# *TP Nº 3: Control de Flujo*

## *Ejercicio 1*

```
#include <stdio.h>

int
main(void)
{
   int letra;

   printf("Ingresar un caracter: ");
   letra = getchar();

   if (letra >= 'A' && letra <= 'Z')
        printf("\nEs una letra mayúscula\n");
   else if (letra >= 'a' && letra <= 'z')
        printf("\nEs una letra minúscula\n");
   else
        printf("\nNo es una letra\n");

   return 0;
}
```

## *Ejercicio 2*

```
#include <stdio.h>

int
main (void)
{
  int a, b;

  printf("Ingresar dos caracteres (sin espacios entre ambos): ");
  a = getchar ();
  b = getchar ();

  if (a > b)
     printf("\nEl caracter '%c' es mayor al caracter '%c'\n", a, b);
  else if (a < b)
     printf("\nEl caracter '%c' es menor al caracter '%c'\n", a, b);
   else
      printf("\nEl caracter '%c' es igual al caracter '%c'\n", a, b);

  return 0;
}
```

## *Ejercicio 3*

```
#include <stdio.h>
#include "getnum.h"

int
main(void)
{
  double venta;

  do
  {
    venta = getfloat("\nIngrese la suma vendida: ");
  }while (venta < 0);

  if (venta > 4000)
    printf("El monto a cobrar es: $%10.2f\n", 300.0 + venta * 0.09);
  if (venta >= 2000 && venta <= 4000)
    printf("El monto a cobrar es: $%10.2f\n", 300.0 + venta * 0.07);
  if (venta >= 1000 && venta < 2000)
    printf("El monto a cobrar es: $%10.2f\n", 300.0 + venta * 0.05);
  if (venta < 1000)
    printf("El monto a cobrar es: $%10.2f\n", 300.0);

  return 0;
}
```

El programa anterior satisface la correctitud pero tiene serios defectos:

- no hace preguntas excluyentes. Si ya sabemos que es superior a 4000, no tiene sentido preguntar luego si está entre 2000 y 4000. Y en caso de cambiar el rango (por ejemplo 3000 por 2000) hay que cambiar en dos lados
- Repite código, por lo que de ser necesario cambiar o corregir la impresión, habría que repetir la modificación 4 veces.
- No utiliza constantes simbólicas. De tener que cambiar el sueldo básico se lo debe cambiar en 4 lados.

```
#include <stdio.h>
#include "getnum.h"

#define BASICO   300.0
#define PORC1000   0.05
#define PORC2000   0.07
#define PORC4000   0.09

int
main(void)
{
  double venta, total = BASICO;

  do
  {
    venta = getfloat("\nIngrese la suma vendida: ");
  }while (venta < 0);

  if (venta > 4000)
    total += venta * PORC4000;
  else if (venta >= 2000)
    total += venta * PORC2000;
  else if (venta >= 1000)
    total +=  venta * PORC1000;

  printf("El monto a cobrar es: $%10.2f\n", total);

  return 0;
}
```

## *Ejercicio 4*

a) **imprime "#######" y "*******"**

b) **imprime "#######"**

c)

```
if ( x < 3 )
{
   printf ("x<3");
   if ( z < 5 )
     printf ("x<3 y z<5");
}
else
   printf ("x >= 3");
```

d)

```
if ( x = = 1)
{
   if ( z > 1)
     printf ( "x = = 1 y z > 1");
}
else
   if ( x > 1 )
     printf ("x > 1");
   else
     printf("x < 1");
```

## *Ejercicio 5*

```
int a, b, c;
a = getchar();
b = getchar();
c = getchar();

if ( a>='B' && a<='Z' && a!='E' && a!='I' && a!='O' && a!='U')
    printf("El caracter %c es una consonante mayúscula \n", a);
else if (a>='b' && a<='z' && a!='e' && a!='i' && a!='o' && a!='u')
  printf("El caracter %c es una consonante minúscula \n", a);
else if (a>='A' && a<='Z' || a>='a' && a<='z')
               printf("El caracter %c es una vocal \n", a);
else
  printf("El caracter %c no es una letra \n", a);


if ( b % 2 == 0 )
    printf("El valor ASCII de %c es par \n", b);

if ( b>='0' && b<= '9' && (b-'0')%2 == 0)
    printf("El caracter %c representa un dígito par \n", b);

if ( c>='A' && c<='Z' || c=='a' ||c=='e' || c=='i' || c=='o' || c=='u' )
    printf("El caracter %c es una vocal o es mayúscula \n", c);

if ( (c>='B' && c<='Z' || c>='b' && c<= 'z') && c!= 'e' && c!='E'
      && c!='i' && c!='I' && c!='o' && c!='O' && c!='u' && c!='U')
    printf("El caracter %c no es una vocal pero es letra \n", c);
```

## *Ejercicio 6*

a)
```
int a = 0;
while (a <= 4)
{
  resultado += a;
  a++;
}
```

**De no estar las llaves solo se ejecuta una instrucción dentro del ciclo.**

b)
```
if ( sueldo >= 1000 )      /* aquí había una coma */
  printf("El sueldo es mayor o igual a 1.000\n"); /* aquí había una coma */
else
  printf("El sueldo es menor a 1.000\n");
```

c) La variable acumuladora ***var2*** no está inicializada. **No** hay que asumir que las variables se inicializan en cero.

d)
```
while ('Y' != (c = getchar()))
  printf ("Pulse Y para abandonar el ciclo\n");
```

**Aquí había un punto y coma, el printf se ejecutaba exactamente una vez pues estaba fuera del while.**

e) **int** c;

El valor retornado por la macro getchar es de tipo **int** debido a que necesita poder devolver además de todos los caracteres posibles (comprendidos en un char) una indicación de error o fin de entrada (EOF) . Por lo tanto utiliza un tipo de dato de mayor rango.

Además hay que tener en cuenta que algunos compiladores consideran al char como **unsigned char** y otros como **signed char**, en el primer caso la condición es siempre falsa pues la variable c jamás puede ser igual a un valor negativo. Pensar qué problemas pueden ocurrir en el segundo caso.

f) Para imprimir el nombre de los días, se debe reemplazar la invocación a **printf** por un **switch**

```
switch(dia) {
case LUN : printf("LUN"); break;
case MAR: printf("MAR"); break;
....
}
```

## *Ejercicio 7*

El programa intenta calcular el promedio de una cantidad *n* de números, ingresados por el usuario, acumulando la suma de los mismos en *suma*. Pero al presentar el promedio en pantalla, divide por el último valor ingresado y no por la cantidad de números *n*.
El único caso en que el programa entrega el valor correcto, es si el último valor ingresado coincide con la cantidad *n* de números.
Ésta condición NO es suficiente como para considerar que el programa funciona correctamente.

## *Ejercicio 8*

### *Kernighan 1.3*

```
#include <stdio.h>

int
main(void)
{
   float fahr, celsius;
   int lower, upper, step;

   lower = 0;
   upper = 300;
   step = 20;

   printf("Tabla de conversión de temperaturas\n");
   printf("Fahr  Celsius\n");
   fahr = lower;

   while (fahr <=  upper)
   {
      celsius = (5.0/9.0) * (fahr-32.0);
      printf("%3.0f %6.1f\n", fahr, celsius);
      fahr = fahr + step;
   }
   return 0;
}
```

### *Kernighan 1.4*

```
#include <stdio.h>

int
main(void)
{
   float fahr, celsius;
   int lower, upper, step;

   lower = 0;
   upper = 300;
   step = 20;

   printf("Tabla de conversion de temperaturas\n");
   printf("Celsius Fahr\n");
   celsius = lower;

   while (celsius <= upper)
   {
        fahr = (9.0 * celsius) / 5.0 + 32.0;
        printf("%3.0f %6.1f\n", celsius, fahr);
        celsius = celsius + step;
     }
     return 0;
}
```

***Kernighan 1.5***

```
#include <stdio.h>

int
main(void)
{
   int fahr;

   printf("Tabla de conversión de temperaturas\n");
   printf("Fahr  Celsius\n");

   for (fahr = 300; fahr >= 0; fahr -=  20)
            printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));

   return 0;
}
```

***Kernighan 2.1***

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

int
main(void)
{
   printf("\nUsando las constantes de limits.h\n");

   /* tipos signados */
   printf("signed char min = %d\n", SCHAR_MIN);
   printf("signed char max= %d\n", SCHAR_MAX);
   printf("signed short min= %d\n", SHRT_MIN);
   printf("signed short max= %d\n", SHRT_MAX);
   printf("signed int min= %d\n", INT_MIN);
   printf("signed int max= %d\n", INT_MAX);
   printf("signed long min= %ld\n", LONG_MIN);
   printf("signed long max= %ld\n", LONG_MAX);

   /* tipos sin signo */
   printf("unsigned char max = %u\n", UCHAR_MAX);
   printf("unsigned short max = %u\n", USHRT_MAX);
   printf("unsigned int max = %u\n", UINT_MAX);
   printf("unsigned long max = %lu\n", ULONG_MAX);

   printf("\nHaciendo cálculos\n");
   /* tipos signados */
   printf("signed char min = %d\n",
          (signed char)((unsigned char) 1 << (sizeof(char) * 8) - 1 ));
   printf("signed char max = %d\n",
          (signed char)((unsigned char) ~0 >> 1 ));
   printf("signed short min = %d\n",
          (short)((unsigned short) 1 << (sizeof(short) * 8) - 1 ));
   printf("signed short max = %d\n",(short)((unsigned short) ~0 >>1 ));
   printf("signed int min = %d\n",
          (int)((unsigned int) 1 << (sizeof(int) * 8) - 1 ));
   printf("signed int max = %d\n",(int)((unsigned int) ~0 >> 1));
   printf("signed long min = %ld\n",
         (long)((unsigned long) 1 << (sizeof(long) * 8) - 1 ));
   printf("signed long max = %ld\n",(long)((unsigned long) ~0 >> 1));

   /* tipos sin signo */
   printf("unsigned char max = %u\n",(unsigned char) ~0);
   printf("unsigned short max = %u\n",(unsigned short) ~0);
   printf("unsigned int max = %u\n",(unsigned int) ~0);
   printf("unsigned long max = %lu\n",(unsigned long) ~0);

   return 0;
}
```

### *Kernighan 2.2*

**El ejercicio 2.2 de K&R pide escribir un ciclo sin usar && ni ||, pero que sea equivalente a for ( i=0; i<lim-1 && (c=getchar()) != '\n' && c != EOF; ++i )**

Código equivalente:

```
enum ciclo { NO, SI };
enum ciclo ciclar = SI;

i = 0;
while ( ciclar == SI )
   if (i >= lim-1)       /* ¿ fuera del rango ? */
      ciclar = NO;
   else if ( (c = getchar()) == '\n')
      ciclar = NO;
   else if (c == EOF)    /* fin de archivo */
      ciclar = NO;
   else
      s[i++] = c;
```

## *Ejercicio 9*

### *Kernighan 1.8*

```
#include <stdio.h>

int
main(void)
{
   int c, blancos, tabs, nuevasLineas;

   blancos = 0;
   tabs = 0;
   nuevasLineas = 0;

   while ((c = getchar()) !=EOF)
   {
      switch(c)
      {
      case ' ':  blancos++; break;
      case '\t': tabs++; break;
      case '\n': nuevasLineas++; break;
      }
   }

   printf("\nBlancos: %d\nTabuladores: %d\nNuevas Lineas: %d\n",
      blancos,tabs,nuevasLineas);

   return 0;
}
```

***Kernighan 1.12***

```
#include <stdio.h>

#define IN_WORD 1
#define OUT_WORD 0

int
main(void)
{
   int c, estado;

   estado = OUT_WORD;
   while ((c = getchar()) != EOF)
      switch(estado)
      {
         case IN_WORD:
            if (c == ' ' || c == '\n'  || c == '\t')
            {
               putchar('\n');
               estado = OUT_WORD;
            }
            else
               putchar(c);
            break;
         case OUT_WORD:
            if (c != ' '&& c != '\n'  && c != '\t')
            {
               estado = IN_WORD;
               putchar(c);
            }
            break;
      }
   return 0;
}
```

## *Ejercicio 10*

a)
```
float x = 1.0 / 3.0;
while (x < 0.52)
    x += .01;
```

**1/3 es división entera y el resultado es 0 (cero)**

**Dos valores reales no deben ser comparados con los operadores "==" o "!="**

b)
```
#define EPSILON 0.001
....
/*  No se puede preguntar por la igualdad en los reales por el
** error de representación. Se debe usar un margen de error (en
** este caso de un milésimo */
for ( y = .1; y -1.0 > EPSILON ||  y -1.0 <  -EPSILON; y += .1)
```

c)
```
switch (n) {
case 1:
     printf("El número es 1\n");
     break;          /* Falta el break */
case 2:
  ...
```

d) Hay que encerrar entre paréntesis la asignación, pues la comparación tiene mayor precedencia que la asignación.

```
while ( (c = getchar() ) != EOF )
```

## *Ejercicio 11*

```
#include <stdio.h>

int
main(void)
{
  int n;
  printf("\n%-10s%-10s%-10s%-10s\n","N","10*n","100*N","1000*N");

  for (n=1; n<=20; n++)
    printf("%-10d%-10d%-10d%-10d\n",n,n*10,n*100,n*1000);

  return 0;
}
```

## *Ejercicio 12*

```
#include <stdio.h>

int
main(void)
{
  int i;

  for (i = 3; i <= 23; i+=5)
    printf("%d,",i);
  printf("\n");

  for (i = 20; i >= -10; i-=6)
    printf("%d,",i);
  printf("\n");
```

**Punto a)**

**Punto b)**

```
   for (i = 19; i <= 51; i+=8)        }      Punto c)
     printf("%d,",i);                 }
   printf("\n");

   return 0;
}
```

### *Ejercicio 13*

```
#include <stdio.h>
#include "getnum.h"

int
main(void)
{
   int i,j, lado;

   do
      lado = getint("Ingrese el lado del cuadrado:");
   while (lado <=0);

   /* Con dos ciclos anidados */
   for (i = 1; i <= lado; i++)
   {
      for (j = 1; j <= lado; j++)
         putchar('*');
      putchar('\n');
   }

   putchar('\n');

   /* con un único ciclo */
   for (i = 1; i <= lado*lado; i++)
   {
      putchar('*');
      if (i % lado == 0)
         putchar('\n');
   }

   return 0;
}
```

## *Ejercicio 14*

```
#include <stdio.h>

#define DELTA ('a' - 'A')

int
main(void)
{
   int car;
   int menor='z';
   int cantMay=0, cantMin=0;
   int eraMay=0, esMay;

   /* Lee caracteres de la entrada estándar mientras no sea fin de entrada y
    *  sea una letra ( mayúscula o minúscula ) del alfabeto ingles
    */
   while ((car=getchar())>= 'A' && car<= 'Z' || car>= 'a' && car<= 'z'
           || car ==' ')
   {
      if ( car != ' ' )
      {
         if (esMay = (car >= 'A' && car <= 'Z'))
         {
            cantMay++;
            car += DELTA;
         }
         else
            cantMin++;

         if (car < menor)
         {
            menor = car;
            eraMay = esMay;
         }
      }
   }

   if ( cantMin + cantMay > 0 )
      printf("\nLa letra menor era: %c\n Cantidad de mayúsculas: %d\n"
           "Cantidad de minúsculas: %d\n",
             eraMay ? menor-DELTA:menor, cantMay, cantMin);
   }
   else
      printf ("\nNo se han ingresado letras\n");

   return 0;
}
```

**En caso de ser mayúscula, la convertimos a minúscula, para poder comparar con la que hasta ahora es la mayor letra.**

**eraMay vale cero si la letra es minúscula y uno si es mayúscula.**

### *Ejercicio 15*

```
#include <stdio.h>
#include "getnum.h"

/* Solo se consideran binarios positivos */
int
main(void)
{
   int binario;
   int decimal=0;
   int base=1;
   int resto;

   binario = getint("Ingrese un número binario (hasta 10 digitos):");

   for (base = 1; binario > 0;  binario /=10, base *=2)
   {
      if ( (resto = binario % 10) > 1 )
      {
         printf("\nNo es un número binario !!!\n");
         return 1;
      }
      else if (resto)
         decimal += base;
   }
   printf("\nEquivalente decimal = %d\n",decimal);

   return 0;
}
```

**Si es un número distinto de 0 o 1 no es binario**

**Si el resto es uno, sumarle una base.**

### *Ejercicio 16*

```
#include <stdio.h>
#include "getnum.h"
#define MAX_ENTERO 1023

int
main(void)
{
   int decimal;
   long binario;
   int pot;

   do
   {
      printf("Ingrese un número entero (entre 0 y %d):", MAX_ENTERO);
      decimal = getint("");
      if (decimal < 0 || decimal > MAX_ENTERO)
         printf("Error de entrada\n");
   }while (decimal < 0 || decimal > MAX_ENTERO);

   for(binario = 0, pot=1;  decimal >0;  decimal /= 2, pot *= 10)
      if (decimal%2)
         binario += pot;

   printf ("\nEquivalente binario: %ld\n",binario);

   return 0;
}
```

### *Ejercicio 17*

```
#include <stdio.h>

int
main(void)
{
   long i=1;

   /* Ejecutar el ciclo mientras los números que se obtienen sean múltiplos
    ** de 10. Dejarán de ser múltiplos cuando el resultado esté fuera del rango
    */
   while ( (i*=10) % 10 == 0 )
      printf("%ld\n",i);

   return 0;
}
```

### *Para pensar: ¿ Es un método seguro ?*

### *Ejercicio 18*

```
#include <stdio.h>
#include "getnum.h"

int
main(void)
{
   int i;
   long factorial=1;

   printf("Cálculo de factorial\n");
   do
   {
      i = getint("Ingrese un numero no negativo: ");
      if (i < 0)
         printf("Número incorrecto\n");
   }while (i < 0);


   while ( i > 1 )
      factorial *= i--;

   /* Este resultado puede ser inválido si excede el rango del tipo long */
   printf(" \nEl factorial es: %ld\n", factorial);

   return 0;
}
```

## *Ejercicio 19*

a) Números enteros

```
#include <stdio.h>
#include "getnum.h"

#define DIGITO 5

int
main(void)
{
   int n;
   int contador=0;

   n = getint("Ingrese un entero: ");

   /* Si es negativo lo convertimos para trabajar en positivo */
   if ( n < 0 )
      n *= -1;

   while ( n != 0 )
   {
      if ( (n % 10) == DIGITO )
         contador++;
      n /= 10 ;
   }

   printf("\nCantidad de dígitos iguales a %d: %d\n", DIGITO,contador);
   return 0;
}
```

b) Números reales

Presentamos dos soluciones alternativas para el caso de números reales.

El primero presenta la solución matemáticamente correcta, pero que en la práctica no funciona debido al error de representación del punto flotante, que pierde precisión en la parte decimal.

Por ejemplo, si ingresamos 34.456, el número almacenado en realidad es el 34.456001282.....A esto se le agrega el error de la multiplicación en cada paso.

```
#include <stdio.h>
#include "getnum.h"

#define DIGITO 5
#define ERROR 0.000000001

int
main(void)
{
   float x;
   long parteEntera;
   int contador=0;

   x = getfloat("Ingrese un número real: ");
   printf("\nNúmero leído: %20.9f\n",x);

   if ( x < 0 )
      x *= -1;

   parteEntera = (long) x;
   x = x - parteEntera;

   while ( parteEntera != 0 )
   {
      if ( (parteEntera % 10) == DIGITO )
         contador++;
      parteEntera /= 10 ;
   }

   printf ("\nDígitos iguales a %d en parte entera:  %d\n",
            DIGITO,contador);
   contador = 0;

   while ( x > ERROR )
   {
      x *= 10;
      parteEntera = (long) x;
      if ( parteEntera == DIGITO )
         contador++;
      x = x - parteEntera;
   }

   printf ("\nDígitos iguales a %d en parte decimal: %d\n",
            DIGITO, contador);
   return 0;
}
```

**Vamos multiplicando por diez, lo cual *teóricamente* introduce ceros a la derecha y en algún momento x debería llegar a valer cero.**

La siguiente solución lee los dígitos como caracteres, sin almacenarlos. A medida que se van leyendo, se van contando cuántos de ellos son igual a '5'. Al estar basada en la lectura de caracteres no hay errores de representación.

```
#include <stdio.h>
#define DIGITO '5'

int
main(void)
{
   int c;
   int huboPunto = 0;
   int cntInt = 0;
   int cntDec = 0;

   printf ("\nIngrese un número real: ");
   while ( ( (c=getchar()) >= '0' &&  c <= '9')  ||
               ( c == '.' &&  !huboPunto ) )
   {
      if ( c >= '0' && c <= '9' )
      {
         if ( c == DIGITO )
            ( huboPunto ) ? cntDec++ : cntInt++;
      }
         else
         huboPunto = 1;
   }
   printf ("\nDigitos iguales a %c en parte entera: %d\n",
             DIGITO,cntInt);
   printf ("Digitos iguales a %c en parte decimal: %d\n",DIGITO,cntDec);

   return 0;
}
```

### *Ejercicio 20*

Presentamos primero una versión que, si bien funciona, no es aceptable pues es ineficiente.

```
#include <stdio.h>

#define EPSILON 0.0000001

int
main(void)
{
   long factorial=1;
   float e=1, anterior=0;
   int i=1, j;

   printf("%-10s %10s\n","N","e");
   while ( e - anterior > EPSILON )
   {
      printf ("%-10d %10.7f\n",i,e);
      anterior = e;
      e += 1.0 / factorial;
      i++;
      for (j=2, factorial=1; j<=i; j++)
         factorial *= j;
   }
   return 0;
}
```

**Calculamos factorial en cada paso**

Presentamos ahora una versión correcta

```
#include <stdio.h>

#define EPSILON 0.0000001

int
main(void)
{
   long factorial=1;
   float e=1, anterior=0;
   int i=1;

   printf("%-10s %10s\n","N","e");
   while ( e - anterior > EPSILON )
   {
      printf ("%-10d %10.7f\n",i,e);
      anterior = e;
      e += 1.0 / factorial;
      i++;
      factorial *= i;
   }
   return 0;
}
```

**Calculamos i! = i * ( i-1)!**

### *Ejercicio 21*

```
#include <stdio.h>
#include "getnum.h"

int
main(void)
{
   int n;
   int n2=0;
   int aux;

   do
   {
      n = getint("Ingrese un número (sin ceros a la izquierda):");
   }while (n<0);

   aux = n;

   while ( aux > 0 )
   {
      n2 = n2 * 10 + ( aux % 10 );
      aux /= 10;
   }

   printf ("\nEl número %ses capicúa\n",(n2==n)?"":"no ");

   return 0;
}
```

## *Ejercicio 22*

```
#include <stdio.h>
#define TOPE 100

int
main(void)
{
  int i, j, invertido, cantCiclos = 0;
  int numDec, numAux, numAct;
  int uno, cinco, diez;

  printf("%-20s%-20s\n", "Num. Decimal", "Num. Romano");
  for (numDec = 1; numDec <= TOPE; numDec++)
  {
     numAux = numDec;
     printf("%-20d",numDec);
     /*
     ** Se invierten los dígitos para poder analizar desde el dígito
     ** de mayor peso hacia el de menor peso
     */
     invertido = 0;
     cantCiclos = 0;
     while (numAux >0)
     {
        invertido = invertido * 10 + (numAux % 10);
        numAux /= 10;
        cantCiclos++;
     }
     numAux = invertido;

     /* Se analiza cada dígito, empezando por el mayor */
     for(i = cantCiclos-1 ; i>=0 ; i--)
     {
        switch(i) /* Se preparan los caracteres necesarios para la posición */
        {
           case 0: uno = 'I'; cinco = 'V'; diez = 'X'; break;
           case 1: uno = 'X'; cinco = 'L'; diez = 'C'; break;
           case 2: uno = 'C'; cinco = 'D';diez = 'M'; break;
        }

        numAct = (numAux % 10) ; /* dígito para analizar */
        numAux = numAux / 10;    /* el resto, en la vuelta siguiente */

        /* Si el dígito involucra una "resta" */
             if (numAct == 4 || numAct == 9)
                 printf("%c%c", uno, numAct==4? cinco:diez);
             else    /* Si solo se suma */
             {
                 if (numAct >= 5)
                 {
                     printf("%c",  cinco);
                     numAct-=5;
                 }
                 for(j=0;j != numAct ;j++)
                     printf("%c",  uno);
             }
     }
     putchar('\n');
  }
  return 0;
}
```

## *Ejercicio 23*

Kernighan 1.9

```
#include <stdio.h>
#define BLANCO     1
#define CARACTER   0

int
main(void)
{
   int c;
   int estado=CARACTER;    /* estado indica si el ultimo caracter leído
                           ** es un blanco u otro caracter  */

   while ((c = getchar()) != EOF)
   {
      switch( estado)
      {
         case CARACTER:
            if ( c == ' ' )
               estado = BLANCO;
            putchar(c);
            break;
         case BLANCO:
            if ( c != ' ' )
            {
               putchar(c);
               estado = CARACTER;
            }
            break;
      }
   }
   return 0;
}
```

Kernighan 1.10

```
#include <stdio.h>

int
main(void)
{
   int c;

   while ((c=getchar()) != EOF)
      switch(c)
      {
         case '\\': printf("\\\\"); break;
         case '\t': printf("\\t"); break;
         case '\b': printf("\\b"); break;
         default: putchar(c); break;
      }

   return 0;
}
```

### *Ejercicio 24*

```
int n,c;

 /* al salir del ciclo n debe ser un número par mayor que cero */
do
  n=getint("");
while(n <= 0 || n % 2 != 0 );

/* otra forma de hacerlo */
 while(!(n > 0 && n % 2 == 0));

/* este ciclo debe incluir la lectura de caracteres y se debe
** ejecutar mientras no sea fin de archivo y los caracteres sean
** dígitos o minúsculas
*/
 while ((c=getchar())!=EOF && (c>='0' && c<='9' || c>='a' && c<='z')
          putchar(c);
```

### *Ejercicio 25*

La salida que se obtiene al ejecutar el programa es la siguiente:

***NO SON iguales: a vale 0.1 que no es igual a 0.1***

Obviamente es absurda e involucra conceptos **MUY IMPORTANTES**:

- Como ya se explicó en Organización Básica de la Computadora, los valores reales se almacenan en memoria según la norma IEEE 754. Es decir que se almacenan expresados como mantisa y exponente. Por este motivo una gran cantidad de valores no tienen representación exacta **como por ejemplo el 0.1**. A la pérdida de precisión debido a este error de representación se le debe agregar que 0.1 es una constante de ***tipo double*** que al ser almacenada en una variable de ***tipo float*** pierde aún más precisión.

- La función **printf** *por default* muestra los valores reales redondeados a una cierta precisión, si uno desea ver una precisión mayor debe colocar la precisión deseada explícitamente como se muestra en el siguiente ejemplo:

```
#include <stdio.h>

int
main(void)
{
    float a=0.1;

    if (a==0.1)
        printf("SON iguales\n");
    else
        printf("NO SON iguales:a vale %.20g que no es igual a 0.1\n", a);

    return 0;
}
```

Y su salida es :

***NO SON iguales: a vale 0.10000000149011611938 que no es igual a 0.1***