CAPÍTULO 4
Estructuras de selección

TEMAS

Han ocurrido muchos avances en los fundamentos teóricos de la programación desde el comienzo de los lenguajes de alto nivel a finales de la década de los años 50. Uno de los más importantes de estos avances fue el reconocimiento a finales de la década de los años 60 que cualquier algoritmo, sin importar cuán complejo fuera, podía ser construido usando combinaciones de cuatro estructuras de ***flujo de control*** *estandarizadas: secuencial, de selección, de repetición y de invocación.*

El término ***flujo de control*** *se refiere al orden en que las instrucciones de un programa son ejecutadas. A menos que se dirijan de otra manera, el flujo de control normal para todos los programas es* ***secuencial****. Esto significa que las instrucciones son ejecutadas en secuencia, una tras otra, en el orden en que son colocadas dentro del programa.*

Las estructuras de selección, repetición e invocación permiten que el flujo de control secuencial sea alterado en formas definidas con precisión. Como podrá haber adivinado, la estructura de selección se usa para seleccionar cuáles instrucciones se han de ejecutar a continuación y la estructura de repetición se usa para repetir un conjunto de instrucciones. En este capítulo presentamos las instrucciones de selección de C++. Las técnicas de repetición e invocación se presentan en los capítulos 5 y 6.

4.1 CRITERIOS DE SELECCIÓN

En la solución de muchos problemas, deben emprenderse diferentes acciones dependiendo del valor de los datos. Los ejemplos de situaciones simples incluyen calcular un área *sólo si* las mediciones son positivas, ejecutar una división *sólo si* el divisor no es cero, imprimir diferentes mensajes *dependiendo* del valor de una calificación recibida, y así en forma sucesiva.

La instrucción `if-else` en C++ se usa para poner en práctica una estructura de decisión en su forma más simple, la de elegir entre dos alternativas. La sintaxis de seudocódigo más usada de esta instrucción es

```
if (condición)
  instrucción ejecutada si la condición es verdadera;
else
  instrucción ejecutada si la condición es falsa;
```

Cuando un programa en ejecución encuentra la instrucción `if`, la condición es evaluada para determinar su valor numérico, el cual es interpretado entonces como verdadero o falso. Si la condición produce cualquier valor numérico positivo o negativo diferente de cero, la condición es considerada como una condición "verdadera" y se ejecuta la instrucción que sigue a `if`. Si la condición produce un valor numérico de cero, la condición es considerada como una condición "falsa" y se ejecuta la instrucción que sigue a `else`. La parte `else` de la instrucción es opcional y puede omitirse.

La condición usada en una instrucción `if` puede ser cualquier expresión válida de C++ (incluyendo, como se verá, una expresión de asignación). Las expresiones más usadas por lo común, sin embargo, se llaman **expresiones relacionales**. Una **expresión relacional simple** consiste en un operador relacional que compara dos operandos, como se muestra en la figura 4.1.

Figura 4.1 Anatomía de una expresión relacional simple.

Mientras cada operando en una expresión relacional puede ser una variable o una constante, los operadores relacionales deben ser uno de los expuestos en la tabla 4.1. Estos operadores relacionales pueden usarse con operados de números enteros, de punto flotante, de precisión doble o de carácter, pero deben escribirse con exactitud como se muestra en la tabla 4.1. Por tanto, mientras todos los siguientes ejemplos son válidos:

```
edad > 40     largo <= 50              temp > 98.6
   3 < 4      indicador == terminar    Num_id == 682
dia != 5      2.0 > 3.3                horas > 40
```

los siguientes son inválidos:

```
largo =< 50             // operador fuera de orden
2.0 >> 3.3              // operador inválido
indicador = = terminar // no se permiten espacios
```

Tabla 4.1

Operador relacional	Significado	Ejemplo
<	menor que	`edad < 30`
>	mayor que	`altura > 6.2`
<=	menor que o igual a	`gravable <= 20000`
>=	mayor que o igual a	`temp >= 98.6`
==	igual a	`calificación == 100`
!=	no es igual a	`número != 250`

Las expresiones relacionales a veces se llaman **condiciones** y se usarán ambos términos para referirnos a ellas. Como todas las expresiones C++, las expresiones relacionales son evaluadas para producir un resultado numérico.[1] En el caso de una expresión relacional, el valor de la expresión sólo puede ser el valor entero de 1 o 0, el cual es interpretado como verdadero y falso, respectivamente. *Una expresión relacional que interpretaríamos como verdadera produce un valor entero de 1, y una expresión relacional falsa produce un valor entero de 0*. Por ejemplo, debido a que la relación 3 < 4 siempre es verdadera, esta expresión tiene un valor de 1, y debido a que la relación 2.0 > 3.3 siempre es falsa, el valor de la expresión en sí es 0. Esto puede verificarse usando las instrucciones

```
cout << "El valor de 3 < 4 es " << (3 < 4) << endl;
cout << "El valor de 2.0 > 3.0 es " << (2.0 > 3.3) << endl;
cout << "El valor de verdadero es " << verdadero << endl;
cout << "El valor de falso es " << falso << endl;
```

[1]En este aspecto C++ difiere de otros lenguajes de computadora de nivel alto que producen un resultado booleano (verdadero, falso).

lo cual produce el despliegue

```
El valor de 3 < 4 es 1
El valor de 2.0 > 3.0 es 0
El valor de verdadero es 1
El valor de falso es 0
```

El valor de una expresión relacional como `horas > 40` depende del valor almacenado en la variable `horas`.

En un programa C++, el valor de una expresión relacional no es tan importante como la interpretación que C++ coloca en el valor cuando se usa la expresión como parte de una instrucción de selección. En estas instrucciones, las cuales se presentan en la siguiente sección, se verá que un valor de cero es usado por C++ para representar una condición falsa y cualquier valor diferente de cero se utiliza para representar una condición verdadera. La selección de cuál instrucción ejecutar a continuación se basa entonces en el valor obtenido.

Además de los operandos numéricos, pueden compararse datos de carácter usando operadores relacionales. Para estas comparaciones, los valores `char` son coaccionados de manera automática a valores `int` para la comparación. Por ejemplo, en el código Unicode, la letra 'A' se almacena usando un código que tiene un valor numérico inferior que la letra 'B', el código para 'B' tiene un valor inferior que el código para 'C', y así en forma sucesiva. Para conjuntos de caracteres codificados de esta manera, las siguientes condiciones se evalúan como sigue:

Expresión	Valor	Interpretación
`'A' > 'C'`	0	falso
`'D' <= 'Z'`	1	verdadero
`'E' == 'F'`	0	falso
`'g' >= 'm'`	0	falso
`'b' != 'c'`	1	verdadero
`'a' == 'A'`	0	falso
`'B' < 'a'`	1	verdadero
`'b' > 'Z'`	1	verdadero

Comparar letras es esencial para alfabetizar nombres o usar caracteres para seleccionar una opción particular en situaciones de toma de decisiones. Las cadenas de caracteres también pueden compararse. Por último, pueden compararse dos cadenas de expresiones usando operadores relacionales o los métodos de comparación de la clase `string` (capítulo 7). En el conjunto de caracteres ASCII, un espacio en blanco precede (y es considerado "menor que") a todas las letras y números; las letras del alfabeto son almacenadas en orden de la A a la Z; y los dígitos son almacenados en orden del 0 al 9. En esta secuencia, las letras minúsculas vienen después (son consideradas "mayores que") las letras mayúsculas, y los códigos de las letras vienen después (son "mayores que") los códigos de los dígitos (véase el apéndice B).

Cuando se comparan dos cadenas, sus caracteres individuales se comparan un par a la vez (ambos primeros caracteres, luego ambos segundos caracteres, etc.). Si no se encuentran diferencias, las cadenas son iguales; si se encuentra una diferencia, la cadena con el primer

carácter inferior es considerada la cadena más pequeña. Los siguientes son ejemplos de comparaciones de cadenas:

Expresión	Valor	Interpretación	Comentario
`"Hola"> "Adios"`	`1`	`verdadero`	La primera 'H' en Hola es mayor que la primera 'A' en Adiós
`"SOLANO" > "JIMENES"`	`1`	`verdadero`	La primera 'S' en SOLANO es mayor que la primera 'J' en JIMENEZ
`"123" > "1227"`	`1`	`verdadero`	El tercer carácter, el '3', en 123 es mayor que el tercer carácter, el '2', en 1227.
`"Bejuco" > "Beata"`	`1`	`verdadero`	El tercer carácter, la 'j', en Bejuco es mayor que el tercer carácter 'a' en Beata.
`"Hombre" == "Mujer"`	`0`	`falso`	La primera 'H' en Hombre no es igual a la primera 'M' en Mujer.
`"planta" <"planeta"`	`0`	`falso`	La 't' en planta es mayor que la 'e' en planeta.

Operadores lógicos

Además de usar expresiones relacionales simples como condiciones, pueden crearse condiciones más complejas usando los operadores lógicos AND, OR y NOT. Estos operadores son representados por los símbolos `&&`, `||`, y `!`, respectivamente.

Cuando el operador AND, `&&`, se usa con dos expresiones simples, la condición es verdadera sólo si ambas expresiones individuales son verdaderas por sí mismas. Por tanto, la condición lógica

```
(voltaje > 48) && (miliamperes < 10)
```

es verdadera sólo si `voltaje` es mayor que 48 y `miliamperes` es menor que 10. En vista que los operadores relacionales tienen una precedencia mayor que los operadores lógicos, podrían haberse omitido los paréntesis en esta expresión lógica.

El operador lógico OR, `||`, también se aplica entre dos expresiones. Cuando se usa el operador OR, la condición se satisface si cualquiera de las dos expresiones o ambas son verdaderas. Por tanto, la condición

```
(voltaje > 48) || (miliamperes < 10)
```

es verdadera si `voltaje` es mayor que 48, `miliamperes` es menor que 10, o si ambas condiciones son verdaderas. Una vez más, los paréntesis que rodean a las expresiones relacionales se incluyen para hacer más fácil de leer la expresión. Debido a la mayor precedencia de los operadores relacionales en relación con los operadores lógicos, se haría la misma evaluación aun si se omitieran los paréntesis.

Para las declaraciones

```
int i, j;
double a, b, completo;
```

las siguientes representan condiciones válidas:

```
a > b
(i == j) || (a < b) || completo
(a/b > 5) && (i <= 20)
```

Antes que puedan ser evaluadas estas condiciones, deben conocerse los valores de `a`, `b`, `i`, `j`, y `completo`. Suponiendo que `a` = 12.0, `b` = 2.0, `i` = 15, `j` = 30 y `completo` = 0.0 las expresiones anteriores producirían los siguientes resultados:

Expresión	Valor	Interpretación
`a > b`	`1`	`completo`
`(i == j) || (a < b) || completo`	`0`	`falso`
`(a/b > 5) && (i <= 20)`	`1`	`verdadero`

El operador NOT se usa para cambiar una expresión a su estado opuesto; es decir, si la expresión tiene cualquier valor diferente de cero (verdadero), `!expression` produce un valor de cero (falso). Si una expresión es falsa para comenzar (tiene un valor de cero), `!expression` es verdadero y produce 1. Por ejemplo, asumiendo que el número 26 es almacenado en la variable `edad`, la expresión `edad > 40` tiene un valor de cero (es falsa), mientras la expresión `!(edad > 40)` tiene un valor de 1. En vista que el operador NOT se usa sólo con una expresión, es un operador unitario.

Los operadores relacionales y lógicos tienen una jerarquía de ejecución similar a la de los operadores aritméticos. La tabla 4.2 enumera la precedencia de estos operadores en relación con los otros operadores usados.

Tabla 4.2

Operador	Asociatividad
`!` unitario `-` `++` `--`	derecha a izquierda
`*` `/` `%`	izquierda a derecha
`+` `-`	izquierda a derecha
`<` `<=` `>` `>=`	izquierda a derecha
`==` `!=`	izquierda a derecha
`&&`	izquierda a derecha
`||`	izquierda a derecha
`=` `+=` `-=` `*=` `/=`	derecha a izquierda

El siguiente ejemplo ilustra el uso de la precedencia y asociatividad de un operador para evaluar expresiones relacionales, suponiendo las siguientes declaraciones:

```
char key = 'm';
int i = 5, j = 7, k = 12;
double x = 22.5;
```

Expresión	Expresión equivalente	Valor	Interpretación
`i + 2 == k - 1`	`(i + 2) == (k - 1)`	0	falso
`3 * i - j <  22`	`(3 * i) - j < 22`	1	verdadero
`i + 2 * j > k`	`(i + (2 * j)) > k`	1	verdadero
`k + 3 <= -j + 3 * i`	`(k + 3) <= ((-j) + (3*i))`	0	falso
`'a' + 1 == 'b'`	`('a' + 1) == 'b'`	1	verdadero
`key - 1 >  'p'`	`(key - 1) > 'p'`	0	falso
`key + 1 == 'n'`	`(key + 1) == 'n'`	1	verdadero
`25 >= x + 1.0`	`25 >= (x + 1.0)`	1	verdadero

Como con todas las expresiones, los paréntesis pueden utilizarse para alterar la prioridad del operador asignado y mejorar la legibilidad de las expresiones relacionales. Al evaluar primero las expresiones dentro de los paréntesis, la siguiente condición compuesta se evalúa como

```
(6 * 3 == 36 / 2) || (13 < 3 * 3 + 4) && !(6 - 2 < 5)
       (18 == 18) ||   (13 < 9 + 4)    && !(4 < 5)
                1 ||   (13 < 13)       && !1
                1 ||          0        &&  0
                1 ||          0
                  1
```

Un problema de exactitud numérica

Un problema que puede ocurrir con las expresiones relacionales de C++ es uno de exactitud numérica sutil relacionado con los números de precisión simple y precisión doble. Debido a la forma en que las computadoras almacenan estos números, deberían evitarse las pruebas de igualdad de valores y variables de precisión simple y precisión doble usando el operador relacional ==.

La razón para esto es que muchos números decimales, como 0.1, por ejemplo, no pueden ser representados con exactitud en binario usando un número finito de bits. Por tanto, la prueba de igualdad exacta para estos números puede fallar. Cuando se desea la igualdad de valores no enteros es mejor requerir que el valor absoluto de la diferencia entre operandos sea menor que algún valor extremadamente pequeño. Por tanto, para operandos de precisión simple y precisión doble, la expresión general

operando_1 == operando_2

debería reemplazarse por la condición

abs(operando_1 – operando_2) < 0.000001

donde el valor 0.000001 puede alterarse con cualquier otro valor aceptablemente pequeño. Por tanto, si la diferencia entre los dos operandos es menor que 0.000001 (o cualquier otra

cantidad seleccionada por el usuario), los dos operandos se consideran iguales en esencia. Por ejemplo, si `x` y `y` son variables de precisión simple, una condición como

```
x/y == 0.35
```

debería ser programada como

```
abs(x/y - 0.35) < EPSILON
```

donde `EPSILON` puede ser una constante fijada en cualquier valor aceptablemente pequeño, como $0.000001._2$ Esta última condición asegura que las ligeras inexactitudes al representar números no enteros en binario no afecten la evaluación de la condición probada. En vista que todas las computadoras tienen una representación binaria exacta de cero, las comparaciones para igualdad exacta a cero no encuentran este problema de exactitud numérica.

Ejercicios 4.1

1. Determine el valor de las siguientes expresiones. Suponga `a` = 5, `b` = 2, `c` = 4, `d` = 6, y `e` = 3.
   - **a.** `a > b`
   - **b.** `a != b`
   - **c.** `d % b == c % b`
   - **d.** `a * c != d * b`
   - **e.** `d * b == c * e`
   - **f.** `!(a * b)`
   - **g.** `!(a % b * c)`
   - **h.** `!(c % b * a)`
   - **i.** `b % c * a`
2. Usando paréntesis, vuelva escribir las siguientes expresiones para indicar en forma correcta su orden de evaluación. Luego evalúe cada expresión suponiendo que `a` = 5, `b` = 2 y `c` = 4.
   - **a.** `a % b * c && c % b * a`
   - **b.** `a % b * c || c % b * a`
   - **c.** `b % c * a && a % c * b`
   - **d.** `b % c * a || a % c * b`
3. Escriba expresiones relacionales para denotar las siguientes condiciones (use nombres variables de su elección):
   - **a.** La distancia es igual a 30 pies.
   - **b.** La temperatura ambiente es 86.4.
   - **c.** Una velocidad es 55 MPH.
   - **d.** El mes actual es 12 (diciembre).
   - **e.** La letra introducida es K.
   - **f.** Una longitud es mayor que dos pies y menor que tres pies.
   - **g.** El día actual es el 15o. día del 1er mes.

[2]El uso de la función `abs()` requiere la inclusión del archivo de encabezado `cmath`. Esto se hace colocando la instrucción de preprocesador `#include<cmath>` ya sea inmediatamente antes o después de la instrucción de preprocesador `#include<iostream>`. Los sistemas basados en UNIX también requieren la inclusión específica de la biblioteca matemática en tiempo de compilación con un argumento `-lm` en la línea de comandos.

h. La velocidad del automóvil es 35 MPH y su aceleración es mayor que 4 MPH por segundo.
i. La velocidad de un automóvil es mayor que 50 MPH y se ha estado moviendo al menos por 5 horas.
j. El código es menor que 500 caracteres y toma más de 2 microsegundos en transmitirse.

4. Determine el valor de las siguientes expresiones, suponiendo que `a` = 5, `b` = 2, `c` = 4 y `d` = 5.

a. `a == 5`
b. `b * d == c * c`
c. `d % b * c > 5 || c % b * d < 7`

4.2 La instrucción `if-else`

La estructura `if-else` dirige a la computadora a ejecutar una serie de una o más instrucciones basadas en el resultado de una comparación. Por ejemplo, suponga que se va a calcular el área de un círculo dado el radio como un valor de entrada. Si la entrada es un número negativo se va a imprimir un mensaje indicando que el radio no puede ser un valor negativo; de lo contrario se va a calcular e imprimir el área del círculo. La estructura `if-else` puede utilizarse en esta situación para seleccionar la operación correcta con base en si el radio es negativo o no. La sintaxis general de la instrucción `if-else` es

```
if (expresion)  instruccion1;
else instruccion2;
```

La expresión se evalúa primero. Si el valor de la expresión es diferente de cero, se ejecuta *instruccion1*. Si el valor es cero, se ejecuta la instrucción después de la palabra clave else. Por tanto, siempre se ejecuta una de las dos instrucciones (ya sea *instruccion1* o *instruccion2*, pero no ambas) dependiendo del valor de la expresión. Hay que observar que la expresión probada debe ponerse entre paréntesis y que se coloca un punto y coma después de cada instrucción

Por claridad, la instrucción `if-else` se escribe de manera típica en cuatro líneas usando la forma

```
if (expresion) <-------- no va punto y coma aquí
    instruccion1;
else <------------------ no va punto y coma aquí
   instruccion2;
```

La forma de la instrucción `if-else` que se selecciona de manera típica depende del largo de las instrucciones 1 y 2. Sin embargo, cuando se usa la segunda forma, no se pone un punto y coma después del paréntesis o de la palabra clave `else`. Los puntos y comas sólo van después del final de las instrucciones.

El diagrama de flujo para la instrucción `if-else` se muestra en la figura 4.2.

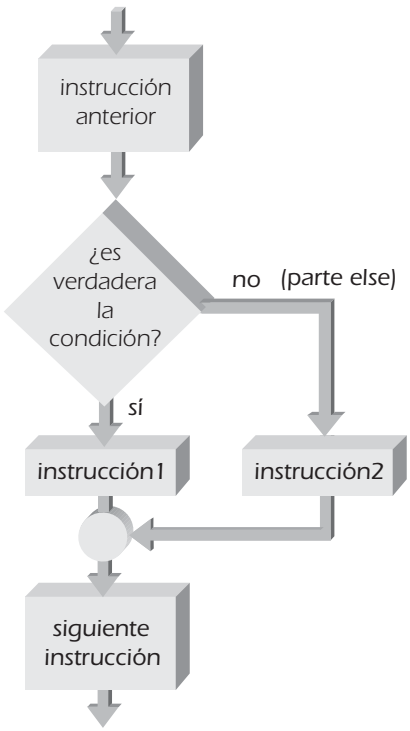


Figure 4.2 *El diagrama de flujo de* `if-else`.

Como un ejemplo específico de una estructura `if-else`, se eleborará un programa C++ para determinar el área de un círculo examinando primero el valor del radio. La condición que se va a probar es si el radio es menor que cero. Una instrucción `if-else` apropiada para esta situación es:

```
if (radio < 0.0)
  cout << "Un radio negativo es invalido" << endl;
else
  cout << "El area de este circulo es " << 3.1416 * pow(radius,2) << endl;
```

Aquí se ha usado el operador relacional < para representar la relación "menor que". Si el valor del radio es menor que 0, la condición es verdadera (tiene un valor de 1) y la instrucción `cout >> "Un radio negativo es invalido";` se ejecuta. Si la condición no es verdadera, el valor de la expresión es cero, y se ejecuta la instrucción posterior a la palabra clave `else`. El programa 4.1 ilustra el uso de esta instrucción en un programa completo.

Programa 4.1

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

int main()
{
  double radio;

  cout << "Por favor introduzca el radio: ";
  cin  >> radio;

  if (radio < 0.0 )
    cout << "Un radio negativo es invalido" << endl;
  else
    cout << "El area de este circulo es " << 3.1416 * pow(radio,2) << endl;

  return 0;
}
```

Se insertó una línea en blanco antes y después de la instrucción `if-else` para resaltarla en el programa completo. Se continuará haciendo esto a lo largo del texto para enfatizar la instrucción que se está presentando.

Para ilustrar la selección en acción, el programa 4.1 se ejecutó dos veces con diferentes datos de entrada. Los resultados fueron

```
Por favor introduzca el radio: -2.5
Un radio negativo es invalido
```

y:

```
Por favor introduzca el radio: 2.5
El area de este circulo es 19.635
```

Al revisar esta salida, observe que el radio en la primera ejecución del programa fue menor que 0 y la parte `if` de la estructura `if-else` ejecutó de manera correcta la instrucción `cout`, indicando al usuario que un radio negativo es inválido. En la segunda ejecución, el radio no es negativo y la parte `else` de la estructura `if-else` se usó para producir un cálculo correcto del área de

$$3.1416 * (2.5)^2 = 19.635$$

Aunque cualquier expresión puede ser probada con una instrucción `if-else`, se usan de manera predominante expresiones relacionales. Sin embargo, instrucciones como

```
if (num)
  cout << "¡Loteria!";
```

```
else
  cout << "¡Perdiste!";
```

son válidas. En vista que `num`, en sí misma, es una expresión válida, el mensaje `¡Loteria!` es desplegado si `num` tiene cualquier valor diferente de cero y el mensaje `¡Perdiste!` se despliega si `num` tiene un valor de cero.

Instrucciones compuestas

Aunque nada más se permite una sola instrucción en las partes `if` y `else` de la instrucción `if-else`, esta instrucción puede ser una sola instrucción compuesta. Una ***instrucción compuesta*** es una secuencia de instrucciones individuales contenidas entre llaves, como se muestra en la figura 4.3.

```
{
   instruccion1;
   instruccion2;
   instruccion3;
        .
        .
        .
   ultima instruccion;
}
```

Figura 4.3 *Una instrucción compuesta consiste de instrucciones individuales encerradas entre llaves.*

El uso de llaves para encerrar un conjunto de instrucciones individuales crea un solo bloque de instrucciones, el cual puede usarse en cualquier parte en un programa en C++ en lugar de una sola instrucción. El siguiente ejemplo ilustra el uso de una instrucción compuesta dentro de la forma general de una instrucción `if-else`.

```
if (expresion)
{
  instruccion1; // pueden colocarse dentro de las llaves
  instruccion2; // tantas instrucciones como sean
                   necesarias
  instruccion3; // cada instruccion debe terminar con un;
}
else
{
  instruccion4;
  instruccion5;
       .
       .
  instruccion_n;
}
```

El programa 4.2 ilustra el uso de una instrucción compuesta en un programa real.

Programa 4.2

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

// un programa para conversion de temperaturas
int main()
{
  char tipo_temp;
  double temp, fahren, celsius;

  cout << "Introduzca la temperatura que se va a convertir: ";
  cin  >> temp;
  cout << "Introduzca una f si la temperatura es en grados Fahrenheit";
  cout << "\n o una c si la temperatura es en grados Celsius: ";
  cin  >> tipo_temp;

    // establecer los formatos de salida
  cout << setiosflags(ios::fixed)
       << setiosflags(ios::showpoint)
       << setprecision(2);

  if (tipo_temp == 'f')
  {
    celsius = (5.0 / 9.0) * (temp - 32.0);
    cout << "\nLa temperatura Celsius equivalente es "
         << celsius << endl;
  }
  else
  {
    fahren =  (9.0 / 5.0) * temp + 32.0;
    cout << "\nLa temperatura Fahrenheit equivalente es "
         << fahren << endl;
  }

  return 0;
}
```

El programa 4.2 verifica si el valor en `tipo_temp` es f. Si el valor es f, se ejecuta la instrucción compuesta correspondiente a la parte `if` de la instrucción `if-else`. Cualquier otra letra produce la ejecución de la instrucción compuesta correspondiente a la parte `else`. A continuación se presenta una muestra de ejecución del programa 4.2.

```
Introduzca la temperatura que se va a convertir: 212
Introduzca una f si la temperatura es en grados Fahrenheit
o una c si la temperatura es en grados Celsius: f

La temperatura Celsius equivalente es 100.00
```

El alcance de un bloque

Todas las instrucciones contenidas dentro de una instrucción compuesta constituyen un solo bloque de código y cualquier variable declarada dentro de dicho bloque sólo tiene significado entre su declaración y las llaves de cierre que definen el bloque. Por ejemplo, considere la siguiente sección de código, la cual consta de dos bloques de código:

```
{   // comienzo del bloque exterior
   int a = 25;
   int b = 17;

   cout << "El valor de a es " << a
        <<" and b is " << b << endl;

   {    // comienzo del bloque interior
     float a = 46.25;
     int c = 10;

     cout << "a es ahora " << a
          << " b es ahora " << b
          << " y c es " << c << endl;
   }   // fin del bloque interior

   cout << "a es ahora " << a
        << " y b es " << b << endl;

}   // fin del bloque exterior
```

La salida que produce esta sección de código es:

```
El valor de a es 25 y b es 17
a es ahora 46.25 b es ahora 17 y c es 10
a es ahora 25 y b es 17
```

Esta salida se produce como sigue: El primer bloque de código define dos variables nombradas `a` y `b`, las cuales pueden usarse en cualquier parte dentro de este bloque después de su declaración, incluyendo cualquier bloque contenido dentro de éste. Dentro del bloque interior, se han declarado dos variables nuevas, nombradas `a` y `c`. En esta etapa entonces, se han declarado cuatro variables diferentes, dos de las cuales tienen el mismo nombre. Cualquier variable referenciada primero produce un intento de acceso a una variable declarada en forma correcta dentro del bloque que contiene la referencia. Si no se define una variable dentro del bloque, se hace un intento de acceso a una variable en el siguiente bloque inmediato exterior hasta que resulta un acceso válido.

Punto de Información

Colocación de llaves en una instrucción compuesta

Una práctica común para algunos programadores en C++ es colocar la llave de apertura de una instrucción compuesta en la misma línea que las instrucciones `if` y `else`. Usando esta convención, la instrucción `if` en el programa 4.2 aparecería como se muestra a continuación. (Esta colocación sólo es una cuestión de estilo, ambos estilos se usan y ambos son aceptables.)

```
if (tipo_temp == 'f') {
  celsius = (5.0 / 9.0) * (temp - 32.0);
  cout << "\nLa temperatura Celsius equivalente es "
       << celsius << endl;
}
else {
  fahren = (9.0 / 5.0) * temp + 32.0;
  cout << "\nLa temperatura Fahrenheit equivalente es "
       << celsius << endl;
}
```

Por tanto, los valores de las variables `a` y `c` referenciadas dentro del bloque interior usan los valores de las variables `a` y `c` declaradas en ese bloque. En vista que ninguna variable nombrada `b` fue declarada dentro del bloque interior, el valor de `b` desplegado desde dentro del bloque interior se obtiene del bloque exterior. Por último, el último objeto `cout`, el cual está fuera del bloque interior, despliega el valor de la variable `b` declarada en el bloque exterior. Si se hiciera un intento por desplegar el valor de `c` en cualquier parte en el bloque exterior, el compilador emitiría un mensaje de error declarando que `c` es un símbolo no definido.

El área dentro de un programa donde una variable puede utilizarse se conoce de manera formal como el **alcance** de la variable, y se ampliará este tema en el capítulo 6.

Selección unidireccional

Una modificación útil de la instrucción `if-else` implica omitir por completo la parte `else` de la instrucción. En este caso, la instrucción `if` adquiere la forma abreviada y con frecuencia útil

```
if (expresion)
  instruccion;
```

La instrucción que sigue a `if (expresion)` sólo se ejecuta si la expresión tiene un valor diferente de cero (una condición verdadera). Como antes, la instrucción puede ser una instrucción compuesta. El diagrama de flujo para esta instrucción se ilustra en la figura 4.4.

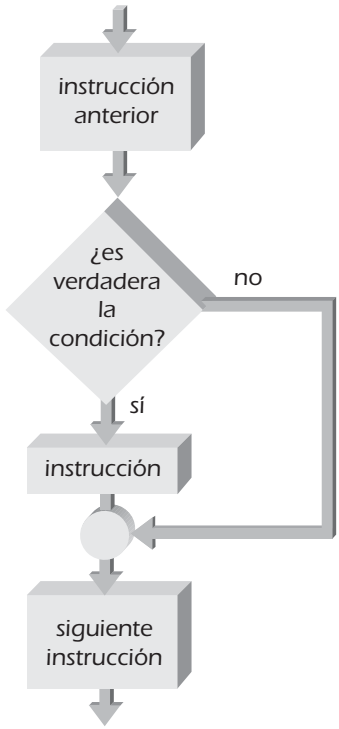


Figura 4.4 *Diagrama de flujo para la instrucción* `if` *unidireccional.*

Esta forma modificada de la instrucción `if` se llama instrucción `if` unidireccional. El programa 4.3 usa esta instrucción para desplegar de manera selectiva un mensaje para automóviles que han sido conducidos más de 3000.0 millas.

Programa 4.3

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

int main()
{
  const double LIMITE = 3000.0;
  int num_id;
  double millas;

  cout << "Por favor introduzca numero de automovil y millas recorridas: ";
  cin  >> num_id >> millas;

  if(millas > LIMITE)
    cout << "  El automovil " << num_id << " esta arriba del limite.\n";

  cout << "Fin de la salida del programa.\n";

  return 0;
}
```

Como una ilustración de estos criterios de selección unidireccionales en acción, el programa 4.3 se ejecutó dos veces, cada vez con diferentes datos de entrada. Sólo los datos de entrada para la primera ejecución causan que se despliegue el mensaje `El automovil 256 esta arriba del limite`.

```
Por favor introduzca numero de automovil y millas
  recorridas: 256 3562.8
   El automovil 256 esta arriba del limite.
Fin de la salida del programa.
```

y

```
Por favor introduzca numero de automovil y millas
  recorridas: 23 2562.3
Fin de la salida del programa.
```

Problemas asociados con la instrucción `if-else`

Dos de los problemas más comunes que se pueden enfrentar al empezar a usar la instrucción `if-else` de C++ son:

1. Comprender mal las implicaciones completas de lo que es una expresión y
2. Usar el operador de asignación, =, en lugar del operador relacional ==.

Recuerde que una expresión es cualquier combinación de operandos y operadores que produce un resultado. Esta definición es demasiado amplia y abarca más de lo que parece al principio. Por ejemplo, todas las siguientes son expresiones C++ válidas:

```
edad + 5
edad = 30
edad == 40
```

Suponiendo que las variables son declaradas de manera adecuada, cada una de las expresiones anteriores produce un resultado. El programa 4.4 usa el objeto `cout` para desplegar el valor de estas expresiones cuando `edad` = 18.

Programa 4.4

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

int main()
{
  int edad = 18;

  cout << "El valor de la primera expresion es " << (edad + 5) << endl;
  cout << "El valor de la segunda expresion es " << (edad = 30) << endl;
  cout << "El valor de la tercera expresion es " << (edad == 40) << endl;

  return 0;
}
```

El despliegue producido por el programa 4.4 es:

```
El valor de la primera expresion es 23
El valor de la segunda expresion es 30
El valor de la tercera expresion es 0
```

Como ilustra la salida del programa 4.4, cada expresión, por sí misma, tiene un valor asociado con ella. El valor de la primera expresión es la suma de la variable `edad` más 5, el cual es 23. El valor de la segunda expresión es 30, el cual también es asignado a la variable `edad`. El valor de la tercera expresión es cero, en vista que `edad` no es igual a 40, y una condición falsa se representa en C++ con un valor de cero. Si el valor en edad hubiera sido 40, la expresión relacional a == 40 hubiera sido verdadera y habría tenido un valor de 1.

Punto de Información

El tipo de datos booleano

Antes del estándar actual ANSI/ISO de C++, éste no tenía un tipo de datos booleano incorporado con sus dos valores booleanos, verdadero y falso. En vista que este tipo de datos originalmente no era parte del lenguaje, una expresión probada podía no evaluarse como un valor booleano. Por tanto, la sintaxis

if(la expresión booleana es verdadera)
ejecute esta instrucción;

tampoco estaba incorporada en C o en C++. Más bien, tanto C como C++ usan la sintaxis que abarca más:

if(expresión)
ejecute esta instrucción;

donde *expresión* es cualquier expresión que produzca un valor numérico. Si el valor de la expresión probada es un valor diferente de cero se considera como verdadero, y sólo un valor de cero es considerado falso.

Como lo especifica el estándar ANSI/ISO de C++, éste tiene un tipo de datos booleano incorporado que contiene los dos valores, **verdadero** y **falso**. Las variables booleanas serán declaradas usando la palabra clave **bool**. Como se ponen en práctica en la actualidad, los valores reales representados por los dos valores booleanos, **verdadero** y **falso**, son los valores enteros 1 y 0, respectivamente. Por ejemplo, considere el siguiente programa, el cual declara dos variables booleanas.

```
#include <iostream>
using namespace std;
int main()
{
  bool t1, t2;
  t1 = verdadero;
  t2 = falso;
  cout << "El valor de t1 es " << t1
       << "\ny el valor de t2 es " << endl;
  return 0;
}
```

La salida producida por este programa es:

```
El valor de t1 es 1
y el valor de t2 es 0
```

Como se puede observar en la salida, los valores booleanos **verdadero** y **falso** están representados por los valores enteros 1 y 0, respectivamente. Los valores booleanos verdadero y falso tienen las siguientes relaciones.

```
!true= es falso
!false= es verdadero
```

Además, aplicar un operador ++ como sufijo o prefijo a una variable del tipo bool establecerá el valor booleano en **verdadero**. Los operadores – – de sufijo y prefijo no pueden aplicarse a variables booleanas.

Los valores booleanos también pueden compararse, como se ilustra en el siguiente código:

```
if (t1 == t2)
  cout << "Los valores son iguales" << endl;
else
  cout << "Los valores no son iguales" << endl;
```

Por último, asignar cualquier valor diferente de cero a una variable booleana produce que la variable se establezca como **verdadera**; es decir, un valor de 1; y asignar un valor de cero a una variable booleana produce que la variable se establezca como **falsa**; es decir, un valor de 0.

Ahora suponga que se pretende usar la expresión relacional `edad == 40` en la instrucción `if`

```
if (edad == 40)
  cout << "¡Feliz Cumpleaños!";
```

pero se escribió mal como `edad = 40`, lo que resulta en

```
if (edad = 40)
  cout << "¡Feliz Cumpleaños!";
```

En vista que el error da como resultado una expresión C++ válida, y cualquier expresión de C++ puede ser probada por una instrucción `if`, la instrucción `if` resultante es válida y causará que se imprima el mensaje `¡Feliz Cumpleaños!` sin importar cuál valor se haya asignado antes a edad. ¿Puede ver por qué?

La condición probada por la instrucción `if` no compara el valor en edad con el número 40, sino que asigna el número `40` a `edad`. Es decir, la expresión `edad = 40` no es una expresión relacional en absoluto, sino una expresión de asignación. Al completar la asignación la expresión en sí tiene un valor de 40. En vista que C++ trata cualquier valor diferente de cero como verdadero, se ejecuta la instrucción `cout`. Otra forma de ver esto es darse cuenta que la instrucción `if` es equivalente a las dos instrucciones siguientes:

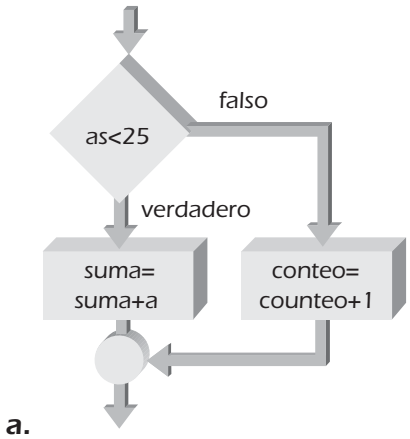
```
edad = 40;      // asigna 40 a edad
if (edad)       // prueba el valor de edad
  cout << "¡Feliz Cumpleaños!";
```

En vista que un compilador de C++ no tiene medio de saber que la expresión que se está probando no es la deseada, debe tenerse especial cuidado cuando se escriben condiciones.

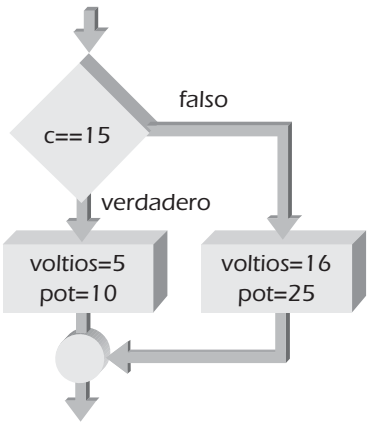
Ejercicios 4.2

1. Escriba instrucciones `if` apropiadas para cada una de las siguientes condiciones:
   - **a.** Si un ángulo es igual a 90 grados imprima el mensaje "El ángulo es un ángulo recto", de lo contrario imprima el mensaje "El ángulo no es un ángulo recto".
   - **b.** Si la temperatura está por encima de 100 grados desplegar el mensaje "arriba del punto de ebullición del agua", de lo contrario desplegar el mensaje "abajo del punto de ebullición del agua".
   - **c.** Si el número es positivo sumar el número a `sumpos`, si no sume el número a `sumneg`.
   - **d.** Si la pendiente es menor que 0.5 fijar la variable `flag` en cero, de lo contrario fijar `flag` en uno.
   - **e.** Si la diferencia entre `voltios1` y `voltios2` es menor que 0.001, fijar la variable `aprox` en cero, de lo contrario calcular `aprox` como la cantidad (`voltios1 – voltios2`) / 2.0.
   - **f.** Si la frecuencia es superior a 60, desplegar el mensaje "La frecuencia es demasiado alta".
   - **g.** Si la diferencia entre `temp1` y `temp2` excede 2.3, calcular `error` como `(`temp1 —temp2`) * factor.`
   - **h.** Si *x* es mayor que *y* y *z* es menor que 20, leer un valor para *p*.
   - **i.** Si la distancia es mayor que 20 y es menor que 35, leer un valor para tiempo.

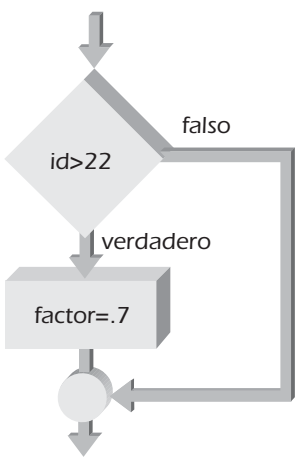
2. Escriba instrucciones `if` correspondientes a las condiciones ilustradas por cada uno de los siguientes diagramas de flujo.



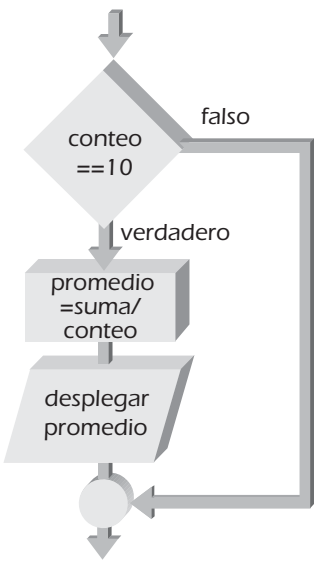
a.



b.



c.



d.

3. Escriba un programa en C++ que le pida al usuario que introduzca dos números. Si el primer número introducido es mayor que el segundo número el programa deberá imprimir el mensaje "El primer número es mayor", de lo contrario deberá imprimir el mensaje "El primer número es menor". Pruebe su programa introduciendo los números 5 y 8 y luego usando los números 11 y 2. ¿Qué piensa que desplegará su programa si los dos números introducidos son iguales? Pruebe este caso.

4. ***a.*** Una cierta onda es de 0 voltios para un tiempo menor que 2 segundos y de 3 voltios para un tiempo igual o mayor que 2 segundos (estas ondas se conocen como funciones de paso). Escriba un programa en C++ que acepte tiempo en la variable nombrada `tiempo` y despliegue el voltaje apropiado dependiendo del valor de entrada.

b. ¿Cuantas ejecuciones debería hacer para el programa escrito en el ejercicio 4a para verificar que opera en forma correcta? ¿Qué datos debería introducir en cada una de las ejecuciones del programa?

5. Una prueba de aislamiento para un cable requiere que el aislamiento resista al menos 600 voltios. Escriba un programa en C++ que acepte una prueba de voltaje e imprima el mensaje "PRUEBA DE VOLTAJE APROBADA" o el mensaje "PRUEBA DE VOLTAJE NO APROBADA", según sea apropiado.

6. a. Escriba un programa en C++ para calcular el valor de la presión en libras por pulgada cuadrada (psi) de una onda descrita como sigue:

Para tiempo, t, igual a o menor que 35 segundos, la presión es $0.46t$ psi y para tiempo mayor que 35 segundos la presión es $0.19t + 9.45$ psi.

El programa deberá solicitar el tiempo como entrada y deberá desplegar la presión como salida.

b. ¿Cuántas ejecuciones deberá hacer para el programa escrito en el ejercicio 6a para verificar que funciona en forma correcta? ¿Qué datos debería introducir en cada una de las ejecuciones del programa?

7. a. Escriba un programa en C++ que despliegue el mensaje "PROCEDER CON EL DESPEGUE" o "CANCELAR EL DESPEGUE" dependiendo de la entrada. Si el carácter g es introducido en la variable codigo, deberá desplegarse el primer mensaje; de lo contrario deberá desplegarse el segundo mensaje.

b. ¿Cuántas ejecuciones deberá hacer para el programa escrito en el ejercicio 7a para verificar que opera en forma correcta? ¿Qué datos debería introducir en cada una de las ejecuciones del programa?

8. Una fábrica pequeña genera su propia energía con un generador de 20 kilowatts y un generador de 50 kilowatts. El gerente de la planta indica cuál generador se requiere al introducir un código de carácter. Escriba un programa en C++ que acepte este código como entrada. Si se introduce el código s deberá desplegarse un mensaje que le indique al capataz de la planta que use el generador más pequeño; de lo contrario deberá ser la salida un mensaje que le indique el uso del generador más grande.

4.3 INSTRUCCIONES `if` ANIDADAS

Como se ha visto, una instrucción `if-else` puede contener instrucciones simples o compuestas. Puede usarse cualquier instrucción de C++ válida, incluyendo otra instrucción `if-else`. Por tanto, pueden incluirse una o más instrucciones `if-else` dentro de cualquier parte de una instrucción `if-else`. La inclusión de una o más instrucciones `if` dentro de una instrucción `if` se llama instrucción `if` *anidada*. Por ejemplo, sustituyendo la instrucción `if` unidireccional

```
if (distancia > 500)
  cout << "oprima";
```

por la instruccion1 en la siguiente instrucción if

```
if (horas < 9)
  instruccion1;
else
  cout << "suelte";
```

resulta la instrucción if anidada

```
if (horas < 9)
{
  if (distancia > 500)
    cout << "oprima";
}
else
  cout << "suelte";
```

Las llaves que rodean al if unidireccional interior son esenciales debido a que en su ausencia C++ asocia un else con el if más cercano que no tenga complemento. Por tanto, sin las llaves, la instrucción anterior es equivalente a

```
if (horas < 9)
  if (distancia > 500)
    cout << "oprima";
  else
    cout << "suelte";
```

Aquí else se empareja con el if interior, lo cual destruye el significado de la instrucción if-else original. Hay que observar también que la sangría es irrelevante en lo que respecta al compilador. Exista la sangría o no, *la instrucción es compilada al asociar el último* else *con el* if *más cercano no emparejado, a menos que se usen llaves para alterar el emparejamiento por omisión.*

El proceso de anidar instrucciones if puede extenderse de manera indefinida, de modo que la instrucción cout << "oprima"; podría reemplazarse por una instrucción if-else completa o por otra instrucción if unidireccional.

La figura 4.5 ilustra la forma general de una instrucción if-else anidada cuando una instrucción if-else es anidada a) dentro de la parte if de una instrucción if-else y b) dentro de la parte else de una instrucción if-else.

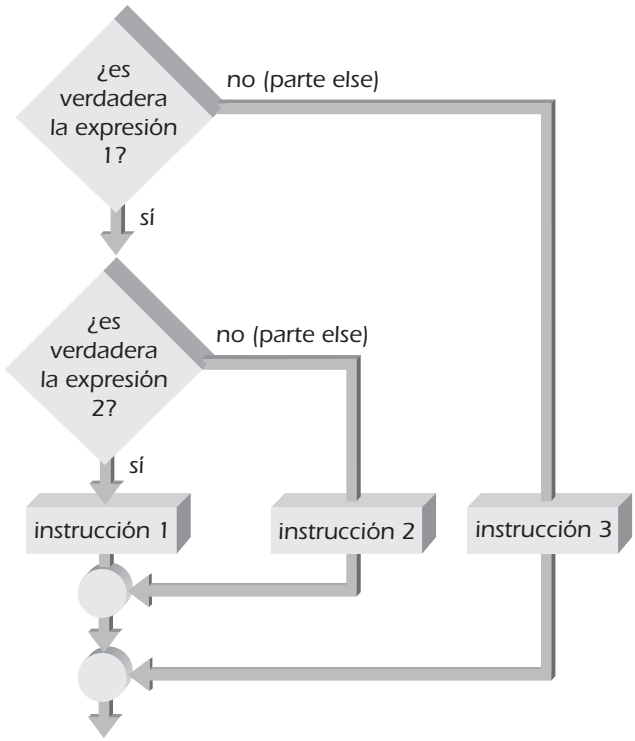


Figura 4.5a *Instrucción* `if-else` *anidada en una* `if`.

La cadena `if-else`

En general, la anidación ilustrada en la figura 4.5a tiende a ser confusa y es mejor evitarla en la práctica. Sin embargo, ocurre una construcción muy útil para la anidación ilustrada en la figura 4.5b, la cual tiene la forma

```
if (expresion_1)
  instruccion1;
else
  if (expresion_2)
    instruccion2;
  else
    instruccion3;
```

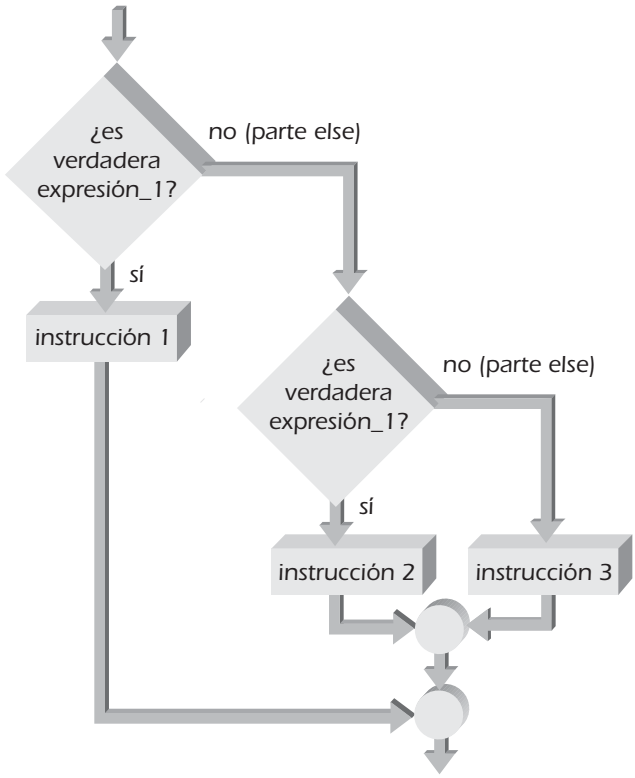


Figura 4.5b *Instrucción* `if-else` *anidada en un* `else`.

Como con todos los programas en C++, en vista que se ignora el espacio en blanco, la sangría mostrada no se requiere. De manera más típica, la construcción anterior se escribe usando el siguiente arreglo:

```
if (expresion_1)
  instruccion1;
else if (expression_2)
  instruccion2;
else
  instruccion3;
```

Esta forma de una instrucción `if` anidada es muy útil en la práctica, y de manera formal se conoce como `cadena if-else`. Cada condición es evaluada en orden, y si cualquier condición es verdadera se ejecuta la instrucción correspondiente y el resto de la cadena se termina. La instrucción asociada con el `else` final sólo se ejecuta si ninguna de las condiciones anteriores se satisface. Esto sirve como un caso por omisión o aplicado a todos los casos que es útil para detectar una condición imposible o errónea.

La cadena puede continuarse en forma indefinida al hacer de manera repetida que la última instrucción sea otra instrucción `if-else`. Por tanto, la forma general de una cadena `if-else` es:

```
if (expresion_1)
  instruccion1;
else if (expresion_2)
  instruccion2;
else if (expresion_3)
  instruccion3;
      .
      .
      .
else if (expresion_n)
  instruccion_n;
else
  ultima_instruccion;
```

Cada condición es evaluada en el orden en que aparece en la instrucción. Para la primera condición que es verdadera, se ejecuta la instrucción correspondiente, y el resto de las instrucciones en la cadena no se ejecutan. Por tanto, si `expresion_1` es verdadera, sólo se ejecuta `instruccion1`; de lo contrario se prueba `expresion_2`. Si `expresion_2` es verdadera entonces, sólo se ejecuta `instruccion2`; de lo contrario se prueba `expresion_3`, y así en forma sucesiva. El `else` final en la cadena es opcional, y `ultima_instruccion` sólo se ejecuta si ninguna de las expresiones previas fue verdadera.

Para ilustrar el uso de la cadena `if-else`, el programa 4.5 despliega el estado de especificación de cada elemento correspondiente a una letra introducida. Se usan los siguientes códigos de letra:

Estado de la especificación	Código de la entrada
Exploración espacial	`S`
Grado militar	`M`
Grado comercial	`C`
Grado juguete	`T`

Programa 4.5

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

int main()
{
  char codigo;

  cout << "Introduzca un codigo de especificacion: ";
  cin  >> codigo;

  if (codigo == 'S')
      cout << "El elemento tiene grado de exploracion espacial.";
  else if (codigo == 'M')
    cout << "El elemento tiene grado militar.";
  else if (code == 'C')
    cout << "El elemento tiene grado comercial.";
  else if (codigo == 'T')
      cout << "El elemento tiene grado de juguete.";
  else
      cout << "Se ha introducido un codigo invalido.";
  cout << endl;

  return 0;
}
```

Como un ejemplo más de una cadena `if-else`, se determina la salida de una unidad convertidora digital usando la siguiente relación entrada/salida:

Peso de entrada	Lectura de salida
mayor que o igual a 90 lbs	1111
menor que 90 lbs pero mayor que 80 lbs	1110
menor que 80 lbs pero mayor que o igual a 70 lbs	1101
menor que 70 lbs pero mayor que o igual a 60 lbs	1100
menor que 60 lbs	1011

Las siguientes instrucciones pueden usarse para determinar la lectura de salida correcta, donde la variable `inlbs` se usa para almacenar la lectura de entrada:

```
if (inlbs >= 90)
    digout = 1111;
```

```
else if (inlbs >= 80)
    digout = 1110;
else if (inlbs >= 70)
    digout = 1101;
else if (inlbs >= 60)
    digout = 1100;
else
    digout = 1011;
```

Hay que observar que este ejemplo aprovecha el hecho que la cadena se detiene una vez que se encuentra una condición verdadera. Esto se logra verificando primero la entrada de peso mayor. Si el valor de entrada es menor que 90, la cadena `if-else` continúa verificando el siguiente peso más alto, y así en forma sucesiva, hasta que se obtiene la categoría de peso correcta.

El programa 4.6 usa una cadena `if-else` para calcular y desplegar la lectura de salida correcta correspondiente a la entrada de peso en la instrucción `cin`.

Programa 4.6

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

int main()
{
  int digout;
  double inlbs;

  cout << "Introduzca el peso: ";
  cin  >> inlbs;

  if (inlbs >= 90)
      digout = 1111;
  else if (inlbs >= 80)
      digout = 1110;
  else if (inlbs >= 70)
    digout = 1101;
  else if (inlbs >= 60)
    digout = 1100;
  else
    digout = 1011;

  cout << "La salida digital es " << digout << endl;

  return 0;
}
```

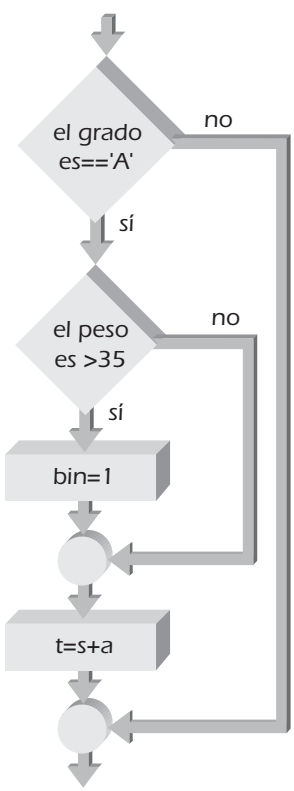
Una muestra de ejecución usando el programa 4.6 se ilustra a continuación.

```
Introduzca el peso: 72.5
La salida digital es 1101
```

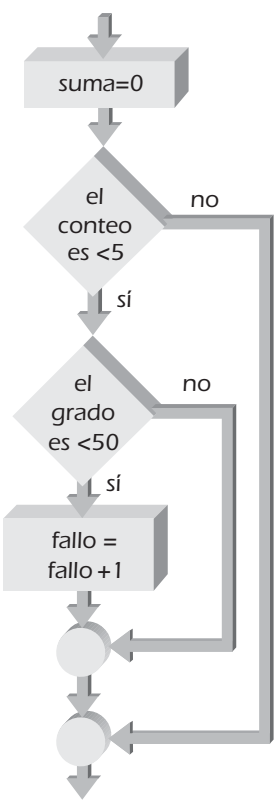
Como con todas las instrucciones de C++, cada instrucción individual dentro de una cadena `if-else` puede reemplazarse por una instrucción compuesta encerrada entre las llaves `{` y `}`.

Ejercicios 4.3

1. Modifique el programa 4.5 para que acepte letras minúsculas y mayúsculas como códigos. Por ejemplo, si un usuario introduce una m o una M, el programa deberá desplegar el mensaje "El elemento tiene grado militar."
2. Escriba instrucciones `if` anidadas correspondientes a las condiciones ilustradas en cada uno de los siguientes diagramas de flujo.



a.



b.

3. Un ángulo es considerado agudo si es menor que 90 grados, obtuso si es mayor que 90 grados y ángulo recto si es igual a 90 grados. Usando esta información, escriba un programa en C++ que acepte un ángulo, en grados, y despliegue el tipo de ángulo correspondiente a los grados introducidos.

4. El nivel de grado de los estudiantes universitarios se determina de manera típica de acuerdo con la siguiente tabla:

Número de créditos completados	Grado
menor que 32	primer año
32 a 63	segundo año
64 a 95	tercer año
96 o más	último año

Usando esta información, escriba un programa en C++ que acepte el número de créditos que ha completado un estudiante, determine el grado del estudiante y lo despliegue.

5. La letra que representa las calificaciones de un estudiante se calcula de acuerdo con la siguiente tabla:

Calificación numérica	Letra
mayor que o igual a 90	A
menor que 90 pero mayor que o igual a 80	B
menor que 80 pero mayor que o igual a 70	C
menor que 70 pero mayor que o igual a 60	D
menor que 60	F

Usando esta información, escriba un programa en C++ que acepte la calificación numérica de un estudiante, convierta la calificación numérica a su calificación en letra equivalente y despliegue la letra.

6. La tolerancia de componentes críticos en un sistema se determina por la aplicación de acuerdo con la siguiente tabla:

Estado de la especificación	Tolerancia
Exploración espacial	Menor que 0.1%
Grado militar	Mayor que o igual a 0.1% y menor que 1%
Grado comercial	Mayor que o igual a 1% y menor que 10%
Grado de juguete	Mayor que o igual a 10%

Usando esta información, escriba un programa en C++ que acepte la lectura de tolerancia de un componente y determine la especificación que debería asignarse al componente.

7. Escriba un programa en C++ que acepte un número seguido por un espacio y luego una letra. Si la letra que sigue al número es f, el programa tratará al número introducido como una temperatura en grados Fahrenheit, convertirá el número a los grados Celsius equivalentes y desplegará un mensaje adecuado. Si la letra que sigue al número es c, el programa tratará al número introducido como una temperatura en Celsius, convertirá el número a los grados Fahrenheit equivalentes y desplegará un mensaje adecuado. Si la letra no es f ni c, el programa imprimirá el mensaje que los datos introducidos son incorrectos y terminará. Use una cadena `if-else` en su programa y use las fórmulas de conversión:

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

8. Usando las relaciones del programa 4.6, el siguiente programa calcula la salida digital:

```
int main()
{
  int digout;
  double inlbs;

  cout << "Introduzca el peso: ";
  cin  >> inlbs;

  if (inlbs >= 90) digout = 1111;
  if (inlbs >= 80) && (inlbs <= 90) digout = 1110;
  if (inlbs >= 70) && (inlbs <= 80) digout = 1101;
  if (inlbs >= 60) && (inlbs <= 70) digout = 1100;
  if (inlbs < 1000) digout = 1011;

  cout << "La salida digital es " << digout << endl;

  return 0;
}
```

a. ¿Este programa producirá la misma salida que el programa 4.6?
b. ¿Cuál programa es mejor y por qué?

9. El siguiente programa fue escrito para producir el mismo resultado que el programa 4.6:

```
int main()
{
  int digout;
  double inlbs;

  cout << "Introduzca el peso: ";
  cin  >> inlbs;
```

```
    if (inlbs < 60)
        digout = 1011;
    else if (inlbs >= 60)
        digout = 1100;
    else if (inlbs >= 70)
      digout = 1101;
    else if (inlbs >= 80)
      digout = 1110;
    else if (inlbs >= 90)
      digout = 1111;

    cout << "La salida digital es " << digout << endl;

    return 0;
}
```

a. ¿Se ejecutará este programa?
b. ¿Qué hace este programa?
c. ¿Para cuáles valores de libras introducidos calculará este programa la salida digital correcta?

4.4 LA INSTRUCCIÓN `switch`

La cadena `if-else` se usa en aplicaciones de programación donde un conjunto de instrucciones debe ser seleccionada entre muchas alternativas posibles. La instrucción `switch` proporciona una alternativa a la cadena `if-else` para casos que comparan el valor de una expresión de número entero con un valor específico. La forma general de una instrucción `switch` es

```
switch (expresion)
{    // inicio de instrucción compuesta
  case valor_1:  <-- termina con dos puntos
    instruccion1;
    instruccion2;
        .
        .
    break;
  case valor_2:  <-- termina con dos puntos
    instruccion;
    instruccion;
        .
        .
    break;
   .
   .
```

```
  case valor_n: ◄──────────── termina con dos puntos
    instruccionw;
    instruccionx;
        .
        .
    break;
  default:      ◄──────────── termina con dos puntos
    instruccionaa;
    instruccionbb;
        .
}    // fin de switch y de la instrucción compuesta
```

La instrucción `switch` usa cuatro palabras clave nuevas: `switch`, `case`, `break` y `default`. Veamos lo que hace cada una de estas palabras.

La palabra clave `switch` identifica el inicio de la instrucción `switch`. La expresión entre paréntesis que sigue a la palabra es evaluada y el resultado de la expresión comparado con diversos valores alternativos contenidos dentro de la instrucción compuesta. La expresión en la instrucción `switch` deben evaluar un resultado en número entero o resultará un error de compilación.

Dentro de la instrucción `switch`, la palabra clave `case` se usa para identificar o etiquetar valores individuales que se comparan con el valor de la expresión `switch`. El valor de la expresión `switch` se compara con cada uno de estos valores `case` en el orden en que se enlistan estos valores hasta que se encuentra una correspondencia. Cuando los valores corresponden, la ejecución comienza con la instrucción que sigue inmediatamente a la correspondencia. Por tanto, como se ilustra en la figura 4.6, el valor de la expresión determina en qué parte de la instrucción `switch` comienza en realidad la ejecución.

Una instrucción `switch` puede contener cualquier número de etiquetas `case`, en cualquier orden. Sin embargo, si el valor de la expresión no corresponde con ninguno de los valores `case`, no se ejecuta ninguna instrucción a menos que se encuentre la palabra clave `default`. La palabra `default` es opcional y opera igual que la última `else` en una cadena `if-else`. Si el valor de la expresión no corresponde con ninguno de los valores `case`, la ejecución del programa comienza con la instrucción que sigue a la palabra `default`.

Una vez que la instrucción `switch` ha localizado un punto de entrada, todas las demás evaluaciones `case` son ignoradas y la ejecución continúa hasta el final de la instrucción compuesta a menos que se encuentre una instrucción `break`. Ésta es la razón por la que la instrucción `break`, la cual identifica el fin de un `case` particular y causa una salida inmediata de la instrucción `switch`. Por tanto, del mismo modo en que la palabra `case` identifica los puntos de partida posibles en la instrucción compuesta, la instrucción `break` determina puntos de terminación. Si se omiten las instrucciones `break`, se ejecutan todos los casos que siguen al valor `case` que tiene la correspondencia, incluyendo el caso `default`.

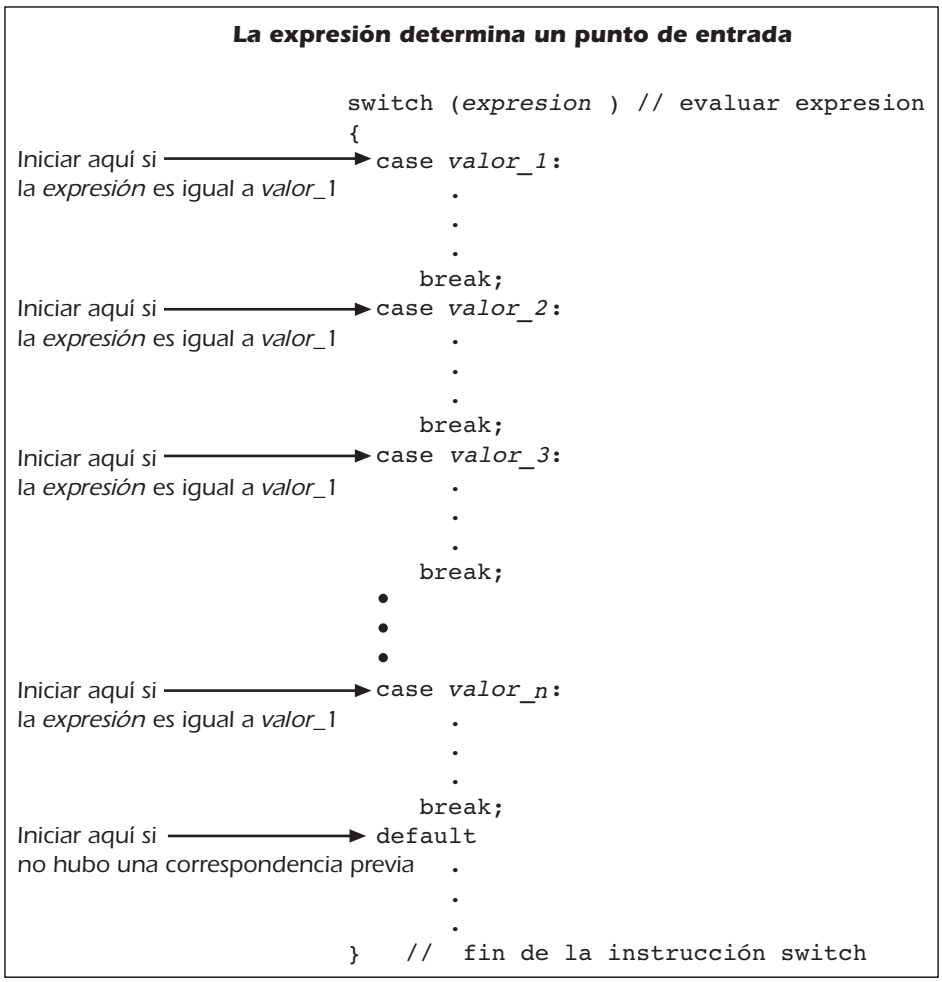


Figura 4.6 *La expresión determina un punto de entrada.*

Cuando escriba una instrucción `switch`, puede usar valores `case` múltiples para referirse al mismo conjunto de instrucciones; la etiqueta `default` es opcional. Por ejemplo, considere lo siguiente:

```
switch (numero)
{
  case 1:
    cout << "Que tenga una buena mañana\n";
    break;
  case 2:
    cout << "Que tenga un buen dia\n";
    break;
  case 3:
  case 4:
  case 5:
    cout << "Que tenga una buena tarde\n";
}
```

Si el valor almacenado en la variable número es 1, se despliega el mensaje `Que tenga una buena mañana`. Del mismo modo, si el valor de número es 2, se despliega el segundo mensaje. Por último, si el valor de número es 3 o 4 o 5, se despliega el último mensaje. En vista que la instrucción a ejecutarse en estos últimos tres casos es la misma, los casos para estos valores pueden "apilarse", como se muestra en el ejemplo. Además, en vista que no hay `default`, no se imprime ningún mensaje si el valor de número no es uno de los valores `case` enumerados. Aunque es una buena práctica de programación enlistar los valores `case` en orden ascendente, no lo requiere la instrucción `switch`. Una instrucción `switch` puede tener cualquier cantidad de valores `case`, en cualquier orden; sólo necesitan enlistarse los valores que se van a probar.

El programa 4.7 usa una instrucción switch para seleccionar la operación aritmética (adición, multiplicación o división) que se va a realizar con dos números dependiendo del valor de la variable `opselect`.

El programa 4.7 se ejecutó dos veces. El despliegue resultante identifica con claridad el caso seleccionado. Los resultados son

```
Por favor introduzca dos numeros: 12 3
Introduzca un codigo seleccionado:
          1 para adicion
          2 para multiplicacion
          3 para division : 2
El producto de los numeros introducidos es 36
```

y:

```
Por favor introduzca dos numeros: 12 3
Introduzca un código seleccionado:
          1 para adicion
          2 para multiplicacion
          3 para division : 3
El primer numero dividido entre el segundo es 4
```

Al revisar el programa 4.7, nótese la instrucción `break` en el último `case`. Aunque este `break` no es necesario, es una buena práctica terminar el último `case` en una instrucción `switch` con un `break`. Esto previene un posible error en el programa más tarde si después se agrega un case adicional a la instrucción `switch`. Con la adición de un `case` nuevo, el `break` entre `cases` se vuelve necesario; tener el `break` en su lugar le asegura que no olvidará incluirlo en el momento de la modificación.

Programa 4.7

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

int main()
{
  int opselect;
  double fnum, snum;

  cout << "Por favor introduzca dos números: ";
  cin  >> fnum >> snum;
  cout << "Introduzca un código seleccionado: ";
  cout << "\n        1 para adicion";
  cout << "\n        2 para multiplicacion";
  cout << "\n        3 para division : ";
  cin  >> opselect;

  switch (opselect)
  {
    case 1:
      cout << "La suma de los numeros introducidos es " << fnum+snum;
      break;
    case 2:
      cout << "El producto de los numeros introducidos es " << fnum*snum;
      break;
    case 3:
      cout << "El primer numero dividido entre el segundo es " << fnum/snum;
      break;
  }     // end of switch

  cout << endl;

  return 0;
}
```

Debido a que los tipos de datos de carácter siempre son convertidos a números enteros en una expresión, también puede usarse una instrucción `switch` para "cambiar" con base en el valor de una expresión de carácter. Por ejemplo, suponiendo que `eleccion` es una variable de carácter, la siguiente instrucción `switch` es válida:

```
switch(eleccion)
{
  case 'a':
  case 'e':
  case 'i':
  case 'o':
  case 'u':
```

```
    cout << "El caracter en eleccion es una vocal\n";
    break;
  default:
    cout << "El caracter en eleccion no es una vocal\n";
    break;    // este break es opcional
}    // fin de la instrucción switch
```

Ejercicios 4.4

1. Vuelva a escribir la siguiente cadena `if-else` usando una instrucción `switch`:

```
if (let_calif == 'A')
  cout << "La calificacion numerica esta entre 90 y 100\n";
else if (let_calif == 'B')
  cout << "La calificacion numerica esta entre 80 y 89.9\n";
else if (let_calif == 'C')
  cout << "La calificacion numerica esta entre 70 y 79.9\n";
else if (let_calif == 'D')
  cout << "Como va a explicar esta\n";
else
{
  cout << "Por supuesto que no tuve nada que ver con mi
           calificacion.\n";
  cout << "Debe ser culpa del profesor.\n";
}
```

2. Vuelva a escribir la siguiente cadena `if-else` usando una instrucción `switch`:

```
if (factor == 1)
  presion = 25.0;
else if (factor == 2)
  presion = 36.0;
else if (factor == 3)
 presion = 45.0;
else if (factor == 4) || (factor == 5) || (factor == 6)
  presion = 49.0;
```

3. Cada unidad de disco en un embarque de estos dispositivos tiene estampado un código del 1 al 4, el cual indica el fabricante de la unidad como sigue:

Código	Fabricante de la unidad de disco
1	3M Corporation
2	Maxell Corporation
3	Sony Corporation
4	Verbatim Corporation

Escriba un programa en C++ que acepte el número de código como una entrada y con base en el valor introducido despliegue el fabricante de la unidad de disco correcto.

4. Vuelva a escribir el programa 4.5 usando una instrucción `switch`.
5. Determine por qué la cadena `if-else` en el programa 4.6 no puede reemplazarse con una instrucción `switch`.
6. Vuelva a escribir el programa 4.7 usando una variable de carácter para el código seleccionado. (*Sugerencia:* revise la sección 3.4 si su programa no opera como pensaba que debería hacerlo.)

4.5 APLICACIONES

Dos usos principales de las instrucciones `if` de C++ son seleccionar rutas de procesamiento apropiadas y prevenir que los datos indeseables sean procesados. En esta sección se proporcionan ejemplos de ambos usos.

Aplicación 1: Validación de datos

Un uso importante de las instrucciones if de C++ es validar datos verificando los casos que son inválidos en forma clara. Por ejemplo, una fecha como 33/5/06 contiene un día que es obvio que es inválido. Del mismo modo, la división de cualquier número entre cero dentro de un programa, como 14/0, no debería permitirse. Estos dos ejemplos ilustran la necesidad de una técnica llamada **programación defensiva**, en la cual el programa incluye código para verificar datos impropios antes que se haga un intento de procesarlos más. La técnica de programación defensiva para verificar los datos introducidos por el usuario en busca de datos erróneos o irrazonables se conoce como **validación de datos de entrada**.

Considere el caso en el cual se escribe un programa en C++ para calcular la raíz cuadrada y el recíproco de un número introducido por un usuario. Antes de calcular la raíz cuadrada, validar que el número no sea negativo, y antes de calcular el recíproco, verificar que el número no sea cero.

Paso 1 Analizar el problema

El planteamiento del problema requiere que se acepte un solo número como entrada, se valide el número introducido y con base en la validación producir dos salidas posibles: si el número no es negativo se determina su raíz cuadrada, y si el número introducido no es cero se determina su recíproco.

Paso 2 Desarrollar una solución

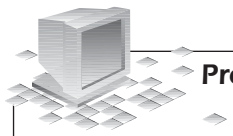
En vista que la raíz cuadrada de un número negativo no existe como un número real, y no puede tomarse el recíproco de cero, nuestro programa debe contener instrucciones de validación de datos de entrada para tamizar los datos introducidos por el usuario y evitar estos dos casos. El seudocódigo que describe el proceso requerido es:

Desplegar un mensaje con el propósito del programa
Aceptar un número introducido por un usuario

If el número es negativo
imprimir un mensaje indicando que no puede calcularse la raíz cuadrada
Else
calcular y desplegar la raíz cuadrada
Endif
If el número es cero entonces
imprimir un mensaje indicando que no puede obtenerse el recíproco
Else
calcular y desplegar el recíproco
Endif

Paso 3 Codificar la solución

El código C++ correspondiente a nuestra solución en seudocódigo se enumera en el programa 4.8.



Programa 4.8

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

int main()
{
  double usenum;

  cout << "Este programa calcula la raiz cuadrada y\n"
       << "el reciproco (1/numero) de un numero\n"
       << "\nPor favor introduzca un numero: ";
  cin  >> usenum;
  if (usenum < 0.0)
    cout << "La raiz cuadrada de un numero negativo no existe.\n";
  else
    cout << "La raiz cuadrada de " << usenum
         << " es " << sqrt(usenum) << endl;
  if (usenum == 0.0)
    cout << "El reciproco de cero no existe.\n";
  else
    cout << "El reciproco de " << usenum
         << " es " <<  (1.0/usenum) << endl;

  return 0;
}
```

El programa 4.8 es un programa bastante sencillo que contiene dos instrucciones `if` separadas (no anidadas). La primera instrucción `if` verifica si se ha introducido un número negativo; si el número es negativo se despliega un mensaje que indica que no puede calcularse la raíz cuadrada de un número negativo, de lo contrario se calcula la raíz cuadrada. La segunda instrucción `if` comprueba si el número introducido es cero; si lo es se despliega un mensaje que indica que no puede obtenerse el recíproco de cero, de lo contrario se calcula el recíproco.

Paso 4 Probar y corregir el programa

Los valores de prueba deberán incluir un número positivo, y valores para los casos limitantes, como un valor de entrada negativo y un cero. A continuación se presenta la ejecución de prueba para dos de estos casos:

```
Este programa calcula la raiz cuadrada y
el reciproco (1/numero) de un numero

Por favor introduzca un numero: 5

La raiz cuadrada de 5 es 2.23607
El reciproco de 5 es 0.2
```

y

```
Este programa calcula la raiz cuadrada y
el reciproco (1/numero) de un numero

Por favor introduzca un numero: -6

La raiz cuadrada de un numero negativo no existe
El reciproco de -6 es -0.166667
```

Aplicación 2: Resolver ecuaciones cuadráticas

Una **ecuación cuadrática** es una ecuación que tiene la forma $ax^2 + bx + c = 0$ o que puede manipularse en forma algebraica en esta forma. En esta ecuación x es la variable desconocida, y a, b y c son constantes conocidas. Aunque las constantes b y c pueden ser cualquier número, incluyendo cero, el valor de la constante a no puede ser cero (si a es cero, la ecuación se volvería una **ecuación lineal** en x). Son ejemplos de ecuaciones cuadráticas

$$5x^2 + 6x + 2 = 0$$
$$x^2 - 7x + 20 = 0$$
$$34x^2 + 16 = 0$$

En la primera ecuación $a = 5$, $b = 6$ y $c = 2$; en la segunda ecuación $a = 1$, $b = -7$ y $c = 20$; y en la tercera ecuación $a = 34$, $b = 0$ y $c = 16$.

Las raíces reales de una ecuación cuadrática pueden calcularse usando la fórmula cuadrática como:

$$\text{raíz 1} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

y

$$\text{raíz 2} = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Usando estas ecuaciones escribiremos un programa en C++ para resolver las raíces de una ecuación cuadrática.

Paso 1 Analizar el problema

El problema requiere que se acepten tres entradas: los coeficientes *a*, *b* y *c* de una ecuación cuadrática, y calcular las raíces de la ecuación usando las fórmulas dadas.

Paso 2 Desarrollar una solución

Un primer intento de solución sería usar los valores de *a*, *b* y *c* introducidos por el usuario para calcular en forma directa un valor para cada una de las raíces. Por tanto, nuestra primera solución sería

```
Desplegar un mensaje con el propósito del programa
Aceptar valores introducidos por el usuario para a, b y c
Calcular las dos raíces
Desplegar los valores calculadas de las raíces
```

Sin embargo, esta solución debe ser definada para tomar en cuenta las posibles condiciones de entrada. Por ejemplo, si un usuario introduce un valor de *D* para *a* y *b*, la ecuación no es ni cuadrática ni lineal y no tiene solución (se hace referencia como un caso degenerado). Otra posibilidad es que el usuario proporcione un valor no cero para *b* por lo que *a* es cero. En este caso la ecuación es un uno lineal con una solución $-c / b$. Una tercera posibilidad es que el valor del término $b^2 - 4ac$, el cual es llamado **diseniminante**, sea negativo. Entonces la raíz cuadrada de un número negativo no se tomará en cuenta, este caso no tendrá raíces verdaderás. Por último, cuando el diseniminante es cero ambas raíces son las mismas (se hace referencia a ello como un caso de repetición de raíces).

Tomando en cuenta estos cuatro casos limitantes, una solución refinada para la correcta determinada de las raíces de la ecuación cuadríatica es expresada por el siguiente seudo código:

```
Desplegar un mensaje con el propósito del programa
Aceptan valores introduccidos por el usuario para a, b y c
If a = 0 y b = 0 entonces
    desplegar un mensaje diciendo que la ecuación no tiene solución
Else if a = cero entonces
    calcular la raíz individual igual a –c/b
    desplegar la raíz individual
Else
    calcular el discriminante
```

```
    If el discriminante > 0 entonces
        resolver ambas raíces usando las fórmulas dadas
        desplegar las dos raíces
    Else if el discriminante < 0 entonces
        desplegar un mensaje de que no hay raíces reales
    Else
        calcular la raíz repetida igual a –b/(2a)
        desplegar la raíz repetida
    Endif
Endif
```

Nótese en el seudocódigo que se han usado instrucciones `if-else` anidadas. La instrucción `if-else` exterior se usa para validar los coeficientes introducidos y determinar que se tiene una ecuación cuadrática válida. La instrucción `if-else` interior se usa entonces para determinar si la ecuación tiene dos raíces reales (discriminante > 0), dos raíces imaginarias (discriminante < 0) o raíces repetidas (discriminante = 0).

Paso 3 Codificar la solución

El código C++ equivalente correspondiente a nuestra solución en seudocódigo se enlista en el programa 4.9.

Paso 4 Probar y corregir el programa

Los valores de prueba deberán incluir valores para a, b y c que produzcan dos raíces reales, además de valores limitantes para a y b que produzcan una ecuación lineal ($a = 0$, $b \neq 0$), una ecuación degenerada ($a = 0$, $b = 0$), y un discriminante negativo y cero. A continuación se muestran dos de estas ejecuciones de prueba del programa 4.9:

```
Este programa calcula las raices de una
   ecuacion cuadratica de la forma
            2
          ax + bx + c = 0
Por favor introduzca valores para a, b y c: 1 2 -35

Las dos raices reales son 5 y -7
```

y

```
Este programa calcula las raices de una
   ecuacion cuadratica de la forma
            2
          ax + bx + c = 0

Por favor introduzca valores para a, b y c: 0 0 16

La ecuacion es degenerada y no tiene raices.
```

Programa 4.9

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

// este programa calcula las raices de una ecuacion cuadratica
int main()
{
  double a, b, c, disc, raiz1, raiz2;

  cout << "Este programa calcula las raices de una\n";
  cout << "   ecuacion cuadratica de la forma\n";
  cout << "             2\n";
  cout << "          ax + bx + c = 0\n\n";
  cout << "Por favor introduzca valores para a, b y c: ";
  cin >>  a >>  b >> c;
  if ( a == 0.0 && b == 0.0)
    cout << "La ecuacion es degenerada y no tiene raices.\n";
  else if (a == 0.0)
    cout << "La ecuacion tiene la raiz unica x = "
       << -c/b << endl;
  else
  {
    disc = pow(b,2.0) - 4 * a * c;    // calcula el discriminante
    if (disc > 0.0)
    {
      disc = sqrt(disc);
      root1 = (-b + disc) / (2 * a);
      root2 = (-b - disc) / (2 * a);
      cout << "Las dos raices reales son "
   << root1 << " y " << raiz2 << endl;
    }
    else if (disc < 0.0)
      cout << "Ambas raices son imaginarias.\n";
    else
      cout << "Ambas raices son iguales a " << -b / (2 * a) << endl;
  }

  return 0;
}
```

La primera ejecución resuelve la ecuación cuadrática $x^2 + 2x - 35 = 0$, la cual tiene las raíces reales $x = 5$ y $x = -7$. Los datos de entrada para la segunda ejecución producen la ecuación $0x^2 + 0x + 16 = 0$. Debido a que esto degenera en la imposibilidad matemática de $16 = 0$, el programa identifica en forma correcta ésta como una ecuación degenerada. Dejamos como ejercicio crear datos de prueba para los otros casos limitantes verificados por el programa.

Ejercicios 4.5

1. **a.** Escriba un programa que acepte dos números reales de un usuario y un código seleccionado. Si el código seleccionado introducido es 1, haga que el programa sume los dos números introducidos con anterioridad y despliegue el resultado; si el código seleccionado es 2, los números deberán multiplicarse, y si el código seleccionado es 3, el primer número deberá ser dividido entre el segundo número.
   b. Determine qué hace el programa escrito en el ejercicio 1a cuando los números introducidos son 3 y 0, y el código seleccionado es 3.
   c. Modifique el programa escrito en el ejercicio 1a de modo que no se permita la división entre 0 y se despliegue un mensaje apropiado cuando se intente dicha división.

2. **a.** Escriba un programa para desplegar los dos indicadores siguientes:

```
Introduzca un mes (use 1 para Ene, etc.):
Introduzca un dia del mes:
```

   Haga que su programa acepte y almacene un número en la variable `mes` en respuesta al primer indicador, y acepte y almacene un número en la variable `dia` en respuesta al segundo indicador. Si el mes introducido no está entre 1 y 12 inclusive, imprima un mensaje informando al usuario que se ha introducido un mes inválido. Si el día introducido no está entre 1 y 31, imprima un mensaje informando al usuario que se ha introducido un día inválido.
   b. ¿Qué hará su programa si el usuario introduce un número con un punto decimal para el mes? ¿Cómo puede asegurar que sus instrucciones `if` comprueben que es un número entero?
   c. En un año que no es bisiesto, febrero tiene 28 días, los meses de enero, marzo, mayo, julio, agosto, octubre y diciembre tienen 31 días y todos los demás meses tienen 30 días. Usando esta información, modifique el programa escrito en el ejercicio 2a para desplegar un mensaje cuando se introduzca un día inválido para un mes introducido por un usuario. Para este programa ignore los años bisiestos.

3. **a.** El cuadrante en el que reside una línea trazada desde el origen es determinado por el ángulo que forma la línea con el eje `x` positivo como sigue:

Ángulo desde el eje *x* positivo	Cuadrante
Entre 0 y 90 grados	I
Entre 90 y 180 grados	II
Entre 180 y 270 grados	III
Entre 270 y 360 grados	IV

Usando esta información, escriba un programa en C++ que acepte el ángulo de la línea como una entrada del usuario y determine y despliegue el cuadrante apropiado a los datos introducidos. (NOTA: Si el ángulo tiene exactamente 0, 90, 180 o 270 grados, la línea correspondiente no reside en ningún cuadrante sino que se encuentra en un eje.)

b. Modifique el programa escrito para el ejercicio 3a de modo que se despliegue un mensaje que identifique un ángulo de cero grados como el eje *x* positivo, un ángulo de 90 grados como el eje y positivo, un ángulo de 180 grados como el eje *x* negativo y un ángulo de 270 grados como el eje *y* negativo.

4. Todos los años que se dividen exactamente entre 400 o que son divisibles exactamente entre cuatro y no son divisibles exactamente entre 100 son años bisiestos. Por ejemplo, en vista que 1600 es divisible exactamente entre 400, el año 1600 fue un año bisiesto. Del mismo modo, en vista que 1988 es divisible exactamente entre cuatro pero no entre 100, el año 1988 también fue un año bisiesto. Usando esta información, escriba un programa en C++ que acepte el año como una entrada del usuario, determine si el año es un año bisiesto y despliegue un mensaje apropiado que le indique al usuario si el año introducido es un año bisiesto o no.

5. Con base en el año del modelo y el peso de un automóvil el estado de Nueva Jersey determina la clase del vehículo y la tarifa de registro que le corresponde usando la siguiente tabla:

Año del modelo	Peso	Clase de peso	Tarifa de registro
1970 o anterior	Menos de 2700 lbs	1	$16.50
	2700 a 3800 lbs	2	25.50
	Más de 3800 lbs	3	46.50
1971 a 1979	Menos de 2700 lbs	4	27.00
	2700 a 3800 lbs	5	30.50
	Más de 3800 lbs	6	52.50
1980 o posterior	Menos de 3500 lbs	7	19.50
	3500 lbs o más	8	52.50

Usando esta información, escriba un programa en C++ que acepte el año y el peso de un automóvil y determine y despliegue la clase y la tarifa de registro para el automóvil.

6. Modifique el programa 4.9 de modo que se calculen y desplieguen las raíces imaginarias cuando el discriminante es negativo. Para este caso las dos raíces de la ecuación son:

$$x_1 = \frac{-b}{2a} + \frac{\text{sqrt}[-(b^2 - 4ac)]}{2a} i$$

y

$$x_2 = \frac{-b}{2a} - \frac{\text{sqrt}[-(b^2 - 4ac)]}{2a} i$$

donde *i* es el símbolo del número imaginario para la raíz cuadrada de –1. (*Sugerencia*: Calcule las partes real e imaginaria de cada raíz por separado.)

7. En el juego del 21, el valor de las cartas del 2 al 10 es el que tienen impreso, sin importar de qué palo sean. Las cartas de personajes (jota, reina y rey) se cuentan como 10, y el as se cuenta como 1 u 11, dependiendo de la suma de todas las cartas en una mano. El as se cuenta como 11 sólo si el valor total resultante de todas las cartas en una mano no excede de 21, de lo contrario se cuenta como 1. Usando esta información, escriba un programa en C++ que acepte los valores de tres cartas como entradas (un 1 correspondiente a un as, un 2 correspondiente a un dos, etc.), calcule el valor total de la mano en forma apropiada y despliegue el valor de las tres cartas con un mensaje impreso.

4.6 ERRORES COMUNES DE PROGRAMACIÓN

Tres errores de programación son comunes en las instrucciones de selección de C++.

1. Usar el operador de asignación, =, en lugar del operador relacional, ==. Esto puede causar una enorme cantidad de frustración debido a que cualquier expresión puede ser probada por una instrucción `if-else`. Por ejemplo, la instrucción

```
if (opselect = 2)
   cout << "Feliz Cumpleaños";
else
   cout << "Buen Dia";
```

   siempre produce que se imprima el mensaje `Feliz Cumpleaños`, sin importar el valor inicial en la variable `opselect`. La razón para esto es que la expresión de asignación `opselect = 2` tiene un valor de 2, el cual se considera un valor verdadero en C++. La expresión correcta para determinar el valor en `opselect` es `opselect == 2`.

2. Permitir que la instrucción `if-else` aparente seleccionar una opción incorrecta. En este problema de depuración típico, el programador se concentra en forma errónea en la condición probada como la fuente del problema. Por ejemplo, suponga que la siguiente instrucción `if-else` es parte de su programa:

```
if (clave == 'F')
{
  contemp = (5.0/9.0) * (intemp - 32.0);
  cout << "Se efectuo la conversion a grados Celsius";
}
else
{
  contemp = (9.0/5.0) * intemp + 32.0;
  cout << "Se efectuo la conversion a grados Fahrenheit";
}
```

Esta instrucción siempre desplegará Se efectuo la `conversion` a grados `Celsius` cuando la variable `clave` contenga una `F`. Por consiguiente, si este mensaje se despliega cuando usted cree que `clave` no contiene una `F`, requerirá investigar el valor de `clave`. Como regla general, siempre que una instrucción de selección no actúa como piensa que debería, pruebe sus suposiciones acerca de los valores asignados a las variables probadas desplegando sus valores. Si se despliega un valor no anticipado, al menos habrá aislado la fuente del problema en las variables mismas, en lugar de buscarla en la estructura de la instrucción `if-else`. A partir de ahí tendrá que determinar dónde y cómo se obtuvo el valor incorrecto.

3. Usar instrucciones `if` anidadas sin incluir llaves para indicar la estructura deseada. Sin llaves, el compilador empareja por omisión los `else` con los `if` sin par más cercanos, lo cual a veces destruye la intención original de la instrucción de selección. Para evitar este problema y crear código que se adapte con facilidad al cambio, es útil escribir todas las instrucciones `if-else` como instrucciones compuestas en la forma

```
if (expresion)
{
  una o mas instrucciones entran aqui
}
else
{
  una o mas instrucciones entran aquí
}
```

Al utilizar esta forma, la integridad y la intención originales de la instrucción `if` se mantienen sin importar cuántas instrucciones se agreguen después.

4.7 RESUMEN DEL CAPÍTULO

1. Las expresiones relacionales, las cuales también se llaman ***condiciones***, se usan para comparar operandos. Si una expresión relacional es verdadera, el valor de la expresión es el entero 1. Si la expresión relacional es falsa, tiene un valor entero de 0. Las expresiones relacionales se crean usando los siguientes operadores relacionales.

Operador relacional	Significado	Ejemplo
<	Menor que	`edad < 30`
>	Mayor que	`altura > 6.2`
<=	Menor que o igual a	`gravable <= 20000`
>=	Mayor que o igual a	`temp >= 98.6`
==	Igual a	`calificación == 100`
!=	No es igual a	`numero != 250`

2. Pueden construirse condiciones más complejas a partir de las expresiones relacionales usando los operadores lógicos de C++, `&&` (AND), `||` (OR) y `!` (NOT).

3. Se usa una instrucción `if-else` para seleccionar entre dos instrucciones alternativas con base en el valor de una expresión. Aunque las expresiones relacionales se usan por lo general para la expresión probada, puede usarse cualquier expresión válida. Al probar una expresión, las instrucciones `if-else` interpretan un valor diferente de cero como verdadero y un valor de cero como falso. La forma general de una instrucción `if-else` es:

```
if (expresion)
  instruccion1;
else
  instruccion2;
```

   Ésta es una instrucción de selección bidireccional. Si la expresión tiene un valor diferente de cero es considerada como verdadera y se ejecuta `instruccion1`; de lo contrario se ejecuta `instruccion2`.

4. Una instrucción `if-else` puede contener otras instrucciones `if-else`. En ausencia de llaves, cada `else` se asocia con el `if` sin par precedente más cercano.

5. La cadena `if-else` es una instrucción de selección de vía múltiple que tiene la forma general

```
if (expresion_1)
  instruccion_1;
else if (expresion_2)
  instruccion_2;
else if (expresion_3)
  instruccion_3;
      .
      .
      .
else if (expresion_m)
  instruccion_m;
else
  instruccion_n;
```

   Cada expresión es evaluada en el orden en que aparece en la cadena. Una vez que una expresión es verdadera (tiene un valor diferente de cero), sólo se ejecuta la instrucción entre esa expresión y el siguiente `else if` o `else`, y no se prueban más expresiones. El `else` final es opcional, y la instrucción correspondiente al `else` final sólo se ejecuta si ninguna de las expresiones anteriores es verdadera.

6. Una instrucción compuesta consiste en cualquier cantidad de instrucciones individuales encerradas dentro del par de llaves `{` y `}`. Las instrucciones compuestas son tratadas como una sola unidad y pueden usarse en cualquier parte en que se use una instrucción individual.

7. La instrucción `switch` es una instrucción de selección de vía múltiple. La forma general de una instrucción switch es

```
switch (expresion)
{    // inicio de la instruccion compuesta
  case valor_1:  <-------------- termina con dos puntos
    instruccion1;
    instruccion2;
        .
        .
    break;
  case valor_2:  <-------------- termina con dos puntos
    instruccionm;
    instruccionn;
        .
        .
    break;
   .
   .
  case valor_n:  <-------------- termina con dos puntos
    instruccionw;
    instruccionx;
        .
        .
    break;
  default:       <-------------- termina con dos puntos
    instruccionaa;
    instruccionbb;
        .
        .
}    // fin de switch y de la instruccion compuesta
```

Para esta instrucción el valor de una expresión es comprada con un número entero o con una constante o con expresiones constantes. La ejecución del programa se transfiere al primer caso de correspondencia y continúa hasta el final de la instrucción `switch` a menos que encuentre una instrucción `break` opcional. Los casos en una instrucción `switch` pueden aparecer en cualquier orden y puede incluirse un caso `default` opcional. El caso `default` se ejecuta si ninguno de los otros casos encuentra una correspondencia.

4.8 APÉNDICE DEL CAPÍTULO: UN ACERCAMIENTO MÁS A FONDO A LA PRUEBA EN PROGRAMACIÓN

En teoría, un conjunto extenso de ejecuciones de prueba revelaría todos los errores posibles del programa y aseguraría que éste funcionará en forma correcta para todas y cada una de las combinaciones de datos de entrada y datos calculados. En la práctica esto requiere verificar todas las combinaciones posibles de ejecución de instrucciones. Debido al tiempo y esfuerzo requeridos, ésta es una meta imposible excepto para programas extremadamente simples. Veamos por qué es así. Considere el programa 4.10.

Programa 4.10

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

int main()
{
  int num;

  cout << "Introduzca un número: ";
  cin  >> num;
  if (num == 5)
    cout << "¡Loteria!\n";
  else
    cout << "¡Lotoria!\n";

  return 0;
}
```

El programa 4.10 tiene dos rutas que pueden recorrerse conforme el programa progresa desde su llave de apertura hasta su llave de cierre. La primera ruta, la cual se ejecuta cuando el número introducido es 5, está en la secuencia

```
cout << "Introduzca un número";
cin  >> num;
cout << "¡Loteria!\n";
```

La segunda ruta, la cual se ejecuta siempre que se introduce cualquier número excepto 5, incluye la secuencia de instrucciones

```
cout << "Introduzca un número";
cin  >> num;
cout << "¡Lotoria!\n";
```

Probar cada ruta posible en el programa 4.10 requiere dos ejecuciones del programa, con una selección juiciosa de los datos de entrada de prueba para asegurar que se emplean ambas rutas de la instrucción `if`. La adición de una instrucción `if` más en el programa incrementa el número de rutas de ejecución posibles por un factor de dos y requiere cuatro ejecuciones (2^2) del programa para una prueba completa. Del mismo modo, dos instrucciones `if` adicionales incrementan el número de rutas por un factor de cuatro y requieren ocho ejecuciones (2^3) para una prueba completa y tres instrucciones `if` adicionales producirían un programa que requiere dieciséis (2^4) ejecuciones de prueba.

Ahora considere un programa de aplicación de tamaño modesto que consta sólo de diez módulos, en el que cada módulo contiene cinco instrucciones `if`. Suponiendo que los módulos siempre son invocados en la misma secuencia, hay 32 rutas posibles a través de cada módulo (2 elevado a la quinta potencia) y más de 1 000 000 000 000 000 (2 elevado a la quincuagésima potencia) de rutas posibles a través del programa completo (todos los módulos ejecutados en secuencia). El tiempo necesario para crear datos de prueba individuales para emplear cada ruta y el tiempo de ejecución real en la computadora requerido para verificar cada ruta hace que la prueba completa de un programa así sea imposible de lograr.

La incapacidad para probar por completo todas las combinaciones de secuencias de ejecución de instrucciones ha conducido al dicho de programación de que "no hay programa libre de errores". También ha conducido a percatarse de que cualquier prueba que se haga deberá pensarse bien para maximizar la posibilidad de localizar errores. Como mínimo, los datos de prueba deberán incluir valores apropiados para los valores de entrada, valores de entrada ilegales que el programa deberá rechazar y valores limitantes que son verificados por las instrucciones de selección dentro del programa.

Consideración de opciones de carrera

Ingeniería civil

El campo de la ingeniería civil se interesa sobre todo en las estructuras y sistemas a gran escala usados por una comunidad. Un ingeniero civil diseña, construye y opera puentes, presas, túneles, edificios, aeropuertos, carreteras y otras obras públicas a gran escala. Los ingenieros civiles también son responsables de los efectos que tienen estos sistemas a gran escala en la sociedad y en el ambiente. Por tanto, los ingenieros civiles están implicados en los recursos hidráulicos, el control de inundaciones, la eliminación de desperdicios y la planeación urbana general. El campo puede subdividirse en tres categorías.

1. Estructuras. Diseño, construcción y operación de edificios a gran escala como presas, edificaciones y carreteras. Las propiedades de los materiales, la geología, la mecánica de suelos, y la estática y dinámica son elementos importantes en su educación. Por ejemplo, qué tan alto puede construirse un edificio antes que lo venza su propio peso es una cuestión que incluye todos estos temas.
2. Planeación urbana. La planeación, diseño y construcción de sistemas de transporte (carreteras, ferrocarriles, desarrollo fluvial, aeropuertos) y el uso general de la tierra. La topografía y la cartografía son habilidades necesarias.
3. Sanidad. Tratamiento de desechos, suministro de agua y sistemas de drenaje. La mecánica de fluidos, hidrología, control de la contaminación, irrigación y economía son consideraciones importantes.