Principios de Programación

El lenguaje C

1. Más sobre Instrucciones de control

Estudiaremos con mayor detalle la repetición y presentaremos estructuras adicionales de control de la repetición a saber las estructuras **for** y **do-while**.

Presentaremos la estructura de selección múltiple **switch**.

Analizaremos los enunciados **break** que sale de las estructuras de control y **continue** que saltea el resto de una estructura de control y continua con la siguiente iteración.

1.1. Lo esencial de la repetición

La mayor parte de los programas incluyen repeticiones o ciclos. Un ciclo es un grupo de instrucciones que la computadora ejecuta en forma repetida en tanto se conserve verdadera alguna condición de continuación del ciclo. Hemos analizado dos procedimientos de repetición:

1. repetición controlada por contador
2. repetición controlada por centinela

La repetición controlada por contador se denomina a veces *repetición definida* porque con anticipación se sabe con exactitud cuantas veces se ejecutará el ciclo. La repetición controlada por centinela a veces se denomina *repetición indefinida* porque no se sabe con anticipación cuantas veces se ejecutará el ciclo.

En la repetición controlada por contador se utiliza una *variable de control* para contar el número de repeticiones. La variable de control es incrementada (normalmente en 1) cada vez que se ejecuta el grupo de instrucciones dentro de la repetición. Cuando el valor de la variable de control indica que se ha ejecutado el número correcto de repeticiones se termina el ciclo y la computadora continúa ejecutando el enunciado siguiente al de la estructura de repetición.

Los valores centinela se utilizan para controlar la repetición cuando:

1. El número preciso de repeticiones no es conocido con anticipación, y
2. El ciclo incluye enunciados que deben obtener datos cada vez que este se ejecuta.

el valor centinela indica "fin de datos". El centinela es introducido una vez que al programa se le han proporcionado todos los datos. Los centinelas deben ser un valor diferente a los valores que toman los datos.

1.2. Repetición controlada por contador

La repetición controlada por contador requiere:

1. El nombre de una variable de control (o contador del ciclo).
2. El valor inicial de la variable de control.
3. El incremento (o decremento) con el cual, cada vez que se termine un ciclo la variable de control será modificada.
4. La condición que compruebe la existencia del valor final de la variable de control.

Considere el siguiente programa que imprime los números del 1 al 10:

Programa

```
#include <stdio.h>

main ()
{
      int contador=1;

      while (contador <= 10) {
            printf (" %d\n", contador);
            ++contador;
      }
      system("PAUSE");
}
```

La declaración contador=1 le da nombre a la variable de control, la declara como un entero, reserva espacio para la misma y pone su valor inicial a 1.

La declaración e inicialización de contador también podría haber sido hecha mediante los enunciados:

```
int contador;

contador=1;
```

El enunciado ++contador; incrementa en 1 el contador del ciclo cada vez que este se ejecuta. La condición del while prueba si el valor de la variable de control es menor o igual a 10 (un valor mayor en este caso 11) hace que la condición se vuelva falsa.

Alternativamente podriamos haber escrito:

```
contador=0;
while (++contador <= 10)
      printf(" %d\n", contador);
```

este trozo de programa incrementa el contador y testea la condición juntos en la condición del while.

2. La estructura de repetición for

La estructura de repetición **for** maneja de forma automática todos los detalles de la repetición controlada por contador.

Para ilustrar los poderes del for reescribiremos el programa anterior:

Programa

```
main ()
{
      int contador;
      for (contador=1; contador <= 10; contador++)
          printf(" %d\n, contador);
      system("PAUSE");
}
```

El programa funciona como sigue: cuando la estructura for inicia su ejecución la variable contador se inicializa en 1. A continuación se verifica la condición de continuación del ciclo que es contador ¡= 10. Como la condición se satisface se imprime contador y luego la variable contador se incrementa en 1, y el ciclo comienza de nuevo con la prueba de continuación del ciclo.

En general:

```
for (expresion1; expresion2; expresion3)
    enunciado;
```

es equivalente a:

```
expresion1;
while (expresion2)
{
      enunciado;
      expresion3;
}
```

expresión 1 y expresión 3 pueden ser cada una de ellas un conjunto de instrucciones separadas por comas que se evaluaran de izquierda a derecha. El valor y el tipo de una lista de expresiones separadas por coma es el valor y el tipo de la expresión de más a la derecha de la lista. El operador coma es utilizado prácticamente solo en estructuras for. Su uso principal es permitir al programador utilizar múltiples expresiones de inicialización y/o múltiples incrementos.

Las tres expresiones de la estructura for son opcionales. Si se omite expresion2, C supondrá la condición es verdadera creando un ciclo infinito. Se puede omitir expresion1 si la variable de control se inicializa en algún otro lado. La expresión3 podría también omitirse si el incremento no es necesario o si se lo calcula mediante enunciados en el cuerpo de la estructura for. Los dos puntos y coma de la estructura for son requeridos.

2.1. La estructura for: notas y observaciones

1. El incremento puede ser negativo (el ciclo ira hacia atrás).
2. Si la condición de continuación del ciclo resulta falsa al inicio, la porción del cuerpo del ciclo no se ejecutará. En vez de ello, la ejecución seguirá adelante con el enunciado que siga a la estructura for.
3. La variable de control, con frecuencia se imprime o se utiliza en cálculos en el cuerpo del ciclo pero esto no es necesario. Es común utilizar la variable de control para controlar la repetición sin necesidad de utilizarla dentro del cuerpo del ciclo.

2.2. Ejemplos utilizando la estructura for

Lo siguientes ejemplos muestran métodos de variar en una estructura for la variable de control:

1. Varie la variable de control de 1 a 100 en incrementos de a 1:

   for (i=1 ; i <= 100; i++)

2. Varie la variable de control de 100 a 1 en incrementos de -1:

   for (i=100 ; i >= 1; i- -)

3. Variar la variable de control de 7 a 77 en pasos de 7:

   for (i=7; i <= 77; i+=7)

4. Variar la variable de control de 20 a 2 en pasos de -2:

   for (i=20; i >= 2; i-=2)

5. Variar la variable de control a lo largo de la siguiente secuencia de valores: 2, 5, 8, 11, 14, 17, 20:

   for (i=2; i <= 20; i+=3)

6. variar la variable de control de acuerdo a la siguiente secuencia de valores: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

   for (i=99; i >= 0; i-=11)

Los siguientes dos ejemplos proporcionan aplicaciones simples de la estructura for. El primer programa utiliza la estructura for para sumar todos los enteros pares desde 2 hasta 100:

Programa 1

```
#include <stdio.h>

main ()
{
      int suma=0, numero;
      for (numero=2; numero <= 100; numero+=2)
          suma += numero;
      printf("Suma es %d\n", suma);
      system("PAUSE");
}
```

Podriamos realizar la cuenta de suma y las inicializaciones dentro del encabezado del for del siguiente modo:

Programa 1

```
#include <stdio.h>

main ()
{
      int numero;
      for (int suma=0, numero=2; numero <= 100; suma += numero, numero+=2);
      printf("Suma es %d\n", suma);
      system("PAUSE");
}
```

El segundo programa calcula el interes compuesto. Considere el siguiente enunciado del problema:

Una persona invierte 1000$ en una cuenta de ahorros que reditúa un interes del 5 %. Suponiendo que todo el interes se queda en depósito dentro de la cuenta, calcule e imprima la cantidad de dinero en la cuenta al final de cada año, durante 10 años. Utilice la fórmula siguiente:

$a = p(1 + r)^n$

donde:
p es la cantidad invertida
r es la tasa anual de interes
n es el numero de años
a es la cantidad en depósito al final del año n

Este problema incluye un ciclo que ejecuta el cálculo indicado para cada uno de los 10 años en los cuales el dinero se queda en depósito. El programa es el siguiente:

Programa 2

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

main ()
{
      int anio;
      double cantidad, principal=1000, tasa=.05;

      printf("%4s%21s\n", "Anio, "Cantidad en deposito");

      for (anio=1;anio <= 10;anio++)
      {
          cantidad = principal * pow(1.0+tasa,anio);
          principal = cantidad;
          printf("%4d%21.2f \n", anio, cantidad);
      }
}
```

pow(x,y) calcula el valor de x elevado a la y. Toma dos argumentos de tipo double y devuelve un valor double. Incluimos la biblioteca math que contiene la definición de esta función. El argumento year es entero. El archivo math.h incluye información que le indica al compilador que convierta el valor de year a una representación temporal double antes de llamar a la función.

El especificador de conversión "%21.2f.[es] utilizado en el programa para imprimir el valor de la variable cantidad. El 21 especifica el ancho del campo en el cual el valor se imprimirá. El 2 especifica la presición (número de posiciones decimales). Si el número de caracteres desplegados es menor que el ancho del campo el valor será automáticamente alineado a la derecha. Si quisiera alinear a la izquierda se debe colocar un signo de menos entre el % y el ancho de campo. El signo de menos puede utilizarse también para alinear enteros y cadenas de caracteres.

3. La estructura de selección múltiple switch

Un programa puede contener una serie de desiciones en las cuales una variable o expresión se probará por separado contra cada uno de los valores constantes enteros que pueda asumir y se tomarán diferentes acciones. Para esta forma de toma de decisiones se proporciona la estructura de decisión múltiple switch.

La estructura switch está formada de una serie de etiquetas **case** y de un caso opcional **default**.

En el siguiente programa se leen calificaciones y se cuenta el total de cada calificación ingresada. Dentro del encabezado de un while se leerá el grado de aprobación (que es un caracter). La repetición termina cuando se ingresa EOF.

Programa

```
#include <stdio.h>

main ()
{
      int grado;
      int cant_a=0, cant_b=0, cant_c=0,
          cant_d=0, cant_e=0, cant_f=0;

      printf ("Ingrese la calificacion\n");
      printf ("Ingrese EOF para terminar el ingreso de datos\n");
      while (( grado=getchar())!=EOF)
      {
            switch (grado)
            {
             case 'A': case 'a': cant_a++;
                                 break;
             case 'B': case 'b': cant_b++;
                                 break;
             case 'C': case 'c': cant_c++;
                                 break;
             case 'D': case 'd': cant_d++;
                                 break;
             case 'E': case 'e': cant_e++;
                                 break;
             case 'F': case 'f': cant_f++;
                                 break;
             case '\n': case ' ': break;
             default :
                       printf("Ha ingresado un grado incorrecto, entre un nuevo grado");
                       break;
            }
      }
printf("\n Totales de cada grado son:\n");
printf("A: %d\n", cant_a);
printf("B: %d\n", cant_b);
printf("C: %d\n", cant_c);
printf("D: %d\n", cant_d);
printf("E: %d\n", cant_e);
printf("F: %d\n", cant_f);
}
```

Una de las caracteristicas de C es que los caracteres pueden ser almacenados en cualquier tipo de datos entero, ya que son representados como enteros de 1 byte. Podemos entonces tratar a un carácter como si fuera un entero o un caracter dependiendo de su uso.

Los caracteres pueden ser leidos utilizando scanf mediante el uso del especificador de conversión $\%c$.

La expresión de control del switch es la que se coloca entre paréntesis a continuación de la palabra clave switch.

En el ejemplo anterior el usuario introduce las calificaciones con el teclado. Cuando se oprime enter los caracteres son leidos por getchar() de a un caracter a la vez. Si el caracter introducido no es igual a EOF (CTRL-Z) se introduce en la estructura switch. En el ejemplo, grado es la expresión de control. El valor de esta expresión es comparado con cada una de las etiquetas case. Suponga el usuario ha escrito la letra C como calificación. La C se compara con cada uno de los case dentro del switch. Si ocurre una coincidencia, se ejecutarán los enunciados correspondientes a dicho case. En este caso se incrementa cant_c en 1, y luego se ejecuta break que hace que se salga del switch.

El enunciado break causa que el control del programa continue con el primer enunciado que sigue despues de la estructura switch. Si no se utilizara el enunciado break los cases siguientes al de la coincidencia se ejecutarian (por ejemplo ingreso solo letras a y las cuenta como a, como b, etc.). Si no hubiera coincidencia se ejecuta el caso default.

Cada case puede tener una o más acciones. La estructura switch es diferente de las otras estructuras en que no se necesitan llaves alrededor de varias acciones de un case de un switch.

4. La estructura de repetición do/while

Es similar a la estructura while. La diferencia es que la condición de continuación del ciclo se testea despues de ejecutar el cuerpo del ciclo y por lo tanto el cuerpo del ciclo se ejecutará al menos una vez.

La forma del enunciado es:

```
do
   enunciado
while (condicion);
```

Cuando termina do/while la ejecución continuará con el enunciado que aparezca despues de la palabra clave while. En el caso de que enunciado sea más de una sentencia se deben colocar entre llaves.

El programa a continuación utiliza una estructura do/while para imprimir los números del 1 al 10.

Programa

```
#include <stdio.h>

main ()
{
      int contador=1;

      do {
         printf (" %d ",contador);
         }
```

```
        while (++contador <= 10);
        system("PAUSE");
}
```

El enunciado:

```
do
   enunciado
while (condicion)
```

es equivalente a :

```
enunciado;
while (condicion)
        enunciado;
```

5. Los enunciados break y continue

5.1. break

Se utilizan para modificar el flujo de control. El enunciado **break**, cuando se utiliza en una estructura while, for, do/while o switch causa la salida inmediata de dicha estructura. La ejecución del programa continua con el primer enunciado después de la estructura.

El programa a continuación muestra el uso del enunciado break es una estructura for.

```
main ()
{
        int x;

        for (x=1; x <= 10; x++)
        {
            if (x == 5) break;
            printf (" %d ",x);
        }
        printf ("\n Salimos del loop con x = %d", x);
        system("PAUSE");
}
```

El programa imprime el 1,2,3,4 y el mensaje Salimos del loop con x = 5.

Programas equivalentes con while y do/while son los siguientes:

```
/* programa con while */
main ()
{
        int x;
```

```
        x=1;
        while (x <= 10)
        {
          if (x == 5) break;
          printf (" %d ",x);
          x++;
        }
        printf ("\n Salimos del loop con x = %d", x);
        system("PAUSE");
}


/* programa con do/while */
main ()
{
        int x;

        x=1;
        do
        {
         if (x == 5) break;
         printf (" %d ",x);
         x++;
        }
        while (x <= 10)
        printf ("\n Salimos del loop con x = %d", x);
        system("PAUSE");
}
```

5.2. continue

El enunciado **continue** cuando se ejecuta en una estructura while, for, do/while, salta los enunciados restantes del cuerpo de dicha estructura y ejecuta la siguiente iteración del ciclo.

En las estructuras while y do/while la prueba de continuación del ciclo se evalúa de inmediato luego del continue. En la estructura for se ejecuta la expresión incremental y a continuación se evalúa la prueba de continuación del ciclo.

A continuación veremos un programa que utiliza continue en una estructura for.

```
main ()
{
        int x;

        for (x=1; x <= 10; x++)
        {
         if (x == 5) continue;
```

```
        printf (" %d ",x);
       }
       printf ("\n Salteamos el valor 5 con continue");
       system("PAUSE");
}
```

este programa imprime:

```
1 2 3 4 6 7 8 9 10
Salteamos el valor 5 con continue
```

programas equivalentes con while y do/while son:

```
/* programa con while */
main ()
{
       int x;

       x=1;
       while (x <= 10)
       {
        if (x == 5) { x++; continue;}
        printf (" %d ",x);
        x++;
       }
       printf ("\n Salteamos el valor 5 con continue");
       system("PAUSE");
}
```

```
/* programa con do/while */
main ()
{
       int x;

       x=1;
       do
       {
        if (x == 5) { x++; continue;}
        printf (" %d ",x);
        x++;
       }
       while (x <= 10);

       printf ("\n Salteamos el valor 5 con continue");
       system("PAUSE");
}
```

6. Ejemplos de programas simples con for y do/while

6.1. Factorial

Programa

```
/* Factorial con do/while */
main ()
{
      int i,n,fac;
      printf("Ingrese el numero del cual calculara factorial \n");
      scanf(" %d", &n);
      i=1;
      fac=1;
      do
          {
            fac*=i;
            i++;
          }
      while (i <= n);
      printf ("Factorial de %d es igual a %d",n,fac);
}


/* Factorial con for */
main ()
{
      int n,fac=1;
      printf("Ingrese el numero del cual calculara factorial \n");
      scanf(" %d", &n);
      for (int i=1; i <= n; fac*=i;i++);
      printf ("Factorial de %d es igual a %d \n",n,fac);
      system("PAUSE");
}
```

6.2. Potencia n-esima

Programa

```
/* Calculo la potencia n-esima de un numero dado con do/while*/
main ()
{
      int i,n,num,potencia;
/* num guarda el numero cuya potencia se calculara,
n guarda la potencia,
potencia se utiliza para calcular la potencia,
i cuenta la cantidad de veces que he multiplicado potencia por num */

      i=1;
```

```
        potencia=1;
        printf (“ Ingrese potencia \n”);
        scanf(“%d”, &n);
        printf (“ Ingrese numero cuya potencia desea calcular \n”);
        scanf (“%d,&num);
        do
        {
          potencia *= num;
          i++;
        }
        while (i¡=n);
        printf (“Potencia %d de %d es igual a %d \n”,n,num,potencia);
}
```

Programa

```
/* Calculo la potencia n-esima de un numero dado con for */
main ()
{
        int n,num,potencia;
/* num guarda el numero cuya potencia se calculara,
n guarda la potencia,
potencia se utiliza para calcular la potencia,
i cuenta la cantidad de veces que he multiplicado potencia por num */

        potencia=1;
        printf (“ Ingrese potencia \n”);
        scanf(“%d”, &n);
        printf (“ Ingrese numero cuya potencia desea calcular \n”);
        scanf (“%d,&num)
;       for (int i=1; i <= n; potencia*=num,i++) ;
        printf (“Potencia %d de %d es igual a %d \n”,n,num,potencia);
}
```

7. Casos de estudio de programación con do/while y for

7.1. Repetición controlada por contador

Promedio de la nota de un examen de una clase con 10 alumnos

Programa

```
/* Promedio programado con do/while */
main ()
{
        total=0;
```

```
        nro_alumno=1;
        do
        {
         scanf(" %d",&calificacion);
         total+=calificacion;
         nro_alumno++;
        }
        while (nro_alumno <= 10);
        promedio=total/10;
        printf("El promedio es %f",promedio);
        system("PAUSE");
}
```

Programa

```
/* Promedio programado con for */
main ()
{
        total=0;
        nro_alumno=1;
        for (nro_alumno=1;nro_alumno <= 10; nro_alumno++)
            {
             scanf(" %d",&calificacion);
             total+=calificacion;
            }
        promedio=total/10;
        printf("El promedio es %f",promedio);
        system("PAUSE");
}
```

7.2. Repetición controlada por centinela

El problema es igual al anterior salvo que en vez de considerar 10 estudiantes consideramos una cantidad arbitraria.

Ingresaremos datos y cuando hayamos terminado con todos los datos ingresaremos -1.

Hay casos por ejemplo éste en el cual el problema se puede resolver de una forma directa con while y no hay una forma directa de resolverlo con do/while. En este caso resolvemos el problema utilizando un if para diferenciar cuando no hay datos de cuando los hay.

Programa

```
/* Programa con do/while */
main ()
{
        int total=0;
        int cant_alumnos=0;
        int centinela;
        int calificacion;
```

```
printf("Ingrese -1 para terminar, cualquier otro valor para continuar\n");
scanf(" %d",&centinela);
if (centinela!=-1)
  do
  {
   scanf(" %d", &calificacion);
   total+=calificacion;
   cant_alumnos++;
   printf("Ingrese -1 para terminar, cualquier otro valor para continuar\n");
   scanf(" %d", &centinela);
  }
  while (centinela != -1);
if (cant_alumnos!=0)
{
  promedio=total/cant_alumnos;
  printf("El promedio es %f", promedio);
}
else
  printf("No se ingresaron calificaciones");
system("PAUSE");
```

Programa

```
/* Programa con for */
main ()
{
    int total=0,centinela,calificacion;
    printf("Ingrese -1 para terminar, cualquier otro valor para continuar\n");
    scanf(" %d",&centinela);
    for(int cant_alumnos=0;centinela!=-1;cant_alumnos++)
      {
       scanf(" %d", &calificacion);
       total+=calificacion;
       printf("Ingrese -1 para terminar, cualquier otro valor para continuar\n");
       scanf(" %d",&centinela);
      }
    if (cant_alumnos!=0)
    {
      promedio=total/cant_alumnos;
      printf("El promedio es %f", promedio);
    }
    else
      printf("No se ingresaron calificaciones");
    system("PAUSE");
```

7.3. Estructuras de control anidadas

Consideremos el problema:

Una universidad ofrece un curso que prepara alumnos para un examen. La universidad desea saber que tan bien salieron sus alumnos en el examen. Se ingresará un 1 si el alumno pasó el examen y un 2 si lo reprobó. Se quiere saber total de aprobados y total de reprobados en un total de 10 alumnos.

Programa

```
/* Programa con do/while*/

main ()
{
      int aprobados=0;
      int reprobados=0;
      cant_alumnos=1;
      do
          {
           scanf(" %d", &calificacion);
           if (calificacion==1) aprobados++;
           else reprobados++;
           cant_alumnos++;
          }
      while (cant_alumnos <= 10);
      printf("Total de aprobados");
      printf(" %d\n", aprobados);
      printf("Total de reprobados");
      printf(" %d", reprobados);
      system("PAUSE");
```

Programa

```
/* Programa con for */

main ()
{
      int aprobados=0;
      int reprobados=0;
      for (int cant_alumnos=1; cant_alumnos <= 10; cant_alumnos ++)
          {
           scanf(" %d", &calificacion);
           if (calificacion==1) aprobados++;
           else reprobados++;
          }
      printf("Total de aprobados");
      printf(" %d\n", aprobados);
      printf("Total de reprobados");
      printf(" %d", reprobados);
      system("PAUSE");
```

8. Una colección de programas utiles

8.1. Copia de entrada en la salida con do/while y con for

Programa

```
main()
{
        int c;

        do
        {
            c=getchar();
            if (c==EOF) break;
            else putchar(c);
        }
        while (c!=EOF);
}
```

Hemos escrito el programa arriba usando break, otra forma es imprimiendo el caracter leido si es distinto de eof como a continuación:

Programa

```
main()
{
        int c;

        do
        {
            c=getchar();
            if (c!=EOF) putchar(c);
        }
        while (c!=EOF);
}
```

Podemos tambien escribir el programa asociando la condición del do/while con el getchar() como a continuación:

Programa

```
main()
{
        int c;

        c=getchar();
        do
            if (c!=EOF) putchar(c);
        while((c=getchar())!=EOF);
```

```
}
```

otra forma (que queda de ejercicio) es colocar la instrucción if antes del do/while de forma que se ejecute el do/while si c!=EOF.

A continuación presentamos el programa con for:

Programa

```
main()
{
        int c;

        for (;c=getchar()!=EOF;putchar(c));
}
```

8.2. Cuenta de la cantidad de caracteres leidos

Programa

```
/* Programa con do/while */
main()
{
      long nc;

      nc=0;
      if (getchar()!=EOF)
        do
            nc++;
        while (getchar()!=EOF);
      printf("Cantidad de caracteres leidos = %ld\n", nc);
      system("PAUSE)";
}
```

Programa

```
/* Programa con do/while */
main()
{
      long nc;

      nc=0;
      for(;getchar()!=EOF;nc++) ;
      printf("Cantidad de caracteres leidos = %ld\n", nc);
      system("PAUSE");
}
```

8.3. Cuenta de la cantidad de lineas leidas

Programa

```
/* Programa con do/while */
main ()
{
      int c,nl;

      nl=0;
      if ((c=getchar())!=EOF)
        do
            if (c=='\n') nl++;
        while ((c=getchar())!=EOF);
      printf("Cantidad de nueva linea %d\n", nl);
      system("PAUSE");
}
```

Programa

```
/* Programa con for */
main ()
{
      int c,nl;

      for (nl=0; (c=getchar())!=EOF;)
          if (c=='\n') nl++;
      printf("Cantidad de nueva linea %d\n", nl);
      system("PAUSE");
}
```

8.4. Cantidad de nueva linea, palabras y caracteres con do/while y for

Cada vez que el programa encuentra el primer caracter de una palabra lo cuenta. La variable en_palabra recuerda cuando el programa está dentro de una palabra y cuando no. Inicialmente no lo es. Cuando no está dentro de una palabra y el siguiente caracter no es espacio, nueva linea ni tabulador seteamos que está dentro de una palabra. Se suma uno a cantidad de palabras.

Programa

```
main()
{
      int c, nl, nw, nc, en_palabra;

      en_palabra = false;
      nl=nw=nc=0;
      if ((c=getchar())!=EOF)
```

```
        do
        {
          ++nc;
          if (c=='\n') ++nl;
          if (c==' ' || c=='\n' || c=='\t') en_palabra=false;
          else if (!en_palabra) { en_palabra=true; ++nw; }
        }
        while ((c=getchar())!=EOF);
      printf(" %d %d %d \n", nl, nw, nc);
      system("PAUSE");
}
```