Fundamentos de los Sistemas Operativos

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

fso

Práctica 1
Programación en C (I)

Contenido

1 Objetivos

Editar programas en lenguaje C y compilarlos en LINUX con *gcc (GNU Compiler Collection*).

- Trabajar con programas en C contenidos en un único archivo
- Aprender a compilar y ejecutar programas sencillos
- Detectar errores básicos de compilación
- Aprender a comunicarse con el programa leyendo y escribiendo en el terminal
- Adquirir experiencia con " if …else if", "switch" y "while"

¡ADVERTENCIA¡: En esta práctica se asume que el alumno ya sabe programar en otro lenguaje.

2 Herramientas necesarias

Para crear programas en C es necesario un editor *(xemacs, kate, vi*, etc.) y un compilador de C.
Esta práctica se realiza en sistema *Unix (Linux)* con compilador *gcc.* Para editar archivos se recomienda utilizar el editor *kate*. En el Anexo1 puede consultar aspectos del gcc y de las funciones printf() y scanf().

2.1 Etapas para crear un programa en C

¡IMPORTANTE!

Tenga en cuenta las siguientes etapas básicas para crear programas en C durante todas las prácticas de FSO:

- Utilice un editor para escribir el código fuente del programa y guárdelo en archivos con extensión *". c"*.

```
$ kate myprogram.c &
```

- Si con el compilador *gcc* utiliza la opción "-o" y no se detectan errores en el programa fuente, automáticamente se genera un archivo con el código ejecutable. Por tanto, el compilador se invoca con al menos dos nombres de archivo: el del código fuente y donde debe almacenar el código ejecutable.

```
$ gcc myprogram.c  -o codeprogram
```

- Para ejecutar el programa debe utilizar el nombre del archivo que contiene código ejecutable

```
$ ./codeprogram
```

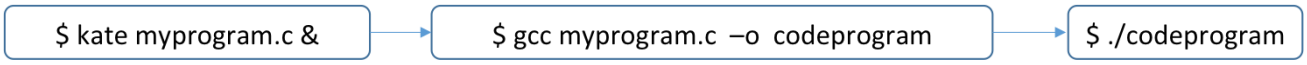

Figura 1: Etapas básicas para crear programas en C

¡RECUERDE! Un programa en C necesita siempre una función *main()*. En C todas las variables deben ser declaradas antes de ser utilizadas y todas las instrucciones se escriben en minúsculas y acaban en ";".

2.2 Warnings y errores

Es importante diferenciar entre warnings y errores cuando se observa el resultado de compilar un programa. Un **warning** (aviso) indica que el código es ambiguo y puede ser interpretado de manera diferente de un compilador a otro, pero el ejecutable puede ser creado. Un **error** indica que no se ha podido compilar completamente el programa y por tanto el ejecutable no puede crearse.
Los errores más típicos son: **errores de compilación y errores de enlazados**. Los errores de compilación suelen darse cuando se omite algo en el código fuente como por ejemplo un ";". Los errores de enlazado son más sutiles y suelen darse cuando el programa se encuentra repartido en varios archivos (a estudiar más adelante).

2.3 Herramienta "*man*" para encontrar información

"*man*" (manual) es una herramienta de UNIX que tiene una entrada para todos los comandos, funciones y llamadas al sistema disponibles. Para saber todo lo relacionado con la aplicación *man* debe poner:

```
$ man man
```

La documentación de las llamadas al sistema se encuentra en la sección 2 del manual, mientras que las de las funciones están en la sección 3. Por ejemplo, para consultar la función printf() o sqrt() debe teclear:

```
$ man 3 printf
$ man 3 sqrt
```

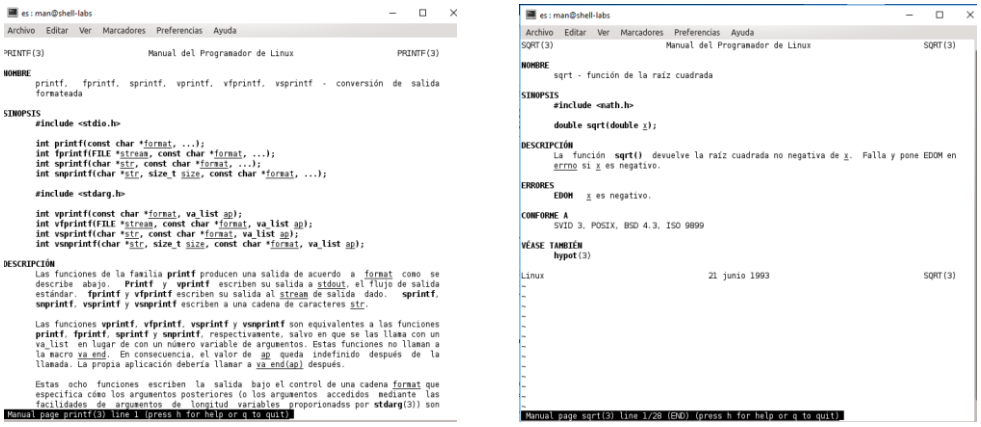

Figura 2: Páginas de *man* para las funciones *printf()* y *sqrt()*.

En cualquier página del manual la sección SYNOPSIS, incluye información del uso de la función. Por ejemplo, en el caso de la Figura 2:

#include <stdio.h>

significa que para utilizar la función printf() se debe tener dicho #include en el archivo del programa. También describe los parámetros necesarios para invocarla y lo que devuelve tras la ejecución. La sección DESCRIPTION proporciona una pequeña descripción de lo que hace la función.

Para salir de una página de *man* hay que pulsar la tecla "q".

3 Programas en C

Aprender C requiere practicar realizando programas y modificándolos.

¡IMPORTANTE! Recuerde mantener siempre una copia de la versión de sus programas cuando le funcionen y antes de hacer grandes cambios en él.

3.1 Ejercicio 1.1 Cree un programa en C: Edite, compile y ejecute

Cree el archivo "numbers.c" con el contenido mostrado en la Figura 3, que corresponde a la estructura básica de un programa en C. El programa "numbers" debe escribir información (números) en pantalla, por lo que se incluye la librería <stdio.h> en el código fuente y se invoca la función printf →Escritura en pantalla.

Los pasos a seguir son:

```
$ kate numbers.c &
```

Lance el editor con el nombre del archivo, cuando aparezca la ventana del editor, escriba las instrucciones. Recuerde salvar cada cierto tiempo para no perder información.

Compile su código fuente con: (para recuperar la consola pulsar Enter)

```
$ gcc numbers.c  -o numbers
```

Compruebe si el compilador le muestra errores.

Es instructivo usar la opción -v de gcc para obtener un informe detallado de todos los pasos de compilación:

```
  $ gcc -v -o numbers numbers.c
```

Cuando ya haya solucionado todos los errores ejecútelo invocando el archivo de contiene código ejecutable

```
$ ./numbers
```

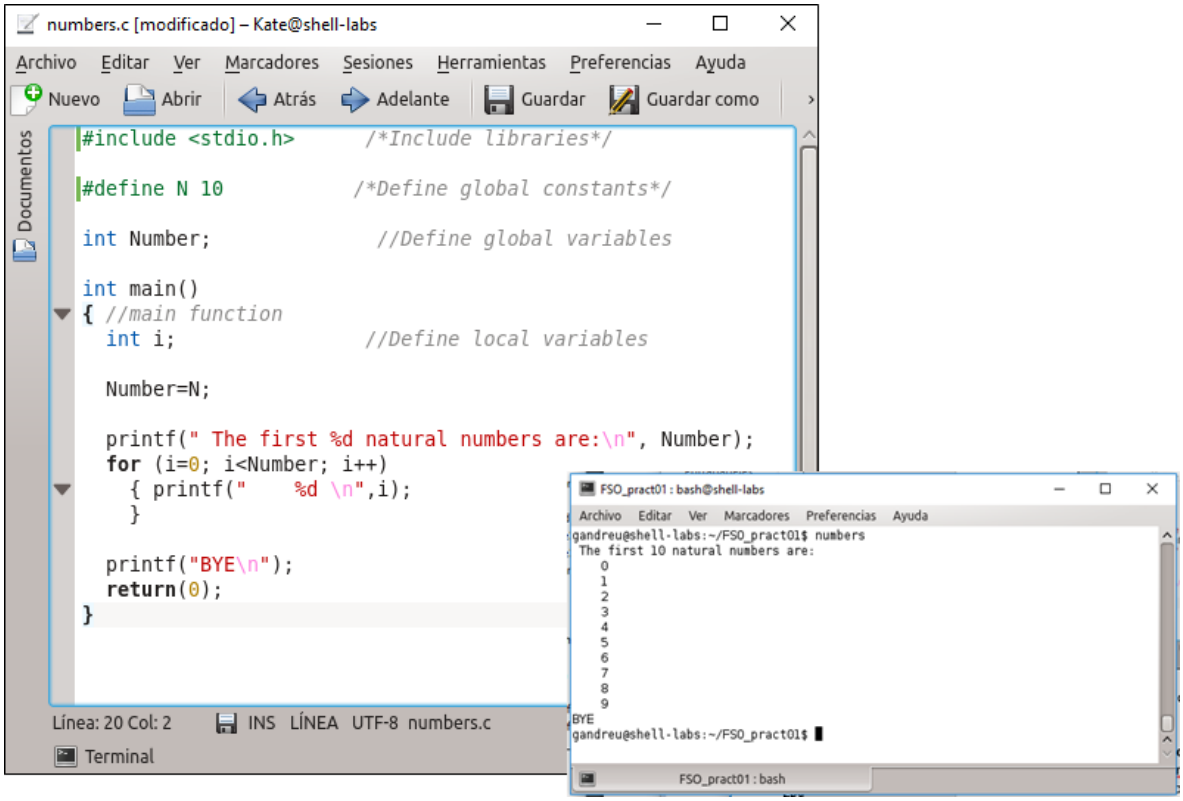

Figura 3: Contenido del archivo "numbers.c" en el editor kate y resultado de la ejecución.

3.2 Ejercicio 1.2 Errores básicos de compilación y *warnings*

Edite el archivo "*numbers.c*" y realice cada una de las acciones que se proponen en la tabla. Tras cada acción debe salvar el archivo, compilar e identificar el tipo de error y el mensaje que le muestra el compilador. Tenga en cuenta que antes de hacer un cambio ha de deshacer el cambio anterior.

Acción	Nº de línea del error	¿Se ha generado el ejecutable?	Error o warning
Comente la declaración de variable i → //int i;	13	No	Error
Sustituya "%d" por "%f"	12/14	No	Warning
Elimine un ";"	12	No	warning y error
Sustituya "main" por otro nombre "many"	8	No	-

3.3 Ejercicio 1.3 Lectura de teclado: función "scanf()"

Para que *"numbers"* escriba un conjunto diferente de números se ha de cambiar el valor de la constante global N por otro y volver a compilar. Sin embargo, el lenguaje C proporciona otras opciones sin necesidad de modificar el código fuente y compilar cada vez.

Copie "*numbers.c*" en otro archivo "*numbers2.c*". Edite el archivo "*numbers2.c*" y modifíquelo para que pregunte al usuario cuantos números quiere visualizar y lea el valor introducido por el usuario. Utilice la función scanf() (Ejemplo: scanf("%d", &Number);. El funcionamiento que se requiere es el que se aprecia en la figura 4.

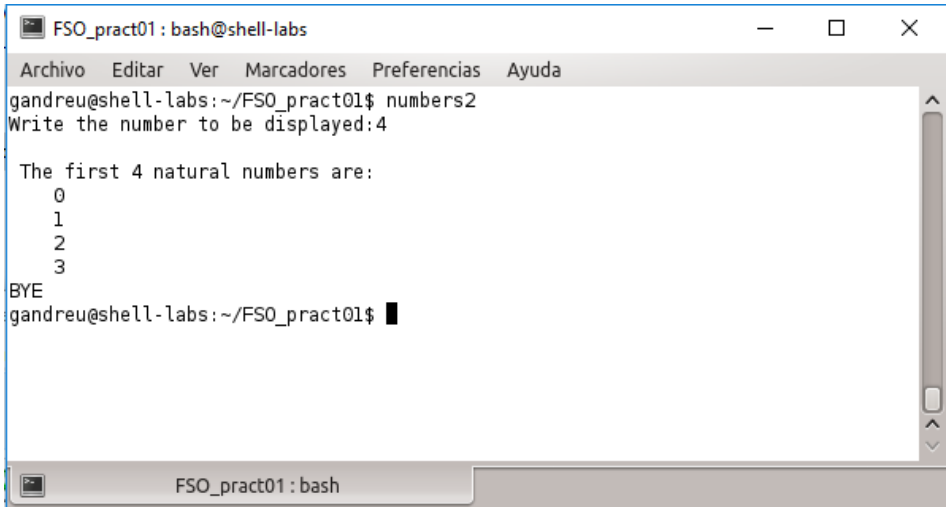

Figura 4: Ejecución de "*numbers2*", visualiza los números solicitados por el usuario.

3.4 Ejercicio 1.4 Control de flujo "if …. else if"

Descargue de Poliformat el archivo "*atm.c*" (*automatic teller machine*), contiene el código mostrado en figura 5, compílelo y ejecútelo. "*atm*" simula las operaciones de un cajero automático y está escrito de una forma básica utilizando "if … else if …". Compruebe su funcionamiento ejecutándolo tantas veces como opciones permite.

```
#include <stdio.h>

#define InitBalance 1000
float Balance;

int main()
{ int operation, value;
  float income, withdraw;

  printf("\nWelcome to the FSO ATM\n");
  Balance=InitBalance;
  operation=0;
  printf("\nIndicate operation to do:\n");
  printf(" 1.Cash Income\n 2.Cash Withdrawal\n 3.Balance Enquiry\n");
  printf(" 4.Account Activity\n 5.Change PIN\n 6.Exit\n\n");
  printf(" Operation:");
  value=scanf("%d",&operation);

  if(operation==1){
    printf(" Cash Income\n");
    printf("\n Enter the amount to deposit:");
     scanf("%f",&income);
     Balance=Balance+income;
    printf(" Successful income\n");

  } else if(operation==2){
    printf(" Cash Withdrawal\n");
    printf("\n Enter the amount to withdraw:");
    scanf("%f",&income);

     if(Balance>income){
       Balance=Balance-income;
     }else{
       printf(" Operation does not allowed\n");
       printf("   Not enough cash\n");
     }

  }else if(operation==3){
   printf(" Balance Enquiry\n");

  }else if((operation==4)|| (operation==5)){
   printf(" This operation is not implemented");

 }else if(operation==6){
   printf(" EXIT\n");

  }else if(operation>6){
   printf(" ERROR: This opertaion does not applied\n");

  }

  printf("\n\n Current Balance: %.2f Euros", Balance);
  printf("\n\n Thanks \n\n");
   return(0);
}
```

```
gandreu@shell-labs:~/FSO_pract01$ atm

Welcome to the FSO ATM

Indicate operation to do:
 1.Cash Income
 2.Cash Withdrawal
 3.Balance Enquiry
 4.Account Activity
 5.Change PIN
 6.Exit

 Operation:2
 Cash Withdrawal

 Enter the amount to withdraw:270


 Current Balance: 730.00 Euros

 Thanks

gandreu@shell-labs:~/FSO_pract01$
```

Figura 5: Código fuente y ejemplo de ejecución del programa "*atm.c*".

3.5 Ejercicio 1.5 Control de flujo "switch"

Para decisiones múltiples donde se comprueba si una variable coincide con uno de un conjunto de valores es aconsejable utilizar la estructura:

```
switch(exp){
  case exp-const: ------------------------;
        break;
  case exp-const: ------------------------;
        break;
  default: ---------------------------------;
        break;
  }
```

Copie el archivo "*atm.c"* en "*atm2.c"* utilizando la orden del *Shell "$cp atm.c atm2.c"*. Modifique "*atm2.c*" utilizando la proposición *switch{}* reemplazando los "*if else*" que comprueban el valor de la variable "*oper*" por la sentencia "*case*" correspondiente. El funcionamiento debe ser el mismo que con "*if … else*". Una vez lo haya conseguido conteste a las siguientes cuestiones:

Pregunta	Respuesta
¿Qué expresión ha utilizado en *switch*(exp)?	switch(operation)
¿Por qué es necesario el *break;*?	para que no se vaya al siguiente case
¿Cómo ha resuelto la comprobación de la línea 43? `}else if((operation==4)|| (operation==5)){`	añadiendo más cases al código (para 4 y para 5)
¿Por qué es necesario el default:?	para tener un caso base

3.6 Ejercicio 1.6 Bucle "while"

Copie "*atm2.c*" en "*atm3.c"*. Modifique atm3.c utilizando la proposición "*while(exp){}"*, de manera que permita más de una operación en cada ejecución, mientras no se seleccione la operación de salida (6). Una vez resuelto intente contestar a las siguientes cuestiones:

Nota. Para sangrar varias líneas al mismo tiempo seleccionar las líneas a sangrar y pulsar en Herramientas -> Alinear

Pregunta	Respuesta

¿Qué expresión ha utilizado en while(exp)?	while(operation != 6) {
	...
	}

4 Propuestas para trabajar por su cuenta

El programa "*atm.c"* puede ser mejorado, aquí se indican algunas funciones por si el alumno desea practicar:

- Función Saldo: Añadir función que pregunte si ver o no el saldo (funciones de la librería string.h.)
- Función Idioma: Añadir función que permita seleccionar el idioma del menú.

- Función Seguridad: Añadir función de seguridad que compruebe su autentificación.
- Función Movimientos: En este caso debe trabajar con archivos. Añadir una función que almacene las operaciones en un archivo y sea capaz de recuperarlas para mostrárselas al cliente

5 Anexo 1: GCC ("GNU Compiler Collection")

GCC ("GNU Compiler Collection") engloba un conjunto de compiladores desarrollados dentro del proyecto GNU, por tanto, es software libre. Actualmente incluye compiladores para C, C++, Objective C, Fortran, Ada. GCC toma un programa fuente y genera un programa ejecutable binario para la máquina donde se ejecuta. GCC puede generar código para Intel x86, ARM, Alpha, Power PC, etc.. Es utilizado como compilador de desarrollo en la mayoría de plataformas. Asi, Linux, Mac OS X, iOS (iPhone e iPad) están íntegramente compilados con GCC.

5.1 Sintaxis y opciones de GCC

La sintaxis de uso del compilador gcc es la siguiente:

```
$ gcc [-options] [source_files] [object_files] -o output_file...
```

Las opciones van precedidas de un guión, como es habitual en UNIX, cada opción puede constar de varias letras y no pueden agruparse varias opciones tras un mismo guión. Algunas opciones requieren después un nombre de archivo o directorio, otras no. Finalmente, pueden darse varios nombres de archivo a incluir en el proceso de compilación.
GCC tiene infinidad de opciones, a continuación, se detallan algunas:

Opción	Descripción
-c	realiza el preprocesamiento, compilación y ensamblado, obteniendo el archivo en código objeto (.o).
-S	realiza el preprocesamiento y compilación, obteniendo el archivo en ensamblador.
-E	realiza solamente el preprocesamiento, enviando el resultado a la salida estándar
-o archivo	indica el nombre del archivo de salida (el ejecutable).
-Iruta	especifica la ruta del directorio de los archivos a incluir en el programa fuente. No lleva espacio entre la I y la ruta, así: -I/usr/include. Por defecto no es necesario indicar las rutas de las "include" estándar.
-Lruta	especifica la ruta del directorio de los archivos de biblioteca con código objeto de las funciones referenciadas en el programa fuente. No lleva espacio entre la L y la ruta, así: -L/usr/lib. Por defecto no es necesario indicar las rutas de las librería estándar.
-lNOMBRE	NOMBRE: librería a enlazar junto con el programa. Le dice al compilador qué biblioteca o librerías tiene que incluir con el programa desarrollado. Para modificar el conjunto de librerías que utiliza el enlazador por defecto. Por ejemplo: -lm incluiría la librería lib**m**.so (librería matemática)
-v	Verbose on; muestra los comandos ejecutado en cada etapa de compilación y la versión del mismo.

5.2 Fases de compilación

El gcc, al igual que otros compiladores, se pueden distinguir 4 etapas en el proceso de compilación (Figura 6):

- **Preprocesado**: Esta etapa interpreta las directivas al preprocesador. Entre otras cosas, las variables inicializadas con #define son sustituidas en el código por su valor en todos los lugares donde aparece su nombre y se incluye el fuente de los #include.
  Ejemplo;

```
$ gcc -E numbers.c > numbers.pp
 Examine numbers.pp con
$ more numbers.pp
 Podra comprobar que la variable N ha sido sustituida por su valor
```

- **Compilación**: transforma el código C en el lenguaje ensamblador propio del procesador de nuestra máquina. Si la máquina destino difiere de la del desarrollo se denomina compilación cruzado.

```
   $ gcc -S numbers.c
realiza las dos primeras etapas creando el archivo numbers.s;  examinándolo con
  $ more numbers.s
puede verse el programa en lenguaje ensamblador.
```

- **Ensamblado**: transforma el programa escrito en lenguaje ensamblador a código objeto, un archivo binario en lenguaje de máquina ejecutable por el procesador. No es frecuente realizar sólo el ensamblado; lo usual es realizar todas las etapas anteriores hasta obtener el código objeto así:

```
 $ gcc -c numbers.c
donde se crea el archivo numbers.o a partir de numbers.c. Puede verificarse el tipo de archivo usando el comando
 $ file numbers.o
```

- **Enlazado**: Las funciones de C/C++ incluidas en nuestro código, tal como printf(), se encuentran ya compiladas y ensambladas en bibliotecas existentes en el sistema. Es preciso incorporar de algún modo el código binario de estas funciones a nuestro ejecutable. En esto consiste la etapa de enlace, donde se reúnen uno o más módulos en código objeto con el código existente en las bibliotecas.El enlazador se denomina ld. El comando para enlazar

```
 $ ld -o circulo circulo.o -lc
```

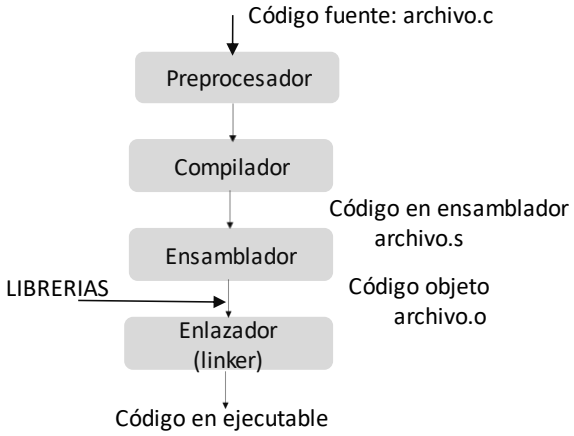

Figura 6: Fases de compilación.

5.3 Todas las fases en un solo paso

En programa con un único archivo fuente todo el proceso anterior puede hacerse en un solo paso:

```
$ gcc -o numbers numbers.c
```

No se crea el archivo numbers.o; el código objeto intermedio se crea y destruye sin verlo el operador, pero el programa ejecutable aparece allí y funciona.
Es instructivo usar la opción -v de gcc para obtener un informe detallado de todos los pasos de compilación:

```
$ gcc -v -o numbers numbers.c
```

Ejemplos de uso:

```
$ gcc hola.c -o hola
 Compila el programa hola.c y genera un archivo ejecutable hola
$ gcc hola.c
  Compila el programa hola.c y genera un archivo ejecutable a.out.
$ gcc -o hola hola.c
  Compila el programa hola.c y genera un archivo ejecutable hola.
```

6 Anexo 2: Librería stdio.h funciones printf() y scanf()

En lenguaje C para la entrada/salida se debe utilizar las funciones de entrada y salida estándar proporcionadas por la biblioteca estándar de lenguaje C stdio.h, como son printf y scanf, entre otras.

6.1 La función printf()

La función printf() envía una cadena de texto con formato a la salida estándar (la pantalla). La cadena de formato contiene dos tipos de objetos: caracteres ordinarios y especificaciones de conversión. Los caracteres ordinarios son copiados al flujo de salida, mientras que las especificaciones de conversión provocan la conversión e impresión de los valores de los argumentos. Cada especificación de conversión debe ir precedida de el símbolo %.

La sintaxis básica para invocar esta función es:

```
printf(<cadena_de_control> [, <lista_de_argumentos> ] )
```

Ejemplo:

Código fuente	Resultado en salida estándar

`#include <stdio.h>`
`int main(){`
`  int num1 = 10;`
`  printf("Number %d", num1);`
`  return 0;`
`}`
	`Number 10`

Figura 7: Ejemplo de programa con printf()

La <cadena_de_control> es, en sí misma, una cadena de caracteres, que se debe escribir entre comillas dobles ("). Los argumentos o parámetros sirven para transferir datos entre funciones o programas. Cuando un programa invoca la función printf() con una lista de argumentos, dichos valores (si no hay error) se muestran en pantalla junto con la <cadena_de_control>. Por tanto, la <cadena_de_control> debe indicar el formato de salida de los datos que se van a mostrar en pantalla y puede constar de:

- Texto ordinario (texto normal).
- Especificadores de formato.
- Secuencias de escape.

Tabla 1: Especificaciones de formato para imprimir con printf()

%caracter	Tipo de argumento que requiere→ convertido a

%c	Int (entero)→ char (caracter): escribe un carácter simple
%d, %i	Int (entero)→ Int : escribe un entero con signo en base decimal
%e, %E	double → reales(double): escribe un numero flotante con exponente, en notación científica indicando el exponente con "e"
%f	double → reales: escribe un número en punto flotante sin exponente, formato de punto flotante mmm.ddd
%g, %G	double → escribe la opción más corta entre "%e" y "%f" o entre "%E" y "%F"
%o	Int → escribe un entero en octal sin signo, sin ceros iniciales
%s	Char * → imprime una cadena de caracteres hasta encontrar el carácter '\0'.
%u	Int → escribe un entero decimal sin signo
%x, %X	Int → escribe un entero hexadecimal sin signo, sin el prefijo 0x
%p	Void * → imprime un puntero

Los especificadores de formato establecen el formato de salida por pantalla de los argumentos.
Las secuencias de escape nos permiten dan formato a la información mostrada por la salida estándar. Una secuencia de escape siempre representa a un carácterl ASCII y se escribe con el carácter barra invertida (\) . Dichos caracteres se pueden clasificar en:

- Gráficos: lo cuales se corresponden con símbolos usados para escribir, ejemplo: ",\, etc...
- No gráficos: representan acciones, por ejemplo, mover el cursor al principio de la línea siguiente, etc..

Tabla 2: Secuencias de escape más utilizadas.

Secuencias escape	Significado /Acción
\n	Nueva línea (ASCII →010)
\t	Tabulación horizontal (ASCII →009)
\v	Tabulación vertical
\f	Nueva página
\r	Retorno de carro
\\	Muestra el carácter barra invertida (ASCII → 092)
\"	Comilla doble muestra (ASCII →034)

6.2 La función scanf()

La función scanf() permite asignar valores a variables desde teclado, su formato es similar a printf. La función scanf() de la biblioteca estándar stdio.h del lenguaje C asigna a una o más variables, uno o más valores (datos) recibidos desde la entrada estándar (teclado). La sintaxis de su llamada es:

scanf(<cadena_de_control> [, <lista_de_argumentos>])

En la <cadena_de_control>, se debe indicar el formato de entrada de los datos que se van a recoger por teclado. Para ello, se deben utilizar los especificadores de formato.
Por ejemplo: scanf("%c %d %f %s", &ch, &i, &x, cad);

Código fuente (ejemplo-A2)	Resultado en salida estándar

`#include <stdio.h>`
`int main(){`
`  int num;`
`  printf("\n Write an integer: ");`
`  scanf("%d", &num);`
`  return 0;`
`}`
	Write an integer: 4

Figura 8 : Ejemplo de programa con scanf()

Al ejecutar el código del ejemplo A2 en la memoria se reservará espacio para la variable *num*. Y si el usuario teclea, por ejemplo, un 4, en pantalla se verá lo mostrado en la columna derecha. Puesto que la variable num es

de tipo entero, en la cadena de control se escribe el especificador de un número entero (%d). Por otro lado, la variable num está precedida del carácter ampersand (&)
Observe que en la función scanf() los nombres de las variables van **precedidos de &** excepto cuando corresponde a una cadena de caracteres. Se trata de un nuevo operador, el operador dirección o puntero a variable. El operador dirección (&) siempre actúa sobre un operando (normalmente una variable) para obtener la dirección de memoria de dicha variable. Las variables se encuentran almacenadas en espacios de memoria y la función scanf() requiere las direcciones de memoria de las variables, de ahí que vayan precedidas de &.