Fonaments dels Sistemes Operatius

Departament de Informàtica de Sistemas i Computadors (DISCA)
Universitat Politècnica de València

fSO

Pràctica 2
Programació en C (II)

Contingut:

1 Objectius

L'objectiu general d'aquesta pràctica consisteix en conèixer nous aspectes que formen part del llenguatge C i aplicar-los de manera pràctica, usant les ferramentes de programació UNIX en el context d'aquest llenguatge.
En concret els objectius són:

- Aprofundir en l'estructura de funcions d'un programa C i veure com això afecta al maneig de variables en els seus diversos àmbits.
- Utilitzar altres característiques d'aquest llenguatge com el treball amb la línia d'arguments, cadenes, punters o estructures.

Per a això es van a realitzar i/o modificar una sèrie de programes en C on l'estudiant haurà de comprovar els resultats obtinguts.

1.1 Entorn necessari

L'entorn per a realitzar les pràctiques consisteix en un sistema Unix (Linux) i el compilador gcc (GNU Compiler) que s'ha descrit en la sessió anterior. Com a entorn d'edició es pot utilitzar KEdit, KWrite o qualsevol altre editor com kate que s'ha proposat en aquesta sessió.
Aquesta pràctica també es pot realitzar en Mac OS X, utilitzant gcc i alguns dels editors disponibles (o millor el XCode)[1].
Els fitxers necessaris per a realitzar la pràctica s'han de descarregar de Poliformat.

2 Funcions en un programa C

En la sessió anterior es va introduir la creació de programes C on s'observa la presència de la crida a la funció principal denominada `main`. A banda d'aquesta funció `main`, en C es distingeixen dos tipus de funcions, les que venen ja definides en forma de biblioteques (per exemple, `printf` que s'utilitza en el següent exemple i que forma part de la biblioteca d'entrada/eixida `stdio.h`) i les que defineix el propi programador. En el següent apartat anem a provar a compilar un programa en C que servirà per a il·lustrar l'ús de funcions definides pel programador. El programa a compilar "`circulo.c`" es molt simple ja que conté d'inici una única funció (`main`) i es mostra a continuació:

```
#include <stdio.h>

#define PI 3.1416
main() {
   float area, radio;
   radio = 10;
   area = PI * (radio * radio);
   printf("Area del circulo de radio %f es %f\n", radio, area);
}
```

Figura-1: Codi del fitxer "*circulo.c*"

Per a compilar aquest programa i generar directament l'executable, es tracta d'introduir la següent ordre tal com s'ha utilitzat es la sessió anterior:

[1] NOTA: tot i que existeix una versió del gcc per a Windows, el seu funcionament difereix a allò descrit en aquesta memòria i per lo tant no es recomana el seu ús.

```
$   gcc -o circulo circulo.c
```

Comprovar que s'ha generat un fitxer executable (amb `ls -la`). Per a poder provar el programa simplement escrivim l'ordre (nota: és necessari ficar ./ al nom del fitxer per a que trobe l'executable en el directorr actual i no lo cerque al PATH).

```
$   ./circulo
```

2.1 Exercici 1: definir i declarar funcions

En aquesta secció es mostra com utilitzar més d'una funció dins d'un programa C. Per a això anem a copiar el contingut de `circulo.c` en un nou arxiu `circulo2.c` que inclourà la definició de la funció `areaC` al final del seu codi per a separar el càlcul de la seua àrea. Aquesta funció utilitzarà un argument de tipus float (`float radio`) i serà invocada des de la funció `main`, substituint la línia "`area = PI * (radio * radio);`" per la crida "`area = areaC(radio);`".

Realitzarem de nou la compilació i comprovarem els missatges que apareixen:

```
$   gcc circulo.c –o circulo
```

2.2 Qüestions

a) Reviseu els missatges i indiqueu a quin tipus corresponen (error o avís).
b) ¿Quin pot ser la interpretació d'aquests missatges?

Amb la finalitat d'evitar els anteriors problemes de compilació anem a analitzar dues alternatives. La primera consisteix en canviar l'ordre en l'aparició de les funcions, de manera que el codi de `areaC` precedisca al de la funció `main` tal com es mostra en la figura-2. Si compilem `circulo2.c` hauríem d'observar que s'ha generat l'executable de forma correcta. La segona alternativa consisteix en declarar la funció `areaC` de manera prèvia a la seua implementació, per exemple, mitjançant la línia "`float areaC(float radio);` ". Cal assenyalar que en la declaració d'una funció només indiquem el tipus de valor que retornen junt amb el del seus arguments. En general, a l'hora de codificar funcions en un programa C es tracta de definir aquestes (o almenys declarar-les) de forma prèvia a la seua invocació.

```
#include <stdio.h>

#define PI 3.1416
float areaC (float radio) {
   return (PI * (radio * radio));
}
main() {
   float area, radio;
   radio = 10;
   area = areaC(radio);
   printf("Area del circulo de radio %f es %f\n", radio, area);
}
```

Figura-2: Codi del fitxer "*circulo2.c*"

2.3 Àmbit de les variables

Un dels aspectes que podem observar en el codi de la figura-2 (`circulo2.c`) es l'aparició de la variable `radio` en dos llocs distints (funcions `areaC` i `main`, respectivament). això es possible degut a l'àmbit de les variables en C que pot correspondre a les següents categories:

- Global: les variables es declaren fora de qualsevol funció i se poden accedir des de qualsevol funció del fitxer.
- Local: es defineixen dins d'una funció i només són accessibles dins d'aquesta funció.
- Estàtiques: són variables locals que conserven el seu valor.

Una de les regles més importants és la que afecta a les variables locals i la seua aplicació és necessària quan apareixen variables globals i locals amb el mateix nom tal com es mostra en el codi `variables.c` de la figura-3. En aquesta situació, l'assignació de les variables locals és la que té prioritat sobre l'assignació de les globals del mateix nom. Per a això es suggereix compilar `variables.c i` analitzar el seu resultat ¿Quin penses que serà el valor de la variable x mostrat per pantalla?

```
#include <stdio.h>
int x=2;
void main(){
    int x=3;
    m();
    printf("%d", x);
}

void m(){
    x = 4;
}
```

Figura-3: Codi del fitxer "*variables.c*"

2.4 Exercici 2: treball amb variables

En el codi font del programa `variablesVarias.c`, mostrat en la figura-3 es tracta de revisar algunes de les situacions que poden sorgir amb l'ús de les variables en els seus diversos àmbits.

```
#include <stdio.h>
int a = 0; /* variable global */

// Esta funció incrementa el valor de la variable global a en 1
void inc_a(void) {
    int a;
    a++;
}

// Esta funció devuelve el valor anterior
// i guarda el nou valor v
int valor_anterior(int v) {
      int temp;
      // declarar aquí la variable s estática.

      temp = s;
      s = v;
      return b;
}
```

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

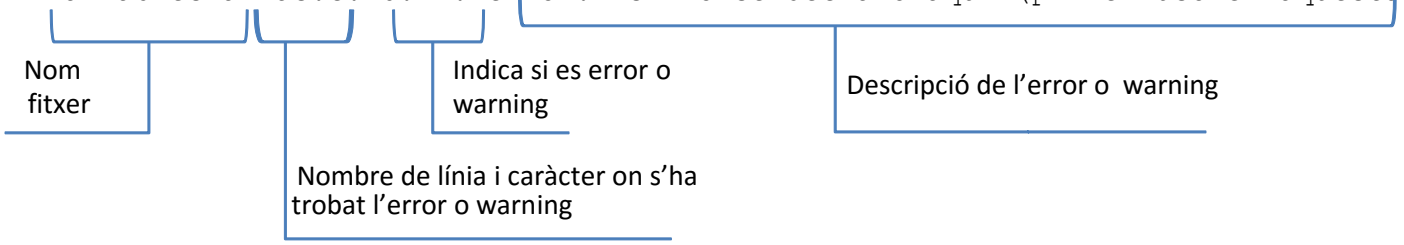
Figura-4: Codi a corregir proporcionat en el fitxer "*variablesVarias.c*"

Compileu `variablesVarias.c`

```
$ gcc -o variablesVarias variablesVarias.c
```

En pantalla apareixeran una sèrie d'errors. Els errors solen tenir el següent format i impedeixen la generació del codi executable tal com es va indicar en la sessió anterior:

```
variablesVarias.c:18:14: error: 's' no se declaró aquí (primer uso en aquesta
```



El programa conté errors de programació que cal que corregiu:

- la funció `inc_a` ha de treballar amb la variable global `a`, per tant no deuria estar definida com local.
- En la funció `valor_anterior` cal que declarar la variable `s` com se indica i per a que retorne el valor anterior el `return` ha de retornar el valor de `temp` (no el de b).

A l'executar `variablesVarias` ha de mostrar-se en consola:

```
a= 1, b= 2
a= 3, b= 2
```

2.5 Qüestions

a) Justifiqueu el canvi produït en el valor de la variable `a` i perquè s'incrementa fins que té un valor 3.
b) Aixímateix, raoneu perquè es manté el valor de la variable b.

3 Paràmetres per línia d'ordres

A l'invocar una ordre en UNIX es normal passar-li paràmetres. Com hem vist, el GCC disposa de multitud de opcions i paràmetres que es poden configurar per línia de comandos.

En un programa en C, podem tractar aquests paràmetres de forma molt simple amb `argc` i `argv`. Per a això cal definir la funció `main` amb aquests dos arguments `int main(int argc, char *argv[])` on:

a) `argc` contindrà el nombre d'arguments passats, que sempre serà major que zero, ja que el nom de l'ordre és el primer argument.
b) `argv` és un vector de cadenes amb els arguments. El primer element d'aquest vector (`argv[0]`) serà sempre el nom de l'ordre.

A partir del pas d'arguments per línies d'ordres es poden plantejar diversos exercicis. En aquest cas es tracta d'elaborar dos programes, partint de la base del fitxer `arguments.c`.

```
#include <stdio.h>

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


     // A completar…


}
```

Figura-5: Contingut inicial del fitxer "*arguments.c*"

3.1 Exercici 3: programa arguments

En aquest exercici es pretén implementar un programa denominat `listaArgs`, que mostre per pantalla el nombre d'arguments i el contingut de cadascun d'ells. Per a això, es deurà utilitzar un bucle que a partir del valor `argc` aplique la funció `printf` per a mostrar cada argument proporcionat. A continuació, es mostra quin ha de ser el resultat de la seua execució amb distints arguments:

```
$ ./listaArgs
Numero de arguments = 1
  Argumento 0 es ./listaArgs
$  ./listaArgs uno dos tres
Numero de arguments = 4
  Argumento 0 es ./listaArgs
  Argumento 1 es uno
  Argumento 2 es dos
  Argumento 3 es tres
```

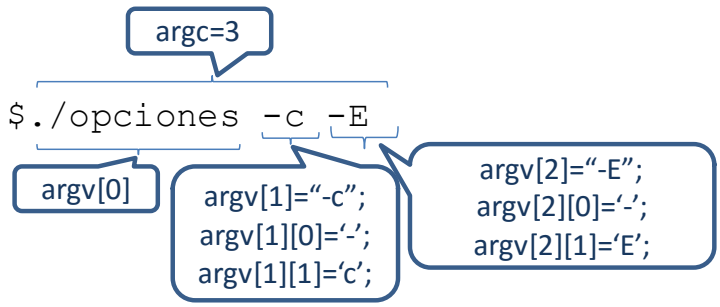
3.2 Exercici 4: programa opcions

Ara es tracta de realitzar un programa denominat `opciones`, que de forma pareguda al gcc identifique les següents opcions i si es el cas mostre la ruta de la opció

```
-c   Mostrará "Compilar"
-E   Mostrará "Preprocesar"
-iruta Mostrará "Incluir + ruta"
```

A continuació es mostra quin ha de ser el resultat de la seua execució amb distints arguments:

```
$ ./opciones -c
   Argumento 1 es Compilar
$  ./opciones -c -E -i/includes
   Argumento 1 es Compilar
   Argumento 2 es Preprocesar
   Argumento 3 es Incluir /includes
```



4 Punters i estructures

Un punter és una variable que conté l'adreça d'altre objecte. Això ens permet accedir i modificar elements de cadenes i estructures de forma simple. En aquesta part de la pràctica anem a completar menuts programes que treballen amb punters, cadenes i estructures.

4.1 Exercici 5: programa majúscules

Completar el programa `mayusculas.c` (figura 6), que ha de convertir un text llegit per consola, convertir-lo en majúscules i després treure-lo per pantalla. En concreto cal completar:

a) Definir les variables `cadena` i `cadena2` com vectors de cadena amb una grandària `TAM_CADENA`.
b) Llegir un text de consola i assignar-ho a la variable `cadena`. Per a llegir tires de caràcters que contiguen blancs es pot utilitzar `scanf("%[^\n]s", str)`.
c) Completar el bucle de conversió a majúscules. Per a això es farà ús de dos punters a cadenes, on `p1` apunta a `cadena` i `p2` apunta a `cadena2`, i per tant s'haurà d'anar copiant l'element apuntat per `p1` a `p2` (restant 32 per a convertir-ho a majúscula, només si es un caràcter en minúscula). Al finalitzar el bucle cal ficar el caràcter zero de final de cadena en la `cadena2`.
d) Treure per consola la `cadena2`, que tindrà el text convertit a majúscules.

```
#include <stdio.h>
#define TAM_CADENA 200
main() {
    // Puntero a caracter per a copiar les cadenas
    char *p1, *p2;

    // A) Definir les variables cadena i cadena2
    // B) Leer cadena de consola
    // C) Convertir a majúscules
    p1 = cadena;
    p2 = cadena2;
    while (*p1 != '\0') {
        //  Copiar p1 a p2 restando 32
    }

    // Acordarse de poner el cero final en cadena2
    // D) Sacar per consola la cadena2.
}
```

Figura-6: Contenido inicial del fitxer "mayusculas.c"

4.2 Exercici 6: programa sumafilas

Completar el programa `sumafilas.c` (figura 7), que suma una sèrie de files i nos retorna la suma de cada fila i la suma total. Cada fila és una estructura que conté dos membres, un vector amb les dades i la suma.

El que cal completar és el següent:

a) Definir una variable `filas` que siga un vector de estructures `FILA` de grandària `NUM_FILAS.`
b) Implementar la funció `suma_fila`. A aquesta funció se li passa un punter a la fila a sumar. Haurà de sumar el vector `datos` i assignar-la al membre `suma`.
c) Completar el bucle per a sumar totes les files. Es necessari cridar a `suma_fila` passant-li la fila, completar el `printf` i actualitzar la variable `suma_total`.

```
#include <stdio.h>

#define TAM_FILA 100
#define NUM_FILAS 10
struct FILA {
    float datos[TAM_FILA];
    float suma;
};
// A) Define una variable filas que sea un vector de estructures FILA de
grandària NUM_FILAS

void suma_fila(struct FILA *pf) {
// B) Implementar suma_fila
}

// Inicia les filas amb el valor i*j
void inicia_filas() {
    int i, j;
    for (i = 0; i < NUM_FILAS; i++) {
        for (j = 0; j < TAM_FILA; j++) {
            filas[i].datos[j] = (float)i*j;
        }
    }
}
main() {
    int i;
    float suma_total;

    inicia_filas();
    // C) Completar bucle
    suma_total = 0;
    for (i = 0; i < NUM_FILAS; i++) {
        // Llamar a suma_fila
        printf("La suma de la fila %u es %f\n", i, /* COMPLETAR */);
        // sumar la fila a suma_total
    }
    printf("La suma final es %f\n", suma_total);
}
```

Figura-7: Contenido inicial del fitxer “sumafilas.c”

El resultat final de l’execució d’aquest programa ha d ser:

```
$  ./sumafilas
La suma de la fila 0 es 0.000000
La suma de la fila 1 es 4950.000000
La suma de la fila 2 es 9900.000000
La suma de la fila 3 es 14850.000000
La suma de la fila 4 es 19800.000000
La suma de la fila 5 es 24750.000000
La suma de la fila 6 es 29700.000000
La suma de la fila 7 es 34650.000000
```

```
La suma de la fila 8 es 39600.000000
La suma de la fila 9 es 44550.000000
La suma final es 222750.000000
```

5 Pas de paràmetres per referència

Per a acabar aquesta sessió, anem a introduir el pas de paràmetres per referència en una funció, que permetrà retornar un o diversos valors a partir de la crida a aquesta funció. Exemples de funcions com `areaC` en `circulos2.c` (figura 2) o `valor_anterior` en `variables-conflicto.c` (figura 4) utilizaven un mecanisme de pas de paràmetres per valor (o per còpia) on en el caso de `areaC` es proporcionava un valor de tipus `float` com paràmetre de entrada i un valor sencer (`int`) per a la funció `valor_anterior`. En el pas de paràmetres per referència, allò que es passa a la funció no és un valor de variable en sí mateix, sinó un punter a la mateixa. Per exemple, la funció `suma_fila` en el codi `sumafilas.c` (figura 7) té com argument un punter a una estructura (`struct FILA *pf`). La principal aportació d'aquest mecanisme consisteix en la possibilitat de modificar el paràmetre passat per referència de forma que puga servir tant d'entrada com d'eixida. En el següent exemple s'utilitza aquest tipus de mecanisme per a actualitzar el valor de la variable `c.` Proveu a compilar-ho i comprovareu el seu efecte.

```
#include <stdio.h>

char F(char *c){
        c[0] = 'f';
        return (*c);
}
main () {
        char c;
        c = 'a';
        printf("%c\n",c);
        printf("%c\n", F(&c));
        printf("%c\n", c);
}
```

Figura-8: Codi del fitxer "*parametroSalida.c*"

5.1 Exercici 7: programa sumafilas2

En el següent exercici, es tracta de realitzar una versió del programa "`sumafilas.c`" que es va plantejar com exercici en l'apartat 4.2 de manera que aquesta nova versió "`sumafilas2.c`" tinga les següents característiques:

a) La variable `filas` passa a ser local i definida en la funció `main`.
b) Implementar la funció `inicia_fila`. A aquesta funció se li passa un punter de la fila a inicialitzar i es cridarà des del `main` abans d'invocar la funció `suma_fila`.

El resultat a obtindre ha de coincidir amb el de l'original "`sumafilas.c`".