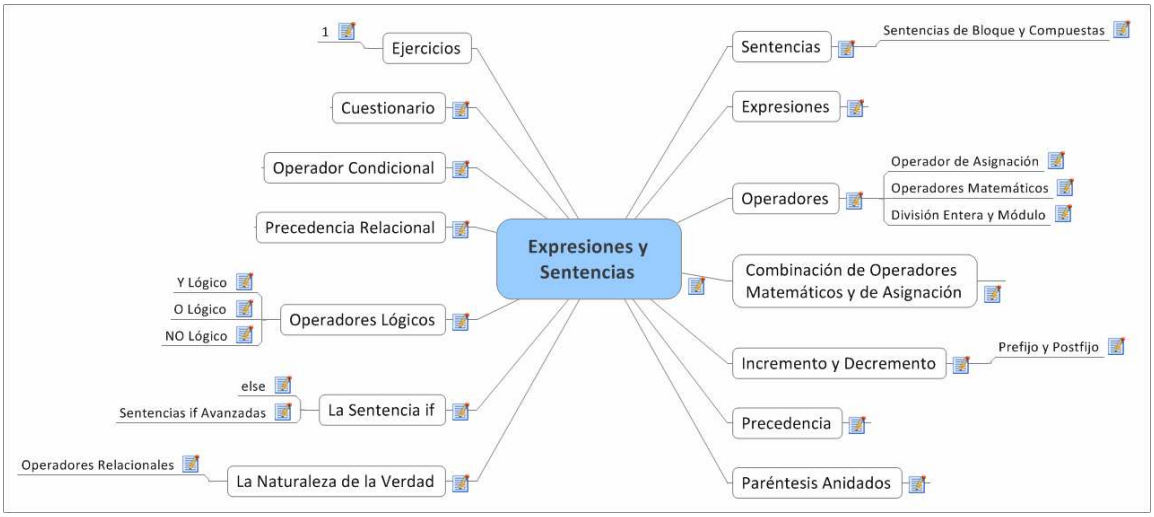
EXPRESIONES Y SENTENCIAS



Un programa no es más que una conjunto de comandos ejecutados en secuencia. El poder de un programa proviene del hecho de ejecutar una comando tras otro, basado en si una particular condición es verdadera o falsa.

Sentencias

En C++, una sentencia controla la secuencia de ejecución o evalúa una expresión. Todas las sentencias en C/C++ finalizan con un punto y coma (;). Una sentencia muy común podría ser:

```
x= a + b;
```

A diferencia del álgebra, la sentencia no significa que x sea igual a a+b, sino que a x se le asignará el resultado de sumar a y b. El operador de asignación (=), asignará el valor resultante de la expresión de su lado derecho en la variable ubicada en su lado izquierdo.

En un sentencia, los espacios en blanco son generalmente ignorados. En tal sentido,

```
x=a+b;
```

y

```
x=  a  +  b;
```

son sentencias equivalente.

Sentencias de Bloque y Compuestas

Las sentencias pueden ser simples o compuestas. Una sentencia compuesta, también llamada de bloque, inicia con una llave abierta ({) y termina con una llave cerrada (}). Cada sentencia dentro del bloque debe finalizar con punto y coma; sin embargo, el bloque como tal no la necesita. Por ejemplo

```
{
    temporal= a;
    a= b;
    b= temporal;
}
```

Este bloque de código actúa como una única sentencia, en este caso intercambiando el contenido de las variables a y b.

PRÁCTICAS SANAS DE PROGRAMACIÓN. Use una llave cerrada cada vez que haya usado una abierta. Finalice las sentencias con un punto y coma. Use espacios en blanco para clarificar el código.

Expresiones

En C/C++ una expresión es cualquier sentencia que evalúe a un valor. Se dice que una expresión devuelve un valor. por ejemplo, 3+2 devuelve el valor 5 y por lo tanto es una expresión. Todas las expresiones son sentencias. Existe una infinidad de piezas de código que califican como expresiones. Por ejemplo:

```
3.2                 // Devuelve el valor 3,2
PI                  // Constante flotante que devuelve el valor 3,14
segundosPorMinuto   / Constante entera que devuelve 60
```

Asumiendo que PI es una constante definida como 3,14 y que segundosPorMinuto es otras constante definida como 60, todas las anteriores sentencias son expresiones. La expresión

```
x= a + b;
```

no sólo suma a y b asignándolo a x, sino que adicionalmente devuelve el valor de dicha asignación. Esta sentencia es también una expresión. Como es una expresión, puede ir en el lado derecho de un operador de asignación:

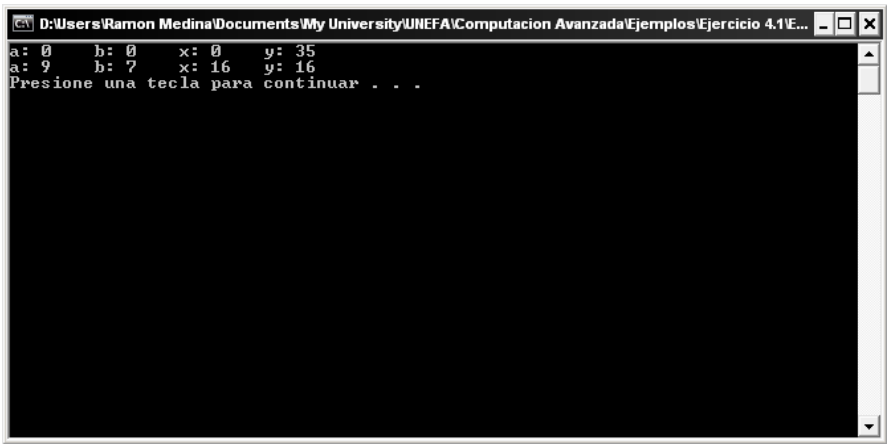
```
y= x= a + b;
```

Esta línea es evaluada de la siguiente manera. Los valores a y b son sumados y su resultado asignado a x. Luego, el resultado de lo asignado a x es asignado a y. Si a, b, x y son todos enteros y tienen los valores, por ejemplo, a igual 2 y b igual a 5, luego de ejecutarse la sentencia, x y y tendrán asignado el valor 7.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:  int main ()
4:  {
5:      int a=0, b=0, x=0, y=35;
6:      cout << "a: " << a << "\tb: " << b;
```

```
7:      cout << "\tx: " << x << "\ty: " << y << endl;
8:
9:      a= 9;
10:     b= 7;
11:     y= x = a+b;
12:     cout << "a: " << a << "\tb: " << b;
13:     cout << "\tx: " << x << "\ty: " << y << endl;
14:
15:     system ("PAUSE");
16:     return 0;
17: }
```

En la línea 5 se declaran e inicializan cuatro variables. Sus valores son mostrados en las líneas 6 y 7.En las líneas 9 y 10 se le asignan valores a las variables a y b. En la línea 11 se calcula la suma de a y b, se asigna el resultado en x y y.



Operadores

Un operador es un símbolo que hace que el compilador tome una acción. Los operadores actúan sobre operandos, y en C y C++, todos los operandos son expresiones. En C/C++ existen varias categorías de operadores. Dos de esas categorías son:

Operadores de asignación

Operadores matemáticos

Operador de Asignación

El operador de asignación (=) hace que el operando en el lado izquierdo del operador, ajuste su valor al resultado del operador de la derecha. La expresión

```
x= a + b;
```

asigna el valor resultante de sumar a y b al operando x.

Un operando que pueda legalmente estar en el lado izquierdo del operador se llama *lvalue*. El que puede estar en el lado derecho se llama *rvalue*. Las constantes son *rvalue* y no pueden ser *lvalue*. Es válido escribir

```
x= 32;                  // bien
```

pero no es válido escribir

```
35 = x;                 // error, no es un lvalue
```

NUEVO TÉRMINO. Un lvalue es un operando que puede estar del lado izquierdo de una expresión. Un *rvalue* es un operando que puede estar en el lado derecho de una expresión. Nótese que todos los lvalue son rvalue, pero no todos los rvalue pueden ser lvalue. Un ejemplo de un rvalue que no puede ser lvalue es una constante literal.

Operadores Matemáticos

Existen cinco operadores matemáticos: adición (+), substracción(-), multiplicación(*), división (/) y residuo (%). Los operadores funcionan según lo esperado a partir de los conocimientos de álgebra. Sin embargo, las operaciones aritméticas con enteros sin signo pueden arrojar resultados inesperados si la expresión evalúa a un número negativo.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:
4:  int main()
5:  {
6:      unsigned int diferencia;
7:      unsigned int numeroGrande = 100;
8:      unsigned int numeroChico = 50;
9:      diferencia = numeroGrande - numeroChico;
10:     cout << "La diferencia es: " << diferencia;
11:     diferencia = numeroChico - numeroGrande;
12:     cout << "\nLa diferencia es: " << diferencia <<endl;
13:
14:     system("PAUSE");
15:     return 0;
```

```
16: }
```

El operador de resta es invocado en la línea 9 y el resultado mostrado en la línea 10, tal y como es lo esperado (100 - 50 = 50). El operador de resta es llamado nuevamente en la línea 11 y el resultado mostrado en la línea 12. El valor resultante debió ser negativo, pero como está siendo interpretado como entero sin signo, se produce un desbordamiento de la variable y como consecuencia se muestra un resultado absurdo.

División Entera y Módulo

La división entera difiere de la división ordinaria. Cuando se divide 21 por 4, el resultado es un número real (parte entera más parte decimal). Los enteros no tienen parte decimal, de tal manera que el resultado es presentado como un cociente entero más un residuo. Por esta razón, el resultado de la división entera de 21 por 4 es 5. El residuo en este caso (21 % 4) es 1. El operador de residuo calcula este valor.

Combinación de Operadores Matemáticos y de Asignación

Resulta bastante común querer añadir un valor al contenido de una variable y asignar el resultado a la misma variable. Si se tiene una variable llamada miEdad y se desea incrementar su valor por 2, se puede escribir:

```
int miEdad= 5;
int temporal;
temporal= miEdad + 2;          //suma 5 + 2 y lo almacena en temporal
miEdad= temporal;              // copia la información de temporal en miEdad
```

Este método es sin embargo, mi complejo e ineficiente. En C/C++ es posible colocar la misma variable a ambos lados del operador de asignación:

```
miEdad= miEdad + 2;
```

que es complemente equivalente al ejemplo anterior y mucho más eficiente. En álgebra esta expresión carece de sentido, pero en C/C++ se lee como "añade 2 al contenido de miEdad y asigna el resultado a miEdad.

Aún más simple de escribir, pero tal vez menos fácil de entender es:

```
miEdad+= 2;
```

El operador de suma con auto asignación (+=)añade el rvalue al lvalue y asigna el resultado al lvalue. Si la variable miEdad contiene 4 antes de ejecutarse la expresión, luego de ella contendrá el número 6.

Existen también operadores de auto asignación para la resta (-=), la división (/=), la multiplicación (*=) y el residuo (%=).

Incremento y Decremento

Un operador muy común de la suma (o resta) con auto asignación es el valor 1. En C/C++ agregar un 1 a una variable se llama incrementar y restar un 1 a una variable decrementar. Existen operadores especiales para llevar a cabo esas acciones.

El operador de incremento (++) aumenta en 1 el contenido de una variable, mientras que el operador de decremento (--) lo disminuye en 1. Por lo tanto, si tiene una variable llamada C y desea incrementarla en 1, puede usar la expresión

```
C++;
```

Esta sentencia es equivalente a escribir

```
C= C + 1;
```

que a su vez es equivalente a

```
C+= 1;
```

Prefijo y Postfijo

Los operadores de incremento (++) y decremento (--) vienen en dos versiones: prefijo y postfijo). La versión prefijo es escrita antes del nombre de la variable (++miEdad) y la postfijo después (miEdad++).

En un sentencia sencilla puede que no importe cuál tipo de operador se usa, pero en expresiones complejas donde se incrementa (o decrementa) el contenido de una variable y luego el resultado es asignado a otra, la diferencia es importante. El operador prefijo es evaluado antes de la asignación; el postfijo es evaluado después.

Si se tiene una variable x cuyo contenido es 5 y se ejecuta la siguiente sentencia

```
int a= ++x;
```

el resultado almacenado en a será 6, puesto que el operador prefijo se ejecuta primero que el operador de asignación. Si por el contrario la expresión ejecutada es la siguiente

```
int b= x++;
```

aunque el contenido de x es 6 luego de ejecutarse la sentencia, el valor almacenado en b será 5, ya que el operador de asignación tiene precedencia sobre el operador de incremento postfijo.

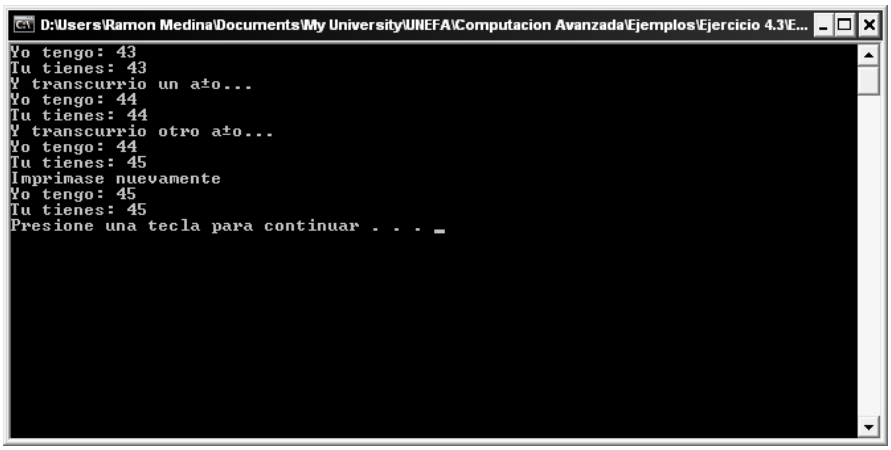
```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:  int main()
4:  {
5:      int miEdad = 43;        // inicializa dos registros
6:      int tuEdad = 43;
7:      cout << "Yo tengo: " << miEdad << endl;
8:      cout << "Tu tienes: " << tuEdad << endl;
9:      miEdad++;               // incremento postfijo
10:     ++tuEdad;               // incremento prefijo
11:     cout << "Y transcurrio un año...\n";
12:     cout << "Yo tengo: " << miEdad << endl;
13:     cout << "Tu tienes: " << tuEdad << endl;
14:     cout << "Y transcurrio otro año...\n";
15:     cout << "Yo tengo: " << miEdad++ << endl;
16:     cout << "Tu tienes: " << ++tuEdad << endl;
17:     cout << "Imprimase nuevamente\n";
18:     cout << "Yo tengo: " << miEdad << endl;
19:     cout << "Tu tienes: " << tuEdad << endl;
20:     system ("PAUSE");
21:     return 0;
```

```
22: }
```

En las líneas 5 y 6 se declaran dos variables enteras, inicializadas con el número 43. Su valores son mostrados en las líneas 7 y 8. Su valores son nuevamente mostrados en las líneas 12 y 13. En la línea 10 miEdad es incrementada usando el operador postfijo y en la 11 lo es tuEdad usando el operador prefijo. Los resultados son mostrados en las líneas 12 y 13 y son idénticos.

En las líneas 15 y 16 las variables miEdad y tuEdad son incrementadas como parte de la sentencia de impresión, utilizando operadores postfijo y prefijo respectivamente. En el primer caso, el contenido de miEdad es mostrado antes de ser incrementado, mientras que en el segundo, el contenido de tuEdad es incrementado antes de ser mostrado. Por esta razón, en esta ocasión los valores impresos son diferentes.

En las líneas 18 y 19 son mostrados nuevamente los contenidos de miEdad y tu Edad, resultando valores idénticos.



Precedencia

En una expresión como la siguiente:

```
x= 5 + 3 * 8;
```

¿qué se ejecuta primero, la suma o la multiplicación?. Si la suma se ejecutara primero, el resultado sería 64. Si fuera la multiplicación, el resultado sería 29. Cada operador tiene un valor de precedencia. La precedencia del operador de multiplicación es mayor que el del operador de suma, razón por la cual la multiplicación se ejecuta primero y el resultado de la expresión anterior es 29.

En el caso de que los operadores tengan la misma precedencia, la expresión es evaluada de izquierda a derecha.. Así

```
x= 5 + 3 + 8 * 9 + 6 * 4;
```

es evaluado multiplicando 8 * 9 (72), luego 6 * 4 (24), luego sumando 5 + 3 (8), después sumando 8 + 72 (90) para finalmente sumar 90 + 24 y obtener el resultado final de 104.

¿Qué sucede si la precedencia de los operadores no se ajusta a las necesidades? Considere por ejemplo la siguiente expresión:

```
segundosTotales= minutosParaPensar + minutosParaEscribir * 60;
```

En esta expresión no se desea multiplicar minutosParaEscribir por 60 y luego sumarle minutosParaPensar. Lo que se quiere hacer es sumar minutosParaPensar más minutosParaEscribir y multiplicar el resultado por 60 para obtener el número total de segundos. En este caso, se usan paréntesis para cambiar el orden de precedencia. Los elementos dentro de los paréntesis se evalúan con una precedencia más alta que la de cualquier otro operador matemático. De esta manera, la expresión:

```
segundosTotales= (minutosParaPensar + minutosParaEscribir) * 60;
```

cumple con el requerimiento establecido.

Paréntesis Anidados

En expresiones complejas, puede que sea necesario anidar paréntesis dentro de paréntesis. Por ejemplo, puede ser necesario calcular el número total de segundos y luego el número total de personas involucradas, antes de multiplicar estos dos valores y obtener el gran total:

```
totalSegundosPersonas= (((minutosParaPensar + minutosParaEscribir) * 60) *
(personasEnOficina + personasDeVacaciones));
```

Esta expresión es leída de adentro hacia afuera. En primer lugar, se suman minutosParaPensar y minutosParaEscribir por que están en el paréntesis más interno y este resultado es multiplicado por 60. Luego se suman personasEnOficina más personasDeVacaciones) Finalmente, el número total de personas es multiplicado por el número total de segundos.

Este ejemplo trae a colación un aspecto importante. Esta expresión es fácil de entender por el computador, pero puede resultar muy compleja para que un ser humano la lea, entienda o modifique. A continuación se presenta la misma expresión, empleando algunas variables temporales:

```
minutosTotales= minutosParaPensar + minutosParaEscribir;
segundosTotales= minutosTotales * 60;
totalDePersonas= personasEnOficina + personasDeVacaciones;
totalSegundosPersonas= totalDePersonas * segundosTotales;
```

Este ejemplo toma más tiempo para ser escrito y emplea más variable temporales, pero es más fácil de comprender. Si se agrega un comentario al principio del código explicando lo que hace y se cambia el número 60 por una constante simbólica, se obtiene un código que es fácil de entender y mantener.

PRÁCTICAS SANAS DE PROGRAMACIÓN. Recuerde que las expresiones devuelven un valor. Use el operador prefijo (++variable) para incrementar o decrementar una variable antes de que sea usada en la expresión. Use el operador postfijo (variable++) para incrementar o decrementar una variable luego de ser usada. Use paréntesis para cambiar el orden de precedencia de los operadores. No anide paréntesis muy profundamente, porque la expresión se hace difícil de comprender y mantener.

La Naturaleza de la Verdad

En C y C++, el cero es considerado falso y cualquier otro valor, verdad. De esta forma, si una expresión es falsa, es igual a cero; si una expresión es igual a cero, es falsa. Si una expresión es verdadera, sólo se puede inferir que es diferente de cero; si una expresión es diferente de cero, entonces es verdadera.

Operadores Relacionales

Los operadores relacionados son usados para determinar cuándo dos números son iguales, o una es mayor o menor que el otro. Los operadores relacionales evalúan a 1 (verdad) o 0 (falso).

Si la variable miEdad tiene como valor 39 y la variable tuEdad tiene como valor 40, es posible determinar si son iguales utilizando el operador relacional correspondiente:

```
miEdad == tuEdad;          // ¿es el valor de miEdad igual al de tuEdad?
```

La expresión anterior evalúa a cero (falso) porque los valores de las variables no son iguales

ADVERTENCIA. Muchos programadores principiantes confunden el operador de asignación (=) que el operador relacional de igualdad (==), lo que puede ocasionar problemas difíciles de descubrir.

Existen seis operadores relacionales que son presentados en la tabla a continuación:

Nombre	Operador
igual	==
diferente	!=
mayor que	>
mayor o igual	>=
menor que	<
menor o igual	<=

PRÁCTICAS SANAS DE PROGRAMACIÓN. Recuerde que los operadores relacionales devuelven una valor 1 (verdad) o 0 (falso). No confunda el operador de asignación (=) con el operador relacional de igualdad (==). Este es uno de los errores de programación más comunes en C y C++.

La Sentencia if

Normalmente un programa fluye de manera secuencial ejecutando las instrucciones en el orden en que aparecen. La sentencia if permite verificar una condición y bifurcar hacia una porción diferente del programa, según el resultado.

La forma más simple de una sentencia if es:

```
if (expresión)
    sentencia;
```

La expresión en el paréntesis puede ser cualquiera; sin embargo, lo más común es que contenga operadores relacionales. Si la expresión evalúa a 0, se considera falsa y la sentencia no es ejecutada. Si la expresión tiene un valor diferente de cero, es considerada verdad, y se ejecuta la sentencia. Considere el siguiente ejemplo:

```
if (numeroGrande > numeroChico)
    numeroGrande= numeroChico;
```

Este código compara los contenidos de las variables numeroGrande y numeroChico. Si el valor de numeroGrande es el mayor, la sentencia asigna el contenido de numeroChico a numeroGrande.

En el caso anterior, sólo una sentencia está bajo la acción de if. Sin embargo, mediante el uso de bloques de sentencias, es posible hacer que más de una lo esté.

```
if (expresión)
{
    sentencia1;
    sentencia2;
    sentencia3;
}
```

He aquí un ejemplo de este último caso:

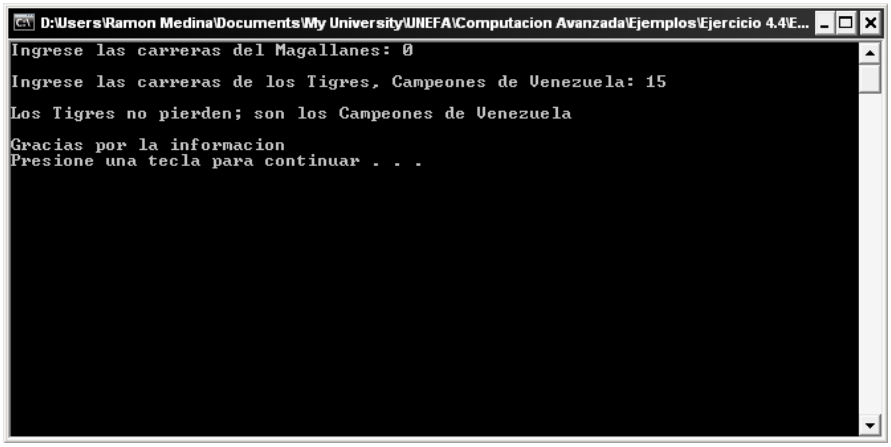
```
if (numeroGrande > numeroChico)
{
    numeroGrande= numeroChico;
    cout << "Número Grande: " << numeroGrande << endl;
    cout << "Número Chico: " << numeroChico << endl;
}
```

En este caso, si numeroGrande es mayor que numeroChico, no sólo ocurre la asignación sino que se muestra un mensaje informativo.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:  int main()
4:  {
5:      int carrerasTigres, carrerasMagallanes;
6:      cout << "Ingrese las carreras del Magallanes: ";
7:      cin >> carrerasMagallanes;
8:
9:      cout << "\nIngrese las carreras de los Tigres, Campeones de Venezuela: ";
10:     cin >> carrerasTigres;
11:
12:     cout << endl;
13:
14:     if (carrerasTigres > carrerasMagallanes)
15:         cout << "Los Tigres no pierden; son los Campeones de Venezuela" << endl;
16:
17:     if (carrerasTigres < carrerasMagallanes)
```

```
18:    {
19:        cout << "Magallanes ganando, " ;
20:        cout << "Debe haber un error\n";
21:    }
22:
22:    if (carrerasTigres == carrerasMagallanes)
23:    {
24:        cout << "Un empate; no puede ser\n";
25:        cout << "Dime las carreras verdaderas de los Tigres: ";
26:        cin >> carrerasTigres;
27:
28:        if (carrerasTigres > carrerasMagallanes)
29:            cout << "Lo sabia..., los Tigres no pierden; son los Campeones de Venezuela"
                << endl;
30:
31:        if (carrerasTigres < carrerasMagallanes)
32:            cout << "Hasta los mejores, algunas vez pierden" << endl;
33:
34:        if (carrerasTigres == carrerasMagallanes)
35:        {
36:            cout << "En serio un empate" << endl;
37:            cout << "Que sucedio?. Suspendieron el juego por lluvia?" << endl;
38:        }
39:    }
40:
41:    cout << "\nGracias por la informacion\n";
42:    system("PAUSE");
43:    return 0;
44: }
```

Este programa la pide al usuario que ingrese el resultado de un partido de béisbol, valores que son almacenados en dos variables enteras. Los valores son comparados en sentencias if en las líneas 14, 17 y 22, mostrándose mensajes relativos al resultado de la comparación. Si las carreras de ambos equipos son iguales, se solicita nuevamente el resultado de uno de los equipos (línea 26) y vuelven a ser comparados en las líneas 28, 31 y 34, para mostrar mensajes alusivos al resultado. Nótese que las sentencias desde la línea 24 hasta la línea 37 (incluyendo las sentencias if contenidas en ese bloque) se ejecutarán sólo si la condición verificada en la línea 22 es verdad.



else

Es frecuente que un programa se desee ejecutar una porción de código si una condición es verdad, y otra diferente si es falsa. La manera más fácil de hacerlo es usando una sentencia else:

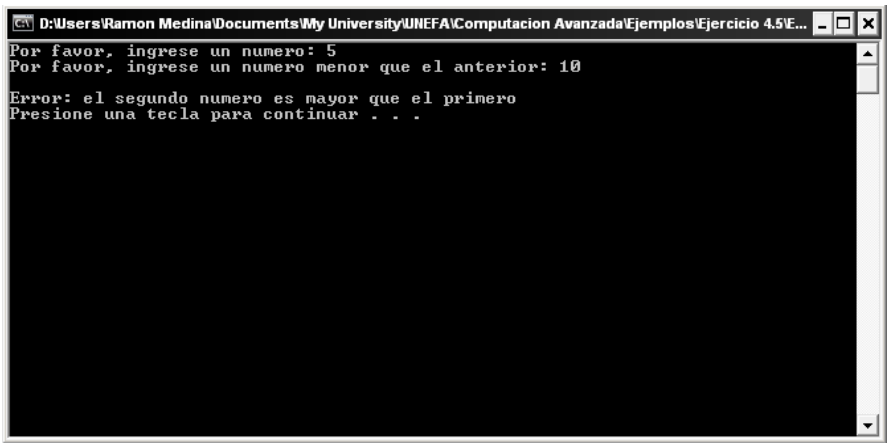
```
if (expresión)
    sentencia1;
else
    sentencia2;
```

En este caso, la sentencia1 se ejecutará si la expresión evaluada arroja un valor verdadero. Si la expresión evalúa a falso, se ejecuta entonces las sentencia2.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:  int main()
4:  {
5:      int primerNumero, segundoNumero;
6:
7:      cout << "Por favor, ingrese un numero: ";
8:      cin >> primerNumero;
9:
10:     cout << "Por favor, ingrese un numero menor que el anterior: ";
11:     cin >> segundoNumero;
12:
13:     if (primerNumero > segundoNumero)
14:         cout << "\nGracias" << endl;
15:     else
16:         cout << "\nError: el segundo numero es mayor que el primero" << endl;
17:
18:     system("PAUSE");
19:     return 0;
```

```
20: }
```

La sentencia if en la línea 23 evalúa si el primer número es mayor que el segundo. Si la condición es verdad, se ejecuta la sentencia de la línea 24. Si no es verdad, se ejecuta la sentencia de la línea 26.



Sentencias if Avanzadas

Las sentencias if pueden ser combinadas de muchas maneras, siendo posible escribir estructuras como la que se muestra a continuación:

```
if (expresión1)
{
    if (expresión2)
        sentencia1;
    else
    {
        if (expresión3)
            sentencia2;
        else
            sentencia3;
    }
}
else
    sentencia4;
```

Esta estructura dice que si la expresión1 es verdadera y la expresión2 es verdadera, se ejecuta la sentencia1. Si la expresión1 es verdad pero la 2 es falsa, entonces si la expresión3 es verdadera, se ejecuta la sentencia2. Si la expresión1 es verdad, pero las expresiones 2 y 3 son falsas, entonces se ejecuta la sentencia3. Si la expresión1 no es verdadera, se ejecuta la sentencia4. ¡Sencillo, no!

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:  int main()
4:  {
5:      // El programa pide dos numeros
6:      // Asigna los numeros ingresados a dos variables, primerNumero y
        // segundoNumero
7:      // Si el primerNumero es mayor o igual que el segundoNumero
8:      // evalua si el mayor es multiplo del menor
9:      // si lo es, verifica si son iguales
10:
11:     int primerNumero, segundoNumero;
12:     cout << "Ingrese dos numeros\nPrimero: ";
13:     cin >> primerNumero;
14:     cout << "\nSegundo: ";
15:     cin >> segundoNumero;
16:     cout << endl << endl;
17:
18:     if (primerNumero >= segundoNumero)
19:     {
20:         if ((primerNumero % segundoNumero) == 0)    // division exacta
21:         {
22:             if (primerNumero == segundoNumero)
23:                 cout << "El primero y segundo numero son iguales" << endl;
24:             else
25:                 cout << "El primer numero es multiplo del segundo" << endl;
26:         }
27:         else
28:             cout << "El primer numero no es multiplo del segundo" << endl;
29:     }
30:     else
31:         cout << "El segundo numero es mayor que el primero" << endl;
32:
33:     system("PAUSE");
34:     return 0;
35: }
```

El programa solicita se ingresen dos números que son comparados. La primera sentencia if en la línea 19 evalúa si el primero es mayor o igual que el segundo. Si no lo es, se ejecuta la sentencia en la línea 31. Si el primer if es verdad, se ejecuta el bloque de código a partir de la línea 20. En la línea 20 se verifica si el primer número es múltiplo del segundo; si lo es, en la línea 22 se comparan para saber si son iguales. Si el if de la línea 20 es falso, se ejecuta la sentencia de la línea 28.

Operadores Lógicos

Los operadores lógicos son conectores que permiten escribir expresiones que combinen más de un operador relacional: "¿es verdad que x es mayor que y, y que y es mayor que z?. Por ejemplo, la lógica de un sofisticado sistema de alarma podría requerir evaluar "si la alarma de la puerta suena Y son más de las 6pm Y no es día feriado NI es fin de semana, entonces llama a la policía". En C y C++ se usan tres tipos de operadores lógicos, para realizar evaluaciones como estas.

Operador	Símbolo
Y	&&
O	||
NO	!

Y Lógico

Un sentencia lógica Y evalúa dos expresiones, y si ambas son verdaderas, la expresión completa es verdad. Por ejemplo:

```
if ((x == 5 ) && (y == 5))
```

evalúa a verdad si es cierto que x es igual a 5 y que y es igual 5. Ambos lados del operador deben ser verdad para que la expresión completa sea verdad.

O Lógico

Una sentencia lógica O evalúa dos expresiones, y basta con que una de ellas sea verdad, para que toda la expresión lo sea. Por ejemplo:

```
if ((x == 5) || (y == 5))
```

evalúa a verdad si x es igual, y es igual a 5 o ambas los son.

NO Lógico

Una sentencia lógica NO evalúa a verdad, si la expresión verificada es falsa. Por ejemplo:

```
if ( !(x == 5))
```

será verdad si x es diferente de 5. Esto es equivalente a escribir:

```
if (x != 5)
```

Precedencia Relacional

Los operadores relacionales y lógicos devuelven valores 1 (verdad) o 0 (falso). Como todas las expresiones, ellos tienen un orden de precedencia que determina que relaciones son evaluadas primero. Esto es importante para determinar el resultado de la sentencia.

```
if (x > 5 && y > 5 || z > 5)
```

En la expresión anterior, puede que el programador deseara que la sentencia fuese verdad si ambos x y y son mayores que 5 o cuando z sea mayor que 5. Por otra parte, podría ser que el programador deseara que la expresión fuese verdad sólo si x es mayor que 5 y también que y o z lo fuesen.

Si x es 3, y 10 y z 10, de acuerdo a la primera interpretación, la expresión sería verdadera (z es mayor que 5, por lo tanto no importa que pasa con x y y). Sin embargo, al considerar la segunda interpretación, la expresión evaluaría a falso, ya que x no es mayor que 5.

Aunque el orden de precedencia determina que relación es evaluada primero, los paréntesis tienen la capacidad para alterarlo.

```
if ((x > 5) && ((y > 5) || (z > 5)))
```

La expresión anterior usa paréntesis para modificar el orden de precedencia y ajustar la sentencia a la segunda interpretación.

NOTA. Es buena idea utilizar paréntesis para clarificar el orden en que deben ser evaluados los operadores relacionales y lógicos. La idea es escribir programas fáciles de leer y mantener.

Operador Condicional

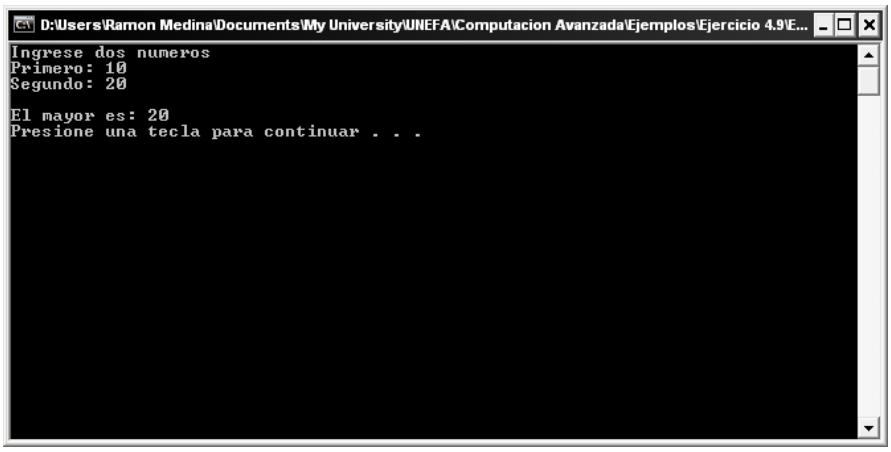
El operador condicional es el único operador ternario en C y C++; esto significa que toma tres operadores. El operador condicional toma tres expresiones y devuelve un valor:

```
(expresión1) ? (expresión2) : (expresión3)
```

La sentencia anterior se lee "si la expresión1 es verdad, devuelve el valor de la expresión2; de lo contrario, devuelve el valor de la expresión3". Este valor es típicamente asignado a una variable.

```
1:  #include <iostream.h>
2:  #include <stdlib.h>
3:  int main()
4:  {
5:      int x, y, z;
6:      cout << "Ingrese dos numeros" << endl;
7:      cout << "Primero: ";
8:      cin >> x;
9:      cout << "Segundo: ";
10:     cin >> y;
11:     cout << endl;
12:
13:     z= (x > y) ? x : y;
14:
15:     cout << "El mayor es: " << z << endl;
16:
17:     system("PAUSE");
18:     return 0;
19: }
```

En la línea 5 se declaran tres variables enteras. Las dos primeras se usan para almacenar los valores ingresados por el usuario. La tercera es empleada para almacenar el resultado de la expresión en la línea 13, que usa el operador condicional. El contenido z es luego mostrado en la línea 15.



Cuestionario

1. ¿Qué es una expresión?
2. ¿Para qué se usan los paréntesis en una expresión?
3. ¿Los número negativos son verdaderos o falsos?
4. ¿Cuál es el resultado de 201 / 4?
5. ¿Cuál es el resultado de 201 % 4?
6. Si las variables miEdad, a y b son enteros, que valor tienen luego de ejecutarse las siguientes instrucciones:

```
miEdad= 43;
a= miEdad++;
b= ++miEdad;
```

7. ¿Cuál es el resultado de 8+2*3?
8. ¿Cuál es la diferencia entre x=3 y x==3?
9. ¿Indique a qué (verdad o falso) evalúan las siguientes expresiones?

```
0
1
-1
```

```
x=0
```

```
x==0    // suponga que x es igual a cero
```

Ejercicios

1

Analice el siguiente programa e intente inferir qué hace

```
#include <iostream.h>
#include <stdlib.h>
int main()
{
    int x;
    cout << "Ingrese un numero menor que 10 o mayor que 100: ";
    cin >> x;
    cout << endl;

    if (x > 10)
        if (x > 100)
            cout << "Es mayor que 100, Gracias" << endl;
    else
        cout << "Es menor que 10, Gracias" << endl;

    system("PAUSE");
    return 0;
}
```

2

Transcriba y compile el programa del ejercicio 1. Cuando lo ejecute y el programa le solicite ingresar un número menor que 10 o mayor que 100, escriba el número 20. Verifique si el resultado del programa es el que usted predijo. Si no es así, explique por qué y haga las correcciones para que el programa funciones de acuerdo a lo esperado.