions printf() i scanf() podríem afegir i mirar les dades de l'estructura.

1.3.12 Funcions

Les funcions són parts del codi que resulten independents del programa principal i de les altres funcions. Les funcions s'han de declarar com les variables però utilitzant un tipus de variable que encara no s'ha comentat, el void. Quan declarem una funció amb: void funció(void), el que estem fent és no definir cap tipus de variable concret i no donar-li arguments. Una funció també necessita ser construïda, tenint dins seu les accions que ha de fer. Un cop declarada i construïda la funció només cal esmentar-la en una part de codi per que s'executi.

Les funcions ens serveixen per modular molt més el codi i facilitar la feina.

Dintre de les funcions distingim entre les ja construïdes (estàndard) i les que ens fem nosaltres mateixos.

- Les funcions que ja estan construïdes perquè són molt usades no cal declarar-les, ja que la seva declaració ve a l'arxiu que hem inclòs a la capçalera, com pot ser stdio.h, i tampoc cal construir-les, ja que ja estan construïdes en un arxiu .c que les conté i que el compilador busca quan li indiquem. Les funcions més típiques, les d'entrada i sortida, com ja hem vist abans printf(), scanf() i d'altres es troben en l'arxiu de capçalera stdio.h (Standart Input and Output Library), aquestes funcions són tan bàsiques que no cal donar-li cap anotació al compilador perquè enllaci amb l'arxiu .c. En canvi, altres funcions les hem de buscar en la taula de llibreries del llenguatge C per trobar en quin arxiu de capçalera es troben i després buscar què li hem d'indicar al compilador, això últim depèn del compilador que utilitzem. Per exemple si volem fer una arrel quadrada necessitem la funció sqrt(), aquesta funció es

troba en l'arxiu de capçalera math.h i perquè el compilador enllaci amb l'arxiu .c (en el cas d'utilitzar el compilador GCC) li hem d'indicar un -lm alhora de compilar:

```
~$gcc -lm arxiu.c -o executable
```

- Ara bé, moltes vegades trobarem que ens hem de construir les nostres pròpies funcions per modular un codi difícil de llegir. Primer buscarem un nom a la funció i la declararem com una variable tipus extern (abans del main()), després utilitzarem aquesta funció on vulguem. La construcció de la funció es pot fer al final del codi o en un altre arxiu .c, per construir-la fem gairebé el mateix que per declarar-la:

```
void funció()
{
accions que fa la funció;
}
```

2 Fem el programa

2.1 Què volem fer? Introducció a la practica

En aquest apartat s'exposaran mètodes per desenvolupar un codi, a més a més, d'explicar com compilar-lo i com arrancar l'arxiu executable.

Com que aquest treball és del llenguatge C he volgut fer un programa utilitzant les mínimes llibreries especials, de manera que el programa només necessita dues llibreries.

L'objectiu de la pràctica és crear un joc. El joc no tindrà gràfics ja que caldria utilitzar unes llibreries que s'aparten del llenguatge C pur. Aquestes llibreries (les SDL, Simple DirectMedia Layer) inclouen funcions que resumeixen tot el codi en poques línies, per exemple una funció pot ser SDL_play(), que fa que es reprodueixi una cançó. Això li treu molt de mèrit al treball, per tant, desenvoluparé un joc en ASCII (tipus d'art explicat a l'apartat següent) que s'executarà des del terminal.

2.2 Un joc en ASCII

ASCII és un tipus d'art que es crea a partir dels caràcters del teclat. Per fer un joc en ASCII necessitem les funcions de la llibreria *curses*, que inclou algunes com: clear() per netejar la terminal, move(y,x) anar a una posició escalar y,x, a més disposa d'un sistema temporitzador per automatitzar el programa.

2.3 El codi pas a pas

El primer que farem serà crear una carpeta amb el nom joc i dins un arxiu amb el nom main.c on escriurem tot el programa. Abans de començar hem d'instal·lar a la llibreria *curses* per poder anar provant si funciona o no el joc.

2.3.1 La capçalera

```
#include <curses.h>          //Incloem les llibreries necessàries: curses, stdio i stdlib ja sabem perquè
#include <stdio.h>           //es fan servir.
#include <stdlib.h>
#include <sys/time.h>        //sys/time.h el que fa és generar una senyal ALARM i sys/signal fa que quan
#include <sys/signal.h>      //es rebi aquesta senyal s'executi la funció handleTime();
#define TAMX 80              //El tamany de la pantalla en un joc ASCII sol ser de 80x25 per això definim
#define TAMY 25              //els tamanys com constants, així si volem canviar el tamany ho podem fer
```

```
#define FPS 25                  //des d'aquí. Igual que els FPS (frames per segon).
#define TRETS_NAU 25            //Definim 25 com el màxim de trets que la nostre nau pot produir a la pantalla.
#define TRETS 1
#define NCURSES_ASSUMED_COLORS "2.0"
```

Tota aquesta part és fàcil d'entendre, potser el que sobta més és l'última definició, encara que en realitat si mirem l'arxiu *curses.h* trobem que si definim això aconseguirem les lletres blanques sobre fons negre (amb un #ifdef), per tant, no té cap dificultat.

2.3.2 Declaracions tipus extern: funcions i variables

```
void llegir_tecles(void);                                   //Les funcions ja les explicaré una a una després.
void dibuixar(void);
void dibuixar_ajuda(void);
void actualitzar_estat(void);
void dibuixar_barra(void);
void handleTimer(int sig);
void sortir(void);
void dibuixarmort(void);

int naux = TAMX/40;                                         //La posició inicial de la nau, en aquest cas a (2, 12)
int nauy = TAMY/2;
int ajuda = 0;  //Variable per activar el menú ajuda, iniciada a 0 perquè no ens surti l'ajuda només començar

typedef struct bala{                                        //Amb typedef definim un tipus de variable "projectil"
        int x;                                             //Amb struct creem l'estructura bala (bala.x, bala.y, etc.)
        int y;
        int activa ;
}projectil;
projectil tret_nau[TRETS_NAU];                              //declarem la variable tret_nau del tipus projectil
typedef struct enemic{                                      //El mateix que amb les bales fem amb els enemics
        int x;
        int y;
        int actiu;
}alien;
alien   extraterrestre[EXTRATERRESTRES];                    //Fem el mateix que amb les bales

int neixalien;                                              //variable neixalien, per ser activada o desactivada
int vides = 3, nivell = 1;                                  //Iniciem les vides, punts i nivell.
long int punts = 0;
int disparat = 0;
unsigned long int frame = 0;
struct itimerval myTimer;                                   //Aquesta estructura controla el temps.
```

2.3.3 El main ()

```
int main(int argc, char *argv[])
{

        initscr();                                          //Iniciem la llibreria curses.
        cbreak();
        noecho();
        curs_set(0);                                        //Eliminem el cursor de la pantalla.

        int i;                                             //Declarem una variable static per utilitzar quan apretem 'f'.
```

```
	//inicialitzem el generador de números aleatoris
	srand(time(NULL));

	//inicialitzem el tret_nau, és a dir: emplenem el vector
	for(i =0; i<TRETS_NAU; i++)
	{
		tret_nau[i].activa = 0;
		tret_nau[i].x =0;
		tret_nau[i].y = 0;
	}

	//inicialitzem els aliens (també emplenem el vector)
	for(i=0; i < EXTRATERRESTRES;i++)
	{
		extraterrestre[i].actiu =0;
		extraterrestre[i].x = 0;
		extraterrestre[i].y = 0;
	}

//aquesta estructura és difícil d'entendre, simplement hem de saber que controla el pas del temps en el joc.
	myTimer.it_value.tv_sec = 0;
	myTimer.it_value.tv_usec=1000000 / FPS;
	myTimer.it_interval.tv_sec=0;
	myTimer.it_interval.tv_usec = 1000000 / FPS;
	setitimer(ITIMER_REAL, &myTimer, NULL);
	signal(SIGALRM, handleTimer);

	//Iniciem aquests colors per poder-los utilitzar més tard
	if(has_colors())
	{
		start_color();
		init_pair(COLOR_BLACK, COLOR_BLACK, COLOR_BLACK);
		init_pair(COLOR_GREEN, COLOR_GREEN, COLOR_GREEN);
		init_pair(COLOR_BLUE, COLOR_BLUE, COLOR_BLUE);
		init_pair(COLOR_YELLOW, COLOR_YELLOW, COLOR_YELLOW);
		init_pair(COLOR_WHITE, COLOR_WHITE, COLOR_WHITE);
		init_pair(COLOR_RED, COLOR_RED, COLOR_RED);
		init_pair(COLOR_MAGENTA, COLOR_MAGENTA, COLOR_MAGENTA);
		init_pair(COLOR_CYAN, COLOR_CYAN, COLOR_CYAN);
	}
	clear();                                                    //Netegem la pantalla
	dibuixar_barra();                                           //Dibuixem la barra d'estat
	dibuixar();                                                 //Dibuixem el aliens, els trets i la nostre nau
	for(;;){                                                    //Aquest for es un bucle infinit perquè vagi llegint tecles sempre
		llegir_tecles();
	}
}
```

2.3.4 Funcions casolanes

Per llegir les tecles:

```
void llegir_tecles(void)
{
	int i;                                                      //Declarem la variable tipus static
	switch(getch()){                                            //Encenem un switch que reculli una tecla
		case 'q':                                           //Amb la tecla 'q' anem a la funció sortir()
			sortir();
```

```
                break;
        case 'j':                                   //Amb la 'j' movem la nau cap a l'esquerra
                if(naux > 1){                       //fins al punt 1 (per no sortir-nos de la pantalla)
                        naux-=2;                    //avançant duess posicions cada toc
                }else{
                        naux = 1;                   //Si la nau està en la posició 1 es quedarà allà
                }
                break;
        case 'l':                                   //Amb la 'l' movem cap a la dreta
                if(naux < TAMX-4){                  //la instrucció és igual que la de la 'j'
                        naux+=2;
                }else{
                        naux = TAMX-4;
                }
                break;
        case 'k':                                   //movem cap a baix amb la 'k'
                if(nauy < TAMY-2){
                        nauy++;                     //amb un increment d'u en u
                }else{
                        nauy = TAMY-2;
                }
                break;
        case 'i':                                   //amb 'i' movem cap a dalt
                if(nauy > 2){
                        nauy--;
                }else{
                        nauy = 2;
                }
                break;
        case 'h':                   //'h' activa el menú ajuda en un toc i en un altre toc el desactiva:
                if(ajuda == 0){
                        ajuda = 1;
                }else if(ajuda == 1){
                        ajuda = 0;
                }
                break;

        /*Possiblement la instrucció més complicada fins al moment. Si premem 'f' s'iniciarà un
bucle for que adquirirà valors de l'u al límit de trets que podem fer (TRETS_NAU). El primer if controla que
la bala no estigui ja activa, el segon que no hàgim disparat ja. En el cas que es compleixin els dos és
donaran les posicions per on han d'anar les bales*/
        case 'f':
                for(i =0;i<TRETS_NAU; i++)
                {
                        if(tret_nau[i].activa == 0){
                                if(disparat == 0){
                                tret_nau[i].activa = 1;
                                tret_nau[i].x = naux+4 ;
                                tret_nau[i].y = nauy;
                                disparat = 1;
                                }

                        }
                }

                disparat = 0;

                break;
        default:
                break;
    }
```

```
}
```

Dibuixem la pantalla cada segon:

```
void handleTimer(int sig)                          //es repeteix cada segon (sys/timer, sys/signal)
{
        frame++;                                   //Anem incrementant el nombre de frames
        clear();                                   //Netegem la pantalla
        dibuixar_barra();                          //Dibuixem la barra

        //Si hem clicat 'h' ajuda serà 1 per tant sortirà l'ajuda.
        if(ajuda ==1)
                dibuixar_ajuda();
        //Si tenim vides seguim jugant, si no surt la pantalla de joc acabat(dibuixarmort())
        if(vides > 0){
        dibuixar();
        actualitzar_estat();
        }else{
        dibuixarmort();
        }
        refresh();                                 //Amb refresh() actualitzem la pantalla
}
```

Dibuixa tota la pantalla:

```
void dibuixar(void)
{
    int i;
        init_pair(2, COLOR_WHITE, COLOR_BLUE);     //iniciem un parell de colors el 2
        attron(COLOR_PAIR(2));                     //attron fa que les funcions següents tinguin colors
                                                   //en aquest cas el parell de colors 2 (blanc i blau)
                                                   //blanc la lletra i blau el fons
        //dibuixar nau
        move(nauy,naux);                           //anem a la posició que tingui la nau
        addstr("}||>");                            //i la dibuixem

        //Dibuixem els trets
        init_pair(3, COLOR_YELLOW, COLOR_BLACK);   //les bales seran grogues
        attron(COLOR_PAIR(3));
        for(i=0;i < TRETS_NAU;i++)                 //recorrem el vector fins TRETS_NAU
        {
                if(tret_nau[i].activa == 1)        //i si el tret de la nau esta actiu
                {
                        move(tret_nau[i].y,tret_nau[i].x);   //anem a les posicions
                        addstr("-");                         //i dibuixem les bales
                }
        }


        //Dibuixem els aliens
        init_pair(4, COLOR_RED, COLOR_BLACK);      //aliens vermells
        attron(COLOR_PAIR(4));
        for(i=0;i < EXTRATERRESTRES; i++)          //fem el mateix que amb les bales
        {
                if(extraterrestre[i].actiu == 1)
                {
                        move(extraterrestre[i].y,extraterrestre[i].x);
                        addstr("d\'_\'b");
                }
        }

}
```

Dibuixa la finestra d'ajuda:

```
void dibuixar_ajuda(void)
{
/*Escollim els colors groc i blau (de fons) i dissenyem una barra d'ajuda, anant a cada posició y, x.*/

        init_pair(6, COLOR_YELLOW, COLOR_BLUE);
        attron(COLOR_PAIR(6));

        move(6,26);
        addstr("+-----------------------------+");
        move(7,26);
        addstr("|            HELP             |");
        move(8,26);
        addstr("|                             |");
        move(9,26);
        addstr("|  KEY 'J'             BACK  |");
        move(10,26);
        addstr("|  KEY 'L'          FORWARD  |");
        move(11,26);
        addstr("|  KEY 'I'             UP    |");
        move(12,26);
        addstr("|  KEY 'K'             DOWN  |");
        move(13,26);
        addstr("|  KEY 'F'             FIRE  |");
        move(14,26);
        addstr("|  KEY 'Q'             EXIT  |");
        move(15,26);
        addstr("|  KEY 'P'            PAUSE  |");
        move(16,26);
        addstr("|                             |");
        move(17,26);
        addstr("| }||>          LA TEVA NAU  |");
        move(18,26);
        addstr("| d'_'b      ALIENS ORELLUTS  |");
        move(19,26);
        addstr("+-----------------------------+");
}
```

Actualitza el moviment de les naus, les bales i els extraterrestres:

```
void actualitzar_estat()
{
        int i,j;
        //calculem el moviment dels trets de la nau
        for(i =0; i<TRETS_NAU; i++)
        {

                if(tret_nau[i].activa == 1)
                {
                        if(tret_nau[i].x > TAMX - 3)                    //Si la bala arriba al final
                        {                                               //de la pantalla l'eliminem
                                tret_nau[i].activa = 0;
                                tret_nau[i].x = 0;
                                tret_nau[i].y = 0;
                        }
                        tret_nau[i].x += 2;                     //les bales es mouen dos espais seguits
                }
        }

        //movem extraterrestres igual que les bales
```

```
      for(i=0;i<EXTRATERRESTRES;i++)
         {
                  if(extraterrestre[i].actiu == 1)
                  {
/*això fa que l'alien segueixi a la nau, ja que si la nau està més endalt que l'extraterrestre, aquest pujarà i si
està més avall baixarà*/
                           if(nauy > extraterrestre[i].y){
                                    extraterrestre[i].y+= rand()%3;   //rand() crea una posició aleatoria (random)
                           }
                           if(nauy < extraterrestre[i].y){
                                    extraterrestre[i].y-= rand()%3;//% fa la resta de la divisió
                           }

                           extraterrestre[i].x--;                                    //la posició x de l'extraterrestre anirà
                                                                                      //disminuint (anirà cap a l'esquerra
                  }
         }
         //si surt per l'esquerra torna a sortir per la dreta
         for(i=0;i<EXTRATERRESTRES;i++)
         {
                  if(extraterrestre[i].x < 0)
                           extraterrestre[i].x = TAMX - 6;
         }

         //Creem l'extraterrestre
          i= (frame/FPS) /3;                                                       //cada interval de temps
          if(extraterrestre[i].actiu ==0)
      {
                  extraterrestre[i].actiu = 1;
                                                                                   //creem l'extraterrestre a un lloc imprevist
            extraterrestre[i].y = (rand() % (TAMY-1))+1 ;
                  extraterrestre[i].x = TAMX-6;
         }

         //matem extraterrestre
         for(i = 0; i<EXTRATERRESTRES;i++)
         {
                  for(j=0;j<TRETS_NAU;j++)
                  {
/*Amb el següent if comparem la posició de l'extraterrestre i de la nau*/
                           if((extraterrestre[i].x > tret_nau[j].x) &&
                           (extraterrestre[i].y == tret_nau[j].y) &&
                           (extraterrestre[i].x < tret_nau[j].x + 6)){
                                    //tret toca a alien
                                    //eliminar alien
                                    extraterrestre[i].actiu = 0;
                                    extraterrestre[i].x =0;
                                    extraterrestre[i].y =0;
                                    //eliminem la bala
                                    tret_nau[j].activa = 0;
                                    tret_nau[j].x = 0;
                                    tret_nau[j].y = 0;
                                    //sumem punts
                                    punts++;
                           }
                  }
                  //nosaltres també podem morir si la posició de l'alien és igual a la nostre
                  if((extraterrestre[i].x > naux) &&
                  (extraterrestre[i].y == nauy) &&
                  (extraterrestre[i].x < naux + 6)){
                           //morim!
```

```
                        vides -=1;
                        // y matem a l'alien en qüestió
                        extraterrestre[i].actiu = 0;
                        extraterrestre[i].x = 0;
                        extraterrestre[i].y = 0;
                }
        }

}
```

Dibuixa la barra d'estat:

```
void dibuixar_barra(void)
{
/*Iniciem els colors verd i negre i dibuixem la barra d'estat*/
        init_pair(5, COLOR_GREEN, COLOR_BLACK);
        attron(COLOR_PAIR(5));
        printw("SPACE BATTLE 1.0          (H)elp          | LIVES: %d | LEVEL: %d | SCORE: %d\n", vides,
nivell, punts);
        addstr("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
}
```

Funció que tanca el joc:

```
void sortir(void)
{
        system("reset");                                        //fem un reset i un clear a la terminal
        system("clear");
        exit(EXIT_SUCCESS);                                     //tanquem les llibreries curses
}
```

Quan s'acaben les vides:

```
void dibuixarmort(void)
{
        move(3,TAMX/4);                                 //Ens movem a aquestes posicions i escrivim el
        printw("No et queden vides, torna-ho a intentar!");                     //missatge final.
        move(5,TAMX/4);
        printw("Joc programat per Jairo Vadillo");
}
```

2.3.5 Compilar el codi amb gcc

Per compilar el codi he triat el compilador gcc, que és gratuït i funciona ràpid. Per compilar el codi ens hem de dirigir a la carpeta on esta guardat des del terminal i teclejar:

```
~$gcc main.c -o joc -lncurses
```

Com podem veure la instrucció és semblant a la de la pàgina 10 d'aquest treball, l'únic que incloem el -lncurses que enllaça amb la llibreria curses.

Ara a la carpeta ens haurà aparegut un nou arxiu anomenat joc, aquest és l'executable.

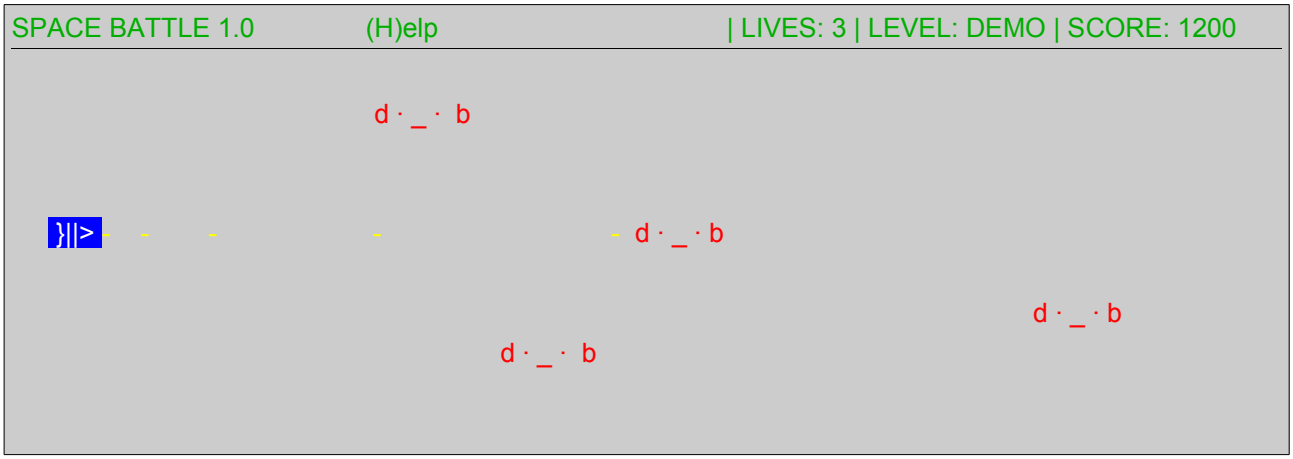
2.3.6 Provem el joc

Ara que ja tenim l'arxiu compilat només ens queda provar el joc, per fer-ho executem l'arxiu que ens ha sortit de la compilació l'anomenat "joc", això ho fem amb:

```
~$./joc
```

La pantalla quedarà totalment ennegrida i aleshores començarà el joc, no oblidem ajustar la terminal a les mesures que hem posat al joc (recordem TAMX = 80 i TAMY = 25).

I ara ja podem començar a jugar!



Conclusions

Crec que he aconseguit tots els objectius que em proposava al principi del treball. He aconseguit explicar el funcionament de totes les parts del llenguatge C, algunes de les funcions més importants, tot això en un manual no massa extens. També s'han pogut veure exemples de diferents llenguatges de programació i perquè el llenguatge C és diferent i millor que els altres.

S'ha demostrat que C és molt portable, ja que tots els programes desenvolupats durant el treball funcionen en qualsevol sistema operatiu.

El llenguatge C és un llenguatge difícil d'aprendre, ja em vist que els primers codis no són fàcils de llegir, però si s'adquireix un bon nivell de llenguatge C es poden fer tot tipus de programes i sistemes operatius amb una gran funcionalitat.

També he pogut aconseguir, encara que amb moltes dificultats, ja que sóc un iniciant, l'objectiu principal que era fer un joc, encara que no un joc complert sinó una demo del joc que seguiré fent. El motiu de que no hagi fet un joc complert no ha sigut ni per temps ni per dificultat sinó perquè amb aquesta demo ja es posa un exemple de llenguatge C i crec que fent el joc més llarg només s'aconseguiria repetir les coses que ja s'han fet, ja que més funcions no tindria.

Agraïments

Vull agrair el treball a l'Ernesto per haver-me deixat el llibre de “Programación en C”, a la meva tutora de TDR la professora Imma Compte per haver-me ajudat durant tot el treball, a en Noel Carriqui per haver-me iniciat en la programació en C, haver-me ensenyat a programar i per ajudar-me amb el joc. També a la gent que m'ha ajudat dels canals de l'IRC Hispano com #badchecksum (www.badchecksum.net), #begur, #c, #ubuntu, #aprende_ubuntu per haver aguantat les meves constants preguntes.

I en general a tots els meus companys i familiars que m'han animat a seguir amb el projecte.

Bibliografia

- Pàgines web consultades:

Com executar un perl:

http://eui.upm.es/CC/Chuletas/Perl/ejecutar.html

Programació:

http://jegsworks.com/lessons-sp/lesson9/lesson9-1.htm

La "hello world coleccion":

http://www.roesler-ac.de/wolfram/hello.htm#ZIM

La llibreria conio.h per Linux:

http://portal53.cl/portal53/ForoInformatica/viewtopic.php?t=211

Art ASCII:

http://es.wikipedia.org/wiki/Arte_ASCII

Joc inspirat en:

http://artax.karlin.mff.cuni.cz/~brain/0verkill/

Llenguatge C:

http://es.wikipedia.org/wiki/Programacion

http://es.wikipedia.org/wiki/Lenguaje_de_programación

http://es.wikipedia.org/wiki/Lenguaje_de_programación_C

http://www.tecnun.es/Asignaturas/Informat2/C/Prog_c.htm

Tutorial de la llibreria *curses*:

http://web.cs.mun.ca/~rod/ncurses/ncurses.html

- Llibres i pdf's:

– "Programación en C. Introducción y conceptos avanzados". Autors: M. Waite, S. Prata, D. Martin, 1987.

– "Aprenda lenguaje ANSI C como si estubiera en primero". Autors varis, Escola Superior d'ingeniers industrials de Navarra, 1998.

Annex

Algunes definicions d'interès:

- **Llibreria**: una llibreria és el conjunt d'un arxiu de capçalera .h i un arxiu .c ja compilat, com és d'entendre l'arxiu .h porta les declaracions de variables i funcions i el .c porta les funcions desenvolupades. Les llibreries poden ser externes o estàndard, la diferència entre aquestes dues és que les externes inclouen funcions que s'allunyen del llenguatge ANSI C o ISO C, en canvi les estàndard inclouen funcions senzilles a partir de les quals es poden fer les funcions de les llibreries externes.

- **"Hola món"**: Tots els exemples de codi font que surten a la part de llenguatges de programació són del tipus "hola món", aquests programes l'únic que fan és escriure a la pantalla "Hola món!". El primer "hola món" va aparèixer per primera vegada al llibre "The C programing language" de Kerninghan i Ritchie.

- ***Kernel* o nucli del sistema operatiu**: el Kernel o nucli del sistema operatiu es la part del sistema operatiu que està destinada a enllaçar el sistema operatiu i les aplicacions amb el hardware i el processador de l'ordinador.

Com que els processadors de textos com el Word o el de l'OpenOffice no permeten el colorejat de sintaxis en els codis aquí incloc una còpia a color de tot el codi del joc.