Estrategias de usos de bucles

David Álvarez

Marzo 2024

Índice

1. Introducción

En el desarrollo de programas es habitual encontrarse ante la necesidad de contabilizar o detectar características en un conjunto de datos. También es común desarrollar programas que resuelvan ecuaciones de tipo sumatorio o productorio. Para resolver este tipo de situaciones existen estrategias metodológicas ayudan, en cierta manera, al proceso de desarrollo de código que calcula la soclusión buscada. Las más comunes son los contadores, acumuladores y las banderas.

2. Contadores

Resulta habitual que entre los requisitos de un programa sea necesario **contabilizar el número de veces que se cumple una condición**. Por ejemplo, suponga que queremos determinar la cantidad de divisores que tiene un número entero (que guardaremos en la variable n. En estos casos se utilizan los contadores.

Un contador es una variable entera cuyo valor inicial es (normalmente) cero y que se incrementa en una unidad cada vez que, durante un proceso iterativo, se cumple la condición que se está buscando (por ejemplo, que un número es divisor de otro).

Como se observa en el programa del ejemplo, la variable *divisores* desempeña el papel de contador. Durante la declaración se inicializa a cero y su valor se incrementa cuando, en cada iteración del bucle que recorre todos los enteros entre 1 y n, se comprueba que uno de los valores cumple la condición de divisibilidad.

```c
/* Ejemplo: Contar los divisores de un número entero
            introducido por el usuario */

#include <stdio.h>

int main(void)
{
    int n, i, divisores=0;
    printf ("Introduzca un numero: ");
    scanf ("%i", &n);
    printf ("Divisores: ");

    for (i=1; i<=n; i++)
    {
        if (n % i == 0)
        {
            printf ("%i ",i);
            divisores++;
        }
    }
    printf ("\nEl numero %i tiene %i divisores\n", n, divisores);

    return 0;
}
```

---

3. Acumuladores

Los acumuladores permiten realizar operaciones de agregación. En ocasiones se utilizan para sumar aquellos datos que cumplen una condición. En otras representan un sumatorio o un productorio de series. Los acumuladores son variables numéricas enteras y su valor inicial es el elemento neutro de la operación de agregación, es decir, el 0 en el caso de sumas y el 1 en el caso de productos.

```c
/* Ejemplo: Suma de los números pares entre 1 y n */

#include <stdio.h>

int main(void)
{
    int n,i,suma=0;
    printf ("Introduzca un número: ");
    scanf ("%i",&n);
    for (i=1;i<=n;i++)
    {
        if (i % 2 == 0)
        {
            suma = suma + i;
        }
    }

    printf ("\nLa suma de los pares entre 1 y %i es:%i \n",n,suma);

    return 0;
}
```

En el caso de las series existe un método sencillo para trasladar las mismas a una secuencia de instrucciones. En las series suma ($\sum$) la variable acumulador se inicializa a cero con objeto de ir agregando los términos de la serie utilizando el operador suma (+) entre el acumulador y la expresión de la serie. El resultado de la suma de cada término se acumula asignando (=), en cada iteración, el resultado al acumulador.

Los valores inicial y final de la serie son respectivamente el valor inicial y la condición de finalización del bucle *for* (si la serie tiene como valor final infinito (∞) es necesario definir un valor final finito con el que obtener una aproximación de la serie). En el caso de las series producto ($\prod$) el mecanismo es similar pero el acumulador se inicializa a uno y el operador será el producto (*).

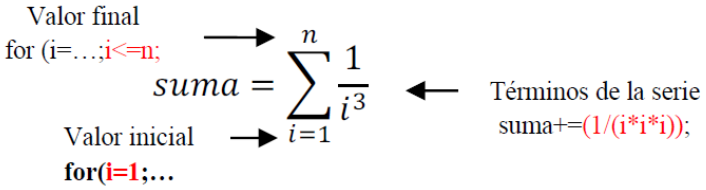

Fig. 1. Ejemplo de un sumatorio y valores en los que fijarse.

```c
/* Ejemplo: Resolver la serie de la imagen anterior a partir
de un valor de n introducido por el usuario */

#include <stdio.h>

int main(void)
{
    int n, i;
    float suma=0.0;

    printf ("Introduzca numero de terminos de la serie:");
    scanf ("%i",&n);

    for (i=1;i<=n;i++)
    {
        // Cuidado con la división entre números enteros
        suma = suma + (1.0/(i*i*i));
    }

    printf ("La suma de la serie es: %f\n",suma);

    return 0;
}
```

El siguiente ejemplo resuelve un productorio que permite calcular el factorial de un número:

$$factorial = \prod_{i=i}^{n} i$$

```c
/* Ejemplo: Resolver la serie de productos
a partir de un valor de n introducido por el usuario.
Observe que esta serie producto representa el factorial de n */

#include <stdio.h>

int main(void)
{
    int n, i, factorial=1;
    printf ("Introduzca un número entero:");
    scanf ("%i",&n);
    for (i=1;i<=n;i++)
    {
        factorial = factorial * i;
    }

    printf ("El factorial de %i es: %i\n",n,factorial);

    return 0;
}
```

4. Banderas (Flags)

Las banderas permiten comprobar/marcar si una propiedad de naturaleza lógica (verdadera o falsa) que que debe comprobasrse repetitivamente en muchos casos, se cumple o no. Por ejemplo, determinar si un número es primo.

En este tipo de problemas, tras definir bajo qué condiciones se puede afirmar que un número es primo (u otra condición que se busque), el primer paso consiste en establecer cual de las dos opciones (ser primo o no serlo) se va a verificar.

En este ejemplo, determinar que un número NO es primo, no requiere comprobar todos y cada uno de los divisores del mismo, porque es suficiente con hallar un divisor distinto de la unidad y del propio número para poder afirmar que el número no es primo. Sin embargo para afirmar que el número es primo, resulta necesario verificar todos los números entre 1 y el propio número, para afirmar que el número es primo. El caso más evidente es el de los número pares, verificar que NO son primos solo requiere comprobar que son divisibles por 2. Sin embargo, verificar que son primos, requier comprobar todos los divisores.

Una bandera es una variable que puede adquirir un valor entre verdadero o falso (0,1). El valor de la bandera al finalizar la verificación permite saber si la propiedad evaluada es verdadero o falso. El valor inicial de la bandera debe ser aquel resultado que implicaría realizar la secuencia completa de verificaciones. En el ejemplo de la comprobación de la condición de primo de un número, la bandera se inicializa a verdadero, inicialmente se considera el número primo.

Durante la secuencia de comprobación, si se encuentra un divisor diferente de la unidad y del propio número, entonces resulta suficiente para cambiar el valor de la bandera. Si durante todas las verificaciones no se encuentra ningun divisor, el valor de la bandera no habrá cambiado y la suposición inicial será correcta, el número es primo.

```c
/* Ejemplo: Determinar si un número entero introducido por teclado es primo o no */

#include <stdio.h>

int main(void)
{
    int n,i;
    int es_primo=1;

    printf ("Introduzca un numero entero:");
    scanf ("%i",&n);

    for (i=2;i<n;i++)
    {
        if (n % i == 0)
        {
            es_primo=0;
        }
    }

    if (es_primo == 1)
    {
        printf ("El numero %i es primo\n",n);
    }
    else
    {
        printf ("El numero %i no es primo\n",n);
    }

    return 0;
}
```