Instrucciones de lectura/escritura

Índice

- Lectura – Escritura de datos.
- Funciones de Lectura/Escritura de datos.
- getchar().
- putchar().
- scanf().
- printf().
- Entrada/Salida de cadenas de caracteres.
- Redireccionamiento de la entrada/salida en MS-DOS.

Lectura – Escritura de datos

- C dispone de una colección de funciones de biblioteca para la entrada/lectura y salida/escritura entre el ordenador y los periféricos E/S.
- Se encuentran declaradas en la librería del sistema stdio.h. Para ello se coloca en la cabecera de los programas la directiva `#include <stdio.h>`
- Se considera como unidad de entrada/lectura el *teclado*  y como unidad de salida/escritura la *pantalla* del ordenador. 

Funciones de Lectura/Escritura de datos

- Transferencia de caracteres sueltos
  - **getchar** - lectura
  - **putchar** - escritura
- Transferencia de caracteres, valores numéricos y cadenas de caracteres
  - **scanf** - lectura con formato
  - **printf** - escritura con formato
- Transferencia de cadenas de caracteres
  - **gets** - lectura
  - **puts** - escritura

Entrada de un carácter – función getchar()

- Devuelve un carácter leído de la entrada estándar (teclado).
- No requiere argumentos. Es necesario colocar los paréntesis vacíos.
- Uso:

```
char c;
. . . .
c = getchar();
```

- Si se encuentra un final de fichero, getchar devuelve el carácter EOF (End Of File).
- Se puede utilizar para leer cadenas de caracteres (uno a uno).

Salida de un carácter – función putchar()

- Imprime en la salida estándar (pantalla) un carácter que se da como argumento.
- Uso:

```
char c;
. . . .
putchar(c);
```

- Se puede utilizar para imprimir una cadena de caracteres imprimiendo dentro de un bucle cada carácter de la cadena.

Ejemplo: I/O con getchar/putchar

```
/*****************************************************************\
* Programa: IO_get_putchar.c                                      *
* Descripción: Lee caracteres mientras sea diferente de EOF (Ctrl-Z)     *
*              y los imprime en mayusculas                        *
* Autor: Pedro Corcuera                                           *
* Revisión: 1.0 2/02/2008                                         *
\*****************************************************************/

#include <stdio.h>

main()
{
 char c;

 /* lee carácter mientras no sea EOF */
 while ((c = getchar())!= EOF )
   if (c >= 'a' && c <= 'z')
      putchar(c - 32);     /* conversion a mayuscula 'a' - 'A' = 32 */


}
```

Entrada de datos – función scanf()

- Función que permite la lectura de cualquier tipo de dato especificado por un formato.
- Usa llamada por *referencia* en los argumentos de las variables leídas.
- Devuelve el número de variables emparejadas correctamente durante la lectura o EOF si se encuentra un final de fichero.
- Sintaxis:

```
scanf(cad_control, arg1, arg2,..,argn);
```

función scanf() – especificación de formato

- La cadena de control contiene información sobre el formato de entrada los argumentos que le siguen.
- El formato se especifica por el carácter % y un *carácter de conversión*.
- Es recomendable iniciar con un espacio en blanco la cadena de control y separar con espacios en blanco los grupos de formato.
- Los argumentos deben coincidir con el número de grupos de formato y deben estar precedidos del carácter **&**, excepto las cadenas (arrays) de caracteres.

función scanf() – caracteres de conversión

Carácter de conversión	Significado
c	carácter
d	entero
e, f ,g	float (real)
h	entero corto
i	entero, octal o hexadecimal
o	octal
s	cadena de caracteres
u	entero sin signo
x	hexadecimal
[. . .]	cadena que incluye caracteres restringidos

Uso de scanf()

Ejemplo

```
int a,*pa;
float x;
char c;
char str[82];

scanf("%d",&a); /* Lee un entero y lo almacena en a */

scanf(" %f %c",&x,&c); /* Lee x y c */

pa=&a; scanf(" %d",pa); /* OK. Lee a */

scanf(" %s",str);  /*Lee hasta un blanco o \n */

scanf(" %[^\n]",str); /* Lee toda la línea */
```

Ejemplo: lectura con función scanf()

```
/*****************************************************************\
* Programa: IO_scanf.c                                            *
* Descripción: Lectura de variables con la función scanf()        *
* Autor: Pedro Corcuera                                           *
* Revisión: 1.0 2/02/2008                                         *
\*****************************************************************/

#include <stdio.h>

main()
{
 int cantidad, a, b;
 float precio;
 char c, articulo[80];     /* articulo[80] - array/cadena de caracteres */

 scanf(" %d %f", &cantidad, &precio);
 scanf("%3d  %3d  %c", &a, &b, &c);  /* lectura rigida de 3 cifras enteras */
 scanf(" %s", articulo);
 scanf("%[ A-Z]", articulo);  /* lectura de caracteres en mayuscula */
 scanf(" %[^\n]", articulo);  /* lee mientras NO COINCIDA(^) con \n (INTRO) */
 scanf(" %s %*d %f", articulo, &cantidad, &precio); /* * lee sin asignar */

}
```

Salida de datos – función printf()

- Sirve para escribir cualquier combinación de valores numéricos, caracteres y cadenas de caracteres.
- Transfiere datos de la memoria del ordenador al dispositivo de salida estándar (pantalla).
- Devuelve un entero que representa el número de caracteres que se imprimen.
- Sintaxis:

```
printf(cad_ctrl, arg1, arg2,...,argn);
```

función printf() – especificación de formato

- La cadena de control contiene dos tipos de objetos: caracteres ordinarios, que se imprimen directamente, y especificadores de conversión que indican el formato de salida de los argumentos que siguen a la cadena de control.
- El formato se especifica por el carácter % y un *carácter de conversión* que indica el tipo de dato correspondiente.
- Los argumentos pueden ser constantes, variables, arrays, expresiones o funciones.
- Los argumentos deben coincidir con el número de especificadores de formato.

función printf() – caracteres de conversión

Carácter de conversión	Significado
c	carácter
d, i	entero con signo
o	octal
x, X	hexadecimal
u	entero sin signo
f	float , double (6 decimales)
e, E	float, double en notación científica
g, G	float, double (general, usa f o e)
lf	double
s	cadena de caracteres
p	puntero

función printf() – indicadores

- Entre el % y el carácter de conversión pueden haber los siguientes indicadores:

Indicador	Significado
–	ajuste a la izquierda del campo
+	datos numéricos c/signo
0	ceros en lugar de blancos (sólo en datos ajust.derec.)
Blanco	espacio blanco precede al número
#	0 precede dato octal, 0x precede dato hexadec., en e, f y g todos los números con un punto, aunque sea entero
[Núm][.][Núm]	Núm. Número que especifica el ancho mínimo del campo
h, l	h para short y l para long
*	ancho en tiempo de ejecución

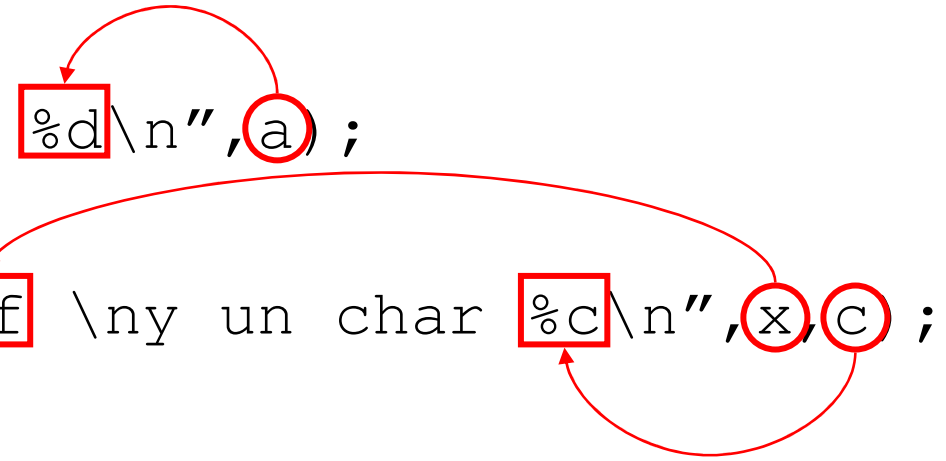
Uso de printf()

Ejemplo:

```
int a=3;
float x=23.0;
char c='A';
printf("Hola mundo!!\n");

printf("Un entero %d\n",a);

printf("Un real %f \ny un char %c\n",x,c);
```



Ejemplo: escritura con función printf()

```
/*****************************************************************\
* Programa: IO_printf.c                                          *
* Descripción: Escritura de variables con la función printf()    *
* Autor: Pedro Corcuera                                          *
* Revisión: 1.0 2/02/2008                                        *
\*****************************************************************/

#include <stdio.h>

main()
{
  char c = 'A';
  char cadena[80] = "Hola este es un ejemplo de cadena";
  int i=37, j=-37;
  float x = 123.456,y;
  double z = 123.456;

  printf("\n\nEjemplos de caracteres y enteros\n");
  printf("12345678901234567890123456789012345678901234567890\n");
  printf("%%c  \t %c \n",c);
  printf("%%d  \t %d \n",c);
  printf("%%s  \t %s \n", cadena);
```

Ejemplo: escritura con función printf()

```
printf("\n\nEjemplos de enteros\n");
printf("12345678901234567890123456789012345678901234567890\n");
printf("%%d %%d \t\t %d %d\n",i,j);
printf("%%.4d %%.4d \t %.4d %.4d\n",i,j);
printf("%%11d %%11d \t %11d %11d\n",i,j);
printf("%%011d %%011d \t %011d %011d\n",i,j);
printf("%% 011d %% 011d \t % 011d % 011d\n",i,j);
printf("%%+11d %%+11d \t %+11d %+11d\n",i,j);
printf("%%+011d %%+011d \t %+011d %+011d\n",i,j);
printf("%%-11d %%-11d \t %-11d %-11d\n",i,j);
printf("%%- 11d %%- 11d \t %- 11d %- 11d\n",i,j);
printf("%%-+11d %%-+11d \t %-+11d %-+11d\n",i,j);
printf("%%11.4d %%11.4d \t %11.4d %11.4d\n",i,j);
printf("%%-11.4d %%-11.4d \t %-11.4d %-11.4d\n",i,j);
printf("%%- 11.4d %%- 11.4d \t %- 11.4d %- 11.4d\n",i,j);
printf("%%-+11.4d %%-+11.4d \t %-+11.4d %-+11.4d\n",i,j);
printf("%%011d %%011d \t %011d %011d\n",i,j);
```

Ejemplo: escritura con función printf()

```
printf("\n\nEjemplos de octal y hexadecimal\n");
printf("12345678901234567890123456789012345678901234567890\n");
printf("%%o %%x \t %o %x\n",i,j);
printf("%%.4o %%.4x \t %.4o %.4x\n",i,j);
printf("%%11o %%11x \t %11o %11x\n",i,j);
printf("%%011o %%011x \t %011o %011x\n",i,j);
printf("%%#011o %%#011x \t %#011o %#011x\n",i,j);
printf("%%#11o %%#11x \t %#11o %#11x\n",i,j);
printf("%%-11o %%-11x \t %-11o %-11x\n",i,j);
printf("%%-#11o %%-#11x \t %-#11o %-#11x\n",i,j);
printf("%%11.4o %%11.4x \t %11.4o %11.4x\n",i,j);
printf("%%-11.4o %%-11.4x \t %-11.4o %-11.4x\n",i,j);
printf("%%-#11.4o %%-#11.4x \t %-#11.4o %-#11.4x\n",i,j);

printf("\n\nEjemplos de reales\n");
printf("12345678901234567890123456789012345678901234567890\n");
printf("%%f %%.3f %%.1f \t\t %f %.3f %.1f\n",x,x,x);
printf("%%e %%.5e %%.3e \t\t %e %.5e %.3e\n",x,x,x);
printf("%%7f %%7.3f %%7.1f \t %7f %7.3f %7.1f\n",x,x,x);
printf("%%12e %%12.5e %%12.3e \t %12e %12.5e %12.3e\n",x,x,x);
printf("%%f %%.3f %%.1f \t\t %f %.3f %.1f\n",z,z,z);
```

Ejemplo: escritura con función printf()

```
x=37.0;
y=-37.0;
printf("\n12345678901234567890123456789012345678901234567890123456789012345678901234567890\n");
printf("%%f %%f \t %f %f\n",x,y);
printf("%%.3f %%.3f \t %.3f %.3f\n",x,y);
printf("%%11f %%11f \t %11f %11f\n",x,y);
printf("%%011f %%011f \t %011f %011f\n",x,y);
printf("%% 011f %% 011f \t % 011f % 011f\n",x,y);
printf("%%+11f %%+11f \t %+11f %+11f\n",x,y);
printf("%%+011f %%+011f \t %+011f %+011f\n",x,y);
printf("%%-11f %%-11f \t %-11f %-11f\n",x,y);
printf("%%- 11f %%- 11f \t %- 11f %- 11f\n",x,y);
printf("%%-+11f %%-+11f \t %-+11f %-+11f\n",x,y);
printf("%%11.3f %%11.3f \t %11.3f %11.3f\n",x,y);
printf("%%11.0f %%11.0f \t %11.0f %11.0f\n",x,y);
printf("%%#11.0f %%#11.0f \t %#11.0f %#11.0f\n",x,y);
printf("%%-11.3f %%-11.3f \t %-11.3f %-11.3f\n",x,y);
printf("%%- 11.3f %%- 11.3f \t %- 11.3f %- 11.3f\n",x,y);
printf("%%-+11.3f %%-+11.3f \t %-+11.3f %-+11.3f\n",x,y);
```

Ejemplo: escritura con función printf()

```
  printf("\n12345678901234567890123456789012345678901234567890\n");
  printf("%%e %%e \t %e %e\n",x,y);
  printf("%%.3e %%.3e \t %.3e %.3e\n",x,y);
  printf("%%15e %%15e \t %15e %15e\n",x,y);
  printf("%%015e %%015e \t %015e %015e\n",x,y);
  printf("%% 015e %% 015e \t % 015e % 015e\n",x,y);
  printf("%%+15e %%+15e \t %+15e %+15e\n",x,y);
  printf("%%+015e %%+015e \t %+015e %+015e\n",x,y);
  printf("%%-15e %%-15e \t %-15e %-15e\n",x,y);
  printf("%%- 15e %%- 15e \t %- 15e %- 15e\n",x,y);
  printf("%%-+15e %%-+15e \t %-+15e %-+15e\n",x,y);
  printf("%%15.3e %%15.3e \t %15.3e %15.3e\n",x,y);
  printf("%%15.0e %%15.0e \t %15.0e %15.0e\n",x,y);
  printf("%%#15.0e %%#15.0e \t %#15.0e %#15.0e\n",x,y);
  printf("%%-15.3e %%-15.3e \t %-15.3e %-15.3e\n",x,y);
  printf("%%- 15.3e %%- 15.3e \t %- 15.3e %- 15.3e\n",x,y);
  printf("%%-+15.3e %%-+15.3e \t %-+15.3e %-+15.3e\n",x,y);

  printf("\n\nEjemplos de especificacion de ancho en run-time\n");
  printf("%%.*d \t %.*d\n",4,37);
  printf("%%*d \t %*d\n",11,37);
  printf("%%*.*f \t %*.*f\n",11,4,37.543);

}
```

Entrada/Salida de cadenas de caracteres

- Para la lectura/escritura de cadenas de caracteres hay dos funciones especiales:
- gets(cadena) – lectura del array de caracteres.
- puts(cadena) – escritura del array de caracteres.
- Ejemplo:

```
#include <stdio.h>

main()
{
 char linea[80];

 gets(linea);
 printf(linea);
}
```

Redireccionamiento de Entrada/Salida en MSDOS

- Cualquier comando (programa) de MSDOS necesita recibir información de algún "lugar" y enviar los resultados del procesamiento a algún "lugar", así como los mensajes de error. Estos "lugares" se llaman, respectivamente, ESTÁNDAR INPUT, ESTÁNDAR OUTPUT y ESTÁNDAR ERROR.
- El e*stándar input* se refiere al medio desde el cual el comando recibe la información. De forma similar, el *estándar output* se refiere al lugar que el comando envia la salida. Cuando se redireccionan los datos el comando recibe o envía la información desde otra fuente.
- El *estándar error* se refiere al medio al que se mandan los mensajes de los errores que se cometen al ejecutar un comando.
- Normalmente (aunque depende de cada comando), el estándar input es el *teclado*, y el estándar output y el estándar error es la *pantalla*.

Redireccionamiento de la salida en MSDOS

- El símbolo para redireccionar la salida es: **>** y se utiliza de la siguiente forma:

  comando > nombre_fichero

- Ejm: dir > salida_dir.txt
- Se puede *añadir* la salida de un comando al final de un fichero ya existente sin borrar su contenido. El símbolo que se utiliza para ello es **>>** y se utiliza de la siguiente forma:

  comando >> nombre_fichero

- Ejm: ls > salida_dir.txt

Redireccionamiento de la entrada en MSDOS

- El símbolo para redireccionar la entrada es **<** y se utiliza de la siguiente manera:

  comando < nombre_fichero

- Ejm: promedio < datos.txt
- Se puede concatenar el redireccionamiento de la entrada y la salida en una sola línea de comando.
- Ejm: promedio < datos.txt > resultado.txt

Instrucciones de control

Índice

- Tipos de instrucciones de control.
- Instrucciones secuenciales.
- Bloques.
- if – else, else - if.
- switch.
- while.
- for.
- do - while.
- break, continue, goto.

Tipos de instrucciones de control

- Las instrucciones de control de flujo de un lenguaje de programación especifican el orden de ejecución de las instrucciones de un programa.
- Secuenciales: Instrucciones
  Bloques
  Instrucción nula
- Selección: if - else
  else - if
  switch
- Repetitivas: while
  for
  do - while
- Control: break
  continue
  return
  goto

Intrucciones secuenciales

- Una instrucción es una expresión terminada con un punto y coma (;). El punto y coma se considera como un terminador de instrucción.
- Sintaxis:

  expresión ;

- *Instrucción nula*: Consiste sólo en un punto y coma y se usa en instrucciones que requieren de una instrucción ejecutable como do, for, if y while.

Bloques

- Un bloque es un grupo de declaraciones e instrucciones encerradas entre llaves { }.
- La *instrucción compuesta* es sintácticamente equivalente a una instrucción simple.
- No se coloca punto y coma después de la llave derecha.
- Sintaxis:

```
{
  [declaración(es)]
  instrucción(es)
  . . .
}
```

if - else

- Sirve para tomar decisiones dependiendo del valor de una expresión lógica:
  - Si la expresión es verdadera (diferente de cero) se ejecuta la instrucción que sigue al if.
  - Si la expresión es falsa (igual a cero) se ejecuta la instrucción que sigue al else. Si no la parte else se omite, continua con la instrucción que sigue al if.
- En caso de una secuencia de if anidados, la parte else se asocia con el if sin else más próximo.
- Sintaxis:

```
if (expresión_lógica)
        instrucción1
[ else
        instrucción2]
```

if - else

```
#include <stdio.h>
/* ej. instr. if - else */
int main()
{
   int a=3,b;

   if(a>2)
   {
       b=100+a;
       printf(“parte if”);
   }
   else
    printf(“parte else”);
}
```

Diagrama de flujo

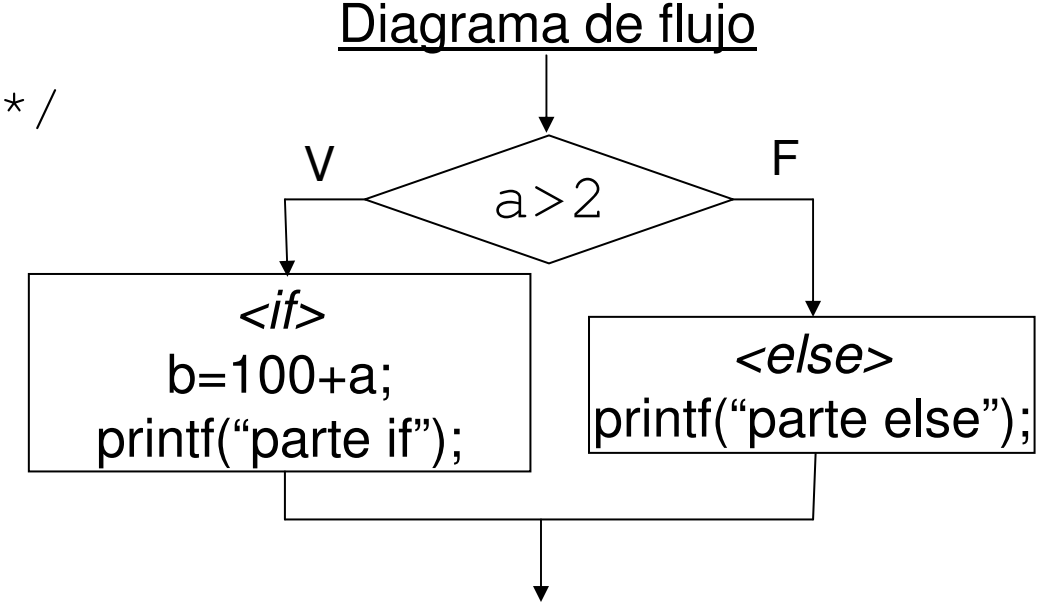
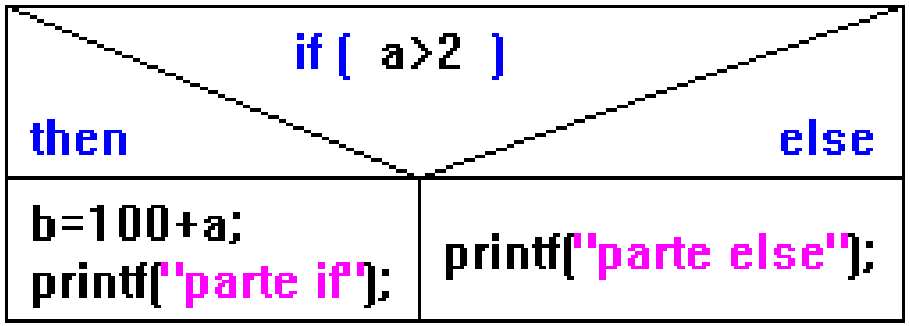


Diagrama de cajas



Ejemplo: if - else

```
/*****************************************************************\
* Programa: pr_ifelse.c                                          *
* Descripción: Ejemplo de if - else                              *
* Autor: Pedro Corcuera                                          *
* Revisión: 1.0 2/02/2008                                        *
\*****************************************************************/

#include <stdio.h>
#include <math.h>    /* Cabecera para funciones matemáticas: sqrt() */

int main( void )
{
  double num;

  printf( "Ingresa un numero no-negativo: " );
  scanf( "%lf", &num);
  if (num < 0)
     printf( "Error: Numero negativo.\n" );
  else
     printf( "La raiz cuadrada es: %f\n", sqrt( num ));

  return 0;
}
```

else - if

- Sirve para tomar decisiones múltiples. Las expresiones se evalúan en orden; si alguna es cierta, se ejecuta la sentencia asociada. Si ninguna expresión es verdadera se puede usar un else final para ejecutar alguna instrucción por defecto.
- Sintaxis:

```
if (expresión1)
        instrucción1
else if (expresión2)
…
else if (expresión_n)
        instrucción_n
[else]
        instrucción_por_defecto
```

Ejemplo: else - if

```
/*****************************************************************\
* Programa: pr_elseif.c                                           *
* Descripción: Prog. que imprime la nota literal para un valor numerico  *
* Autor: Pedro Corcuera                                           *
* Revisión: 1.0 2/02/2008                                         *
\*****************************************************************/
#include <stdio.h>

main()
{
  float nota;

  printf("Introduce la nota: "); scanf (" %f", &nota);
  if (nota == 10)
    printf("Matricula de Honor\n");
  else if (nota >= 9.0)
    printf("Sobresaliente\n");
  else if (nota >= 7.0)
    printf("Notable\n");
  else if (nota >= 5.0)
    printf("Aprobado\n");
  else
    printf("Suspenso\n");
}
```

switch

- Sirve para tomar seleccionar un grupo de instrucciones entre varios grupos disponibles.
- Sintaxis:

```
switch (expresión) {
    [case expr-cte1 : ] [instruc1]
    [case expr-cte2 : ] [instruc2]
    . . . . .
    [case expr-cte_n : ] [instruc_n]
    [default : ] [instruc_por_defecto]
}
```

switch

- switch evalúa la expresión entera y compara su valor con todos los casos, si alguno corresponde con ese valor , la ejecución se transfiere a la instrucción que sigue al case y continúa hasta el final del switch.
- La opción default es opcional y se ejecuta si no se satisfacen ninguno de los case.
- La expresión del switch debe ser de tipo **integral** (int, long, short o char).
- Para salir de inmediato de la instrucción switch se puede usar un **break** para continuar con la instrucción que sigue al switch o un return para volver a la función de llamada .

switch

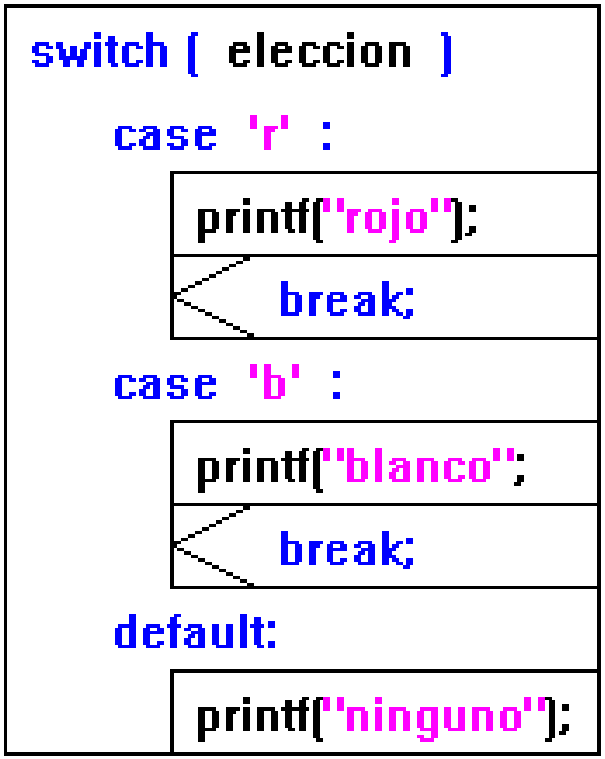
```
switch(ch)
{
  case 'A': printf("A");
            break;
  case 'B':
  case 'C': printf("B o C");
  case 'D': printf("B, C o D");
            break;
  default:  printf("Otra letra");
}
```

switch

```
#include <stdio.h>
/* ej. instr. switch */
int main()
{
   char eleccion;

   switch(eleccion=getchar())
   {
        case ‘r’: case ‘R’:
          printf(“ROJO”);break;
        case ‘b’: case ‘B’:
          printf(“BLANCO”);break;
        case ‘a’: case ‘A’:
          printf(“AZUL”);break;
        default:
          printf(“NINGUNO”);
   }
}
```

Diagrama de cajas



Ejemplo: switch

```
/*****************************************************************\
* Programa: pr_switch.c                                           *
* Descripción: Prog. que detecta teclas numericas y blancos       *
* Autor: Pedro Corcuera                                           *
* Revisión: 1.0 2/02/2008                                         *
\*****************************************************************/
#include <stdio.h>

main()
{
  char c;

  printf("Pulsa una tecla: "); scanf (" %c", &c);
  switch (c) {
    case '0':  case '1':  case '2':  case '3':  case '4': case '5':  case '6':
    case '7':  case '8':  case '9':
        printf ("tecla numerica\n"); break;
    case ' ':  case '\n':  case '\t':
       printf ("blanco\n"); break;
    default: printf ("Otra tecla\n"); break;
    }
}
```

while

- El cuerpo del while se ejecuta mientras el valor de la expresión sea diferente de cero.
- Sintaxis:

```
while (expresión)
        instrucción
```

while

```
#include <stdio.h>

int main()
{
  int i=0,ac=0;

  while( i < 100 )
  {
     printf("%d", i*i);
     ac += i;
     i++;
  }
}
```

Diagrama de flujo

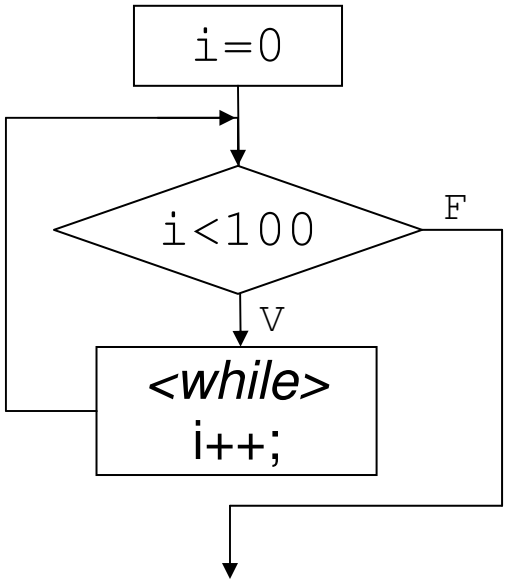
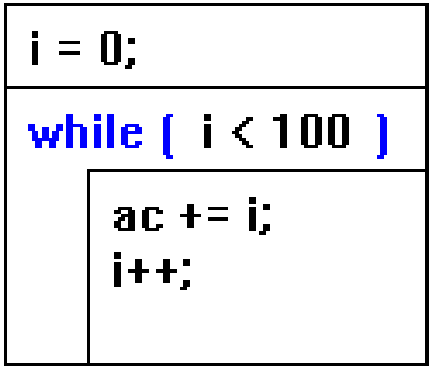


Diagrama de cajas



Ejemplo 1: while

```
/*****************************************************************\
* Programa: f2c_while.c                                         *
* Descripción: Prog. que imprime tabla de conversion Fahrenheit-Celsius  *
* Autor: Pedro Corcuera                                         *
\*****************************************************************/
#include <stdio.h>
#define RANGO_INICIAL     0       /* limite inf. tabla */
#define RANGO_FINAL       100     /* limite sup. tabla */
#define PASO              10      /* tamaño paso */

main()
{
  int fahrenheit;
  double celsius;

  fahrenheit=RANGO_INICIAL;
  while (fahrenheit <= RANGO_FINAL )
  {
    celsius = 5.*(fahrenheit - 32)/9;
    printf("Fahrenheit = %d \t Celsius = %.3f \n", fahrenheit, celsius);
    fahrenheit += PASO;
  }
  return 0;
}
```

Ejemplo 2: while

```
/*****************************************************************\
* Programa: cuenta_pal.c                                          *
* Descripción: Prog. que cuenta número de líneas, palabras y caracteres  *
\*****************************************************************/
#include <stdio.h>
#define EN       1
#define FUERA    0

main()
{
  int c, nl = 0, np = 0, nc = 0, estado = FUERA;  /* inicia contadores */

  while ((c=getchar()) != EOF ) {
     ++nc;                    /* contador de caracteres */
     if (c=='\n') ++nl;       /* si cambio de linea incrementa cont. lineas*/
     if (c==' '||c=='\n'||c=='\t')  /* si es blanco reinicia estado */
        estado = FUERA;
     else if (estado == FUERA) {
         estado = EN; /* dentro de palabra */ ++np; /* cont. de palabras */
     }
  }
  printf("numero de lineas: %d  caracteres: %d  palabras: %d\n",nl, nc, np);
  return 0;
}
```

Ejemplo 3: while

- Calcular el valor de **seno(x)** en los puntos resultantes de dividir el intervalo [0,π] en N subintervalos, con un error menor que un valor dado ε especificados por el usuario.

  Utilizar la expresión:

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \quad \equiv \quad sen(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \quad x \in \Re$$

  se considera que el error cometido es menor que el valor absoluto del último término que se toma.

Ejemplo 3: observaciones previas

- Cálculo del **seno(x)**

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

- Se observa que los términos de la serie son recurrentes:

$$t_i = -t_{i-1} \cdot \frac{x \cdot x}{2 \cdot i \cdot (2 \cdot i + 1)} \qquad t_0 = x$$

- Patrón de programación de sumatorios:

$$suma = \sum_{i=0}^{N} termino$$

```
suma=0; i = 0;
while (i <= N)
{
    suma = suma + termino;
    i = i + 1;
}
```

Ejemplo 3: pseudocódigo

```
leer eps, N_interv
dx = pi / N_interv
x = 0
while (x <= pi)
        t = x
        i = 1
        seno = x
        while | t | > eps
             t = − t * (x*x)/(2*i*(2*i + 1)
             seno = seno + t
             i = i + 2
        end_while
        print x, seno
        x = x + dx
 end_while
```

Ejemplo 3: while

```
/*****************************************************************\
* Programa: seno.c                                                *
* Descripción: Prog. que imprime el seno para valores en el intervalo    *
* entre 0 y PI y precison especificadas por el usuario            *
* mediante una serie de Taylor y el valor según la funcion de biblioteca *
* Autor: Pedro Corcuera                                           *
* Revisión: 1.0 2/02/2008                                         *
\*****************************************************************/
#include <stdio.h>
#include <math.h>

#define PI 3.141592

main()
{
 float seno, x, term, dx, eps;
 int i,ninterv;

 printf("Ingresa precision ");
 scanf("%f",&eps);
 printf("Ingresa numero de intervalos ");
 scanf("%d",&ninterv);
 dx = PI/ninterv;
```

Ejemplo 3: while

```
x = 0;
while (x <= PI)
{
  term=x;
  seno=x;
  i=1;
  while (fabs(term)> eps)
  {
    term = -term*x*x/(2*i*(2*i+1));
    seno += term;
    i += 1;
  }
  printf("seno ( %f ) = %f \t sin = %5f\n",x,seno,sin(x));
  x += dx;
 }
}
```

for

- Es la instrucción de repetición que con más frecuencia se utiliza.
- Sintaxis:

```
for ( [expr-inic] ; [expr-cond] ; [expr-ciclo] )
        instrucción
```

La instrucción for es equivalente a:

```
[expr-inic] ;
while ([expr-cond]) {
        instrucción
        [expr-ciclo] ;
}
```

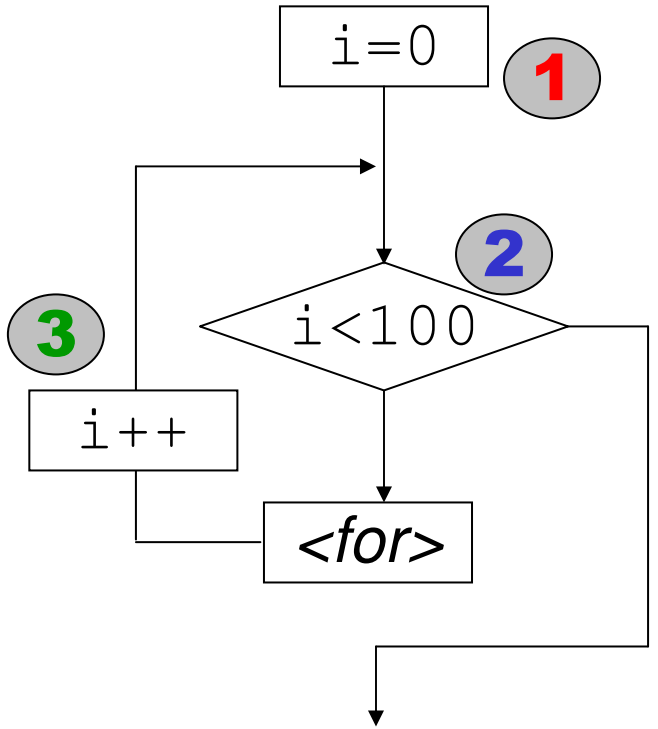
for

- Se ejecuta la primera expresión, llamada de *inicialización*.
- Se evalúa la segunda expresión, llamada de *condición*. Si es verdadera (valor diferente a 0) se ejecuta el bloque del ciclo
- Después de ejecutar las instrucciones del ciclo, se ejecuta la tercera expresión, llamada de *incremento*.
- La expresión de inicialización e incremento no pueden ser llamadas a función.
- Puede omitirse cualquiera de las tres partes, aunque deben permanecer los puntos y coma. Si la expresión de condición se omite se supone que es cierta.

for

```
int main()
{
   int i,ac=0;

   for(i=0; i<100; i++)
   {
       printf("%d",i*i);
       ac += i;
   }
}
```



Sintaxis:

for(**inicialización**,**condición_permanencia**,**incremento**)

Ejemplo: for

```
/*****************************************************************\
* Programa: f2c_for.c                                             *
* Descripción: Prog. que imprime tabla de conversion Fahrenheit-Celsius  *
* Autor: Pedro Corcuera                                           *
\*****************************************************************/
#include <stdio.h>
#define RANGO_INICIAL     0       /* limite inf. tabla */
#define RANGO_FINAL       100     /* limite sup. tabla */
#define PASO              10      /* tamaño paso */

main()
{
  int fahrenheit;
  double celsius;

  for(fahrenheit = RANGO_INICIAL;
      fahrenheit <= RANGO_FINAL ;
      fahrenheit += PASO)
  {
    celsius = (5.0/9.0)*(fahrenheit - 32);
    printf("Fahrenheit = %d \t Celsius = %.3f \n", fahrenheit, celsius);
  }
  return 0;
}
```

do - while

- Es otra variación del ciclo while que comprueba la condición *al final* del ciclo. Por ello, siempre se realiza la instrucción del ciclo por lo menos una vez.
- Si la expresión condicional es verdadera, el bloque de instrucciones del ciclo se ejecuta nuevamente. Si es falsa se sale del ciclo.
- Aun cuando no es necesario, es mejor colocar la(s) instrucción(es) del ciclo entre llaves.
- Sintaxis:

```
do
        instrucción
while (expresión) ;
```

do - while

```
int main()
{
   int i=0,ac=0;

   do
   {
       printf("%d",i*i);
       ac += i;
       i++;
   }
   while( i<100 );
}
```

Diagrama de flujo

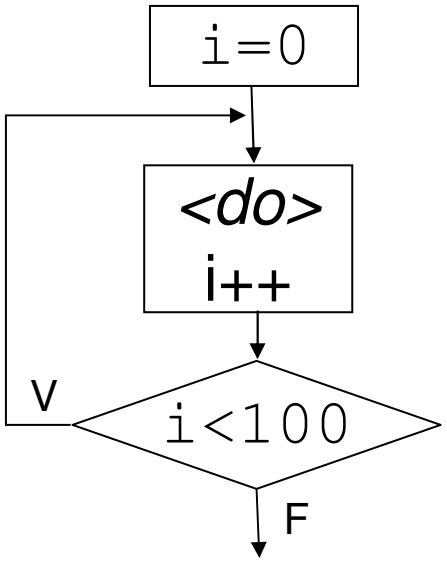
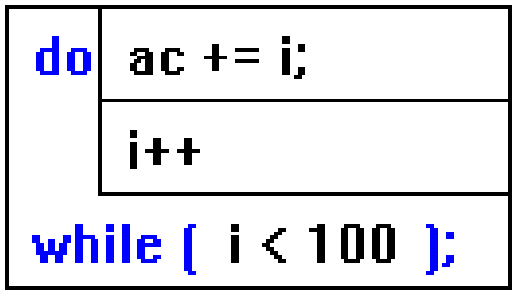


Diagrama de cajas



Ejemplo: do - while

```
/*****************************************************************\
* Programa: promedio_dowhile.c                                    *
* Descripción: Prog. que calcula el promedio de una serie de numeros *
* Autor: Pedro Corcuera                                           *
\*****************************************************************/
#include <stdio.h>

main()
{
  int n, cont = 1;
  float x, promedio, suma = 0;

  printf("Cuantos numeros? "); scanf(" %d", &n);
  do
  {
    printf("x = ");
    scanf(" %f", &x);
    suma += x;
    ++ cont;
  } while ( cont <= n)
  promedio = suma / n;
  printf("El promedio es: %f\n", promedio);
  return 0;
}
```

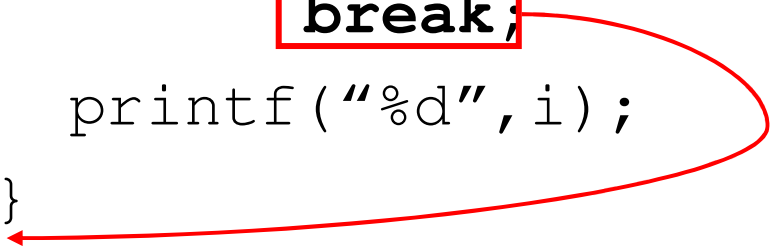
break

- Se usa para terminar la ejecución de las sentencias do-while, for, switch, o while en la que aparece. El control pasa a la siguiente instrucción que sigue a la que termina.
- Se usa para terminar un ciclo sin necesidad de asignar banderas (flags).
- Sintaxis:

```
break ;
```

break

```
int main()
{
  int i;
  for(i = 0; i < 100; i++)
  {
    if(i % 17 == 0)   /* Si multiplo de 17 */
          break;              /*Sale del bucle*/
    printf("%d",i);
  }
}
```



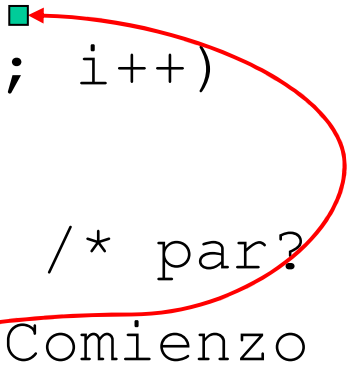
continue

- La instrucción continue causa que la expresión condicional del ciclo (for, while, do-while) se re-evalúe inmediatamente.
- Sintaxis:

```
continue;
```

continue

```
int main()
{
   int i;
   for(i = 0; i < 100 ; i++)
   {
      if(i % 2 == 0)    /* par? */
            continue;/*Comienzo la iteración*/
      printf("%d ",i);
   }
}
```



goto

- goto permite el salto *no estructurado* hacia una instrucción etiquetada. ⟶
- Para *etiquetar* una instrucción se coloca antes de ella un nombre seguida de :
- Debe **evitarse** su uso (*instrucción maldita*)
- Sintaxis:

```
goto nombre ;
. . .
nombre : instrucción
```

Librería <math.h> (I)

double acos(double x) Calcula el arco coseno de x.

double asin(double x) Calcula el arco seno de x.

double atan(double x) Devuelve el arco tangente en radianes.

double atan2(double y, double x) Calcula el arco tangente de las dos variables x e y. Es similar a calcular el arco tangente de y / x, excepto en que los signos de ambos argumentos son usados para determinar el cuadrante del resultado.

double ceil(double x) Redondea x hacia arriba al entero más cercano.

double cos(double x) devuelve el coseno de x, donde x está dado en radianes.

double cosh(double x) Devuelve el coseno hiperbólico de x.

double exp(double x) Devuelve el valor de e (la base de los logaritmos naturales) elevado a la potencia x.

double fabs(double x) Devuelve el valor absoluto del número en punto flotante x.

double floor(double x) Redondea x hacia abajo al entero más cercano.

double fmod(double x, double y) Calcula el resto de la división de x entre y. El valor devuelto es x - n * y, donde n es el cociente de x / y.

Librería <math.h> (II)

double frexp(double x, int *exp) Se emplea para dividir el número x en una fracción normalizada y un exponente que se guarda en exp .

long int labs(long int j) Calcula el valor absoluto de un entero largo.

double ldexp(double x, int exp) Devuelve el resultado de multiplicar el número x por 2 elevado a exp (inversa de frexp).

double log(double x) Devuelve el logaritmo neperiano de x.

double log10(double x) Devuelve el logaritmo decimal de x.

double modf(double x, double *iptr) Divide el argumento x en una parte entera y una parte fraccional. La parte entera se guarda en iptr.

double pow(double x, double y) Devuelve el valor de x elevado a y.

double sin(double x) Devuelve el seno de x.

double sinh(double x) Regresa el seno hiperbólico de x.

double sqrt(double x) Devuelve la raíz cuadrada no negativa de x.

double tan(double x) Devuelve la tangente de x.

double tanh(double x) Devuelve la tangente hiperbólica de x.