CAPÍTULO 5
Instrucciones de repetición

TEMAS

5.1 ESTRUCTURAS BÁSICAS DEL CICLO
- CICLOS DE PRUEBA PRELIMINAR Y PRUEBA POSTERIOR
- CICLOS DE CUENTA FIJA FRENTE A LOS DE CONDICIÓN VARIABLE

5.2 CICLOS `while`

5.3 CICLOS `while` INTERACTIVOS
- CENTINELAS
- INSTRUCCIONES `break` Y `continue`
- LA INSTRUCCIÓN NULA

5.4 CICLOS `for`

5.5 TÉCNICAS DE PROGRAMACIÓN CON CICLOS
- TÉCNICA 1: ENTRADA INTERACTIVA DENTRO DE UN CICLO
- TÉCNICA 2: SELECCIÓN DENTRO DE UN CICLO
- TÉCNICA 3: EVALUACIÓN DE FUNCIONES DE UNA VARIABLE
- TÉCNICA 4: CONTROL INTERACTIVO DE UN CICLO

5.6 CICLOS ANIDADOS

5.7 CICLOS `do while`
- VERIFICACIONES DE VALIDEZ

5.8 ERRORES COMUNES DE PROGRAMACIÓN

5.9 RESUMEN DEL CAPÍTULO
- CONSIDERACIÓN DE OPCIONES DE CARRERA: INGENIERÍA INDUSTRIAL

Los programas examinados hasta ahora han ilustrado los conceptos de programación implicados en las capacidades de entrada, salida, asignación y selección. En este punto ya debe tener suficiente experiencia para sentirse cómodo con estos conceptos y la mecánica de ponerlos en práctica usando C++. Sin embargo, muchos problemas requieren una capacidad de repetición en la cual el mismo cálculo o secuencia de instrucciones se repita, una y otra vez, usando diferentes conjuntos de datos. Los ejemplos de dicha repetición incluyen la verificación continua de las entradas de datos del usuario hasta que se introduce una entrada aceptable, como una contraseña válida; contar y acumular

totales corrientes; y la aceptación constante de datos de entrada y el recálculo de valores de salida que sólo se detiene al introducir un valor centinela.

Este capítulo explora los diferentes métodos que usan los programadores para construir secciones repetitivas de código y cómo pueden ponerse en práctica en C++. Por lo común, a una sección de código que se repite se le conoce como ***ciclo****, porque después que se ejecuta la última instrucción en el código el programa se ramifica, o regresa, a la primera instrucción y comienza otra repetición a través del código. Cada repetición se conoce también como una* ***iteración*** *o* ***paso a través del ciclo****.*

5.1 ESTRUCTURAS BÁSICAS DEL CICLO

El poder real de un programa se demuestra cuando debe hacerse una y otra vez el mismo tipo de operación. Por ejemplo, considere el programa 3.17 en la sección 3.6, donde el mismo conjunto de instrucciones se repite tres veces. Volver a escribir este mismo conjunto de instrucciones es tedioso, consume tiempo y está sujeto a errores. Por supuesto que sería conveniente si pudiéramos escribir esas instrucciones repetitivas sólo una vez y luego implementar un método para informar al programa que repita la ejecución de las instrucciones tres veces. Dicho método está disponible usando secciones de código repetitivas.

Construir una sección de código repetitiva requiere que estén presentes cuatro elementos. El primer elemento necesario es una instrucción de repetición. Esta ***instrucción de repetición*** define los límites que contienen la sección de código repetitiva y controla si el código se ejecutará o no. En general, hay tres formas diferentes de instrucciones de repetición, todas las cuales son proporcionadas en C++:

1. `while`
2. `for`
3. `do while`

Cada una de estas instrucciones requiere una condición que debe evaluarse, la cual es el segundo elemento requerido para construir secciones de código repetitivas. Las condiciones válidas son idénticas a las usadas en las instrucciones de selección. Si la condición es verdadera, el código es ejecutado; de lo contrario, no lo es.

El tercer elemento requerido es una instrucción que establece la condición al inicio. Esta instrucción debe colocarse siempre antes que la condición sea evaluada por primera vez para asegurar la ejecución correcta del ciclo.

Por último, debe haber una instrucción dentro de la sección de código repetitiva que permita que la condición se vuelva falsa. Esto es necesario para asegurar que, en algún punto, se detengan las repeticiones.

Ciclos de prueba preliminar y prueba posterior

La condición que se está probando puede evaluarse al principio o al final de la sección de código que se repite. La figura 5.1 ilustra el caso donde la prueba ocurre al principio del ciclo. Este tipo de ciclo se conoce como **ciclo de prueba preliminar** porque la condición se prueba antes que se ejecuten cualesquiera instrucciones dentro del ciclo. Si la condición es verdadera, se ejecutan las instrucciones ejecutables dentro del ciclo. Si el valor inicial de la condición es falso, las instrucciones ejecutables dentro del ciclo nunca se ejecutan y el control se transfiere a la primera instrucción que se halla después del ciclo. Para evitar repeticiones infinitas, la condición debe actualizarse dentro del ciclo. Los ciclos de prueba preliminar también se conocen como **ciclos controlados en la entrada**. Las estructuras de ciclo `while` y `for` son ejemplos de estos ciclos.

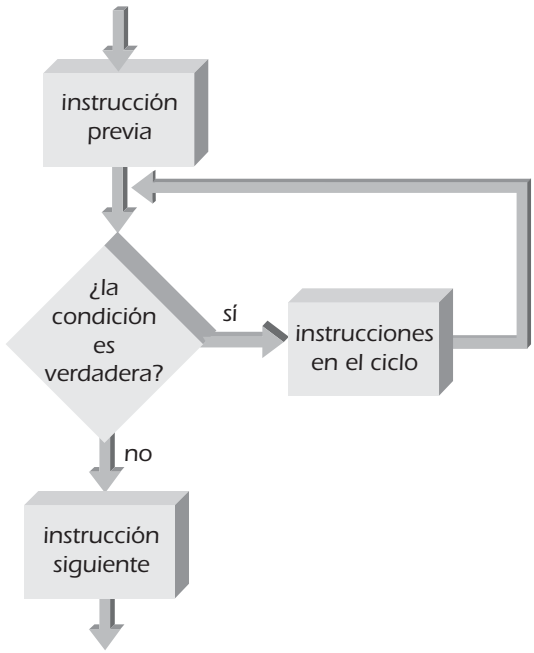


Figura 5.1 Un ciclo de prueba preliminar.

Un ciclo que evalúa una condición al final de la sección de código repetitiva, como se ilustra en la figura 5.2, se conoce como un **ciclo de prueba posterior** o **ciclo controlado en la salida**. Estos ciclos siempre ejecutan las instrucciones del ciclo al menos una vez antes que la condición se pruebe. En vista que las instrucciones ejecutables dentro del ciclo se ejecutan de manera continua hasta que la condición se vuelve falsa, siempre debe haber una instrucción dentro del ciclo que actualice la condición y permita que se vuelva falsa. El constructo `do while` es un ejemplo de un ciclo de prueba posterior.

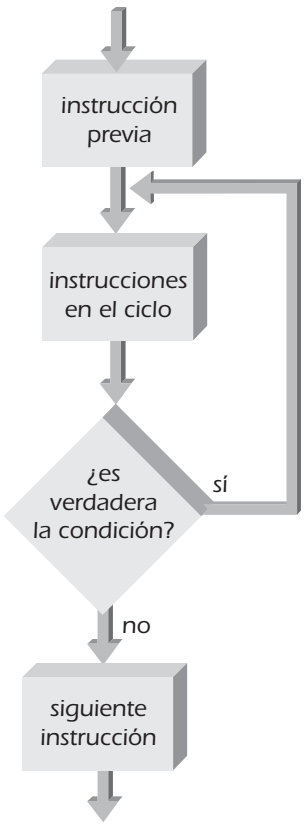


Figura 5.2 *Un ciclo de prueba posterior.*

Ciclos de cuenta fija frente a los de condición variable

Además del lugar donde se prueba la condición (prueba preliminar o prueba posterior), las secciones de código repetitivas también se clasifican según el tipo de condición que se prueba. En un **ciclo de cuenta fija**, la condición se usa para dar seguimiento al número de repeticiones que han ocurrido. Por ejemplo, podríamos desear producir una tabla de 10 números, incluyendo sus cuadrados y cubos, o un diseño fijo como

```
**************************
**************************
**************************
**************************
```

En cada uno de estos casos, se ejecuta un número fijo de cálculos o se imprime un número fijo de líneas, punto en el cual la sección de código repetitiva finaliza. Todas las instrucciones de repetición de C++ pueden utilizarse para producir ciclos de cuenta fija.

En muchas situaciones no se conoce con anticipación el número exacto de repeticiones o los elementos son demasiado numerosos para contarlos con antelación. Por ejemplo, cuando se introduce una cantidad grande de datos experimentales podríamos no desear tomar el tiempo para contar el número de elementos de datos real que se van a introducir. En casos como éste se usa un ciclo de condición variable. En un **ciclo de condición variable** la condi-

ción probada no depende de que se alcance una cuenta, sino más bien de una variable que puede cambiar de manera interactiva con cada paso a través del ciclo. Cuando se encuentra un valor especificado, sin importar cuántas iteraciones han ocurrido, las repeticiones se detienen. Todas las instrucciones de repetición de C++ pueden usarse para crear ciclos de condición variable.[1] En este capítulo se encontrarán ejemplos de ciclos de cuenta fija y de condición variable.

Ejercicios 5.1

1. Enumere las tres instrucciones de repetición que se proporcionan en C++.
2. Enumere los cuatro elementos que deben estar presentes en una instrucción de repetición.
3. **a.** ¿Qué es un ciclo controlado en la entrada?
   b. ¿Cuál de las instrucciones de repetición de C++ produce ciclos controlados en la entrada?
4. **a.** ¿Qué es un ciclo controlado en la salida?
   b. ¿Cuál de las instrucciones de repetición de C++ produce ciclos controlados en la salida?
5. **a.** ¿Cuál es la diferencia entre un ciclo de prueba preliminar y uno de prueba posterior?
   b. Si la condición que se está probando en un ciclo de prueba preliminar es falsa para comenzar, ¿cuántas veces se ejecutarán las instrucciones internas del ciclo?
   c. Si la condición que se está probando en un ciclo de prueba posterior es falsa para comenzar, ¿cuántas veces se ejecutarán las instrucciones internas del ciclo?
6. ¿Cuál es la diferencia entre un ciclo de cuenta fija y uno de condición variable?

5.2 Ciclos `while`

En C++, un **ciclo `while`** se construye usando una instrucción `while`. La sintaxis de esta instrucción es

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

La *`expresión`* contenida dentro del paréntesis es la condición probada para determinar si se ejecuta la *`instrucción`* que sigue al paréntesis. La expresión es evaluada exactamente en la misma manera que la contenida en una instrucción `if-else`; la diferencia está en cómo se usa la expresión. Como se ha visto, cuando la expresión es verdadera (tiene un valor diferente de cero) en una instrucción `if-else`, la instrucción que sigue a la expresión se ejecuta una vez. En una instrucción `while`, la instrucción que sigue a la expresión se ejecuta en forma repetida hasta que la expresión reconozca un valor diferente de cero. Considerando

[1] En esto, C y C++ difieren de los lenguajes de alto nivel anteriores, en los que la instrucción `for` (la cual se ponía en práctica usando una instrucción `DO` en FORTRAN) sólo podía usarse para producir ciclos de cuenta fija. La instrucción `for` de C++, como se verá en breve, es casi intercambiable con su instrucción `while`.

sólo la expresión y la instrucción que sigue al paréntesis, el proceso usado por la computadora al evaluar una instrucción `while` es

1. *Probar la expresión*
2. *Si la expresión tiene un valor diferente de cero (verdadero)*
   a. ejecutar la instrucción que sigue al paréntesis
   b. regresar al paso 1
   de lo contrario
   salir de la instrucción `while` *y ejecutar la siguiente instrucción ejecutable que sigue a la instrucción* `while`

Hay que observar que el paso 2b obliga a que el control del programa se transfiera de nuevo al paso 1. Esta transferencia del control de vuelta al inicio de una instrucción `while` a fin de reevaluar la expresión es lo que forma el ciclo del programa. La instrucción `while` literalmente se enrrolla en sí misma para volver a verificar la expresión hasta que se evalúe en cero (se vuelva falsa). Esto naturalmente significa que en alguna parte en el ciclo debe estipularse una disposición que permita que se altere el valor de la expresión probada. Como se verá, esto es así.

Este proceso de repetición producido por una instrucción `while` se ilustra en la figura 5.3. Se usa una forma de diamante para mostrar los dos puntos de entrada y los dos puntos de salida requeridos en la parte de decisión de la instrucción `while`.

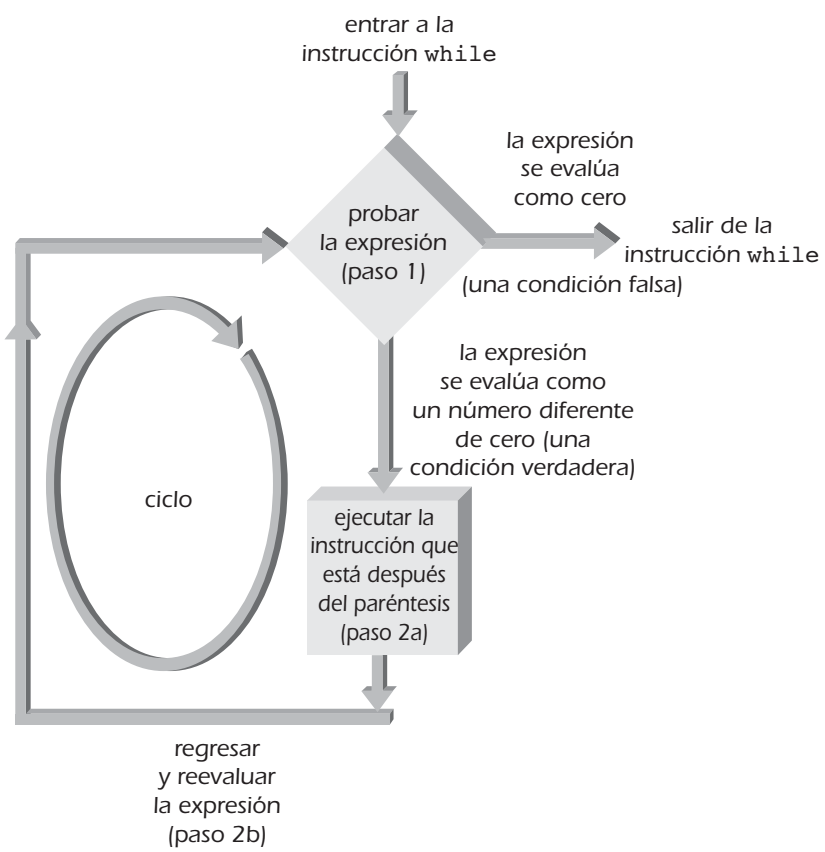


Figura 5.3 Anatomía de un ciclo `while`.

Para hacer esto un poco más tangible, considere la expresión relacional `cuenta <= 10` y la instrucción `cout << cuenta;`. Uando estas dos instrucciones, se puede escribir la siguiente instrucción `while` válida:

```
while (cuenta <= 10)
  cout << cuenta;
```

Aunque la instrucción anterior es válida, el lector alerta se percatará que se ha creado una situación en la que el objeto `cout` es invocado de manera indefinida (hasta que detengamos el programa) o no es invocado en absoluto. Veamos por qué sucede esto.

Si `cuenta` tiene un valor menor que o igual a 10 cuando la expresión se evalúa por primera vez, se ejecuta la instrucción `cout`. Entonces la instrucción `while` regresa automáticamente y vuelve a probar la expresión. En vista que no se ha cambiado el valor almacenado en cuenta, la expresión aún es verdadera y se hace otra llamada a `cout`. Este proceso continúa de manera indefinida, hasta que el programa que contiene esta instrucción sea detenido por el usuario. Sin embargo, si `cuenta` inicia con un valor mayor que 10, la expresión es falsa para comenzar y el objeto `cout` nunca se usa.

¿Cómo establecemos un valor inicial en `cuenta` para controlar lo que hace la instrucción `while` la primera vez que se evalúa la expresión? La respuesta, por supuesto, es asignar valores a cada variable en la expresión probada antes que se encuentre la instrucción `while`. Por ejemplo, la siguiente secuencia de instrucciones es válida:

```
cuenta = 1;
while (cuenta <= 10)
  cout << cuenta;
```

Utilizando esta secuencia de instrucciones, se asegura que `cuenta` comience con un valor de 1. Podría asignarse cualquier valor a `cuenta` en la instrucción de asignación, lo importante es asignar algún valor. En la práctica, el valor asignado depende de la aplicación.

Todavía debemos cambiar el valor de `cuenta` de modo que al final podamos salir de la instrucción `while`. Hacer esto requiere una expresión como `cuenta = cuenta + 1` para incrementar el valor de `cuenta` cada vez que se ejecute la instrucción `while`. El hecho que una instrucción `while` proporcione la repetición de una instrucción única no impide que se incluya una instrucción adicional para cambiar el valor de `cuenta`. Todo lo que tiene que hacerse es reemplazar la instrucción única con una instrucción compuesta. Por ejemplo,

```
cuenta = 1;          // inicializa cuenta
while (cuenta <= 10)
{
  cout << cuenta;
  cuenta++;          // incrementa cuenta
}
```

Nótese que, por claridad, se ha colocado cada instrucción en la instrucción compuesta en una línea diferente. Esto es consistente con la convención adoptada para instrucciones compuestas en el capítulo anterior. Ahora se analizará la secuencia anterior de instrucciones.

La primera instrucción de asignación establece `cuenta` igual a 1. Entonces se introduce la instrucción `while` y la expresión se evalúa por primera vez. En vista que el valor de `cuenta` es menor que o igual a 10, la expresión es verdadera y se ejecuta la instrucción compuesta. La primera instrucción en la instrucción compuesta utiliza el objeto `cout` para desplegar el valor de `cuenta`. La siguiente instrucción agrega 1 al valor almacenado en la actualidad

en `cuenta`, haciendo este valor igual a 2. Ahora la instrucción `while` regresa para volver a probar la expresión. En vista que `cuenta` todavía es menor que o igual a 10, la instrucción compuesta se ejecuta de nuevo. Este proceso continúa hasta que el valor de `cuenta` llega a 11. El programa 5.1 ilustra estas instrucciones en un programa real.

Programa 5.1

```
#include <iostream>
using namespace std;

int main()
{
  int cuenta;

  cuenta = 1;              // inicializa cuenta
  while (cuenta <= 10)
  {
    cout << cuenta << "   ";
    cuenta++;              // incrementa cuenta
  }

  return 0;
}
```

La salida del programa 5.1 es:

```
1   2   3   4   5   6   7   8   9   10
```

No hay nada especial respecto al nombre `cuenta` usado en el programa 5.1. Podría haberse utilizado cualquier variable entera válida.

Antes que consideremos otros ejemplos de la instrucción `while`, se ameritan dos comentarios concernientes al programa 5.1. Primero, la instrucción `cuenta++` puede reemplazarse con cualquier instrucción que cambie el valor de `cuenta`. Una instrucción como `cuenta = cuenta + 2`, por ejemplo, causaría que se desplegara cada segundo entero. En segundo lugar, es responsabilidad del programador asegurar que `cuenta` sea cambiada en una forma que conduzca al final a una salida normal de `while`. Por ejemplo, si se reemplaza la expresión `cuenta++` con la expresión `cuenta--`, el valor de cuenta nunca excederá 10 y se creará un ciclo infinito. Un ***ciclo infinito*** es un ciclo que nunca termina; el programa sigue desplegando números hasta que usted se percata que el programa no está funcionando como esperaba.

Ahora que tiene alguna familiaridad con la instrucción `while`, vea si puede leer y determinar la salida del programa 5.2.

Programa 5.2

```
#include <iostream>
using namespace std;

int main()
{
  int i;

  i = 10;
  while (i >= 1)
  {
    cout << i << "  ";
    i--;              // resta 1 de i
  }

  return 0;
}
```

La instrucción de asignación en el programa 5.2 establece inicialmente la variable `int i` en 10. La instrucción `while` verifica entonces si el valor de `i` es mayor que o igual a 1. Mientras la expresión es verdadera, el valor de `i` es desplegado por el objeto `cout` y el valor de `i` es disminuido en 1. Cuando al fin i llega a cero, la expresión es falsa y el programa sale de la instrucción `while`. Por tanto, se obtiene el siguiente despliegue cuando se ejecuta el programa 5.2:

```
10  9  8  7  6  5  4  3  2  1
```

Para ilustrar el poder de la instrucción `while`, considere la tarea de imprimir una tabla de números de 1 a 10 con sus cuadrados y cubos. Esto puede hacerse con una instrucción `while` simple como se ilustra en el programa 5.3.

Cuando se ejecuta el programa 5.3, se produce el siguiente despliegue:

```
NUMERO    CUADRADO    CUBO
------    --------    ----
  1           1          1
  2           4          8
  3           9         27
  4          16         64
  5          25        125
  6          36        216
  7          49        343
  8          64        512
  9          81        729
 10         100       1000
```

Programa 5.3

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  int num;

  cout << "NUMERO    CUADRADO    CUBO\n"
       << "------    --------    ----\n";

  num = 1;
  while (num < 11)
  {
    cout << setw(3) << num << "         "
         << setw(3) << num * num << "       "
         << setw(4) << num * num * num << endl;
    num++;          // incrementa num
  }
  return 0;
}
```

Hay que observar que la expresión usada en el programa 5.3 es `num < 11`. Para la variable entera `num`, esta expresión es exactamente equivalente a la expresión `num <= 10`. La elección de cuál usar es suya por completo.

Si se desea usar el programa 5.3 para producir una tabla de 1000 números, todo lo que se hace es cambiar la expresión en la instrucción `while` de `num < 11` a `num < 1001`. Cambiar el 11 a 1001 produce una tabla de 1000 líneas, nada mal para una simple instrucción `while` de cinco líneas.

Todos los ejemplos de programa que ilustran la instrucción `while` representan ciclos de cuenta fija porque la condición probada es un contador que verifica un número fijo de repeticiones. Puede hacerse una variación en el ciclo de cuenta fija cuando el contador no se incremente en uno cada vez que pasa por el ciclo, sino por algún otro valor. Por ejemplo, considere la tarea de producir una tabla de conversión de temperatura Celsius a Fahrenheit. Suponga que se van a desplegar temperaturas Fahrenheit correspondientes a las temperatu-

ras Celsius que varían de 5 a 50 grados en incrementos de cinco grados. El despliegue deseado puede obtenerse con la serie de instrucciones

```
celsius = 5;     // valor Celsius inicial
while (celsius <= 50)
{
  fahren = (9.0/5.0) * celsius + 32.0;
  cout << celsius << "      "
       << fahren;
  celsius = celsius + 5;
}
```

Como antes, la instrucción `while` consiste en todo desde la palabra `while` hasta la llave de cierre de la instrucción compuesta. Antes de introducir el ciclo `while` nos hemos asegurado de asignar un valor al contador que se va a evaluar, y hay una instrucción para alterar el valor del contador dentro del ciclo (en incrementos de 5 grados Celsius) para asegurar una salida del ciclo `while`. El programa 5.4 ilustra el uso de código similar en un programa completo.

Programa 5.4

```
#include <iostream>
#include <iomanip>
using namespace std;

// un programa para convertir Celsius en Fahrenheit
int main()
{
  const int MAX_CELSIUS = 50;
  const int VAL_INICIAL = 5;
  const int TAMAÑO_PASO = 5;
  int celsius;
  double fahren;

  cout << "GRADOS     GRADOS\n"
       << "CELSIUS  FAHRENHEIT\n"
       << "-------  ----------\n";

  celsius = VAL_INICIAL;

    // establecer los formatos de salida solo para numeros en punto flotante
  cout << setiosflags(ios::showpoint)
       << setprecision(2);
```

(Continúa)

(Continuación)

```
    while (celsius <= MAX_CELSIUS)
    {
      fahren = (9.0/5.0) * celsius + 32.0;
      cout << setw(4)  << celsius << fixed
           << setw(13) << fahren << endl;
      celsius = celsius + TAMAÑO_PASO;
    }

    return 0;
}
```

El despliegue obtenido cuando se ejecuta el programa 5.4 es

```
GRADOS    GRADOS
CELSIUS   FAHRENHEIT
-------   ----------
   5         41.00
  10         50.00
  15         59.00
  20         68.00
  25         77.00
  30         86.00
  35         95.00
  40        104.00
  45        113.00
  50        122.00
```

EJercicios 5.2

1. Vuelva a escribir el programa 5.1 para imprimir los números 2 a 10 en incrementos de dos. La salida de su programa deberá ser

   ```
   2  4  6  8  10
   ```

2. Vuelva a escribir el programa 5.4 para producir una tabla que empiece en un valor Celsius de −10 y termine con un valor Celsius de 60, en incrementos de diez grados.

3. a. Para el siguiente programa determine el número total de elementos desplegados. Determine además el primer y último números impresos.

```
#include <iostream>
using namespace std;

int main()
{
  int num = 0;
  while (num <= 20)
  {
    num++;
    cout << num << " ";
  }

  return 0;
}
```

b. Introduzca y ejecute el programa del ejercicio 3a en una computadora para verificar sus respuestas al ejercicio.

c. ¿Cómo afectaría a la salida si las dos instrucciones dentro de la instrucción compuesta se invirtieran (es decir, si la llamada a `cout` se hiciera antes de la instrucción `++ num`)?

4. Escriba un programa en C++ que convierta galones en litros. El programa deberá desplegar galones de 10 a 20 en incrementos de un galón y los litros equivalentes correspondientes. Use la relación que 1 galón contiene 3.785 litros.

5. Escriba un programa en C++ que convierta pies en metros. El programa deberá desplegar pies de 3 a 30 en incrementos de tres pies y los metros equivalentes correspondientes. Use la relación que hay 3.28 pies en un metro.

6. Una máquina comprada por 28 000 dólares se deprecia 4000 dólares por año durante siete años. Escriba y ejecute un programa en C++ que calcule y despliegue una tabla de depreciación para siete años. La tabla deberá tener la forma

Año	Depreciación	Valor al final del año	Depreciación acumulada
1	4000	24000	4000
2	4000	20000	8000
3	4000	16000	12000
4	4000	12000	16000
5	4000	8000	20000
6	4000	4000	24000
7	4000	0	28000

7. Un automóvil viaja a una velocidad promedio de 55 millas por hora durante cuatro horas. Escriba un programa en C++ que despliegue la distancia, en millas, que el automóvil ha recorrido después de 1, 2, etc., horas hasta el final del viaje.

8. a. Una fórmula de conversión aproximada para convertir temperatura Fahrenheit en Celsius es

Celsius = (Fahrenheit – 30) / 2

Usando esta fórmula, y empezando con una temperatura Fahrenheit de cero grados, escriba un programa C++ que determine cuándo la temperatura Celsius equivalente aproximada difiere del valor equivalente exacto por más de cuatro grados. (*Sugerencia:* Use un ciclo `while` que termine cuando la diferencia entre los equivalentes Celsius aproximados y exactos exceda de cuatro grados.)

b. Usando la fórmula de conversión a Celsius aproximada dada en el ejercicio 8a, escriba un programa en C++ que produzca una tabla de temperaturas Fahrenheit, temperaturas Celsius equivalentes exactas y la diferencia entre los valores Celsius equivalentes correctos y aproximados. La tabla deberá comenzar en cero grados Fahrenheit, usar incrementos de dos grados Fahrenheit y terminar cuando la diferencia entre los valores exactos y aproximados difiera por más de cuatro grados.

9. El valor del número de Euler, *e*, puede aproximarse usando la fórmula

$e = 1 + 1/1! + ½! + 1/3! + 1/4! + 1/5! + \cdots$

Usando esta fórmula, escriba un programa en C++ que aproxime el valor de *e* usando un ciclo `while` que termine cuando la diferencia entre dos aproximaciones sucesivas difiera por menos que 10 e–9.

10. El valor del seno de *x* puede aproximarse usando la fórmula

$$\text{sen } x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \cdots$$

Utilizando esta fórmula, determine cuántos términos se necesitan para aproximar el valor devuelto por la función `sin()` intrínseca con un error menor que 1 e–6, cuando *x* = 30 grados. (*Sugerencias:* Use un ciclo `while` que termine cuando la diferencia entre el valor devuelto por la función `sin()` intrínseca y la aproximación es menor que 1 e–6. Además observe que *x* debe convertirse primero a una medida en radianes y que el signo alternante en la serie aproximada puede determinarse como (–1)*(*n* + 1) donde *n* es el número de términos usados en la aproximación.)

5.3 CICLOS while INTERACTIVOS

La combinación de la introducción de datos interactivos con las capacidades de repetición de la instrucción `while` produce programas muy adaptables y potentes. Para entender el concepto implicado, considere el programa 5.5, donde una instrucción `while` se usa para aceptar y luego desplegar cuatro números introducidos por el usuario, uno a la vez. Aunque usa una idea muy simple, el programa resalta los conceptos del flujo de control necesarios para producir programas más útiles.

Programa 5.5

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  const int MAXNUMS = 4;
  int cuenta;
  double num;

  cout << "\nEste programa le pedira que introduzca "
       << MAXNUMS << " numeros.\n";
  cuenta = 1;
  while (cuenta <= MAXNUMS)
  {
    cout << "\nIntroduzca un numero: ";
    cin  >> num;
    cout << "El número introducido es " << num;
    count++;
  }
  cout << endl;

  return 0;
}
```

La siguiente es una muestra de la ejecución del programa 5.5.

```
Este programa le pedira que introduzca 4 numeros.

Introduzca un numero: 26.2
El número introducido es 26.2
Introduzca un numero: 5
El número introducido es 5
Introduzca un numero: 103.456
El número introducido es 103.456
Introduzca un numero: 1267.89
El número introducido es 1267.89
```

Revisemos el programa para entender con claridad cómo se produjo la salida. El primer mensaje desplegado es causado por la ejecución del primer objeto `cout` invocado. Esta llamada está afuera de la instrucción `while` y es anterior a ella, así que se ejecuta una vez antes que cualquier instrucción en el ciclo `while`.

Una vez que se introduce el ciclo `while`, las instrucciones dentro de la instrucción compuesta son ejecutadas mientras la condición probada sea verdadera. La primera vez que pasa por la instrucción compuesta, se despliega el mensaje `Introduzca un numero:`. Entonces el programa llama a `cin`, que obliga a la computadora a esperar que se introduzca un número en el teclado. Una vez que se ha introducido un número y se ha oprimido la tecla Retorno o Entrar, el objeto `cout` despliega el número. Entonces la variable `cuenta` se incrementa en uno. Este proceso se repite hasta que se han realizado cuatro pasos a través del ciclo y el valor de `cuenta` es 5. Cada paso causa que se despliegue el mensaje `Introduzca un numero:` causa que se haga una llamada a `cin` y causa que se despliegue el mensaje `El numero introducido` es. La figura 5.4 ilustra este flujo de control.

En lugar de tan sólo desplegar los números introducidos, el programa 5.5 puede modificarse para usar los datos introducidos. Por ejemplo, hagamos que se sumen los números introducidos y se despliegue el total. Para hacer esto, se debe tener mucho cuidado con la forma en que se suman los números, en vista que se usa la misma variable, `num`, para cada número introducido. Debido a esto la introducción de un número nuevo en el programa 5.5 causa de manera automática que se pierda el número previo almacenado en `num`. Por tanto, cada número introducido debe sumarse al total antes que se introduzca otro número. La secuencia requerida es

Introducir un número
Sumar el número al total

¿Cómo se suma un número individual a un total? Una instrucción como `total = total + num` hace el trabajo a la perfección. Ésta es la instrucción de acumulación introducida en la sección 3.1. Después de introducir cada número, la instrucción de acumulación suma el número en el total, como se ilustra en la figura 5.5. El flujo de control completo requerido para sumar los números se ilustra en la figura 5.6.

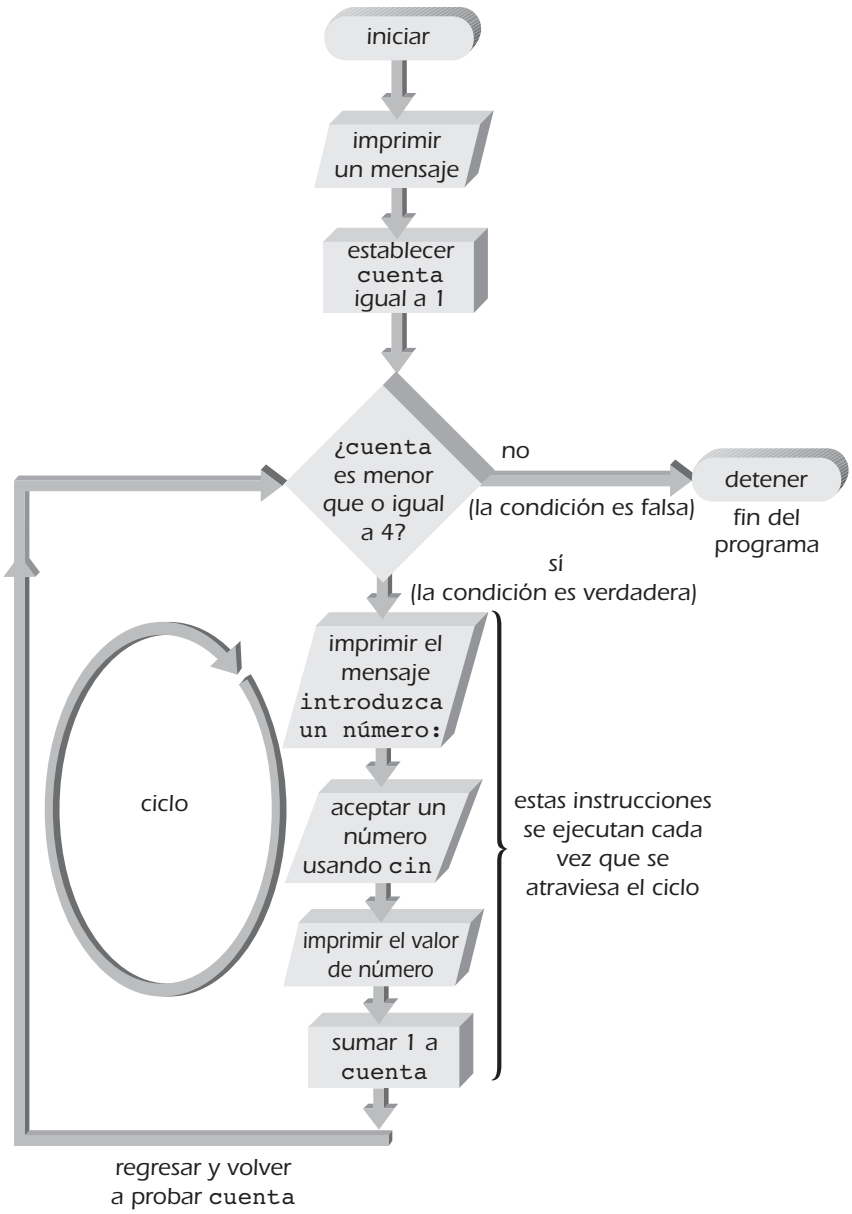


Figura 5.4 *Diagrama de flujo de control para el programa 5.5.*

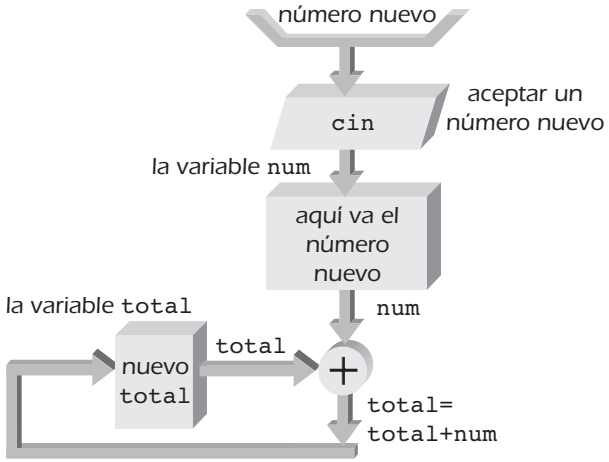


Figura 5.5 *Aceptar y sumar un número a un total.*

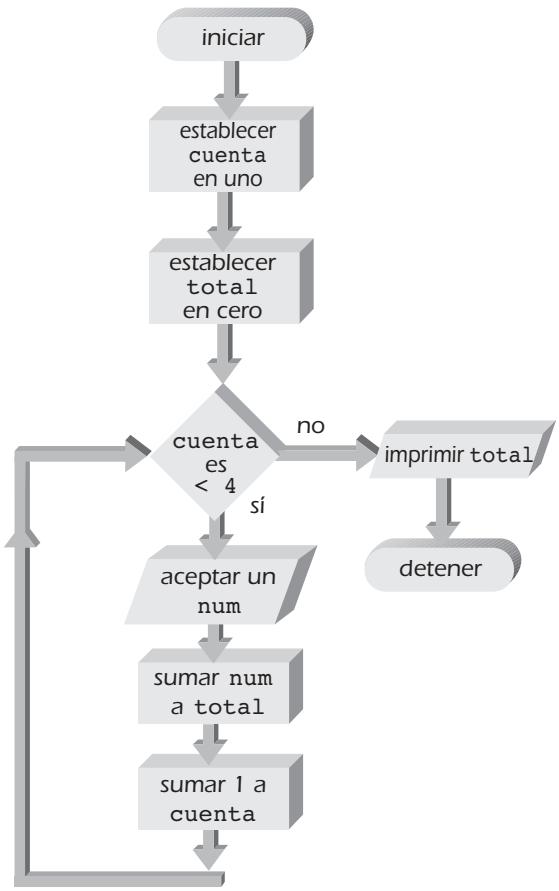


Figura 5.6 *Flujo de control de acumulación.*

Al revisar la figura 5.6, observe que se ha previsto establecer al inicio el total en cero antes que se introduzca el ciclo `while`. Si se quiere despejar el total dentro del ciclo `while`, se establecería en cero cada vez que se ejecutara el ciclo y se borraría cualquier valor almacenado con anterioridad.

El programa 5.6 incorpora las modificaciones necesarias al programa 5.5 para obtener el total de los números introducidos. Como se indica en el diagrama de flujo mostrado en la figura 5.6, la instrucción `total = total + num;` se coloca inmediatamente después de la instrucción `cin`. Poner la instrucción de acumulación en este punto en el programa asegura que el número introducido es "capturado" de inmediato en el total.

Programa 5.6

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  const int MAXNUMS = 4;
  int cuenta;
  double num, total;

  cout << "\nEste programa le pedira que introduzca "
       << MAXNUMS << " numeros.\n";
  cuenta = 1;
  total = 0;

  while (cuenta <= MAXNUMS)
  {
    cout << "\nIntroduzca un numero: ";
    cin  >> num;
    total = total + num;
    cout << "El total es ahora " << setprecision(7) << total;
    cuenta++;
  }

  cout << "\nEl total final es " << setprecision(7) << total << endl;

  return 0;
}
```

Revisemos el programa 5.6. La variable `total` fue creada para almacenar el total de los números introducidos. Antes de introducir la instrucción `while` el valor de `total` se establece en cero. Esto asegura que cualquier valor previo presente en la ubicación o ubicaciones de almacenamiento asignadas a la variable `total` es borrado. Dentro del ciclo `while` la instrucción `total = total + num;` se usa para sumar el valor del número introducido a total. Conforme se introduce cada valor, se suma al `total` existente para crear un total nuevo. Por tanto, `total` se vuelve un subtotal corriente de todos los valores introducidos. Sólo cuando todos los números son introducidos `total` contiene la suma final de todos los números. Después que termina el ciclo `while`, se usa una instrucción `cout` para desplegar esta suma.

Usando los mismos datos que se introdujeron en la muestra de ejecución para el programa 5.5, se hizo la siguiente muestra de ejecución del programa 5.6:

```
Este programa le pedira que introduzca 4 numeros.

Introduzca un numero: 26.2
El número introducido es 26.2
Introduzca un numero: 5
El total es ahora 31.2
Introduzca un numero: 103.456
El total es ahora 134.656
Introduzca un numero: 1267.89
El total es ahora 1402.546

El total final es 1402.546
```

Habiendo usado una instrucción de asignación de acumulación para sumar los números introducidos, ahora se puede ir más allá y calcular el promedio de los números. ¿Dónde se calculará el promedio, dentro del ciclo `while` o fuera de él?

En el presente caso, calcular un promedio requiere que estén disponibles tanto una suma final como el número de elementos en esa suma. Entonces se calcula el promedio dividiendo la suma final entre el número de elementos. En este punto, se debe preguntar: "¿En qué punto en el programa está disponible la suma correcta y en qué punto está disponible el número de elementos?" Al revisar el programa 5.6 se puede observar que la suma correcta necesaria para calcular el promedio está disponible después que termina el ciclo `while`. De hecho, el propósito del ciclo `while` es asegurar que los números son introducidos y sumados en forma correcta para producir una suma correcta. Después que termina el ciclo, también tenemos una cuenta del número de elementos usados en la suma. Sin embargo, debido a la forma en que fue construido el ciclo `while`, el número en `cuenta` (5) cuando termina el ciclo es 1 más que el número de elementos (4) usados para obtener el total. Sabiendo esto, tan sólo se resta uno de cuenta antes de utilizarlo para determinar el promedio. Con esto como antecedentes, veamos si puede leer y entender el programa 5.7.

Programa 5.7

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  const int MAXNUMS = 4;
  int cuenta;
  double num, total, promedio;

  cout << "\nEste programa le pedira que introduzca "
       << MAXNUMS << " numeros.\n";
  cuenta = 1;
  total = 0;

  while (cuenta <= MAXNUMS)
  {
    cout << "Introduzca un número: ";
    cin  >> num;
    total = total + num;
    cuenta++;
  }

  cuenta--;
  promedio = total / cuenta;
  cout << "\nEl promedio de los numeros es " << promedio << endl;

  return 0;
}
```

El programa 5.7 es casi idéntico al programa 5.6, excepto por el cálculo del promedio. También se ha eliminado el despliegue constante del total dentro y después del ciclo `while`. El ciclo en el programa 5.7 se usa para introducir y sumar cuatro números. Inmediatamente después de salir del ciclo, se calcula y se despliega el promedio.

A continuación se presenta una muestra de ejecución del programa 5.7:

```
Este programa le pedira que introduzca 4 numeros.
Introduzca un numero: 26.2
Introduzca un numero: 5
Introduzca un numero: 103.456
Introduzca un numero: 1267.89

El promedio de los numeros es 350.637
```

Centinelas

Todos los ciclos que hemos creado hasta ahora han sido ejemplos de ciclos de cuenta fija, donde se ha usado un contador para controlar el número de iteraciones del ciclo. Por medio de una instrucción `while`, también pueden construirse ciclos de condición variable. Por ejemplo, cuando se introducen calificaciones puede ser que no se desee contar el número de calificaciones que se introducirán, sino que se preferiría introducirlas en forma continua y, al final, introducir un valor de datos especial para señalar el final de la introducción de datos.

En la programación por computadora, los valores de datos usados para señalar el inicio o el fin de una serie de datos se llaman **centinelas**. Los valores centinelas deben seleccionarse, por supuesto, de modo que no entren en conflicto con los valores de datos legítimos. Por ejemplo, si se fuera a elaborar un programa para procesar las calificaciones de un estudiante, y suponiendo que no se dan créditos extras que pudieran producir una calificación mayor que 100, podría usarse cualquier calificación mayor que 100 como un valor centinela. El programa 5.8 ilustra este concepto. En el programa 5.8 se solicitan y aceptan datos en forma continua hasta que se introduce un número mayor que 100. La introducción de un número mayor que 100 alerta al programa para que salga del ciclo `while` y despliegue la suma de los números introducidos.

A continuación puede verse una muestra de ejecución usando el programa 5.8. En tanto se introduzcan calificaciones menores que o iguales a 100, el programa continúa solicitando y aceptando datos adicionales. Cuando se introduce un número menor que o igual a 100, el programa suma este número al total. Cuando se introduce un número mayor que 100, se sale del ciclo y se despliega la suma de las calificaciones que se introdujeron.

```
Para dejar de introducir calificaciones, introduzca
 cualquier numero mayor que 100.

Introduzca una calificacion: 95
Introduzca una calificacion: 100
Introduzca una calificacion: 82
Introduzca una calificacion: 101

El total de las calificaciones es 277
```

Programa 5.8

```
#include <iostream>
using namespace std;

int main()
{
  const int CALIF_ALTA = 100;
  double calificacion, total;

  calificacion = 0;
  total = 0;
  cout << "\nPara dejar de introducir calificaciones, introduzca cualquier numero";
  cout << "\n mayor que 100.\n\n";

  while (calificacion <= CALIF_ALTA)
  {
    total = total + calificacion;
    cout << "Introduzca una calificacion: ";
    cin  >> calificacion;
  }

  cout << "\nEl total de las calificaciones es " << total << endl;
  return 0;
}
```

Instrucciones `break` y `continue`

Dos instrucciones útiles en conexión con las instrucciones de repetición son las instrucciones **`break`** y **`continue`**. Se ha encontrado la instrucción `break` en relación con la instrucción `switch`. La sintaxis de esta instrucción es

break;

Una instrucción `break`, como su nombre implica, obliga a una interrupción inmediata, o salida, de `switch`, `while` y las instrucciones `for` y `do-while` presentadas en las siguientes secciones.

Por ejemplo, la ejecución del siguiente ciclo `while` es terminada de inmediato si se introduce un número mayor que 76.

```
while(cuenta <= 10)
{
  cout << "Introduzca un número: ";
  cin  >> num;
  if (num > 76)
  {
    cout << "¡Perdiste!\n";
    break;        // interrumpe el ciclo
  }
  else
    cout << "¡Sigue intentandolo!\n";
  cuenta++
}
// break salta hasta aquí
```

La instrucción `break` viola los principios puros de la programación estructurada porque proporciona una segunda salida no estándar de un ciclo. No obstante, la instrucción `break` es muy útil y valiosa para interrumpir ciclos cuando se detecta una condición inusual. La instrucción `break` también se usa para salir de una instrucción `switch`, pero esto se debe a que el caso deseado se ha detectado y procesado.

La instrucción `continue` es similar a la instrucción `break` pero sólo se aplica a ciclos creados con instrucciones `while`, `do-while` y `for`. El formato general de una instrucción `continue` es

continue;

Cuando `continue` se encuentra en un `ciclo`, la siguiente iteración del ciclo comienza de inmediato. Para ciclos `while` esto significa que la ejecución es transferida de manera automática al principio del ciclo y se inicia una reevaluación de la expresión probada. Aunque la instrucción `continue` no tiene un efecto directo en una instrucción `switch`, puede incluirse dentro de una instrucción `switch` que esté contenida en un `ciclo`. Aquí el efecto de `continue` es el mismo: se comienza la siguiente iteración del ciclo.

Como una regla general, la instrucción `continue` es menos útil que la instrucción `break`, pero es conveniente para saltarse datos que no deberían ser procesados mientras permanecen en un ciclo. Por ejemplo, las calificaciones inválidas simplemente son ignoradas en la siguiente sección de código y sólo se suman las calificaciones válidas al total:[2]

[2]Sin embargo, la instrucción `continue` no es esencial y la selección podría haberse escrito como

```
if (calificacion >= 0 && calificacion <= 100)
{
  total = total + calificacion;
  cuenta++;
}
```

```
while (cuenta < 30)
{
   cout << "Introduzca una calificacion: ";
   cin  >> calificacion;
   if(calificacion < 0 || calificacion > 100)
     continue;
   total = total + calificacion;
   cuenta++;
}
```

La instrucción nula

Todas las instrucciones deben terminarse con un punto y coma. Un punto y coma sin nada que lo preceda también es una instrucción válida, llamada instrucción nula. Por tanto, la instrucción

```
;
```

es una instrucción nula. Ésta es una instrucción que no hace nada y que se utiliza donde se requiere una instrucción desde el punto de vista sintáctico, pero no invoca ninguna acción. Las instrucciones nulas se usan de manera típica con instrucciones `while` o `for`. Un ejemplo de una instrucción `for` que usa una instrucción nula se encuentra en el programa 5.10c en la siguiente sección.

Ejercicios 5.3

1. Vuelva a escribir el programa 5.6 para calcular el total de ocho números.

2. Vuelva a escribir el programa 5.6 para desplegar el indicador:

```
Por favor introduzca el número total de valores de datos
que se van a sumar:
```

En respuesta a este indicador, el programa deberá aceptar un número introducido por un usuario y luego usar este número para controlar el número de veces que se ejecuta el ciclo `while`. Por tanto, si el usuario introduce *5* en respuesta al indicador, el programa deberá solicitar la introducción de cinco números y desplegar el total después que se hayan introducido los cinco números.

3. a. Escriba un programa en C++ para convertir grados Celsius a Fahrenheit. El programa deberá solicitar el valor Celsius inicial, el número de conversiones que se harán y el incremento entre valores Celsius. El despliegue deberá tener encabezados apropiados y enlistar el valor Celsius y el valor Fahrenheit correspondiente. Use la relación

$$Fahrenheit = (9.0 / 5.0) * Celsius + 32.0$$

b. Ejecute el programa escrito en el ejercicio 3a en una computadora. Verifique que su programa empiece en el valor Celsius inicial correcto y contenga el número exacto de conversiones especificado en sus datos de entrada.

4. a. Modifique el programa escrito en el ejercicio 3a para solicitar el valor Celsius inicial, el valor Celsius final y el incremento. Por tanto, en lugar que la condición verifique una cuenta fija, verificará el valor Celsius final.

b. Ejecute el programa escrito en el ejercicio 4a en una computadora. Verifique que su salida comience en el valor inicial correcto y termine en el valor final correcto.

5. Vuelva a escribir el programa 5.7 para calcular el promedio de diez números.

6. Vuelva a escribir el programa 5.7 para desplegar el siguiente indicador:

```
Por favor introduzca el número total de valores de datos
que se van a promediar:
```

En respuesta a este indicador, el programa deberá aceptar un número introducido por un usuario y luego usar este número para controlar el número de veces que se ejecutará el ciclo `while`. Por tanto, si el usuario introduce 6 en respuesta al indicador, el programa deberá solicitar la introducción de seis números y desplegar el promedio de los siguientes seis números introducidos.

7. Por error, un programador puso la instrucción `promedio = total / cuenta;` dentro del ciclo `while` inmediatamente después de la instrucción `total = total + num;` en el programa 5.7. Por tanto, el ciclo while se vuelve

```
while (cuenta <= MAXNUMS)
{
  cout << "Introduzca un numero: ";
  cin  >> num;
  total = total + num;
  promedio = total / cuenta;
  cuenta++;
}
```

¿El programa producirá el resultado correcto con este ciclo `while`?
Desde una perspectiva de programación, ¿cuál ciclo `while` es mejor usar y por qué?

8. Una serie aritmética se define por

$$a + (a + d) + (a + 2d) + (a + 3d) + \cdots + [(a + (n - 1)d)]$$

donde a es el primer término, d es la "diferencia común" y n es el número de términos que se van a sumar. Usando esta información, escriba un programa en C++ que use un ciclo `while` para desplegar cada término y para determinar la suma de la serie aritmética si se tiene que $a = 1$, $d = 3$ y $n = 100$. Asegúrese que su programa despliegue el valor que ha calculado.

9. Una serie geométrica se define por

$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1}$$

donde a es el primer término, r es la "razón común" y n es el número de términos en la serie. Usando esta información, escriba un programa en C++ que utilice un ciclo `while` para desplegar cada término y para determinar la suma de una serie geométrica si se tiene que $a = 1$, $r = .5$ y $n = 10$. Asegúrese que su programa despliega el valor que se ha calculado.

10. Además del promedio aritmético de un conjunto de números, se puede calcular una media geométrica y una media armónica. La media geométrica de un conjunto de n números $x_1, x_2, \ldots x_n$ se define como

$$\sqrt[n]{x_1 \cdot x_2 \cdot \cdots \cdot x_n}$$

y la media armónica como

$$\frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\cdots+\frac{1}{x_n}}$$

Usando estas fórmulas, escriba un programa en C++ que continúe aceptando números hasta que se introduzca el número 999 y luego calcule y despliegue tanto la media geométrica como la armónica de los números introducidos. (*Sugerencia:* Será necesario que su programa cuente en forma correcta el número de valores introducidos.)

11. a. Los siguientes datos se recolectaron en un viaje reciente en automóvil.

Millaje	Gallones
22495	Full tank
22841	12.2
23185	11.3
23400	10.5
23772	11.0
24055	12.2
24434	14.7
24804	14.3
25276	15.2

Escriba un programa en C++ que acepte un valor de millaje y galones y calcule las millas por galón (mpg) logradas para ese segmento del viaje. Las millas por galón se obtienen como la diferencia en millaje entre llenadas del tanque dividido entre el número de galones de gasolina utilizados desde que se llenó el tanque.

b. Modifique el programa escrito para el ejercicio 11a para calcular y desplegar adicionalmente las mpg acumuladas después de cada llenada de tanque. Las mpg acumuladas se calculan como la diferencia entre el millaje en cada llenada y el millaje al principio del viaje dividido entre la suma de los galones usados hasta ese punto en el viaje.

5.4 Ciclos `for`

En C++, un **`ciclo for`** se construye usando una instrucción `for`. Esta instrucción realiza las mismas funciones que la instrucción `while`, pero usa una forma diferente. En muchas situaciones, en especial aquellas que usan una condición de cuenta fija, el formato de la instrucción `for` es más fácil de usar que su instrucción `while` equivalente.

La sintaxis de la instrucción `for` es

```
for (lista de inicialización; expresión; lista de alteración)
    instrucción;
```

Aunque la instrucción `for` parece un poco complicada, en realidad es bastante simple si se considera cada una de sus partes por separado.

Dentro del paréntesis de la instrucción `for` hay tres elementos, separados por puntos y comas. Cada uno de estos elementos es opcional y puede describirse de manera individual, pero los puntos y comas deben estar presentes siempre.

En su forma más común, la lista de *inicialización* consiste de una sola instrucción usada para establecer el comienzo (valor inicial) de un contador, la *expresión* contiene el valor máximo o mínimo que puede tener el contador y determina cuándo se termina el ciclo, y la lista de *alteración* proporciona el valor de incremento que se suma o se resta del contador cada vez que se ejecuta el ciclo. Son ejemplos de instrucciones `for` simples que tienen esta forma

```
for (cuenta = 1; cuenta < 10; cuenta = cuenta + 1)
  cout << cuenta;
```

y

```
for (i = 5; i <= 15; i = i + 2)
  cout << i;
```

En la primera instrucción `for`, la variable contadora se llama `cuenta`, el valor inicial asignado a `cuenta` es 1, el ciclo continúa en tanto el valor en `cuenta` sea menor que 10, y el valor de `cuenta` se incrementa en uno cada vez que pasa por el ciclo. En la siguiente instrucción `for`, la variable contadora es nombrada `i`, el valor inicial asignado a `i` es 5, el ciclo continúa en tanto el valor de i sea menor que o igual a 15 y el valor de `i` es incrementado en 2 cada vez que pasa por el ciclo. En ambos casos se usa una instrucción `cout` para desplegar el valor del contador. Otro ejemplo de un ciclo `for` está dado en el programa 5.9.

Cuando se ejecuta el programa 5.9, se produce el siguiente despliegue:

```
NUMERO     RAIZ CUADRADA
------     -------------
   1          1.00000
   2          1.41421
   3          1.73205
   4          2.00000
   5          2.23607
```

Las primeras dos líneas desplegadas por el programa son producidas por las dos instrucciones `cout` colocadas antes de la instrucción `for`. La salida restante es producida por el ciclo `for`. Este ciclo comienza con la instrucción `for` y es ejecutado como sigue.

El valor inicial asignado a la variable contadora `cuenta` es 1. En vista que el valor en cuenta no excede del valor final de 5, la ejecución de la instrucción `cout` dentro del ciclo produce el despliegue

```
1          1.00000
```

Entonces se transfiere el control de nuevo a la instrucción `for`, la cual entonces incrementa el valor en `cuenta` a 2, y el ciclo se repite, produciendo el despliegue

```
2          1.41421
```

Programa 5.9

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
  const int CUENTAMAX = 5;
  int cuenta;

  cout << "NUMERO    RAIZ CUADRADA\n";
  cout << "------    -----------\n";

  cout << setiosflags(ios::showpoint);
  for (cuenta = 1; cuenta <= CUENTAMAX; cuenta++)
    cout << setw(4) << cuenta
         << setw(15) << sqrt(double(cuenta)) << endl;

  return 0;
}
```

Este proceso continúa hasta que el valor en cuenta excede el valor final de 5, produciendo la tabla de salida completa. Con propósitos de comparación, un ciclo `while` equivalente al ciclo `for` contenido en el programa 5.9 es:

```
cuenta = 1
while (cuenta <= MAXCOUNT)
{
  cout << setw(4) << cuenta
       << setw(15) << sqrt(cuenta) << endl;
  cuenta++;
}
```

Como se puede observar en este ejemplo, la diferencia entre los ciclos `for` y `while` es la colocación de los elementos de inicialización, prueba de condición e incremento. El agrupamiento de estos elementos en la instrucción `for` es muy conveniente cuando deben construirse ciclos de cuenta fija. Vea si puede determinar la salida producida por el programa 5.10.

Programa 5.10

```
#include <iostream>
using namespace std;

int main()
{
  int cuenta;

  for (cuenta = 2; cuenta <= 20; cuenta = cuenta + 2)
    cout << cuenta << "  ";

  return 0;
}
```

¿Pudo encontrar la respuesta? El ciclo comienza con una cuenta inicializada en 2, se detiene cuando cuenta excede de 20 e incrementa cuenta en pasos de 2. La salida del programa 5.10 es

```
2  4  6  8  10  12  14  16  18  20
```

La instrucción `for` no requiere que cualquiera de los elementos entre paréntesis esté presente o que sean usados para inicializar o alterar los valores en las instrucciones de expresión. Sin embargo, los dos puntos y comas deben estar presentes dentro del paréntesis de `for`. Por ejemplo, la construcción `for ( ; cuenta <= 20 ;)` es válida.

Si falta la lista de inicialización, el paso de inicialización se omite cuando se ejecuta la instrucción `for`. Esto significa, por supuesto, que el programador debe proporcionar las inicializaciones requeridas antes que se encuentre la instrucción `for`. Del mismo modo, si falta la lista de alteración, cualesquiera expresiones necesarias para alterar la evaluación de la expresión probada deben incluirse en forma directa dentro de la parte de instrucción del ciclo. La instrucción `for` sólo asegura que todas las expresiones en la lista de inicialización se ejecutan una vez, antes de la evaluación de la expresión probada, y que todas las expresiones en la lista de alteración se ejecuten al final del ciclo antes que se vuelva a verificar la expresión probada. Por tanto, el programa 5.10 puede volverse a escribir en cualquiera de las tres formas mostradas en los programas 5.10a, 5.10b y 5.10c.

Programa 5.10a

```
#include <iostream>
using namespace std;

int main()
{
  int cuenta;

  cuenta = 2;    // inicializador fuera de la instrucción for
  for ( ; cuenta <= 20; cuenta = cuenta + 2)
    cout << cuenta << "  ";

  return 0;
}
```

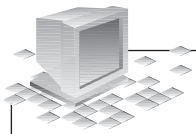
Programa 5.10b

```
#include <iostream>
using namespace std;

int main()
{
  int cuenta;

  cuenta = 2;    // inicializador fuera del ciclo for
  for( ; count <= 20; )
  {
    cout << cuenta << "  ";
    cuenta = cuenta + 2;     // instrucción de alteración
  }

  return 0;
}
```



Programa 5.10c

```
#include <iostream>
using namespace std;

int main()   // todas las expresiones dentro de los paréntesis de for
{
  int cuenta;

  for (cuenta = 2; cuenta <= 20; cout << cuenta << "  ", cuenta = cuenta + 2);

  return 0;
}
```

En el programa 5.10a, `cuenta` es inicializada fuera de la instrucción `for` y la primera lista dentro del paréntesis se deja en blanco. En el programa 5.10b, tanto la lista de inicialización como la lista de alteración son eliminadas de dentro del paréntesis. El programa 5.10b también usa una instrucción compuesta dentro del ciclo `for`, con la instrucción de alteración de la expresión incluida en la instrucción compuesta. Por último, el programa 5.10c ha incluido todos los elementos dentro del paréntesis, de modo que no hay necesidad de ninguna instrucción útil después del paréntesis. Aquí la instrucción nula satisface el requerimiento sintáctico de que una instrucción siga al paréntesis de `for`.

Observe también en el programa 5.10c que la lista de alteración (último conjunto de elementos en el paréntesis) consiste en dos elementos, y que se ha usado una coma para separar estos elementos. El uso de comas para separar elementos en las listas de inicialización y de alteración se requiere si cualquiera de estas dos listas contiene más de un elemento. Por último, nótese el hecho que los programas 5.10a, 5.10b y 5.10c son inferiores al programa 5.10, y aunque pueda encontrarlos en su carrera de programación, no debería usarlos. Agregar elementos distintos a las variables de control del ciclo y sus condiciones de actualización dentro de la instrucción `for` tiende a confundir su legibilidad y puede introducir efectos indeseados. Mantener "limpia" la estructura de control del ciclo, como se hizo en el programa 5.10, es importante y es una buena práctica de programación.

Aunque las listas de inicialización y alteración pueden omitirse de una instrucción `for`, omitir la expresión probada produce un ciclo infinito. Por ejemplo, un ciclo así es creado por la instrucción

```
for (cuenta = 2; ; cuenta = cuenta + 1)
  cout << cuenta;
```

Como con la instrucción `while`, pueden usarse las instrucciones `break` y `continue` dentro de un ciclo `for`. Un `break` obliga a una salida inmediata del ciclo `for`, como lo hace en el ciclo `while`. Sin embargo, un `continue` obliga a que el control se pase a la lista de alteración en una instrucción `for`, después de lo cual se reevalúa la expresión probada. Esto difiere de la acción de un `continue` en una instrucción `while`, donde el control se pasa en forma directa a la reevaluación de la expresión probada.

Punto de Información

Dónde colocar las llaves de apertura

Hay dos estilos para escribir ciclos `for` que son usados por programadores profesionales en C++. Estos estilos sólo se utilizan cuando el ciclo `for` contiene una instrucción compuesta. El estilo ilustrado y usado en el texto toma la forma

```
for (expresión)
{
   la instrucción compuesta va aquí
}
```

Un estilo igual de aceptable que ha sido usado por muchos programadores coloca la llave inicial de la instrucción compuesta en la primera línea. Usando este estilo, un ciclo `for` aparece como

```
for (expresión) {
   la instrucción compuesta va aquí
}
```

La ventaja del primer estilo es que las llaves se alinean una bajo la otra, facilitando localizar los pares de llaves. La ventaja del segundo estilo es que hace más compacto el código y ahorra una línea, permitiendo que se vea más código en la misma área de despliegue. Ambos estilos se usan pero casi nunca se mezclan. Seleccione cualquier estilo que le atraiga y sea consistente en su uso. Como siempre, la sangría que use dentro de la instrucción compuesta (dos o cuatro espacios, o un tabulador) también deberá ser consistente en todos sus programas. La combinación de estilos que seleccione se convierte en una "firma" para su trabajo de programación.

La figura 5.7 ilustra el funcionamiento interno de un ciclo `for`. Como se muestra, cuando el ciclo `for` es completado, el control se transfiere a la primera instrucción ejecutable que se encuentre después del ciclo. Para evitar ilustrar siempre estos pasos de manera forzada, se dispone de un conjunto simplificado de símbolos de diagrama de flujo para describir los ciclos `for`. Usando el hecho que una instrucción for puede representarse con el símbolo de diagrama de flujo

los ciclos `for` completos pueden ilustrarse en forma alternativa como se muestra en la figura 5.8.

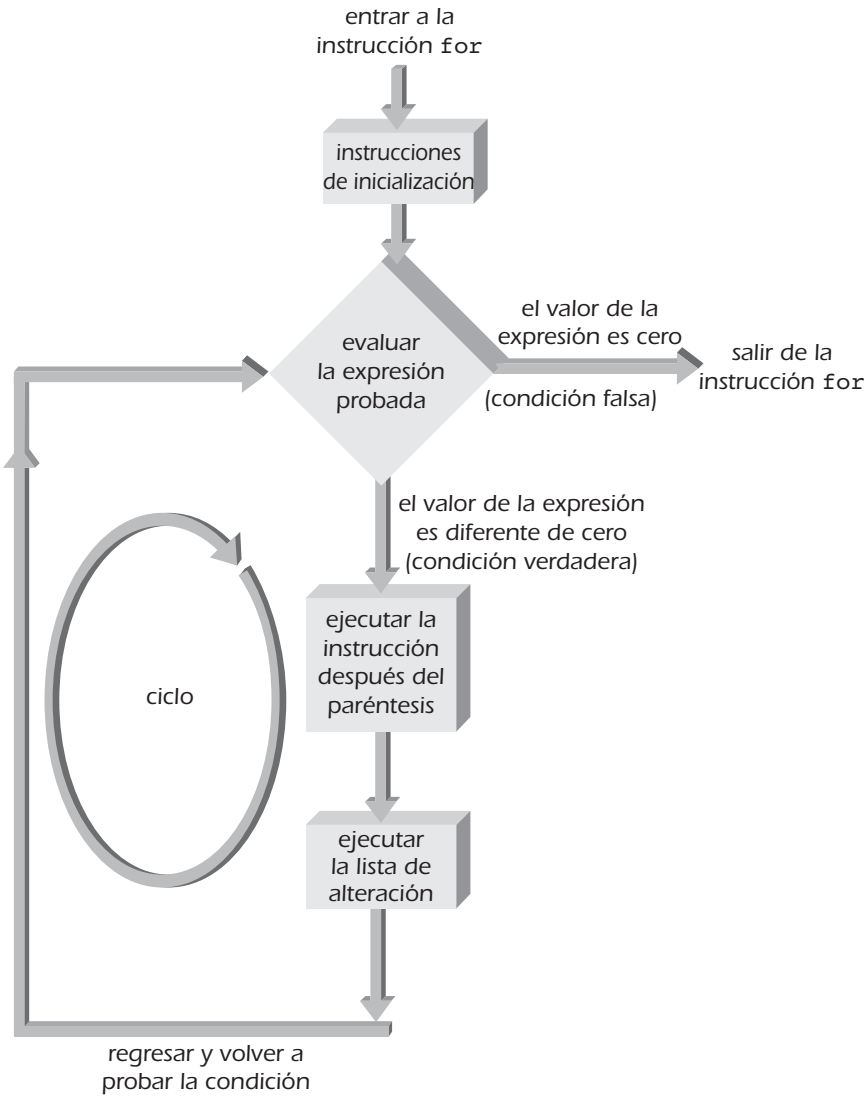


Figura 5.7 Diagrama de flujo de un ciclo `for`.

Punto de Información

¿Se debe utilizar un ciclo `for` o uno `while`?

Una pregunta que plantean por lo común los programadores principiantes es cuál estructura de ciclo deberían usar, un ciclo `for` o uno `while`. Ésta es una buena pregunta porque ambas estructuras de ciclo son ciclos de prueba preliminar, que, en C++, pueden usarse para construir tanto ciclos de cuenta fija como ciclos de condición variable.

En casi todos los otros lenguajes de computadora, incluyendo Visual Basic y Pascal, la respuesta es relativamente sencilla, porque la instrucción `for` sólo puede usarse para construir ciclos de cuenta fija. Por tanto, en estos lenguajes las instrucciones `for` se usan para construir ciclos de cuenta fija y las instrucciones `while` se usan por lo general sólo cuando se construyen ciclos de condición variable.

En C++, esta distinción tan fácil no se sostiene, pues cada instrucción puede usarse para crear cada tipo de ciclo. La respuesta en C++, entonces, en realidad es cuestión de estilo. En vista que un ciclo `for` y uno `while` son intercambiables en C++, cualquier ciclo es apropiado. Algunos programadores profesionales siempre usan una instrucción `for` para todos los ciclos de prueba preliminar que crean y casi nunca usan una instrucción `while`; otros siempre utilizan una instrucción `while` y rara vez una instrucción `for`. Un tercer grupo tiende a conservar la convención utilizada en otros lenguajes: un ciclo `for` se usa por lo general para crear ciclos de cuenta fija y un ciclo `while` para crear ciclos de condición variable. En C++ todo es cuestión de estilo y encontrará los tres estilos en su carrera de programación.

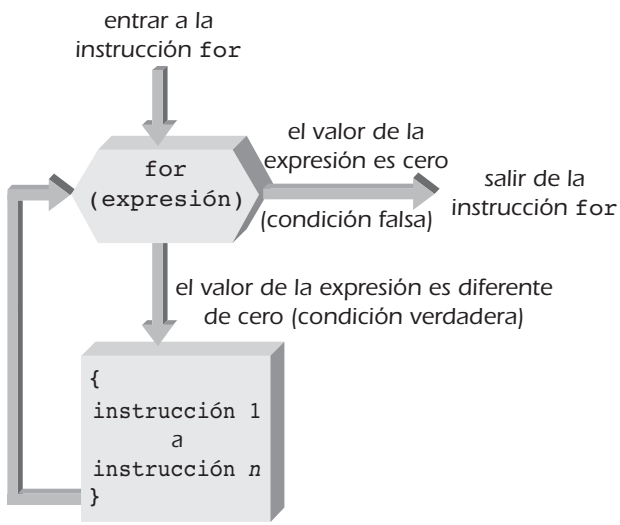


Figura 5.8 *Diagrama de flujo simplificado de un ciclo* `for`.

Para entender el poder enorme de los ciclos `for`, considere la tarea de imprimir una tabla de números del 1 al 10, incluyendo sus cuadrados y cubos, usando esta instrucción. Dicha tabla fue producida con anterioridad usando un ciclo `while` en el programa 5.3. Puede desear revisar el programa 5.3 y compararlo con el programa 5.11 para obtener un sentido mayor de la equivalencia entre los ciclos `for` y `while`.

Programa 5.11

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  const int MAXNUMS = 10;
  int num;

  cout << "NUMERO   CUADRADO   CUBO\n"
       << "------   --------   ----\n"

  for (num = 1; num <= MAXNUMS; num++)
    cout << setw(3) << num << "         "
         << setw(3) << num * num << "       "
         << setw(4) << num * num * num << endl;

  return 0;
}
```

Cuando el programa 5.11 se ejecuta, el despliegue producido es

```
NUMERO     CUADRADO     CUBO
------     --------     ----
  1            1          1
  2            4          8
  3            9         27
  4           16         64
  5           25        125
  6           36        216
  7           49        343
  8           64        512
  9           81        729
 10          100       1000
```

Con sólo cambiar el número 10 en la instrucción `for` del programa 5.11 a 1000 crea un ciclo que se ejecuta 1000 veces y produce una tabla de números del 1 al 1000. Como con la instrucción `while`, este pequeño cambio produce un aumento inmenso en el procesamiento y salida proporcionados por el programa. Nótese también que la expresión `num++` se usó en la lista de alteración en lugar del usual `num = num + 1`.

Ejercicios 5.4

1. Escriba instrucciones `for` individuales para los siguientes casos.
   a. Al usar un contador nombrado `i` que tiene un valor inicial de 1, un valor final de 20 y un incremento de 1.
   b. Al usar un contador nombrado `icuenta` que tiene un valor inicial de 1, un valor final de 20 y un incremento de 2.
   c. Al usar un contador nombrado `j` que tiene un valor inicial de 1, un valor final de 100 y un incremento de 5.
   d. Al usar un contador nombrado `icuenta` que tiene un valor inicial de 20, un valor final de 1 y un incremento de –1.
   e. Al usar un contador nombrado `icuenta` que tiene un valor inicial de 20, un valor final de 1 y un incremento de –2.
   f. Al usar un contador nombrado `cuenta` que tiene un valor inicial de 1.0, un valor final de 16.2 y un incremento de 0.2.
   g. Al usar un contador nombrado `xcnt` que tiene un valor inicial de 20.0, un valor final de 10.0 y un incremento de –0.5.

2. Determine el número de veces que se ejecuta cada ciclo for para la instrucción for escrita para el ejercicio 1.

3. Determine el valor en total después que se ejecuta cada uno de los siguientes ciclos.

   a.
   ```
   total = 0;
   for (i = 1; i <= 10; i = i + 1)
     total = total + 1;
   ```
   b.
   ```
   total = 1;
   for (cuenta = 1; cuenta <= 10; cuenta = cuenta + 1)
     total = total * 2;
   ```
   c.
   ```
   total = 0
   for (i = 10; i <= 15; i = i + 1)
     total = total + i;
   ```
   d.
   ```
   total = 50
   for (i = 1; i <=10; i = i + 1)
     total = total - i;
   ```
   e.
   ```
   total = 1
   for (icnt = 1; icnt <= 8; ++icnt)
     total = total * icnt;
   ```
   f.
   ```
   total = 1.0
   for (j = 1; j <= 5; ++j)
     total = total / 2.0;
   ```

4. Determine la salida del siguiente programa.

```
#include <iostream>
using namespace std;

int main()
{
  int i;

  for (i = 20; i >= 0; i = i - 4)
    cout << i << " ";

  return 0;
}
```

5. Modifique el programa 5.11 para producir una tabla de los números cero a 20 en incrementos de 2, con sus cuadrados y cubos.

6. Modifique el programa 5.11 para producir una tabla de números de 10 a 1, en lugar de 1 a 10 como lo hace ahora.

7. Escriba y ejecute un programa en C++ que despliegue una tabla de 20 conversiones de temperatura de Fahrenheit a Celsius. La tabla deberá comenzar con un valor Fahrenheit de 20 grados e incrementarse en valores de 4 grados. Recuerde que

$$Celsius = (5.0/9.0) * (Fahrenheit - 32)$$

8. Modifique el programa escrito para el ejercicio 7 para solicitar al inicio el número de conversiones que se van a hacer.

9. La expansión de un puente de acero conforme se calienta hasta una temperatura Celsius final, T_F, desde una temperatura Celsius inicial, T_0, puede aproximarse usando la fórmula

$$Aumento\ de\ longitud = a * L * (T_F - T_0)$$

donde a es el coeficiente de expansión (el cual para el acero es 11.7 e-6) y L es el largo del puente a la temperatura T_0. Usando esta fórmula, escriba un programa en C++ que despliegue una tabla de longitudes de expansión para un puente de acero que tiene 7365 metros de largo a 0 grados Celsius, conforme la temperatura incrementa a 40 grados en incrementos de 5 grados.

10. La probabilidad que una llamada telefónica individual dure menos de t minutos puede aproximarse por la función de probabilidad exponencial

$$Probabilidad\ que\ una\ llamada\ dure\ menos\ de\ t\ minutos = 1 - e^{-t/a}$$

donde a es la duración de la llamada promedio y e es el número de Euler (2.71828). Por ejemplo, suponiendo que la duración de la llamada promedio es 2 minutos, la probabilidad que una llamada durará menos de 1 minuto se calcula como $1 - e^{-1/2} = 0.3297$.

Usando esta ecuación de probabilidad, escriba un programa en C++ que calcule y despliegue una lista de probabilidades de la duración de una llamada menor que 1 a menor que 10 minutos, en incrementos de un minuto.

11. a. El índice de llegadas de clientes en un banco concurrido en Nueva York puede estimarse usando la función de probabilidad de Poisson

$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

donde x = el número de clientes que llegan por minuto; λ = el número promedio de llegadas por minuto y e el número de Euler (2.71828). Por ejemplo, si el número promedio de clientes que entran en el banco es de tres clientes por minuto, entonces λ es igual a tres. Por tanto, la probabilidad que un cliente llegue en cualquier minuto es

$$P(x = 1) = \frac{3^1 e^{-3}}{1!} = 0.149561$$

y la probabilidad que lleguen dos clientes en cualquier minuto es

$$P(x = 2) = \frac{3^2 e^{-3}}{2!} = 0.224454$$

Usando la función de probabilidad de Poisson, escriba un programa en C++ que calcule y despliegue la probabilidad de llegada de 1 a 10 clientes en cualquier minuto cuando el índice de llegadas promedio es de tres clientes por minuto.

b. La fórmula dada en el ejercicio 11a también es aplicable para estimar el índice de llegadas de aviones en un aeropuerto concurrido (aquí, un "cliente" que llega es un avión que aterriza). Usando esta misma fórmula, modifique el programa escrito en el ejercicio 11a para aceptar el índice de llegadas promedio como un elemento de datos de entrada. Luego ejecute el programa modificado para determinar la probabilidad de cero a 10 aviones que intenten aterrizar en cualquier periodo de un minuto en un aeropuerto durante las horas de mayor tráfico de llegadas. Suponga que el índice de llegadas promedio para las horas de mayor tráfico es de dos aviones por minuto.

12. Una pelota de golf es soltada desde un avión. La distancia, d, que cae la pelota en t segundos está dada por la ecuación $d = (\frac{1}{2})gt^2$, donde g es la aceleración debida a la gravedad y es igual a 32 pies/s^2. Usando esta información, escriba y ejecute un programa en C++ que despliegue la distancia que cae en cada intervalo de un segundo para diez segundos y la distancia que cae la pelota de golf al final de cada intervalo. La salida deberá completar la siguiente tabla:

Tiempo	Distancia en el intervalo actual	Distancia total
0	0.0	0.0
1	16.0	16.0
.	.	.
.	.	.
.	.	.
10	.	.

13. Suponga que el avión en el ejercicio 12 vuela a una altura de 50 000 pies. Modifique el programa escrito para el ejercicio 12 para determinar cuánto tiempo le tomará a la pelota llegar al suelo. A fin de incrementar la precisión de su resultado sin un número indebido de cálculos, disminuya el intervalo de tiempo de 1 segundo a 0.1 segundos conforme la pelota se acerca al suelo.

14. La secuencia de Fibonacci es 0, 1, 1, 2, 3, 5, 8, 13, ... donde los primeros dos términos son 0 y 1, y cada término a partir de entonces es la suma de los dos términos precedentes; es decir $Fib[n] = Fib[n - 1] + Fib[n - 2]$. Usando esta información, escriba un programa en C++ que calcule el *enésimo* número en una secuencia de Fibonacci, donde n sea introducido de manera interactiva en el programa por el usuario. Por ejemplo si $n = 6$, el programa deberá desplegar el valor 5.

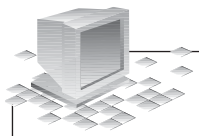
5.5 TÉCNICAS DE PROGRAMACIÓN CON CICLOS

En esta sección se presentan cuatro técnicas de programación comunes asociadas con ciclos de prueba preliminar (`for` y `while`). Todas estas técnicas son de uso común para los programadores experimentados.

Técnica 1: Entrada interactiva dentro de un ciclo

En la sección 5.2 se presentó el efecto de incluir una instrucción `cin` dentro de un ciclo `while`. Introducir datos en forma interactiva dentro de un ciclo es una técnica general que es aplicable por igual a los ciclos `for`. Por ejemplo, en el programa 5.12 se usa una instrucción `cin` para permitir a un usuario introducir de manera interactiva un conjunto de números. Conforme se introduce cada número, es sumado a un total. Cuando se sale del ciclo `for`, se calcula y se despliega el promedio.

La instrucción `for` en el programa 5.12 crea un ciclo que se ejecuta cuatro veces. Se le indica al usuario que introduzca un número cada vez que pasa el ciclo. Después que es introducido cada número, se suma de inmediato al total. Nótese que `total` es inicializado en cero como parte de la lista de inicialización que se ejecuta en la instrucción `for`. El ciclo en el programa 5.12 es ejecutado en tanto el valor en `cuenta` es menor que o igual a cuatro y es terminado cuando `cuenta` se vuelve cinco (el incremento a cinco, de hecho, es lo que causa que termine el ciclo). La salida producida por el programa 5.12 es en esencia la misma que en el programa 5.7.



Programa 5.12

```
#include <iostream>
using namespace std;

// Este programa calcula el promedio de CUENTAMAX
// numeros introducidos por el usuario
int main()
{

  const int CUENTAMAX = 4;
  int cuenta;
  double num, total, promedio;

  total = 0.0;

  for (cuenta = 0; cuenta < CUENTAMAX; cuenta++)
  {
    cout << "Introduzca un número: ";
    cin  >> num;
    total = total + num;
  }

  promedio = total / CUENTAMAX;
  cout << "El promedio de los datos introducidos es "
       << promedio << endl;
  return 0;
}
```

Técnica 2: Selección dentro de un ciclo

Otra técnica de programación común es usar un ciclo `for` o `while` para hacer un ciclo a través de un conjunto de números y seleccionar aquellos números que satisfagan uno o más criterios. Por ejemplo, suponga que desea encontrar la suma positiva y negativa de un conjunto de números. Los criterios aquí son si el número es positivo o negativo, y la lógica para poner en práctica este programa está dada por el seudocódigo

```
Mientras la condición del ciclo sea verdadera
    Introducir un número
     Si el número es mayor que cero
       sumar el número a la suma positiva
     De lo contrario
       sumar el número a la suma negativa
    Endif
Endwhile
```

El programa 5.13 describe este algoritmo en C++ para un ciclo de cuenta fija donde se han de introducir cinco números.

Programa 5.13

```
#include <iostream>
using namespace std;

// Este programa calcula las sumas positiva y negativa de un conjunto
// de numeros introducidos por el usuario en NUMSMAX
int main()
{
  const int NUMSMAX = 5;
  int i;
  double usenum, totpos, totneg;

  totpos = 0; // esta inicializacion puede hacerse en la declaracion
  totneg = 0; // esta inicializacion puede hacerse en la declaracion

  for (i = 1; i <= NUMSMAX; i++)
  {
    cout << "Introduzca un numero (positivo o negativo) : ";
    cin  >> usenum;
    if (usenum > 0)
      totpos = totpos + usenum;
    else
      totneg = totneg + usenum;
   }
   cout << "El total positivo es " << totpos << endl;
   cout << "El total negativo es " << totneg << endl;

   return 0;
}
```

La siguiente es una muestra de ejecución usando el programa 5.13.

```
Introduzca un numero (positivo o negativo) : 10
Introduzca un numero (positivo o negativo) : -10
Introduzca un numero (positivo o negativo) : 5
Introduzca un numero (positivo o negativo) : -7
Introduzca un numero (positivo o negativo) : 11
El total positivo es 26
El total negativo es -17
```

Técnica 3: Evaluación de funciones de una variable

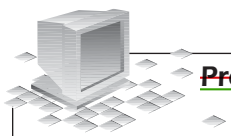
Los ciclos pueden construirse convenientemente para determinar y desplegar los valores de una sola variable de función matemática para un conjunto de valores sobre cualquier intervalo especificado. Por ejemplo, suponga que desea conocer los valores de la función

$$y = 10x^2 + 3x - 2$$

para x entre 2 y 6. Suponiendo que x ha sido declarada como una variable de número entero, puede utilizarse el siguiente ciclo `for` para calcular los valores requeridos.

```
for (x = 2; x <= 6; x++)
{
  y = 10 * pow(x,2) + 3 * x - 2;
  cout << setw(4) << x
       << setw(11) << y << endl;
}
```

Para este ciclo se ha usado la variable `x` como la variable del contador y como la incógnita (variable independiente) en la función. Para cada valor de `x` de dos a cinco se calcula y se despliega un nuevo valor de `y`. Este ciclo `for` está contenido dentro del programa 5.14, el cual también despliega encabezados apropiados para los valores impresos.



Programa 5.14

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
  int x, y;

  cout << "valor x   valor y\n"
       << "-------   --------\n"
  for (x = 2; x <= 6; x++)
  {
    y = 10 * pow(x,2) + 3 * x - 2;
    cout << setw(4) << x
         << setw(11) << y << endl;
  }

  return 0;
}
```

Cuando se ejecuta el programa 5.14 se despliega lo siguiente:

```
valor x    valor y
-------    --------
   2          44
   3          97
   4         170
   5         263
   6         376
```

Aquí son de importancia dos elementos. El primero es que cualquier ecuación con una incógnita puede evaluarse usando un solo ciclo `for` o un ciclo `while` equivalente. El método requiere sustituir la ecuación deseada en el ciclo en lugar de la ecuación usada en el programa 5.14, y ajustar los valores del contador para que correspondan con el rango de solución deseado.

El segundo elemento que se debe considerar es que no estamos restringidos a usar valores enteros para la variable del contador. Por ejemplo, al especificar un incremento no entero, pueden obtenerse soluciones para valores fraccionarios. Esto se muestra en el programa 5.15, donde la ecuación $y = 10x^2 + 3x - 2$ es evaluada en el rango $x = 2$ a $x = 6$ en incrementos de 0.5.

Programa 5.15

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
  double x, y;

  cout << "valor x     valor y\n";
       << "-------     ---------\n"
  cout << setiosflags(ios::fixed)
       << setiosflags(ios::showpoint)
       << setprecision(5);
  for (x = 2.0; x <= 6.0; x = x + 0.5)
  {
    y = 10.0 * pow(x,2.0) + 3.0 * x - 2.0;
    cout << setw(7) <<  x
         << setw(14) << y << endl;
  }

  return 0;
}
```

Hay que observar que *x* y *y* han sido declaradas como variables de punto flotante en el programa 5.15 para permitir que estas variables adopten valores fraccionarios. La siguiente es la salida producida por este programa.

```
valor x      valor y
-------     ---------
2.00000      44.00000
2.50000      68.00000
3.00000      97.00000
3.50000     131.00000
4.00000     170.00000
4.50000     214.00000
5.00000     263.00000
5.50000     317.00000
6.00000     376.00000
```

Técnica 4: Control interactivo de un ciclo

Los valores usados para controlar un ciclo pueden establecerse usando variables en lugar de valores constantes. Por ejemplo, las cuatro instrucciones

```
i = 5;
j = 10;
k = 1;
for (cuenta = i; cuenta <= j; cuenta = cuenta + k)
```

producen el mismo efecto que la instrucción única

```
for (count = 5; count <= 10; count = count + 1)
```

Del mismo modo, las instrucciones

```
i = 5;
j = 10;
k = 1;
cuenta = i;
while (cuenta <= j)
  cuenta = cuenta + k;
```

producen el mismo efecto que el siguiente ciclo `while`

```
cuenta = 5;
while (cuenta <= 10)
  cuenta = cuenta + 1;
```

La ventaja de usar variables en las expresiones de inicialización, condición y alteración es que permiten asignar valores para estas expresiones que son externos a la instrucción `for`

o `while`. Esto es útil en especial cuando se usa una instrucción `cin` para establecer los valores reales. Para hacer esto un poco más tangible, considere el programa 5.16.

Programa 5.16

```
#include <iostream>
#include <iomanip>
using namespace std;

// este programa despliega una tabla de numeros, sus cuadrados y cubos
// empezando por el numero 1. El numero final en la tabla es
// introducido por el usuario

int main()
{
  int num, final;

  cout << "Introduzca el numero final para la tabla: ";
  cin  >> final;

  cout << "NUMERO CUADRADO CUBO\n";
  cout << "------ -------- ----\n";

  for (num = 1; num <= final; num++)
    cout << setw(3) << num
         << setw(8) << num*num
         << setw(7) << num*num*num << endl;

  return 0;
}
```

En el programa 5.16, se ha usado una variable para controlar la expresión de condición (la de en medio). Aquí se ha colocado una instrucción `cin` antes del ciclo para permitir al usuario decidir cuál debería ser el valor final. Nótese que este arreglo permite al usuario establecer el tamaño de la tabla en tiempo de ejecución, en lugar de hacer que el programador establezca el tamaño de la tabla en tiempo de compilación. Esto también hace que el programa sea más general, en virtud que ahora puede usarse para crear una variedad de tablas sin la necesidad de reprogramar y volver a compilar.

Ejercicios 5.5

1. **`cin` dentro de un ciclo:** Escriba y ejecute un programa en C++ que acepte seis temperaturas Fahrenheit, una a la vez, y convierta cada valor introducido en su equivalente Celsius antes que se solicite el siguiente valor. Use un ciclo `for` en su programa. La conversión requerida es *Celsius* = (5.0/9.0) · (*Fahrenheit* – 32).

2. **`cin` dentro de un ciclo:** Escriba y ejecute un programa en C++ que acepte 10 valores individuales de galones, uno a la vez, y convierta cada valor introducido a su equivalente en litros antes que se solicite el siguiente valor. Use un ciclo `for` en su programa. Use el hecho que hay 3.785 litros en un galón.

3. **Control interactivo del ciclo:** Modifique el programa escrito para el ejercicio 2 para solicitar inicialmente el número de datos que se introducirán y convertirán.

4. **Control interactivo del ciclo:** Modifique el programa 5.13 de modo que el número de entradas que va a haber sea especificado por el usuario cuando se ejecute el programa.

5. **Selección:** Modifique el programa 5.13 de modo que despliegue el promedio de los números positivos y negativos. (*Sugerencia*: Tenga cuidado de no contar el número cero como un número negativo.) Pruebe su programa introduciendo los números 17, –10, 19, 0, –4. El promedio positivo desplegado por su programa deberá ser 18 y el promedio negativo, –7.

6. a. **Selección:** Escriba un programa en C++ que seleccione y despliegue el valor máximo de cinco números que se introducirán cuando el programa se ejecute. (*Sugerencia:* Use un ciclo `for` con una instrucción `cin` y una `if` internas en el ciclo.)

   b. Modifique el programa escrito para el ejercicio 6a de modo que despliegue tanto el valor máximo como la posición en el conjunto de números introducido donde ocurre el máximo.

7. **Selección:** Escriba un programa en C++ que seleccione y despliegue los primeros 20 números enteros que sean divisibles entre 3.

8. **Selección:** Los padres de una niña le prometieron darle 10 dólares cuando cumpliera 12 años de edad y duplicar el regalo en cada cumpleaños subsiguiente hasta que el regalo excediera 1000 dólares. Escriba un programa en C++ para determinar qué edad tendrá la niña cuando se le dé la última cantidad y la cantidad total recibida.

9. **Funciones matemáticas:** Modifique el programa 5.15 para producir una tabla de valores y para lo siguiente:

   a. $y = 3x^5 - 2x^3 + x$

   para x entre 5 y 10 en incrementos de 0.2

   b. $y = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$

   para x entre 1 y 3 en incrementos de 0.1

   c. $y = 2e^{0.8t}$ para t entre 4 y 10 en incrementos de 0.2

10. Funciones matemáticas: Un modelo de población mundial, en miles de millones de personas, está dado por la ecuación

$Población = 6.0e^{0.02t}$)

donde t es el tiempo en años ($t = 0$ representa enero de 2000 y $t = 1$ representa enero de 2001). Usando esta fórmula, escriba un programa en C++ que despliegue una tabla de población anual para los años de enero de 2005 a enero de 2010.

11. Funciones matemáticas: Las coordenadas x y y como una función del tiempo, t, de un proyectil lanzado con una velocidad inicial v en un ángulo de θ con respecto al suelo están dadas por:

$x = v\ t$ coseno(θ)
$y = v\ t$ seno(θ)

Usando estas fórmulas, escriba un programa en C++ que despliegue una tabla de valores de x y y de y para un proyectil lanzado con una velocidad inicial de 500 pies/s en un ángulo de 22.8 grados. (*Sugerencia:* Recuerde convertir la medida a radianes.) La tabla deberá contener valores correspondientes al intervalo de tiempo de 0 a 10 segundos en incrementos de $\frac{1}{2}$ segundo.

5.6 CICLOS ANIDADOS

En muchas situaciones es conveniente usar un ciclo contenido dentro de otro ciclo. Estos ciclos se llaman **ciclos anidados**. Un ejemplo simple de un ciclo anidado es

```
for(i = 1; i <= 5; i++)       // inicio del ciclo exterior
{
  cout << "\ni es ahora " << i << endl;

  for(j = 1; j <= 4; j++)     // inicio del ciclo interior
    cout << "  j = " << j;    // fin del ciclo interior
}                             // fin del ciclo exterior
```

El primer ciclo, controlado por el valor de `i`, se llama *ciclo exterior*. El segundo ciclo, controlado por el valor de `j`, se llama *ciclo interior*. Hay que observar que todas las instrucciones en el ciclo interior están contenidas dentro de los límites del ciclo exterior y que se ha usado una variable diferente para controlar cada ciclo. En cada ciclo del ciclo exterior, el ciclo interior recorre su secuencia por completo. Por tanto, cada vez que el contador `i` aumenta en 1, el ciclo `for` interior se ejecuta por completo. Esta situación se ilustra en la figura 5.9.

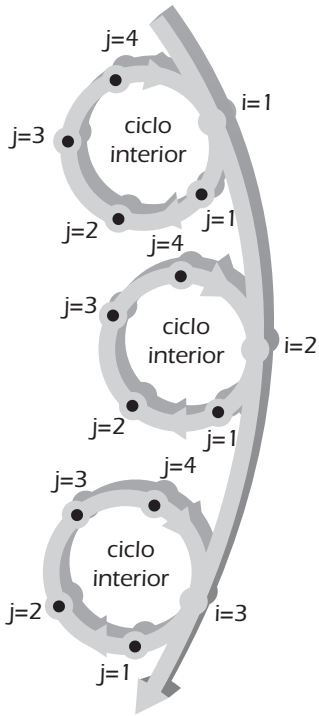


Figura 5.9 Ciclos para cada `i`, `j`.

El programa 5.17 incluye este tipo de código en un programa funcional.

Un ejemplo de salida del programa 5.17 es

```
i es ahora 1
  j = 1  j = 2  j = 3  j = 4
i es ahora 2
  j = 1  j = 2  j = 3  j = 4
i es ahora 3
  j = 1  j = 2  j = 3  j = 4
i es ahora 4
  j = 1  j = 2  j = 3  j = 4
i es ahora 5
  j = 1  j = 2  j = 3  j = 4
```

Programa 5.17

```
#include <iostream>
using namespace std;

int main()
{
  const int MAXI = 5;
  const int MAXJ = 4;
  int i, j;

  for(i = 1; i <= MAXI; i++)     // inicio del ciclo exterior <----+
  {                              //                               |
    cout << "\ni es ahora " << i << endl;  //                     |
                                 //                               |
    for(j = 1; j <= MAXJ; j++)   // inicio del ciclo interior      |
      cout << "  j = " << j;     // fin del ciclo interior         |
  }                              // fin del ciclo exterior   <-----+

  cout << endl;

  return 0;
}
```

Para ilustrar la utilidad de un ciclo anidado, se usará uno para calcular la calificación promedio para cada estudiante en una clase de 20 estudiantes. Cada estudiante ha presentado cuatro exámenes durante el semestre. La calificación final se calcula como el promedio de las calificaciones en estos exámenes. El seudocódigo que describe cómo puede hacerse este cálculo es

```
For 20 veces
   Establecer el total de calificaciones del estudiante en cero
   For 4 veces
      introducir una calificación
      sumar la calificación al total
   Endfor  // fin del ciclo for interior
   Calcular el promedio de calificaciones del estudiante
   Imprimir el promedio de calificaciones del estudiante
Endfor  // fin del ciclo for exterior
```

Como describe el seudocódigo, se usará un ciclo exterior consistente en 20 ciclos para calcular el promedio de calificaciones para cada estudiante. El ciclo interior consiste de cuatro ciclos. La calificación de un examen se introduce en cada pasada del ciclo interior. Conforme se introduce cada calificación se suma al total para el estudiante, y al final del ciclo se calcu-

la y se despliega el promedio. En vista que tanto el ciclo exterior como el interior son ciclos de cuenta fija de 20 y 4, respectivamente, se usarán instrucciones `for` para crear estos ciclos. El programa 5.18 proporciona el código C++ correspondiente al seudocódigo.

Programa 5.18

```
#include <iostream>
using namespace std;

int main()
{

  const int NUMCALIF = 4;
  const int NUMESTUDIANTES = 20;
  int i, j;
  double calificacion, total, promedio;

  for (i = 1; i <= NUMESTUDIANTES; i++) // inicio del ciclo exterior
  {
    total = 0;                        // limpia el total para este estudiante
    for (j = 1; j <= NUMCALIF; j++) // inicio del ciclo interior
    {
      cout << "Introduzca una calificacion de examen para este estudiante: ";
      cin  >> calificacion;
      total = total + calificacion;     // suma la calificacion en el total
    }                                   // fin del ciclo for interior
    promedio = total / NUMCALIF;        // calcula el promedio
    cout << "\nEl promedio para el estudiante " << i
         << " es " << promedio << "\n\n";
  }                                     // fin del ciclo for exterior

  return 0;
}
```

Al revisar el programa 5.18, ponga particular atención en la inicialización de total dentro del ciclo exterior, antes que se introduzca el ciclo interior. `total` se inicializa 20 veces, una para cada estudiante. También hay que observar que el promedio se calcula y despliega inmediatamente después de terminado el ciclo interior. En vista que las instrucciones que calculan e imprimen el promedio también están contenidas dentro del ciclo exterior, se calculan y despliegan 20 promedios. La introducción y adición de cada calificación dentro del ciclo interior utilizan técnicas que ya se habían visto antes, con las cuales ya debe estar familiarizado ahora.

Ejercicios 5.6

1. Se llevaron a cabo cuatro experimentos, cada uno consistente en seis resultados de prueba. Los resultados para cada experimento se dan a continuación. Escriba un programa usando un ciclo anidado para calcular y desplegar el promedio de los resultados de prueba para cada experimento.

Resultados del primer experimento: 23.2 31 16.9 27 25.4 28.6
Resultados del segundo experimento: 34.8 45.2 27.9 36.8 33.4 39.4
Resultados del tercer experimento: 19.4 16.8 10.2 20.8 18.9 13.4
Resultados del cuarto experimento: 36.9 39 49.2 45.1 42.7 50.6

2. a. Modifique el programa escrito para el ejercicio 1 de modo que el número de resultados de prueba para cada experimento sea introducido por el usuario. Escriba su programa de modo que pueda introducirse un número diferente de resultados de prueba para cada experimento.

b. Vuelva a escribir el programa escrito para el ejercicio 2a para eliminar el ciclo interior.

3. a. Un fabricante de equipo eléctrico prueba cinco generadores midiendo sus voltajes de salida en tres momentos diferentes. Escriba un programa en C++ que use un ciclo anidado para introducir los resultados de prueba de cada generador y luego calcule y despliegue el voltaje promedio para cada generador. Suponga los siguientes resultados de prueba de los generadores:

Primer generador: 122.5 122.7 123.0
Segundo generador: 120.2 127.0 125.1
Tercer generador: 121.7 124.9 126.0
Cuarto generador: 122.9 123.8 126.7
Quinto generador: 121.5 124.7 122.6

b. Modifique el programa escrito para el ejercicio 3a para calcular y desplegar el voltaje promedio para todos los generadores. (*Sugerencia:* Use una segunda variable para almacenar el total de todos los voltajes de los generadores.)

4. Vuelva a escribir el programa escrito para el ejercicio 3a para eliminar el ciclo interior. Para hacer esto, tendrá que introducir tres voltajes para cada generador en lugar de uno a la vez. Cada voltaje debe almacenarse en su propio nombre de variable antes que se calcule el promedio.

5. Escriba un programa que calcule y despliegue valores para y cuando

$$y = xz/(x - z)$$

Su programa deberá calcular y para valores de x que varían entre 1 y 5 y valores de z que varían entre 2 y 6. La variable x deberá controlar el ciclo exterior e incrementarse en pasos de 1 y z deberá incrementarse en pasos de 1. Su programa deberá desplegar también el mensaje `Funcion Indefinida` cuando los valores de x y de z sean iguales.

6. Los lenguajes ensambladores para algunos microprocesadores no tienen una operación de multiplicación. Aunque hay algoritmos sofisticados para llevar a cabo la multiplicación en estos casos, un método simple multiplica por adición repetida. En este caso la eficiencia del algoritmo puede incrementarse usando ciclos anidados. Por

ejemplo, para multiplicar un número por doce, primero suma el número tres veces y luego suma el resultado cuatro veces. Esto sólo requiere siete adiciones en vez de doce. Usando esta información escriba un programa en C++ que multiplique 33, 47 y 83 por 1001 usando tres ciclos y luego despliegue el resultado. (*Sugerencia:* 1001 = 7 · 11 · 13)

5.7 Ciclos do while

Tanto las instrucciones `while` como las `for` evalúan una expresión al inicio del ciclo de repetición; como tales siempre se usan para crear ciclos de prueba preliminar. Los ciclos de prueba posterior, los cuales también se conocen como ciclos controlados a la salida, pueden construirse de igual forma en C++. La estructura básica de un ciclo así, el cual se conoce como ciclo `do while`, se ilustra en la figura 5.10. Nótese que un ciclo do `while` continúa las iteraciones a través del ciclo mientras la condición es verdadera y sale del ciclo cuando la condición es falsa.

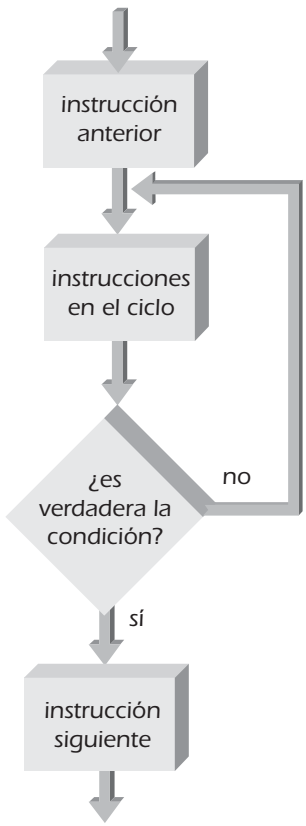


Figura 5.10 Estructura del ciclo `do while`.

En C++, un ciclo `do while` de prueba posterior se crea usando una instrucción `do`. Como su nombre implica, esta instrucción permite hacer algunas instrucciones antes que sea evaluada una expresión al final del ciclo. La forma general de la instrucción `do` de C++ es

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

no olvide el final `;`

Como con todos los programas en C++, la instrucción única en `do` puede reemplazarse por una instrucción compuesta. En la figura 5.11 se muestra un diagrama de control de flujo que ilustra la operación de la instrucción `do`.

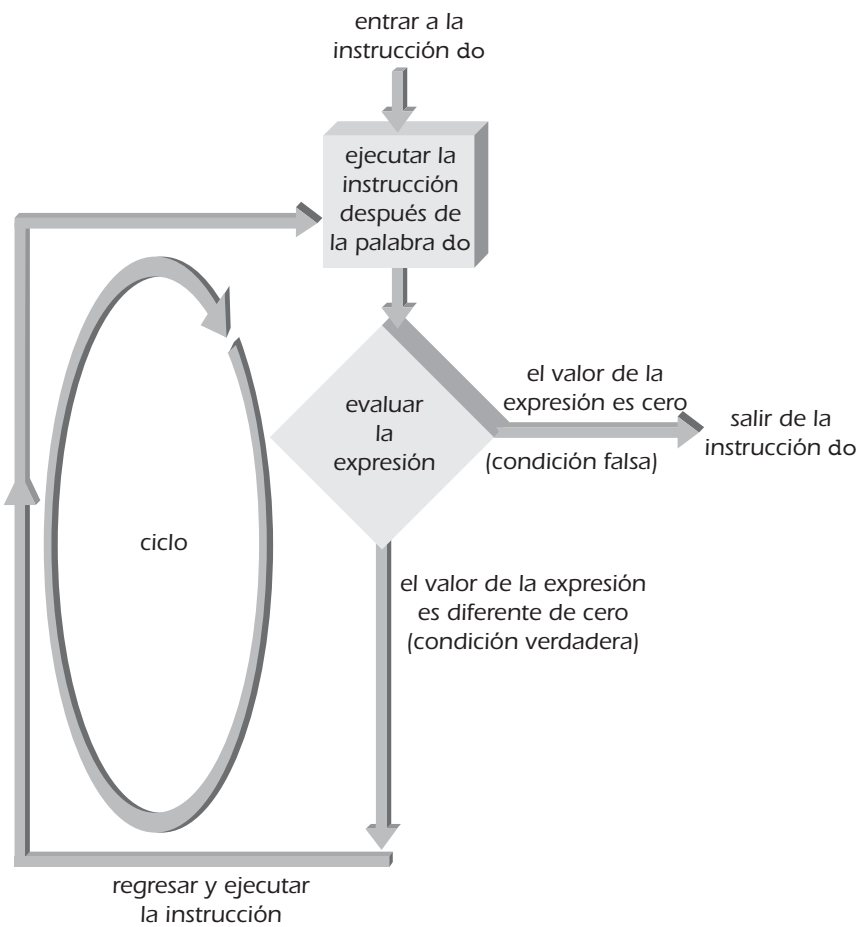


Figura 5.11 Flujo de control de la instrucción `do`.

Como se ilustró, todas las instrucciones dentro de la instrucción do son ejecutadas al menos una vez antes que la expresión sea evaluada. Luego, si la expresión tiene un valor diferente de cero, las instrucciones se ejecutan de nuevo. Este proceso continúa hasta que la expresión se evalúa en cero (se vuelve falsa). Por ejemplo, considere la siguiente instrucción do:

```
do
{
  cout << "\nIntroduzca un precio: ";
  cin  >> precio;
  if (abs(precio - CENTINELA) < 0.0001)
    break;
  impuestoventa = TASA * precio;
  cout << setiosflags(ios::showpoint)
       << setprecision(2)
       << "El impuesto sobre la venta es $ " << impuestoventa;
}
while (precio != CENTINELA);
```

Obsérvese que el indicador y la instrucción cin se incluyen dentro del ciclo debido a que la expresión probada es evaluada al final del ciclo.

Como todas las instrucciones de repetición, la instrucción do siempre puede reemplazar o ser reemplazada por una instrucción while o for equivalente. La elección de cuál instrucción usar depende de la aplicación y el estilo preferido por el programador. En general, se prefieren las instrucciones while y for porque permiten que cualquiera que lea el programa sepa con claridad lo que se está probando "por adelantado" en la parte superior del ciclo del programa.

Verificaciones de validez

La instrucción do es útil en particular para filtrar datos introducidos por el usuario y proporcionar verificaciones de validación de los datos. Por ejemplo, suponga que se requiere un operador para introducir un número de identificación del cliente válido entre los números 1000 y 1999. Un número que esté fuera de este rango se rechazará y se hará una nueva solicitud para un número válido. La siguiente sección de código proporciona el filtrado de datos necesario para verificar la introducción de un número de identificación válido:

```
do
{
  cout << "\nIntroduzca un numero de identificacion: ";
  cin  >> num_id;
}
while (num_id < 1000 || num_id > 1999);
```

Aquí, una solicitud para la introducción de un número de identificación se repite hasta que se introduce un número válido. Esta sección de código es "el meollo" ya que ni alerta al operador de la causa de la nueva solicitud de datos ni permite la salida prematura del ciclo si no puede encontrar un número de identificación válido. Una alternativa para eliminar el primer inconveniente es

```
do
{
  cout << "\nIntroduzca un numero de identificacion: ";
  cin  >> num_id;
  if (num_id < 1000 || num_id > 1999)
  {
     cout << "Se acaba de introducir un numero invalido\n";
     cout << "Por favor verifique el numero de identificacion
             y vuelva a introducirlo\n";
  }
  else
     break;   // interrumpe si se introdujo un numero de
                 identificacion valido
} while(1);  // esta expresion siempre es verdadera
```

Aquí se ha usado una instrucción `break` para salir del ciclo. En vista que la expresión que está evaluando la instrucción `do` siempre es 1 (verdadera), se ha creado un ciclo infinito que sólo se interrumpe cuando se encuentra la instrucción `break`.

Ejercicios 5.7

1. a. Usando una instrucción `do`, escriba un programa para aceptar una calificación. El programa deberá solicitar una calificación en forma continua en tanto se introduzca una calificación inválida. Una calificación inválida es cualquier calificación menor que 0 o mayor que 100. Después que se ha introducido una calificación válida, su programa deberá desplegar el valor de la calificación introducida.

b. Modifique el programa escrito para el ejercicio 1a de modo que el usuario sea alertado cuando se ha introducido una calificación inválida.

c. Modifique el programa escrito para el ejercicio 1b de modo que permita al usuario salir del programa introduciendo el número 999.

d. Modifique el programa escrito para el ejercicio 1b de modo que termine en forma automática después que se han introducido cinco calificaciones inválidas.

2. a. Escriba un programa que solicite en forma continua que se introduzca una calificación. Si la calificación es menor que 0 o mayor que 100, su programa deberá imprimir un mensaje apropiado que informe al usuario que se ha introducido una calificación inválida, de lo contrario la calificación deberá sumarse a un total. Cuando se introduzca una calificación de 999 el programa deberá salir del ciclo de repetición y calcular y desplegar el promedio de las calificaciones válidas introducidas.

b. Ejecute el programa escrito en el ejercicio 2a en una computadora y verifique el programa usando datos de prueba apropiados.

3. a. Escriba un programa para invertir los dígitos de un número entero positivo. Por ejemplo, si se introduce el número 8735, el número desplegado deberá ser 5378. (*Sugerencia:* Use una instrucción do y continuamente quite y despliegue el dígito de las unidades del número. Si la variable `num` en un inicio contiene el número introducido, el dígito de las unidades se obtiene como (`num % 10`). Después que se despliega un dígito de unidades, dividir el número entre 10 establece el número para la siguiente iteración. Por tanto (8735 % 10) es 5 y (8735 / 10) es 873. La instrucción `do` deberá continuar en tanto el número remanente no sea cero.

b. Ejecute el programa escrito en el ejercicio 3a en una computadora y verifique el programa usando datos de prueba apropiados.

4. Repita cualquiera de los ejercicios en la sección 5.3 usando una instrucción `do` en lugar de una instrucción `for`.

5. Dado un número n, y una aproximación para su raíz cuadrada, puede obtenerse una aproximación más cercana a la raíz cuadrada real usando la fórmula:

$$\textit{aproximación nueva} = \frac{(n \,/\, \textit{aproximación previa}) + \textit{aproximación previa}}{2}$$

Usando esta información, escriba un programa en C++ que indique al usuario que introduzca un número y una estimación inicial de su raíz cuadrada. Usando estos datos de entrada su programa deberá calcular una aproximación a la raíz cuadrada que tenga una precisión hasta 0.00001 (*Sugerencia:* Detenga el ciclo cuando la diferencia entre dos aproximaciones sea menor que 0.00001.)

6. Aquí hay un problema desafiante para aquellos que saben un poco de cálculo. El método de Newton-Raphson puede utilizarse para encontrar las raíces de cualquier ecuación $y(x) = 0$. En este método la $(i + 1)$*ésima* aproximación, x_{i+1}, a una raíz de $y(x) = 0$ está dada en términos de la iésima aproximación, x_i, por la fórmula

$$x_{i+1} = x_i - y(x_i) \,/\, y'(x_i)$$

Por ejemplo, si $y(x) = 3x^2 + 2x - 2$, entonces $y'(x) = 6x + 2$, y las raíces se encuentran haciendo una estimación razonable para una primera aproximación x_1 y haciendo iteraciones usando la ecuación

$$x_{i+1} = x_i - (3x_i^2 + 2x_i - 2) \,/\, (6x_i + 2)$$

a. Usando el método de Newton-Raphson, encuentre las dos raíces de la ecuación $3x^2 + 2x - 2 = 0$. (*Sugerencia:* Hay una raíz positiva y una raíz negativa.)

b. Extienda el programa escrito para el ejercicio 6a de modo que encuentre las raíces de cualquier función $y(x) = 0$, cuando la función para $y(x)$ y la derivada de $y(x)$ son colocadas en el código.

5.8 Errores comunes de programación

Por lo general los programadores en C++ principiantes cometen seis errores cuando usan instrucciones de repetición. El más problemático de éstos es el error de "fallar por uno", donde el ciclo se ejecuta ya sea una vez más o una vez menos de lo que se pretendía. Por ejemplo, el ciclo creado por la instrucción `for(i = 1; i < 11; i++)` se ejecuta diez veces, no once, aun cuando se use el número 11 en la instrucción. Por tanto, un ciclo equivalente puede construirse usando la instrucción `for(i = 1; i <= 10; i ++)`. Sin embargo, si el ciclo empieza con un valor inicial de i = 0, usando la instrucción `for(i = 0; i < 11; i++)`, el ciclo se repetirá 11 veces, al igual que un ciclo construido con la instrucción `for (i = 0; i <= 10; i++)`. Por tanto, al construir ciclos, debe ponerse atención particular a las condiciones inicial y final usadas para controlar el ciclo para asegurar que el número de repeticiones no falle por una ejecución de más o una de menos.

Los siguientes dos errores se relacionan con la expresión probada, y ya se han encontrado con las instrucciones `if` y `switch`. El primero es el uso inadvertido del operador de asignación, =, por el operador de igualdad, ==, en la expresión probada. Un ejemplo de este error es mecanografiar la expresión de asignación a = 5 en lugar de la expresión relacional deseada a==5. En vista que la expresión probada puede ser cualquier expresión válida en C++, incluyendo expresiones aritméticas y de asignación, este error no es detectado por el compilador.

Como con la instrucción `if`, las instrucciones de repetición no deben usar el operador de igualdad, ==, cuando prueben operandos de punto flotante o de precisión doble. Por ejemplo, la expresión `fnum == 0.01` deberá reemplazarse por una prueba que requiera que el valor absoluto de `fnum - 0.01` sea menor que una cantidad aceptable. La razón para esto es que todos los números son almacenados en forma binaria. Usando un número finito de bits, los números decimales como 0.01 no tienen un equivalente binario exacto, así que las pruebas que requieren igualdad con dichos números pueden fallar.

Los siguientes dos errores son particulares de la instrucción `for`. El más común es colocar un punto y coma al final de los paréntesis de `for`, lo cual con frecuencia produce un ciclo `no hacer nada`. Por ejemplo, considere las instrucciones

```
for(cuenta = 0; cuenta < 10; cuenta++);
  total = total + num;
```

Aquí el punto y coma al final de la primera línea de código es una instrucción nula. Esto tiene el efecto de crear un ciclo que es ejecutado 10 veces sin hacer nada excepto el incremento y prueba de `cuenta`. Este error tiende a ocurrir debido a que los programadores en C++ están acostumbrados a terminar la mayor parte de las líneas con un punto y coma.

El siguiente error ocurre cuando se usan comas para separar los elementos en una instrucción `for` en lugar de los puntos y comas requeridos. Un ejemplo de esto es la instrucción

```
for (cuenta = 1, cuenta < 10, cuenta++)
```

Las comas deben usarse para separar elementos dentro de las listas de inicialización y de alteración, pero los puntos y comas deben usarse para separar estas listas de la expresión probada.

El último error ocurre cuando se omite el punto y coma final de la instrucción `do`. Este error por lo general es cometido por programadores que han aprendido a omitir el punto y coma después del paréntesis de una instrucción `while` y conservan este hábito cuando encuentran la palabra reservada `while` al final de una instrucción `do`.

5.9 RESUMEN DEL CAPÍTULO

1. Una sección de código repetitivo se conoce como un *ciclo*.

El ciclo es controlado por una instrucción de repetición que prueba una condición para determinar si se ejecutará el código. Cada una de estas pruebas a través del ciclo se conoce como una *repetición* o *iteración*. La condición probada siempre debe establecerse de manera explícita antes de su primera evaluación por la instrucción de repetición. Dentro del ciclo siempre debe haber una instrucción que permita la alteración de la condición de modo que pueda salirse del ciclo, una vez comenzado.

2. Hay tres tipos básicos de ciclo:

a. `while`

b. `for`

c. `do while`

Los ciclos `while` y `for` son ciclos de prueba *preliminar* o *controlados en la entrada*. En este tipo de ciclo la condición probada se evalúa al principio del ciclo, el cual requiere que la condición probada se establezca de manera explícita antes de la entrada al ciclo. Si la condición es verdadera, comienzan las repeticiones del ciclo; de lo contrario no se entra al ciclo. Las iteraciones continúan en tanto la condición permanece verdadera. En C++, los ciclos `while` y `for` se construyen usando instrucciones `while` y `for`, respectivamente.

El ciclo `do while` es un ciclo de prueba posterior o *controlado en la salida*, donde la condición probada se evalúa al final del ciclo. Este tipo de ciclo siempre se ejecuta al menos una vez. En tanto la condición probada permanezca verdadera, los ciclos `do while` continúan ejecutándose.

3. Los ciclos también se clasifican según el tipo de condición probada. En un *ciclo de cuenta fija*, la condición se usa para dar seguimiento a cuantas repeticiones han ocurrido. En un ciclo de *condición variable* la condición probada se basa en una variable que puede cambiar de manera interactiva con cada pasada a través del ciclo.

4. En C++, un ciclo `while` se construye usando una instrucción `while`. La forma más utilizada de esta instrucción es

while (expresión)
{
  instrucciones;
}

La expresión contenida dentro del paréntesis es la condición probada para determinar si se ejecuta la instrucción que sigue al paréntesis, la cual por lo general es una instrucción compuesta. La expresión es evaluada exactamente de la misma manera que si estuviera contenida en una instrucción `if-else`; la diferencia es cómo se usa la expresión. En una instrucción `while` la instrucción que sigue a la expresión es ejecutada de manera repetida en tanto la expresión mantenga un valor diferente de cero, en lugar de sólo una vez, como en una instrucción `if-else`. Un ejemplo de un ciclo `while` es

```
cuenta = 1;                // inicializar cuenta
while (cuenta <= 10)
{
  cout << cuenta << "  ";
  cuenta++;                   // incrementar cuenta
}
```

La primera instrucción de asignación establece cuenta igual a 1. Luego se introduce la instrucción `while` y la expresión es evaluada por primera vez. En vista que el valor de cuenta es menor que o igual a 10, la expresión es verdadera y la instrucción compuesta se ejecuta. La primera instrucción en la instrucción compuesta usa el

objeto `cout` para desplegar el valor de cuenta. La siguiente instrucción agrega 1 al valor almacenado en la actualidad en cuenta, haciendo este valor igual a 2. La instrucción `while` regresa ahora a probar de nuevo la expresión. En vista que `cuenta` aún es menor que o igual a 10, la instrucción compuesta se ejecuta de nuevo. Este proceso continúa hasta que el valor de `cuenta` llega a 11.

La instrucción `while` siempre verifica esta expresión al principio del ciclo. Esto requiere que cualesquiera variables en la expresión probada deben tener valores asignados antes que se encuentre `while`. Dentro del ciclo `while` debe haber una instrucción que altere el valor de la expresión probada.

5. En C++, un ciclo `for` se construye usando una instrucción `for`. Esta instrucción ejecuta las mismas funciones que la instrucción `while`, pero utiliza una forma diferente. En muchas situaciones, en especial aquellas que usan una condición de cuenta fija, el formato de la instrucción `for` es más fácil de usar que su instrucción `while` equivalente. La forma más usada de la instrucción `for` es

```
for (lista de inicialización; expresión; lista de alteración)
{
   instrucciones;
}
```

Dentro del paréntesis de la instrucción `for` hay tres elementos, separados por puntos y comas. Cada uno de estos elementos es opcional pero los puntos y comas deben estar presentes.

La lista de inicialización se usa para establecer los valores iniciales antes de entrar al ciclo; por lo general se usa para inicializar un contador. Las instrucciones dentro de la lista de inicialización sólo se ejecutan una vez. La expresión en la instrucción `for` es la condición que se prueba al inicio del ciclo y antes de cada iteración. La lista de alteración contiene instrucciones de ciclo que no están contenidas dentro de la instrucción compuesta: por lo general se usa para incrementar o disminuir un contador cada vez que se ejecuta el ciclo. Las instrucciones múltiples dentro de una lista se separan con comas. Un ejemplo de un ciclo `for` es

```
for (total = 0, cuenta = 1; cuenta < 10; cuenta++)
{
   cout << "Introduzca una calificacion: ";
   total = total + calificacion;
}
```

En esta instrucción `for`, la lista de inicialización se usa para inicializar tanto total como cuenta. La expresión determina que el ciclo se ejecutará en tanto el valor en cuenta sea menor que 10, y el valor de cuenta se incrementa en uno cada vez que se pasa por el ciclo.

6. La instrucción `for` es útil en extremo para crear ciclos de cuenta fija. Esto se debe a que las instrucciones de inicialización, la expresión probada y las instrucciones que afectan a la expresión probada pueden incluirse en el paréntesis al inicio de un ciclo `for` para una fácil inspección y modificación.

7. La instrucción `do` se usa para crear ciclos de prueba posterior porque verifica su expresión al final del ciclo. Esto asegura que el cuerpo de un ciclo `do` se ejecute al menos una vez. Dentro de un ciclo `do` debe haber al menos una instrucción que altere el valor de la expresión probada.

Consideración de opciones de carrera

Ingeniería industrial

Cada una de las disciplinas de la ingeniería tradicional (civil, mecánica, eléctrica, química y metalúrgica/minera) se fundamenta en un área particular de las ciencias naturales. La ingeniería industrial, sin embargo, incorpora el conocimiento de las ciencias sociales al diseño de mejoras en sistemas hombre-máquina. Los ingenieros industriales son responsables del diseño, instalación y evaluación de máquinas y sistemas y también de la supervisión de su interfaz con las personas para mejorar la productividad general. Este trabajo puede implicar la comprensión de las características conductuales humanas y sus efectos en el diseño de máquinas o del lugar de trabajo. Los ingenieros industriales hacen mucho énfasis en el conocimiento de economía, administración de negocios y finanzas, al igual que en las ciencias naturales. Las áreas de especialización del ingeniero industrial pueden dividirse en cuatro categorías:

1. Investigación de operaciones. Esta área implica la aplicación de técnicas analíticas y modelos matemáticos a fenómenos como control de inventarios, simulación, teoría de las decisiónes y teoría de colas para optimizar los sistemas totales necesarios para la producción de bienes.
2. Ingeniería administrativa. La interacción cada vez más compleja de la administración con las habilidades de producción en las operaciones industriales modernas ha generado una gran demanda de gerentes con capacitación técnica. Estos gerentes evalúan y planean empresas corporativas e interactúan con la fuerza laboral, los departamentos de ingeniería y los subcontratistas. Un ingeniero administrativo también puede participar en las operaciones financieras de una compañía, debido a sus conocimientos de economía, administración de negocios y leyes.
3. Ingeniería de manufactura y producción. Antes que se fabrique un producto, el proceso completo de manufactura debe ser diseñado e instalado para optimizar la economía implicada y la calidad final del artículo. Esta tarea requiere un conocimiento amplio de diseño de procesos, trazado de plantas, diseño de herramientas, robótica e interacciones hombre-máquina.
4. Sistemas de información. Esta área implica el uso de computadoras para recopilar y analizar datos para la toma de decisiones y la planeación y para mejorar la actividad hombre-máquina.

La siguiente lista incluye las responsabilidades más comunes de los ingenieros industriales que respondieron a una encuesta reciente realizada por el Instituto Estadounidense de Ingenieros Industriales:

Planeación y diseño de instalaciones
Ingeniería de métodos
Diseño de sistemas de trabajo
Ingeniería de producción
Sistemas de información y control administrativo
Análisis y diseño de organizaciones
Medición del trabajo
Administración de salarios
Control de calidad
Administración de proyectos
Control de costos
Control de inventarios
Conservación de energía
Control de procesos computarizados
Empaque, manejo y prueba de productos
Selección de herramientas y equipo
Control de producción
Estudio del mejoramiento de productos
Mantenimiento preventivo
Programas de seguridad
Programas de capacitación