UNIVERSIDAD NACIONAL DE LOJA

NOMBRE:

JUAN DIEGO ENCALADA

CURSO:

1ERO A

ASIGNATURA:

TEORIA DE LA INFORMACIÓN

LOJA – ECUADOR

SCANF:

La función scanf( ): lee datos de la corriente de entrada estándar stdin en las ubicaciones dadas por cada entrada en *lista-argumentos*. Cada *argumento* debe ser un puntero a una variable con un tipo que corresponda a un especificador de tipo en *serie-formato*. La *serie-formato* controla la interpretación de los campos de entrada y es una serie de caracteres de varios bytes que empieza y termina en su estado de desplazamiento inicial.

lee *serie-formato* de izquierda a derecha. Se espera que los caracteres fuera de las especificaciones de formato coincidan con la secuencia de caracteres en stdin; los caracteres coincidentes en stdin se exploran pero no se almacenan. Si un carácter de stdin entra en conflicto con *serie-formato*, scanf() finaliza. El carácter conflictivo se deja en stdin como si no se hubiera leído.

Cuando se encuentra la primera especificación de formato, el valor del primer campo de entrada se convierte de acuerdo con la especificación de formato y se almacena en la ubicación especificada por la primera entrada en *lista-argumentos*. La segunda especificación de formato convierte el segundo campo de entrada y lo almacena en la segunda entrada de *lista-argumentos*, y así sucesivamente hasta el final de *serie-formato*.

Un campo de entrada se define como todos los caracteres hasta el primer carácter de espacio en blanco (espacio, tabulador o línea nueva), hasta el primer carácter que no se puede convertir de acuerdo con la especificación de formato o hasta que se alcance el campo *ancho* , lo que ocurra primero. Si hay demasiados argumentos para las especificaciones de formato, se ignoran los argumentos adicionales. Los resultados no están definidos si no hay suficientes argumentos para las especificaciones de formato.

Cada campo de la especificación de formato es un carácter único o un número que indica una opción de formato determinada. El carácter *tipo* , que aparece después del último campo de formato opcional, determina si el campo de entrada se interpreta como un carácter, una serie o un número. La especificación de formato más simple contiene sólo el signo de porcentaje y un carácter de *tipo* (por ejemplo, %s).

Cada campo de la especificación de formato se describe en detalle a continuación. Si un signo de porcentaje (%) va seguido de un carácter que no tiene ningún significado como carácter de control de formato, el comportamiento no está definido. Una excepción a este comportamiento es %%. Para especificar un carácter de signo de porcentaje, utilice %%.

Las restricciones siguientes se aplican a la impresión y exploración de punteros:

- Si se imprime un puntero y se vuelve a escanear desde el mismo grupo de activación, el puntero de retroceso escaneado se comparará con el puntero que se imprime.
- Si una función de la familia scanf() explora un puntero que ha imprimido un grupo de activación diferente, la función de la familia scanf() establecerá el puntero en NULL.

EJEMPLO:

Este ejemplo explora varios tipos de datos.

```
#include <stdio.h>

int main(void)
{
   int i;
   float fp;
   char c, s[81];

   printf("Enter an integer, a real number, a character "
          "and a string : \n");
   if (scanf("%d %f %c %s", &i, &fp, &c, s) != 4)
      printf("Not all fields were assigned\n");
   else
   {
      printf("integer = %d\n", i);
      printf("real number = %f\n", fp);
      printf("character = %c\n", c);
      printf("string = %s\n",s);
   }
}

/*****************  If input is: 12 2.5 a yes,  *******************
**************  then output should be similar to:  ****************

Enter an integer, a real number, a character and a string :
integer = 12
real number = 2.500000
character = a
string = yes
*/
```

Este ejemplo convierte un entero hexadecimal en un entero decimal. El bucle while finaliza si el valor de entrada no es un entero hexadecimal.

```
#include <stdio.h>

int main(void)
{
   int number;

   printf("Enter a hexadecimal number or anything else to quit:\n");
   while (scanf("%x",&number))
   {
      printf("Hexadecimal Number = %x\n",number);
      printf("Decimal Number     = %d\n",number);
   }
}

/***************  If input is: 0x231 0xf5e 0x1 q,  ****************
*****************  then output should be similar to:  **************

Enter a hexadecimal number or anything else to quit:
Hexadecimal Number = 231
Decimal Number     = 561
Hexadecimal Number = f5e
Decimal Number     = 3934
Hexadecimal Number = 1
Decimal Number     = 1
*/
```

Este ejemplo lee de `stdin` y asigna datos utilizando la serie de formato posicional alternativa.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int i;
   char s[20];
   float f;

   scanf("%2$s %3$f %1$d",&i, s, &f);

   printf("The data read was \n%i\n%s\n%f\n,i,s,f);

   return 0;
}

/***************  If the input is : test 0.2 100  *****************
**************  then the output will be similar to: ***************

The data read was
100
test
0.20000
*/
 ---------------------------------------------------------------------
```

Este ejemplo lee una serie de caracteres de varios bytes en una serie Unicode amplia. El ejemplo se puede compilar con LOCALETYPE (*LOCALEUCS2) o LOCALETYPE (*LOCALEUTF).

```
#include <locale.h>
#include <stdio.h>
#include <wchar.h>

void main(void)
{
   wchar_t uString[20];

   setlocale(LC_UNI_ALL, "");
   scanf("Enter a string %ls",uString);

   printf("String read was %ls\n",uString);
}

/* if the input is : ABC
   then the output will be similiar to:

   String read was ABC

 */
```

PRINTF:

La función **printf()** combina la capacidad de realizar un seguimiento de los datos, al igual que la función **trace()**, con la capacidad de generar los datos y otro texto con el formato específico que describa. La función **printf()** indica a DTrace que debe realizar un seguimiento de los datos asociados a cada argumento ubicado detrás del primer argumento y, a continuación, debe aplicar un formato a los resultados mediante las reglas descritas en el primer argumento **printf()**, conocido como *cadena de formato*.

La cadena de formato es una cadena normal que contiene una serie de conversiones de formato; cada una de ellas comienza con el carácter % y describe el formato que debe aplicarse al argumento correspondiente. La primera conversión de la cadena de formato

se corresponde con el segundo argumento **printf()**, la segunda conversión con el tercer argumento, y así sucesivamente. Todo el texto que se encuentra entre una conversión y otra se imprime de forma literal. El carácter que se incluye después del carácter de conversión % describe el formato que se utilizará para el argumento correspondiente.

A diferencia de printf(3C), DTrace **printf()** es una función incorporada que reconoce el compilador del lenguaje D. Este compilador proporciona varios servicios de utilidad para la función **printf()** de DTrace que no se encuentran en la función **printf()** de la biblioteca de C:

- El compilador de D compara los argumentos con las conversiones en la cadena de formato. Si un tipo de argumento es incompatible con la conversión de formato, el compilador de D proporciona un mensaje de error que describe el problema.
- No es necesario utilizar prefijos de tamaño con las conversiones de formato **printf()** para el compilador de D. Para la rutina **printf()** de C, es necesario indicar el tamaño de los argumentos mediante la adición de prefijos como, por ejemplo, %ld para long o %lld para long long. El compilador de D conoce el tamaño y el tipo de los argumentos, por lo que no es necesario utilizar prefijos en las instrucciones **printf()** del lenguaje D.
- DTrace proporciona caracteres de formato adicionales que resultan de utilidad para las funciones de depuración y supervisión. Por ejemplo, la conversión de formato %a puede utilizarse para imprimir un puntero en forma de nombre de símbolo y desplazamiento.

Para implementar estas funciones, la cadena de formato de la función **printf()** debe especificarse como constante de cadena en el programa D. Las cadenas de formato no pueden ser variables dinámicas del tipo string.

Ejemplo de formato: Se incluyen como parte de la cadena que se pasa como primer parámetro a printf ofrece muchas posibilidades. A continuación se muestran algunos ejemplos:

- Especificar el ancho mínimo: Podemos poner un entero entre el símbolo de porcentaje (%) y el especificador de formato, para indicar que la salida alcance un ancho mínimo. Por ejemplo, %10f asegura que la salida va a tener al menos 10 espacios de ancho. Esto es útil cuando se van a imprimir datos en forma de columnas. Por ejemplo, si tenemos:

```
int num = 12;
int num2 = 12345;
printf("%d\n",num2);
printf("%5d\n",num);
```

se imprime:

```
12345
   12
```

Ejemplos:

Este ejemplo imprime datos en diversos formatos.

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

int main(void)
{
    char ch = 'h', *string = "computer";
    int count = 234, hex = 0x10, oct = 010, dec = 10;
    double fp = 251.7366;
    wchar_t wc = (wchar_t)0x0058;
    wchar_t ws[4];

    printf("1234567890123%n4567890123456789\n\n", &count);
    printf("Value of count should be 13; count = %d\n\n", count);
    printf("%10c%5c\n", ch, ch);
    printf("%25s\n%25.4s\n\n", string, string);
    printf("%f    %.2f    %e    %E\n\n", fp, fp, fp, fp);
    printf("%i    %i     %i\n\n", hex, oct, dec);
}
/*****************  Output should be similar to:  *****************

234   +234    000234     EA    ea     352

12345678901234567890123456789

Value of count should be 13; count = 13


  h    h
          computer
              comp

251.736600    251.74    2.517366e+02    2.517366E+02

16    8     10

*********************************************************************/
```

Ejemplo que utiliza `printf()`

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
/* This program is compile LOCALETYPE(*LOCALE)  and              */
/* SYSIFCOPT(*IFSIO)    */
int main(void)
{
    wchar_t wc = (wchar_t)0x00C4;     /*  D */
    wchar_t ws[4];
    ws[0] = (wchar_t)0x00C1;        /* A   */
    ws[1] = (wchar_t)0x00C2;        /* B   */
    ws[2] = (wchar_t)0x00C3;        /* C   */
    ws[3] = (wchar_t)0x0000;
    /* The output displayed is CCSID 37  */
    printf("%lc   %ls\n\n",wc,ws);

    /* Now let's try a mixed-byte CCSID example  */
    /* You would need a device that can handle mixed bytes to  */
    /* display this correctly.                          */

    setlocale(LC_ALL,
    "/QSYS.LIB/JA_JP.LOCALE"); /* a CCSID 5026 locale */

    /* big A means an A that takes up 2 bytes on the screen   */
    /* It will look bigger than single byte A                 */

    ws[0] = (wchar_t)0x42C1;        /* big A   */
    ws[1] = (wchar_t)0x42C2;        /* big B   */
    ws[2] = (wchar_t)0x42C3;        /* big C   */
    ws[3] = (wchar_t)0x0000;
    wc = 0x42F1;                    /* big 1   */

    printf("%lc   %ls\n\n",wc,ws);

    /* The output of this printf is not shown below and it   */
    /* will differ depending on the device you display it on,*/
    /* but if you looked at the string in hex it would look  */
    /* like this:  0E42F10F404040400E42C142C242C30F          */
    /* 0E is shift out, 0F is shift in, and 42F1 is the      */
    /* big 1 in CCSID   5026  */
```

```
    printf("%lc   %.4ls\n\n",wc,ws);

    /* The output of this printf is not shown below either. */
    /* The hex would look like:                             */
    /* 0E42F10F404040400E42C10F                             */
    /* Since the precision is in bytes we only get 4 bytes  */
    /* of the string.         */

    printf("%lc   %#.2ls\n\n",wc,ws);

    /* The output of this printf is not shown below either. */
    /* The hex would look like:                             */
    /* 0E42F10F404040400E42C142C20F                         */
    /* The # means precision is in characters regardless   */
    /* of size.  So we get 2 characters of the string.      */
}
/*****************  Output should be similar to:  *****************


D     ABC


*********************************************************************/
```

Ejemplo que utiliza `printf()`

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
/* This program is compiled with LOCALETYPE(*LOCALEUCS2) and             */
/* SYSIFCOPT(*IFSIO)                                                  */
/* We will assume the locale setting is the same as the CCSID of the */
/* job.  We will also assume any files involved have a CCSID of      */
/* 65535 (no convert).  This way if printf goes to the screen or     */
/* a file the output will be the same.                               */
int main(void)
{
    wchar_t wc = 0x0058;     /* UNICODE X */
    wchar_t ws[4];
    setlocale(LC_ALL,
     "/QSYS.LIB/EN_US.LOCALE"); /* a CCSID 37 locale */
    ws[0] = 0x0041;        /* UNICODE A   */
    ws[1] = (wchar_t)0x0042;        /* UNICODE B   */
    ws[2] = (wchar_t)0x0043;        /* UNICODE C   */
    ws[3] = (wchar_t)0x0000;
    /* The output displayed is CCSID 37  */
    printf("%lc   %ls\n\n",wc,ws);
    printf("%lc   %.2ls\n\n",wc,ws);

    /* Now let's try a mixed-byte CCSID example  */
    /* You would need a device that can handle mixed bytes to  */
    /* display this correctly.                          */

    setlocale(LC_ALL,
    "/QSYS.LIB/JA_JP.LOCALE");/* a CCSID 5026 locale */

    /* big A means an A that takes up 2 bytes on the screen   */
    /* It will look bigger then single byte A                 */
    ws[0] = (wchar_t)0xFF21;        /* UNICODE big A   */
    ws[1] = (wchar_t)0xFF22;        /* UNICODE big B   */
    ws[2] = (wchar_t)0xFF23;        /* UNICODE big C   */
    ws[3] = (wchar_t)0x0000;
    wc = 0xff11;                    /* UNICODE big 1   */

    printf("%lc   %ls\n\n",wc,ws);
```

```
    /* The output of this printf is not shown below and it   */
    /* will differ depending on the device you display it on,*/
    /* but if you looked at the string in hex it would look  */
    /* like this:  0E42F10F404040400E42C142C242C30F          */
    /* 0E is shift out, 0F is shift in, and 42F1 is the      */
    /* big 1 in CCSID   5026  */

    printf("%lc   %.4ls\n\n",wc,ws);

    /* The output of this printf is not shown below either. */
    /* The hex would look like:                             */
    /* 0E42F10F404040400E42C10F                             */
    /* Since the precision is in bytes we only get 4 bytes  */
    /* of the string.         */

    printf("%lc   %#.2ls\n\n",wc,ws);

    /* The output of this printf is not shown below either. */
    /* The hex would look like:                             */
    /* 0E42F10F404040400E42C142C20F                         */
    /* The # means precision is in characters reguardless   */
    /* of size.  So we get 2 characters of the string.      */
}
/*****************  Output should be similar to:  *****************


X     ABC

X     AB

*********************************************************************/
```