Principios de Programación

El lenguaje C

1. Instrucciones de entrada/salida

Una parte importante de la solución de cualquier problema es la presentación de los resultados. Analizaremos a fondo las caracteristicas de formato de printf y scanf.

Ya hemos visto algunas caracteristicas de scanf y printf. Resumiremos esas características y presentaremos otras.

1.1. Flujos

Toda entrada y salida se ejecuta mediante *flujos* (secuencias de caracteres organizadas en lineas). Cada linea está formada de cero o más caracteres y está terminada por el caracter de nueva linea.

Cuando empieza la ejecución del programa de forma automática se conectan tres flujos al programa. Por lo regular el *flujo estandar de entrada* se conecta al teclado y el *flujo estandar de salida* se conecta a la pantalla. Un tercer flujo, el *error estandar* se conecta a la pantalla. Los mensajes de error son extraidos o desplegados al flujo estandar de error.

Con frecuencia los sistemas operativos permiten que estos flujos sean redirigidos a otros dispositivos.

stdin y stdout pueden redirigirse a archivos o conductos (pipes), por ejemplo:

prog < infile

hace que prog lea el contenido de infile en vez de leer del teclado. Otro ejemplo:

prog | otroprog

hace que la salida estandar de prog sea la entrada estandar de otroprog.

1.2. fgetc, fputc, getchar, putchar, fscanf, fprintf

Cuando un programa comienza, hay 3 archivos que se abren automáticamente y proveen punteros a archivos: los apuntadores de archivo asignados a la entrada estandar (**stdin**) , a la salida estandar (**stdout**) y al error estandar (**stderr**). La idea es que cuando quiero leer o imprimir en las entrada y salida estandar coloco como nombre de archivo stdin y stdout.

Puedo sino leer o imprimir a archivos (en cuyo caso doy el nombre del archivo en la intrucción de entrada/salida).

La biblioteca estandar provee funciones para leer datos de archivos y para escribir datos a los archivos.

Los flujos proporcionan canales de comunicación entre archivos y programas. Abrir un archivo regresa un apuntador a una estructura FILE (FILE es

un nombre de tipo definido en stdio) que contiene información utilizada para procesar dicho archivo.

La operación **fgetc**, declarada como:

int fgetc(FILE *stream)

devuelve el siguiente caracter del flujo de entrada al cual señala stream. Si el flujo está al fin de archivo fgetc devuelve EOF. Idem si hay un error de lectura (devuelve EOF).

Podemos definir getchar() en terminos de getc con la sentencia:

#define getchar() fgetc(stdin)

La operación **fputc**, declarada como:

int fputc(int c,FILE *stream)

escribe el caracter c en el flujo de salida stream. Si hay un error de escritura devuelve EOF.

Podemos definir putchar(c) en terminos de fputc con la sentencia:

#define putchar(c) fputc(c,stdout)

Trabajaremos con la entrada y salida estandar.

2. Salida con formato utilizando printf

Cada llamada a printf contiene una *cadena de control de formato* que describe el formato de la salida. La cadena de control de formato consiste de especificadores de conversión, banderas, anchos de campo, precisiones y caracteres literales.

La función printf puede llevar a cabo las siguientes capacidades de formato:

1. Redondear valores de punto flotante a un número indicado de valores decimales.
2. Alinear una columna de números con puntos decimales apareciendo uno por encima del otro.
3. Salidas justificadas a la derecha o a la izquierda.
4. Insertar caracteres literales en posiciones precisas en una linea de salida.
5. Representación en formato exponencial de números de punto flotante.
6. Representación en formato octal y hexadecimal de enteros no signados.
7. Despliegue de todo tipo de datos con anchos de campo de tamaño fijo y precisiones.

La forma de la operación **printf** es la siguiente:

printf(cadena de control de formato, otros argumentos)

los otros argumentos son opcionales y se corresponden con las especificaciones de conversión, las que comienzan con % y forman parte de la cadena de control de formato. La cadena de control de formato se coloca entre comillas.

2.1. Como imprimir enteros

A continuación se describen cada uno de los especificadores de conversión de enteros.

d	Despliega un entero decimal signado
i	Muestra un entero decimal signado. i y d son diferentes cuando se usan con scanf y son iguales cuando se usan en printf.
o	Muestra un entero octal no signado
u	Muestra un entero decimal no signado
x o X	Muestra un entero hexadecimal no signado. con X se imprimen las letras A hasta F en mayuscula, con x se imprimen las mismas letras en minuscula.
h o l	se coloca antes de cualquier especificador de conversión de enteros para indicar que se muestra un entero short o long respectivamente.

Abajo presentamos un programa ejemplo que muestra el uso de los especificadores de conversión. Notar que solo se imprime el signo menos (-), más adelante veremos como imprimir los signos +. Notar también que el uso de u en un número negativo cambia su valor.

Programa

```
#include <stdio.h>

main ()
{
      printf(“ %d\n”,455);
      printf(“ %i\n”,455);
      printf(“ %d\n”,+455);
      printf(“ %d\n”,-455);
      printf(“ %hd\n”,32000);
      printf(“ %ld\n”,2000000000);
      printf(“ %o\n”,455);
      printf(“ %u\n”,455);
      printf(“ %d\n”,-455);
      printf(“ %x\n”,455);
      printf(“ %X\n”,455);
      system(”PAUSE”); }
```

El programa anterior imprime:

455
455
455
-455
32000
2000000000
707
455
65081 (u otro valor que depende del compilador)
1c7
1C7

Notar que 455 decimal es 707 octal y que 455 decimal es 1c7 hexadecimal.

El uso de hd para valores muy grandes puede dar error (desplega cualquier valor).

2.2. Como imprimir números de punto flotante

Un valor de punto flotante contiene un punto decimal, como en 33.5 o 657.983. Los valores de punto flotante se pueden imprimir en uno de varios formatos.

Los especificadores de conversión son los siguientes:

e o E	Muestra un valor en punto flotante en notación exponencial.
f	Muestra valores en punto flotante.
g o G	Despliega un valor en punto flotante ya sea en la forma de punto flotante f o en la notación exponencial e (o E).
L	Se coloca antes de cualquier especificador de conversión de punto flotante para indicar que está mostrando un valor de punto flotante long double.

En forma preestablecida los valores impresos con los especificadores de conversión e, E y f son extraidos con 6 digitos de precisión a la derecha del punto decimal.

El especificador de conversión f siempre imprime por lo menos un digito a la izquierda del punto decimal (puede ser 0).

Los especificadores de conversión e y E imprimen respectivamente la e en minusculas y la E en mayusculas antes del exponente y siempre se imprime exactamente un digito a la izquierda del punto decimal (distinto de 0).

El especificador de conversión g (G) imprime en formato e (E) o en formato f sin ceros después del punto decimal. Los valores se imprimen con e (E) si despues si despues de convertir el valor a notación exponencial, el exponente del valor es menor a -4 o es mayor o igual a la precisión especificada (por omisión en el caso de g o G, 6 dígitos significativos). De lo contrario se utilizará f.

Si se utiliza g o G los valores 0.0000875, 8750000.0,8.75,87.50 y 875 se imprimen como: 8.75e-005,8.75e+006,8.75,87.5 y 875.

La precisión de los especificadores de conversión g y G indican el número máximo de digitos significativos impresos incluyendo el dígito a la izquierda del punto decimal. Cuando decimos la precisión es de 6 dígitos significativos, nos referimos a 5 dígitos despues del punto decimal y 1 a la izquierda del punto decimal. Por ejemplo, si quisiera imprimir 1234567.89 con %g se imprime: 1.23457e+006 (se corre el punto decimal 6 posiciones y se imprimen 5 decimales), si quisiera imprimir 12345678.9 con %g se imprime: 1.23457e+007 (se corre el punto decimal 7 posiciones y se imprimen 5 decimales). Si quisiera imprimir 1234.567 se imprime 1234.57.

Considere el siguiente programa:

```
#include <stdio.h>

main ()
{
      printf(“ %e\n”, +1234567.89);
      printf(“ %E\n”, -1234567.89);
      printf(“ %e\n”, 12.3456789e+05);
      printf(“ %f\n”, 1234567.89);
      printf(“ %f\n”, 12.3456789e+05);
      printf(“ %g\n”, 1234567.89);
      printf(“ %G\n”, 12.3456789);
}
```

se imprime:

```
1.234568e+006
-1.234568E+006
1.234568+006
1234567.890000
1234567.890000
1.23457e++006
12.3457
```

2.3. Cómo imprimir caracteres

Para imprimir caracteres individuales se utiliza el especificador de conversión ” %c”. Se requiere de un argumento char.

Ejemplo

```
main ()
{
      char caracter=’A’;
      printf(“ %c\n”,caracter);
      system(“PAUSE”);
}
```

imprime el caracter 'A'.

2.4. Cómo imprimir con anchos del campo y precisiones

El tamaño exacto de un campo en el cual se imprimen datos se especifica por el ancho del campo. Si el ancho del campo es mayor que los datos que se están imprimiendo, a menudo los datos dentro de dicho campo quedarán justificados a la derecha. Un entero que representa el ancho del campo es insertado en la especificación de conversión, entre el signo de % y el especificador de conversión.

En el caso de que el ancho del campo es menor que los datos que se están imprimiendo este es aumentado automáticamente. Además el signo de menos de un valor negativo ocupa una posición.

Los anchos de campo pueden ser utilizados con todos los especificadores de conversión.

La función printf también da la capacidad de definir la **precisión** con la cual los datos se imprimirán. La precisión tiene significados distintos para diferentes tipos de datos. Cuando se utiliza con especificadores de conversión de enteros, la precisión indica el número mínimo de digitos a imprimirse. Si el valor impreso contiene menos dígitos que la precisión especificada al valor impreso se le antepondrán ceros hasta que el número de digitos sea igual a la precisión. La precisión por omisión para enteros es 1.

Cuando se utiliza con especificadores de conversión de punto flotante e,E y f la precisión es el número de dígitos que aparecerá despues del punto decimal. Cuando se utilice con los especificadores de conversión g y G, la precisión es el número máximo de digitos significativos a imprimirse (se consideran digitos a la izquierda del punto decimal más digitos a la derecha).

Para especificar la precisión, coloque un punto decimal seguido por un entero que representa la precisión entre el ancho del campo y el especificador de conversión. Cuando un valor en punto flotante se imprime con una precisión menor que el número original de decimales el valor se redondea.

Ejemplo:

printf(“%9.3f”,123.456789);

Imprime 123.457 justificado a la derecha en un campo de 9 dígitos.

Utilizando expresiones enteras en la lista de argumentos a continuación de la cadena de control de formato es posible especificar el ancho de campo y la precisión. Para utilizar esta característica inserte un asterisco (*) en el lugar del ancho de campo y/o la precisión. El argumento coincidente en la lista de argumentos se evalua y se utiliza en lugar del asterisco. El valor correspondiente al ancho de campo puede ser negativo, el correspondiente a la precisión debe ser positivo. Un valor negativo para el ancho de campo hace que la salida se justifique a la izquierda.

Ejemplo:

printf(“%*.*f”,7,2,98.736);

utiliza 7 para el ancho de campo y 2 para la precisión. Imprime 98.74 justificado a la derecha.

Ejemplo:

```
#include <stdio.h>


main ()
{
/* Uso de ancho del campo */
      printf(“%4d\n”, 12);
      printf(“%4d\n”, -123);
      printf(“%4d\n”, 123456);
      printf(“%3f\n”, 12.34);
      printf(“%15f\n”, 12.34);
/* Uso de precision */
      printf(“%.4d\n”, 873);
      printf(“%.9d\n”, 873);
      printf(“%5.4d\n”, 12);
      printf(“%8.5d\n”, 123);
      printf(“%4.5d\n”, 123);
      printf(“%9.3f\n”, 1234.5678);
      printf(“%20.3e\n”, 1234.5678);
      printf(“%5.4g\n”, 1234.5678);
      printf(“%4.4g\n”, 111.6);
```

imprime: (b significa espacio)

```
bb12
-123
123456
12.340000
bbbbbb12.340000
0873
000000873
b0012
bbb00123
00123
b1234.568
bbbbbbbbbb1.235e+003
b1235
111.6
```

2.5. Uso de banderas en la cadena de control de formato de printf

La función printf tiene también *banderas* para complementar sus capacidades de formato de salida. Están disponibles cinco banderas, para uso en cadenas de control de formato. Las banderas son las siguientes:

-	signo de menos. Justificación de la salida a la izquierda. El signo de menos se coloca entre el
+	signo de más. Despliega un signo de más antes de valores positivos y un signo de menos antes de valores negativos.
espacio	Imprime un espacio antes de un valor positivo que no se imprima con la bandera +.
#	Antecede un 0 al valor extraido cuando se utiliza con el especificador de conversión octal o. Antecede 0x o 0X al valor de salida cuando se utiliza con los especificadores de conversión hexadecimales x o X. Obliga a un punto decimal para un número de punto fl otante impreso con e,E,f,g o G que no contenga una parte fraccionaria (se imprimen 0's a la derecha del punto decimal). En el caso de los especificadores g y G no se eliminan los ceros a la derecha.
0	cero. Rellena un campo con ceros a la izquierda.

Las banderas se colocan de inmediato a la derecha del signo de %. En una especificación de conversión se pueden conmbinar varias banderas.

Ejemplo 1

```
main ()
{
      printf(" %5d %15f\n", 22, 12.34);
      printf(" % -5d % -15f\n",22,12.34);
      system("PAUSE");
}
```

imprime:

```
bbb22bbbbbbb12.340000
b22bbb12.340000
```

observar en el ejemplo anterior además del uso del menos (-) el uso del espacio.

Ejemplo 2

```
main ()
{
	printf(" %d\n %d\n", 786,-786);
	printf(" %+d\n %+d\n", 786, -786);
	system("PAUSE");
}
```

se imprime:

```
786
-786
+786
-786
```

Ejemplo 3

```
main ()
{
	int c= 1427;
	float p= 235.7;
	float q= 123;

	printf(" %#o\n", c);
	printf(" %#x\n", c);
	printf(" %#X\n", c);
	printf("\n %g\n", p);
	printf(" %#g\n", p);
	printf(" %#g\n", q);
	system("PAUSE");
}
```

se imprime:

```
02623
0x593
0X593

235.7
235.700
123.000
```

recordar que g o G imprime sin ceros despues del punto decimal. Si queremos que no se eliminen los ceros a la derecha debemos usar #. Observar que g se

declaro sin punto decimal ni ceros a la derecha y cuando se imprime se imprime con punto decimal y ceros. Pasa lo mismo si imprimiera con f o con e (en este ultimo caso se usa la notación exponencial).

Ejemplo 4

```
main ()
{
      printf(" %+09d\n", 452);
      printf(" %09d\n", 452);
      system("PAUSE");
}
```

se imprime:

```
+00000452
000000452
```

3. Entrada con formato utilizando scanf

Cada enunciado scanf contiene una cadena de control de formato que describe el formato de los datos que se leen. La cadena de control está formada de especificaciones de conversión y de caracteres literales. La función scanf tiene las siguientes capacidades de formato de entrada:

1. Entrada de todo tipo de datos.
2. Entrada de caracteres especificos desde un flujo de entrada.
3. Omitir caracteres especificos del flujo de entrada.

La función scanf se escribe en la forma siguiente:

```
scanf(string-de-control-de-formato, otros-argumentos)
```

el primer argumento describe los formatos de la entrada y los otros-argumentos son apuntadores a variables en las cuales se almacena la entrada.

A continuación se resumen los especificadores de conversión utilizados para introducir todos los tipos de datos.

Enteros

d	Lee un entero decimal, opcionalmente signado. El argumento correspondiente es un apuntador a un entero.
i	Lee un entero decimal, octal o hexadecimal, opcionalmente signado. El argumento correspondiente es un apuntador a un entero.
o	Lee un entero octal. El argumento correspondiente es un apuntador a un entero no signado.
u	Lee un entero decimal no signado. El argumento correspondiente es un apuntador a un entero no signado.
x o X	Lee un entero hexadecimal. El argumento correspondiente es un apuntador a un entero no signado.
h o l	Se coloca antes de cualquiera de los especificadores de conversión de enteros, para especificar que un entero short o long será introducido.

Números de punto flotante

e, E, f, g o G	Lee un valor en punto flotante. El argumento correspondiente es un apuntador a una variable de punto flotante.
L	Se coloca delante de cualquier especificador de conversión de punto flotante para indicar que un valor double o long double será introducido.

Caracteres

c	Lee un caracter. El argumento correspondiente es un apuntador a char.

EJEMPLOS

Ejemplo 1

```
main ()
{
      int a,b,c,d,e,f,g;

      printf("Entre siete enteros:\n");
      scanf(" %d %i %i %i %o %u %x", &a, &b, &c, &d, &e, &f, &g);
      printf("La entrada desplegada como enteros decimales es:\n");
      printf(" %d %d %d %d %d %d %d\n", a, b, c, d, e, f, g);
      system("PAUSE");
}
```

lee e imprime lo siguiente:

```
Entre siete enteros:
-70 -70 070 0x70 70 70 70
La entrada desplegada como enteros decimales es:
-70 -70 56 112 56 70 112
```

El primer $\%d$ lee -70.
El primer $\%i$ lee -70.
El segundo $\%i$ lee 070 que es 70 octal que es 56 decimal.
El tercer $\%i$ lee 0x70 que es 70 hexadecimal que es 112 decimal.
El $\%o$ lee 70 octal que es 56 decimal.
El $\%u$ lee 70 sin signo que es 70 decimal.
El $\%x$ lee 70 hexadecimal que es 112 decimal.

Ejemplo 2

```
main ()
{
      float a,b,c;

      printf("Entre tres numeros punto flotante:\n");
      scanf(" %e %f %g", &a, &b, &c);
      printf("Los numeros en notacion punto flotante son:\n");
      printf("%f\n %f\n %f\n", a, b, c);
      system("PAUSE");
}
```

lee e imprime lo siguiente:

```
Entre tres numeros punto flotante:
1.27987 1.27987e+003 3.38476e-006
Los numeros en notacion punto flotante son:
1.279870
1279.869995
0.000003
```

En la especificación de conversión scanf se puede utilizar un ancho de campo para leer un número especifico de caracteres a partir del flujo de entrada. Por ejemplo:

```
main ()
{
      int x, y;
      printf("Entre un entero de 6 digitos\n");
      scanf("%2d %d", &x, &y);
      printf("Los enteros ingresados son %d y %d\n",x,y);
      system("PAUSE");
}
```

La entrada/salida de este programa es:

```
Entre un entero de 6 digitos
123456
Los enteros ingresados son 12 y 3456
```

3.1. Cómo saltear caracteres de la entrada

A menudo es necesario omitir ciertos caracteres del flujo de entrada. Por ejemplo una fecha podría ser introducida como

7-9-91

Queremos guardar el dia, mes, año y saltear los guiones de la entrada. Para eliminar caracteres innecesarios los incluimos en la cadena de control de formato de scanf junto con *. Por ejemplo, leemos la fecha con:

scanf(" %d %*c %d %*c %d", &dia, &mes, &anio);

El caracter de supresión de asignación le permite a scanf leer cualquier tipo de datos a partir de la entrada y descartarlos sin asignarlos a una variable. Por ejemplo la instrucción:

scanf(" %2d %f %*d", &i, &x);

con la entrada:

56789 0123 a

asignará 56 a i, 789.0 a x y saltea 0123. La proxima llamada a una instrucción de entrada leerá la letra 'a'.

3.2. Ancho de campo en la lectura de reales

Podemos asociar un ancho de campo a los valores reales leidos. El ancho abarca la parte entera, el punto y los decimales. Por ejemplo:

```
scanf(" %4f%2f, &f1, &f2);
printf(" %f %f", f1, f2);
```

cuando introduzco la entrada:

12.34 12.0

imprimirá

12.300000 4.000000

idem si utilizo e, E, g o G.