CAPÍTULO 4

Estructuras de selección: sentencias `if` y `switch`

Contenido

4.1. Estructuras de control
4.2. La sentencia `if`
4.3. Sentencia condición doble `if_else`
4.4. Sentencias `if_else` anidadas
4.5. Sentencia de `switch`: condiciones múltiples
4.6. Expresiones condicionales: el operador `?:`
4.7. Evaluación en cortocircuito de expresiones lógicas
4.8. Puesta a punto de programas
4.9. Errores frecuentes de programación
RESUMEN
EJERCICIOS
PROBLEMAS
EJERICIOS RESUELTOS
PROBLEMAS RESUELTOS

INTRODUCCIÓN

Los programas definidos hasta este punto se ejecutan de modo secuencial, es decir, una sentencia después de otra. La ejecución comienza con la primera sentencia de la función y prosigue hasta la última sentencia, cada una de las cuales se ejecuta una sola vez. Esta forma de programación es adecuada para resolver problemas sencillos. Sin embargo, para la resolución de problemas de tipo general se necesita la capacidad de controlar cuáles son las sentencias que se ejecutan y en qué momentos. Las *estructuras* o *construcciones de control* controlan la secuencia o flujo de ejecución de las sentencias. Las estructuras de control se dividen en tres grandes categorías en función del flujo de ejecución: *secuencia, selección* y *repetición*.

Este capítulo considera las *estructuras selectivas* o *condicionales* —sentencias `if` y `switch`— que controlan si una sentencia o lista de sentencias se ejecutan en función del cumplimiento o no de una condición. Para soportar estas construcciones, el estándar ANSI/ISO C++ soporta el tipo lógico `bool`.

CONCEPTOS CLAVE

- Estructura de control.
- Estructura de control selectiva.
- Sentencia `break`.
- Sentencia compuesta.
- Sentencia `enum`.
- Sentencia `if`.
- Sentencia `switch`.
- Tipo de dato `bool`.

4.1. ESTRUCTURAS DE CONTROL

Las **estructuras de control** controlan el flujo de ejecución de un programa o función. Las estructuras de control permiten combinar instrucciones o sentencias individuales en una simple unidad lógica con un punto de entrada y un punto de salida.

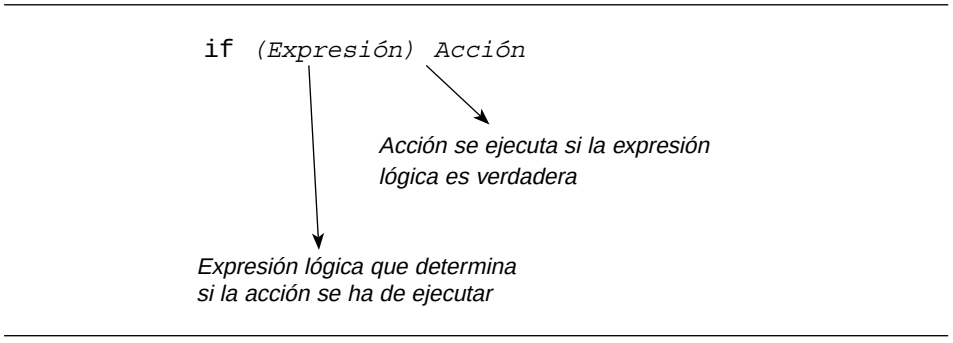
Las instrucciones o sentencias se organizan en tres tipos de estructuras de control que sirven para controlar el flujo de la ejecución: *secuencia*, *selección* (*decisión*) y *repetición*. Hasta este momento sólo se ha utilizado el flujo secuencial. Una **sentencia compuesta** es un conjunto de sentencias encerradas entre llaves ({ y }) que se utiliza para especificar un flujo secuencial.

```
{
  sentencia1;
  sentencia2;
  .
  .
  .
  sentencian;
}
```

El control fluye de la *sentencia*$_1$ a la *sentencia*$_2$, y así sucesivamente. Sin embargo, existen problemas que requieren etapas con dos o más opciones o alternativas a elegir en función del valor de una condición o expresión.

4.2. LA SENTENCIA `IF`

En C++, la estructura de control de selección principal es una sentencia `if`. La sentencia `if` tiene dos alternativas o formatos posibles. El formato más sencillo tiene la sintaxis siguiente:



La sentencia `if` funciona de la siguiente manera. Cuando se alcanza la sentencia `if` dentro de un programa, se evalúa la *expresión* entre paréntesis que viene a continuación de `if`. Si *Expresión* es verdadera, se ejecuta *Acción*; en caso contrario no se ejecuta *Acción* (en su formato más simple, *Acción* es una sentencia simple, y en los restantes formatos, es una sentencia compuesta). En cualquier caso la ejecución del programa continúa con la siguiente sentencia del programa. La Figura 4.1 muestra un *diagrama de flujo* que indica el flujo de ejecución del programa.

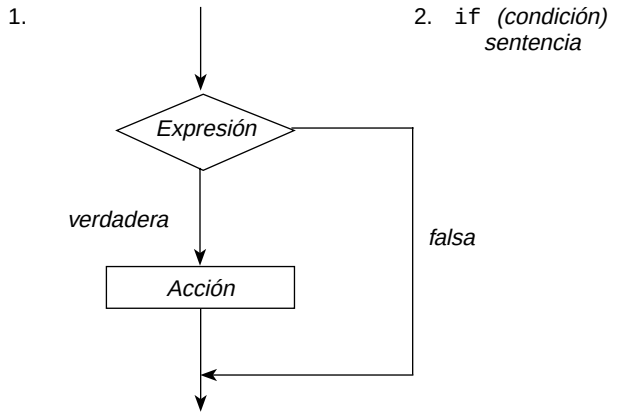


Figura 4.1. Diagrama de flujo de una sentencia básica `if`.

Otro sistema de representar la sentencia `if` es:

```
if (condición) sentencia;
```

condición *es una expresión entera*
sentencia *es cualquier sentencia ejecutable, que se ejecutará sólo si la condición toma un valor distinto de cero.*

Ejemplo 4.1

Prueba de divisibilidad.

```
void main()
{
    int n, d;
    cout << "Introduzca dos enteros:";
    cin >> n >> d;
    if (n%d == 0) cout << n << "es divisible por" << d << endl;
}
```

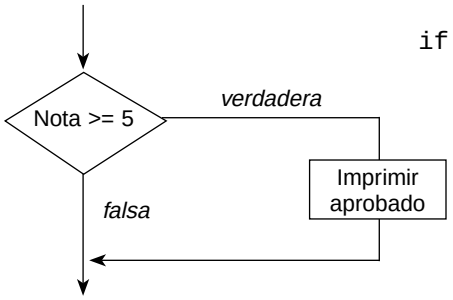
Ejecución

```
Introduzca dos enteros: 36 4
36 es divisible por 4
```

Este programa lee dos números enteros y comprueba cuál es el valor del resto de la división `n` entre `d` (`n % d`). Si el resto es cero, `n` es divisible por `d`. (En nuestro caso 36 es divisible por 4, ya que 36 : 4 = 9 y el resto es 0.)

Ejemplo 4.2

Representar la superación de un examen (Nota >= 5, Aprobado)




```
if (Nota >= 5)
    cout << "Aprobado" << endl;
```

Ejemplo 4.3

```
// programa demo1 if
#include <iostream>     // E/S de C++
using namespace std;

void main()
{
   float numero;
   // obtener número introducido por usuario
   cout << "Introduzca un número positivo o negativo:";
   cin >> número;

   // comparar número con cero
   if (número > 0)
      cout << número << " es mayor que cero" << endl;
}
```

La ejecución de este programa produce

```
Introduzca un número positivo o negativo: 10.15
10.15 es mayor que cero
```

Si en lugar de introducir un número positivo se introduce un número negativo, ¿qué sucede? Nada. El programa es tan simple que sólo puede comprobar si el número es mayor que cero.

```
// programa demo2 if
#include <iostream>
using namespace std;

void main()
{
   float numero;

   // obtener numero introducido por usuario
   cout "introduzca un número positivo o negativo:";
   cin >> número;
   // comparar numero a cero
   if (numero > 0)
      cout << numero << "es mayor que cero" << endl;
   if (numero < 0)
      cout << numero << "es menor que cero" << endl;
```

```
   if (numero == 0)
      cout << numero << "es igual a cero" << endl;
}
```

Este programa simplemente añade otra sentencia `if` que comprueba si el número introducido es menor que cero. Realmente, una tercera sentencia `if` se añade también que, comprueba si el número es igual a cero.

Ejercicio 4.1

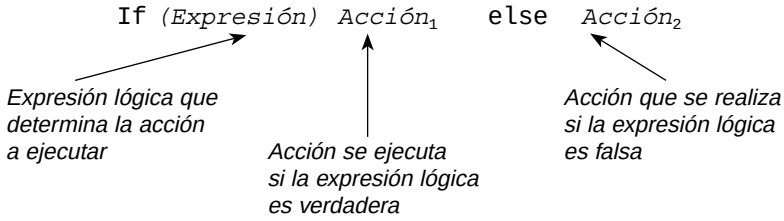
Visualizar el valor absoluto de un número leído del teclado.

```
// Programa de cálculo del valor absoluto de la entrada
#include <iostream>
#include <string>
using namespace std;

int main() {
   cout << "Introduzca un número:";
   int Valor;
   cin >> Valor;
   if (Valor < 0)
      Valor = -Valor;
   cout << Valor << "es positivo" << endl;
   return 0;
}
```

4.3. SENTENCIA: CONDICIÓN DOBLE `IF-ELSE`

Un segundo formato de la sentencia `if` es la sentencia `if-else`. Este formato de la sentencia `if` tiene la siguiente sintaxis:



En este formato *Acción*$_1$ y *Acción*$_2$ son individualmente o bien una única sentencia que termina en un punto y coma (;) o un grupo de sentencias encerrado entre llaves. Cuando se ejecuta la sentencia `if-else`, se evalúa *Expresión*. Si *Expresión* es verdadera, se ejecuta *Acción*$_1$ y en caso contrario se ejecuta *Acción*$_2$. La Figura 4.2 muestra la semántica de la sentencia `if-else`.

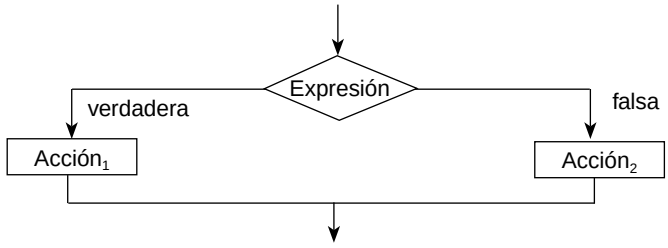


Figura 4.2. Diagrama de flujo de la representación de una sentencia `if-else`.

Ejemplos

1.
```
if (salario >= 100.000)
   salario_neto = salario – impuestos;
else
   salario_neto = salario;
```

Si `salario` es mayor que `100.000`, se calcula el salario neto, restándole los impuestos; en caso contrario (`else`), el salario neto es igual al salario (bruto).

2.
```
if (Nota >= 5)
    cout << "Aprobado" << endl;
else
  cout << "Suspenso" << endl;
```

Formatos

1.
```
if (expresión_lógica)
   sentencia
```

2.
```
if (expresión_lógica)
   sentencia1
else
   sentencia2
```

3.
```
if (expresión_lógica) sentencia
```

4.
```
if (expresión_lógica) sentencia1 else sentencia2
```

Si *expresión lógica* es verdadera, se ejecuta *sentencia* o bien $sentencia_1$, si es falsa (*si no*, en caso contrario), se ejecuta $sentencia_2$.

Ejemplos

```
1. if (x > 0.0)
      producto = producto * x;
2. if (x != 0.0)
      producto = producto * x;
   // se ejecuta la sentencia de asignación cuando x no es igual a 0.
   // en este caso producto se multiplica por x y el nuevo valor se
   // guarda en producto reemplazando el valor antiguo.
   // si x es igual a 0, la multiplicación no se ejecuta.
```

Ejemplo 4.4

Prueba de visibilidad (igual que 4.1, al que se ha añadido la cláusula `else`)

```
void main()
{
   int n, d;
   cout << "Introduzca dos enteros:";
```

```
   cin >> n >> d;
   if (n%d == 0) cout << n << "es divisible por" << d << endl;
   else
      cout << n << "no es divisible por" << d << endl;
}
```

Ejecución

```
Introduzca dos enteros 36 5
36 no es divisible por 5
```

Comentario

`36` no es divisible por `5`, ya que `36` dividido entre `5` produce un resto de 1 (`n % d = = 0`, es falsa, y se ejecuta la cláusula `else`).

Ejemplo 4.5

Calcular el mayor de dos números leídos del teclado y visualizarlo en pantalla.

```
void main()
{
   int x, y;
   cout << "Introduzca dos enteros:";
   cin >> x >> y;
   if (x > y) cout << x << endl;
   else
      cout << y << endl;
}
```

Ejecución

```
Introduzca dos enteros: 17 54
54
```

Comentario

La condición es (`x > y`). Si `x` es mayor que `y`, la condición es "verdadera" (*true*) y se evalúa a 1; en caso contrario la condición es "falsa" (*false*) y se evalúa a `0`. De este modo se imprime `x` cuando es mayor que `y`, como en el ejemplo de la ejecución.

4.4. SENTENCIAS `IF-ELSE` ANIDADAS

Hasta este momento, las sentencias `if` implementan decisiones que implican una o dos alternativas. En esta sección se mostrará cómo se puede utilizar la sentencia `if` para implementar decisiones que impliquen diferentes alternativas.

Una sentencia `if` es anidada cuando la sentencia de la rama verdadera o la rama falsa es a su vez una sentencia `if` . Una sentencia `if` anidada se puede utilizar para implementar decisiones con varias alternativas o multi-alternativas.

Sintaxis

```
if (condición1)
   sentencia1
else if (condición2)
   sentencia2
      .
      .
      .
else if (condiciónn)
   sentencian
else
   sentencia_e
```

Ejemplo 4.6

```
// incrementar contadores de números positivos, números
// negativos o ceros

if (x > 0)
   num_pos = num_pos + 1;
else
   if (x < 0)
      num_neg = num_neg + 1;
   else
      num_ceros = num_ceros + 1;
```

La sentencia `if` anidada tiene tres alternativas. Se incrementa una de las tres variables (`num_pos`, `num_neg` y `num_ceros`) en 1, dependiendo de que `x` sea mayor que cero, menor que cero o igual a cero, respectivamente. Las cajas muestran la estructura lógica de la sentencia `if` anidada; la segunda sentencia `if` es la acción o tarea falsa (a continuación de `else`) de la primera sentencia `if`.

La ejecución de la sentencia `if` anidada se realiza como sigue: se comprueba la primera condición (`x > 0`); si es verdadera, `num_pos` se incrementa en 1 y se salta el resto de la sentencia `if`. Si la primera condición es falsa, se comprueba la segunda condición (`x < 0`); si es verdadera `num_neg` se incrementa en uno; en caso contrario se incrementa `num_ceros` en uno. Es importante considerar que la segunda condición se comprueba *sólo si* la primera condición es falsa.

4.4.1. Sangría en las sentencias `if` anidadas

El formato multibifurcación se compone de una serie de sentencias `if` anidadas, que se pueden escribir en cada línea una sentencia `if`. La sintasxis multibifurcación anidada es:

Formato 1

```
if (expresión_lógica1)
   sentencia1
```

```
else
   if (expresión_lógica2)
   else
      if (expresión_lógica3)
            sentencia3
         else
            if (expresión_lógica4)
              sentencia4
            else
               sentencia5
```

Formato 2

```
if (expresión_lógica1)
   sentencia1
else if (expresión_lógica2)
   sentencia2
else if (expresión_lógica3)
   sentencia3
else if (expresión_lógica4)
   sentencia4
else
   sentencia5
```

Ejemplos

```
1. if (x > 0)
      if (y > 0)
         z = sqrt(x) + sqrt(y);

2. if (x > 0)
      if (y > 0)
         z = sqrt(x) + sqrt(y);
   else
      cerr << "\n *** Imposible calcular z" << endl;
```

Ejemplo 4.7

```
// comparación_if
// ilustra las sentencias compuestas if-else
#include <iostream>
using namespace std;

void main()
   cout << " introduzca un número positivo o negativo: ";
   cin >> número;

   // comparar número a cero
   if (numero > 0)
   {
      cout << numero << " es mayor que cero" << endl;
```

```
      cout << "pruebe de nuevo introduzca un número negativo" << endl;
   }
   else if (numero < 0)
      {
      cout << numero << " es menor que cero" << endl;
      cout << "pruebe de nuevo introduciendo un número negativo"
           << endl;
      }
      else
      {
         cout << numero << " es igual a cero" << endl;
         cout << "  ¿por qué no introduce un número negativo? << endl;
      }
   }
```

4.4.2. Comparación de sentencias `if` anidadas y secuencias de sentencias `if`

Los programadores tienen dos alternativas: (1) usar una secuencia de sentencias `if`; (2) una única sentencia `if` anidada. Por ejemplo, la sentencia `if` del Ejemplo 4.6 se puede reescribir como la siguiente secuencia de sentencias `if`.

```
if (x > 0)
   num_pos = num_pos + 1;
if (x < 0)
   num_neg = num_neg + 1;
if ( x == 0)
   num_ceros = num_ceros + 1;
```

Aunque la secuencia anterior es lógicamente equivalente a la original, no es tan legible ni eficiente. Al contrario que la sentencia `if` anidada, la secuencia no muestra claramente cuál es la sentencia a ejecutar para un valor determinado de `x`. Con respecto a la eficiencia, la sentencia `if` anidada se ejecuta más rápidamente cuando `x` es positivo ya que la primera condición (`x > 0`) es verdadera, lo que significa que la parte de la sentencia `if` a continuación del primer `else` se salta. En contraste, se comprueban siempre las tres condiciones en la secuencia de sentencias `if`. Si `x` es negativa, se comprueban dos condiciones en las sentencias `if` anidadas frente a las tres condiciones de las secuencias de sentencias `if`.

Una estructura típica `if-else` anidada permitida es:

```
if (número > o)
{
   // ...
}
else
   {
   if (// ...)
      {
         // ...
      }
   else
      {
```

```
      if (// ...)
         {
            // ...
         }
      }
   // ...
   }
```

Ejercicio 4.2

Existen diferentes formas de escribir sentencias `if` *anidadas.*

1.
```
if (a > 0) if (b > 0) ++a; else if (c > 0)
if (a < 5) ++b; else if (b < 5) ++c; else --a;
else if (c < 5) --b; else --c; else a = 0
```

2.
```
if (a > 0)                          // forma más legible
   if (b > 0) ++a;
   else
      if (c > 0)
         if (a < 5) ++b;
         else
            if (b < 5) ++c;
            else --a;
      else
         if (c < 5) --b;
         else --c;
else
   a = 0;
```

3.
```
if (a > 0)                          // forma más legible
   if (b > 0) ++a;
   else if (c > 0)
      if (a < 5) ++b;
      else if (b < 5) ++c;
      else --a;
   else if (c < 5) --b;
   else --c;
else
   a = 0
```

Ejercicio 4.3

Calcular el mayor de tres números enteros.

```
void main()
{
   int a, b, c, mayor;
   cout << "Introduzca tres enteros:";
   cin >> a >> b >> c;
```

```
   if (a > b)
      if (a > c) mayor = a;
      else mayor = c;
   else
      if (b > c) mayor = b;
      else mayor = c;
   cout << "El mayor es " << mayor << endl;
}
```

Ejecución

```
Introduzca tres enteros: 77 54 85
El mayor es 85
```

Análisis

Al ejecutar el primer `if`, la condición (`a > b`) es verdadera, entonces se ejecuta la segunda `if`. En el segundo `if` la condición (`a > c`) es falsa, en consecuencia el primer `else` y `mayor = 85` y se termina la sentencia `if` y se ejecuta la última línea y se visualiza `El mayor es 85`.

4.5. SENTENCIA `switch`: CONDICIONES MÚLTIPLES

La sentencia `switch` es una sentencia C++ que se utiliza para seleccionar una de entre múltiples alternativas. La sentencia `switch` es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada *expresión de control* o *selector*. El valor de esta expresión puede ser de tipo `int` o `char`, pero no de tipo `double` .

Sintaxis

```
switch (selector)
{
   case etiqueta1 : sentencias1;
                    break;
   case etiqueta2 : sentencias2;
                    break;
      .
      .
      .
   case etiquetan : sentencias n;
                    break;
   default:         sentenciasd;        // opcional
}
```

La expresión de control o *selector* se evalúa y se compara con cada una de las etiquetas de `case`. La expresión *selector* debe ser un tipo ordinal (por ejemplo, `int, char, bool` pero no `float o string`). Cada *etiqueta* es un valor único, constante, y cada etiqueta debe tener un valor diferente de los otros. Si el valor de la expresión selector es igual a una de las etiquetas `case` —por ejemplo, $etiqueta_i$— entonces la ejecución comenzará con la primera sentencia de la secuencia $secuencia_i$ y continuará hasta que se encuentra una sentencia `break` (o hasta que se encuentra el final de la sentencia de control `switch`).

El tipo de cada etiqueta debe ser el mismo que la expresión de *selector*. Las expresiones están permitidas como etiquetas pero sólo si cada operando de la expresión es por sí misma una constante —por ejemplo, `4 + 8` o bien `m * 15`, siempre que `m` hubiera sido definido anteriormente como constante con nombre.

Si el valor del selector no está listado en ninguna etiqueta `case`, no se ejecutará ninguna de las opciones a menos que se especifique una acción por defecto (omisión). La omisión de una etiqueta `default` puede crear un error lógico difícil de prever. Aunque la etiqueta `default` es opcional, se recomienda su uso a menos que se esté absolutamente seguro de que todos los valores de *selector* estén incluidos en las etiquetas `case`.

Una sentencia `break` consta de la palabra reservada `break` seguida por un punto y coma. Cuando la computadora ejecuta las sentencias siguientes a una etiqueta `case`, continúa hasta que se alcanza una sentencia `break`. Si la computadora encuentra una sentencia `break`, termina la sentencia `switch`. Si se omiten las sentencias `break`, después de ejecutar el código de `case`, la computadora ejecutará el código que sigue a la siguiente `case`.

Ejemplo 1

```
switch (opción)
{
   case 0:
      cout << "Cero!" << endl;
      break;
   case 1:
      cout << "Uno!" << endl;
      break;
   case 2:
      cout << "Dos!" << endl;
      break;
   default:
      cout << "Fuera de rango" << endl;
}
```

Ejemplo 2

```
switch (opción)
{
case 0:
case 1:
case 2:
      cout << "Menor de 3";
      break;
case 3:
      cout << "Igual a 3";
      break;
default:
      cout << "Mayor que 3";
}
```

Ejemplo 4.8

Comparación de las sentencias `if-else-if` *y* `switch`.

Se necesita saber si un determinado carácter `car` es una vocal.

Solución con *if-else-if*

```
if ((car == 'a') || (car == 'A'))
   cout << car << "es una vocal" << endl;
else if ((car == 'e') || (car == 'E'))
   cout << car << "es una vocal" << endl;
else if ((car == 'i') || (car == 'I'))
   cout << car << "es una vocal" << endl;
else if ((car == 'o') || (car == 'O'))
   cout << car << "es una vocal" << endl;
else if ((car == 'u') || (car == 'U'))
   cout << car << "es una vocal" << endl;
else
   cout << car << "no es una vocal" << endl;
```

Solución con *switch*

```
switch (car) {
   case 'a': case 'A':
   case 'e': case 'E':
   case 'i': case 'I':
   case 'o': case 'O':
   case 'u': case 'U':
      cout << car << "es una vocal" << endl;
      break;
   default
      cout << car << "no es una vocal" << endl;
   }
```

Ejemplo 4.9

```
// Programa de ilustración de la sentencia switch
#include <iostream>
using namespace std;

int main()
{
   char nota;
   cout << "Introduzca calificación (A-H) y pulse Intro:";
   cin >> nota;

   switch (nota)
   {
      case 'A':  cout << "Excelente."
                      << "Examen superado\n";
                 break;
      case 'B':  cout << "Notable.";
                 cout << "Suficiencia\n";
                 break;
      case 'C':  cout << "Aprobado\n";
                 break;
```

```
      case 'D':
      case 'F':  cout << "Suspendido\n";
                 break;
      default:
         cout "no es posible esta nota";
   }
   cout << "Final de programa" << endl;
   return 0;
}
```

Cuando se ejecuta la sentencia `switch`, se evalúa `nota`; si el valor de la expresión es igual al valor de una etiqueta, entonces se transfiere el flujo de control a las sentencias asociadas con la etiqueta correspondiente. Si ninguna etiqueta coincide con el valor de `nota` se ejecuta la sentencia `default` y las sentencias que vienen detrás de ella. Normalmente, la última sentencia de las sentencias que vienen después de una `case` es una sentencia `break`. Esta sentencia hace que el flujo de control del programa salte a la última sentencia de `switch`. Si no existiera `break`, se ejecutarían también las sentencias restantes de la sentencia `switch`.

Ejecución de prueba 1

```
Introduzca calificación (A-H) y pulse Intro: A
Excelente. Examen superado
Final de programa
```

Ejecución de prueba 2

```
Introduzca calificación (A-H) y pulse Intro: B
Notable. Suficiencia
Final de programa
```

Ejecución de prueba 3

```
Introduzca calificación (A-H) y pulse Intro: E
No es posible esta nota
Final de programa
```

Precaución

Si se olvida `break` en una sentencia `switch`, el compilador no emitirá un mensaje de error, ya que se habrá escrito una sentencia `switch` correcta sintácticamente, pero no realizará las tareas previstas.

Ejemplo 4.10

```
int tipo_vehículo;
cout << "Introduzca tipo de vehículo:";
cin >> tipo_vehículo, peaje;
```

```
switch(tipo_vehículo)
{
   case 1:
      cout << "turismo";
      peaje = 500;
      break;  ──────►  Si se omite esta break, el vehículo primero será turismo y luego autobús.
   case 2:
      cout << "autobús";
      peaje = 3000;
      break;
   case 3:
      cout << "motocicleta";
      peaje = 300;
      break;
   default:
      cout << "vehículo no autorizado";
}
```

Cuando la computadora comienza a ejecutar una sentencia `case`, no detiene su ejecución hasta que se encuentra o bien una sentencia `break` o bien una sentencia `switch`.

4.5.1. Caso particular `case`

Está permitido tener varias expresiones `case` en una alternativa dada dentro de la sentencia `switch`. Por ejemplo, se puede escribir:

```
switch(c) {
   case '0': case '1': case '2': case '3': case '4':
   case '5': case '6': case '7': case '8': case '9':
      num_digitos++;  // se incrementa en 1 el valor de num_digitos
      break;
   case '': case '\t': case '\n':
      num_blancos++;  // se incrementa en 1 el valor de num_blancos
      break;
   default:
      num_distintos++;
}
```

4.5.2. Uso de sentencias `switch` en menús

La sentencia `if-else` es más versátil que la sentencia `switch` y se puede utilizar unas sentencias `if-else` anidadas o multidecisión, en cualquier parte que se utiliza una sentencia `case`. Sin embargo, normalmente, la sentencia `switch` es más clara. Por ejemplo, la sentencia `switch` es idónea para implementar menús.

Un *menú* de un restaurante presenta una lista de alternativas para que un cliente elija entre sus diferentes opciones. Un menú en un programa de computadora hace la misma función: presentar una lista de alternativas en la pantalla para que el usuario elija una de ellas.

4.6. EXPRESIONES CONDICIONALES: EL OPERADOR ?:

Las sentencias de selección (`if` y `switch`) consideradas hasta ahora son similares a las sentencias previstas en otros lenguajes, tales como C y Pascal. Sin embargo, C++ ha heredado un tercer mecanismo de selección de su lenguaje raíz C, una expresión que produce uno de dos valores, resultado de una expresión lógica o booleana (también denominada condición). Este mecanismo se denomina *expresión condicional*. Una expresión condicional tiene el formato `C? A : B` y es realmente una operación ternaria (tres operandos) en el que `C,A` y `B` son los tres operandos y `?` es el operador.

Sintaxis

condición ? expresión$_1$ *: expresión*$_2$

condición	es una expresión lógica
expresión$_1$/*expresión*$_2$	son expresiones compatibles de tipos

Se evalúa *condición*, si el valor de *condición* es verdadera (distinto de cero) entonces se devuelve como resultado el valor de *expresión*$_1$; si el valor de *condición* es falsa (cero), se devuelve como resultado el valor de *expresión*$_2$.

Uno de los medios más sencillos del operador condicional (`?:`) es utilizar el operador condicional y llamar a una de dos funciones.

Ejemplos

1.1 `a == b ? función1() : función2();`
es equivalente a la siguiente sentencia:

```
if (a == b)
  función1();
else
  función2();
```

1.2 El operador `?:` se utiliza en el siguiente segmento de código para asignar el menor de dos valores de entrada asignados a `Menor`.

```
int Entrada1;
int Entrada2;
cin >> Entrada1 >> Entrada2;
int Menor = Entrada1 <= Entrada2 ? Entrada1 : Entrada2
```

Ejemplo 4.11

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

void main()
{
   float n1, n2;

   cout << "Introduzca dos números positivos o negativos:";
   cin >> n1 >> n2;
```

```
//if-else
   cout << endl << "if-else";
   if (n1 > n2)
      cout << n1 << " > " << n2;
   else
      cout << n1 << " > " << n2;

   // operador condicional
   cout << endl << condicional:";
   n1 > n2 ? cout << n1 << " > " << n2
           : cout << n1 << " < " << n2;
}
```

4.7. EVALUACIÓN EN CORTOCIRCUITO DE EXPRESIONES LÓGICAS

Cuando se evalúan expresiones lógicas en C++ se puede emplear una técnica denominada *evaluación en cortocircuito*. Este tipo de evaluación significa que se puede detener la evaluación de una expresión lógica tan pronto como su valor pueda ser determinado con absoluta certeza. Por ejemplo, si el valor de `(soltero == 's')` es falso, la expresión lógica `(soltero == 's') && (sexo = 'h') && (edad > 18) && (edad <= 45)` será falsa con independencia de cuál sea el valor de las otras condiciones. La razón es que una expresión lógica del tipo

```
falso && (...)
```

debe ser siempre falsa, cuando uno de los operandos de la operación `AND` es falso. En consecuencia, no hay necesidad de continuar la evaluación de las otras condiciones cuando `(soltero == 's')` se evalúa a falso.

El compilador C++ utiliza este tipo de evaluación. Es decir, la evaluación de una expresión lógica de la forma `a1 && a2` se detiene si la subexpresión `a1` de la izquierda se evalúa a falsa.

C++ realiza evaluación en cortocircuito con los operadores `&&` y `||`, de modo que evalúa primero la expresión más a la izquierda de las dos expresiones unidas por `&&` o bien por `||`. Si de esta evaluación se deduce la información suficiente para determinar el valor final de la expresión (independiente del valor de la segunda expresión), el compilador de C++ no evalúa la segunda expresión.

Ejemplo

Si `x` es negativo, la expresión

```
(x >= 0) && (y > 1)
```

se evalúa en cortocircuito ya que `x >= 0` será falso y, por tanto, el valor final de la expresión será falso.

En el caso del operador `||` se produce una situación similar. Si la primera de las dos expresiones unidas por el operador `||` es *verdadera*, entonces la expresión completa es *verdadera*, con independencia de que el valor de la segunda expresión sea *verdadero* o *falso*. La razón es que el operador `|| OR` produce resultado verdadero si el primer operando es verdadero.

Otros lenguajes, distintos de C++, utilizan evaluación completa. En evaluación completa, cuando dos expresiones se unen por un símbolo `&&` o `||`, se evalúan siempre ambas expresiones y, a continuación, se utilizan las tablas de verdad de `&&` o bien `||` para obtener el valor de la expresión final.

Ejemplo

Si `x` es cero, la condición

```
if ((x != 0.0) && (y/x > 7.5))
```

es falsa ya que (`x != 0.0`) es falsa. Por consiguiente, no hay necesidad de evaluar la expresión (`y / x > 7.0`) cuando `x` sea cero. Sin embargo, si altera el orden de las expresiones, al evaluar el compilador la sentencia `if`

```
if ((y / x > 7.5) && (x != 0.0))
```

se produciría un error en tiempo de ejecución de división por cero («division by zero»).

> El orden de las experiencias con operadores `&&` y `||` puede ser crítico en determinadas situaciones.

4.8. PUESTA A PUNTO DE PROGRAMAS

Estilo y diseño

1. El estilo de escritura de una sentencia `if` e `if-else` es el sangrado de las diferentes líneas en el formato siguiente:

```
if (expresión_lógica)
   sentencia1
else
   sentencia2
```

```
if (expresión_lógica)
      {
        sentencia1
           .
        .
        .
      sentencia_k
   }
else
   {
      sentencia_k+1
         .
         .
         .
      sentencia_n
   }
```

   En el caso de sentencias `if-else-if` utilizadas para implementar una estructura de selección multialternativa se suele escribir de la siguiente forma:

```
if (expresión_lógica1)
    sentencia1
else if (expresión_lógica2)
    sentencia2
  .
  .
  .
```

```
else if (expresión_lógica_n)
    sentencia_n
else
    sentencia_n+1
```

2. Una construcción de selección múltiple se puede implementar más eficientemente con una estructura `if-else-if` que con una secuencia de sentencias independientes `if`. Por ejemplo,

```
cout <<"Introduzca nota";
cin >> nota
if (nota < 0 || nota > 100)
{
  cout << nota << " no es una nota válida.\n";
  return'?';
}
if (nota >= 90( && (nota <= 100)
  return 'A';
if (nota >= 80) && (nota < 90)
  return 'B';
if (nota >=70) && (nota < 80)
  return 'C';
if (nota >= 60) && (nota < 70)
  return 'D';
if (nota < 60)
  return 'F';
```

Con independencia del valor de `nota` se ejecutan todas las sentencias `if`; 5 de las expresiones lógicas son expresiones compuestas, de modo que se ejecutan 16 operaciones con independencia de la nota introducida. En contraste, las sentencias `if` anidadas reducen considerablemente el número de operaciones a realizar (3 a 7), todas las expresiones son simples y no se evalúan todas ellas siempre.

```
cout << "Introduzca nota";
cin >> nota
if (nota < 0 || nota > 100)
{
   cout << nota << " no es una nota válida.\n";
   return '?';
}
else if (nota >= 90)
   return 'A';
else if (nota >= 80)
   return 'B';
else if (nota >= 70)
   return 'C';
else if (nota >= 60)
   return 'D';
else
   return 'F';
```

4.9. ERRORES FRECUENTES DE PROGRAMACIÓN

1. *Uno de los errores más comunes en una sentencia `if` es utilizar un operador de asignación (=) en lugar de un operador de igualdad (==).*

2. *En una sentencia* `if` *anidada, cada claúsula* `else` *se corresponde con la* `if` *precedente más cercana. Por ejemplo, en el segmento de programa siguiente.*

```
if (a > 0)
if (10 > 0)
c = a + b;
else
c = a + abs(b);
d = a * b * c;
```

¿Cuál es la sentencia `if` asociada a `else`?

El sistema más fácil para evitar errores es el sangrado o indentación, con lo que ya se aprecia que la cláusula `else` se corresponde a la sentencia que contiene condición `b > 0`.

```
if (a > 0)
   if (b > 0)
      c = a + b;
   else
      c = a + abs(b);
d = a * b * c;
```

3. *Las comparaciones con operadores* `==` *de cantidades algebraicamente iguales pueden producir una expresión lógica falsa, debido a que la mayoría de los números reales no se almacenan exactamente*. Por ejemplo, aunque las expresiones reales siguientes son equivalentes:

```
a * (1 / a)
1.0
```

son algebraicamente iguales, la expresión

```
a * (1 / a) == 1.0
```

puede ser falsa debido a que `a` es real.

4. *Cuando en una sentencia* `switch` *o en un bloque de sentencias falsas una de las llaves (*`{,}`*) aparece un mensaje de error tal como:*

```
Error ...: Cumpound statement missing } in function
```

Si no se tiene cuidado con la presentación de la escritura del código, puede ser muy difícil localizar la llave que falta.

5. *El selector de una sentencia* `switch` *debe ser de tipo entero o compatible entero*. Así, las constantes reales

```
2.4, -4.5, 3.1416
```

no pueden ser utilizadas en el selector.

6. *Cuando se utiliza una sentencia* `switch`*, asegúrese que el selector de* `switch` *y las etiquetas* `case` *son del mismo tipo (*`int, char` *o* `bool` *pero no* `float`*). Si el selector se evalúa a un valor no listado en ninguna de las etiquetas* `case`*, la sentencia* `switch` *no gestionará ninguna acción; por esta causa se suele poner una etiqueta* `default` *para resolver este problema.*

RESUMEN

Sentencia `if`

Una alternativa

```
if (a != 0)
   resultado = a/b;
```

Múltiples alternativas

```
if (x < 0)
{
   cout << "Negativo" << endl;
   abs_x = -x;
}
else if (x == 0)
{
   cout << "Cero" << endl;
   abs_x = 0;
}
else
{
   cout << "Positivo" << endl;
   abs_x = x;
}
```

Dos alternativas

```
if (a >= 0)
   cout << a << " es positivo" << endl;
else
   cout << a << " es negativo" << endl;
```

Sentencia `switch`

```
switch (sig_car)
{
   case 'A': case'a':
      cout << "Sobresaliente" << endl;
      break;
   case 'B': case 'b':
      cout << "Notable" << endl;
      break;
   case 'C': case 'c':
      cout << "Aprobado" << endl;
      break;
   case 'D': case 'd':
      cout << "Suspenso" << endl;
      break;
   default
      cout << "nota no válida" << endl;
   }  // fin de switch
```

EJERCICIOS

4.1. ¿Qué valor se asigna a consumo en la sentencia `if` siguiente si velocidad es 120?

```
if (velocidad > 80)
     consumo = 10.00;
else if (velocidad > 100)
     consumo = 12.00;
else if (velocidad > 120)
     consumo = 15.00;
```

4.2. Explique las diferencias entre las sentencias de la columna de la izquierda y de la columna de la derecha. Para cada una de ellas deducir el valor final de x si el valor inicial de x es 0.

```
if (x >= 0)          if (x >= 0)
     x = x+1;             x = x+1;
else if (x >= 1);    if (x >= 1)
     x = x+2;             x = x+2;
```

4.3. ¿Qué salida producirá el siguiente código cuando se inserta en un programa completo?

```
int x = 2;
cout << "Arranque\n";
if (x <= 3)
if (x != 0)
    cout << "Hola desde el segundo if.\n";
else
    cout << "Hola desde el else.\n";
cout << "Fin\n";
cout << "Arranque de nuevo\n";
if (x > 3)
       if (x != 0)
          cout << "Hola desde el segundo
            if.\n";
       else
          cout << "Hola desde el else.\n";
    cout << "De nuevo fin\n";
```