El proceso de acumulación y operadores de asignación

En este documento iniciamos recordando lo que es la asignación de un valor a una variable de cualquier tipo, con el operador =, luego revisamos el concepto de acumulador y lo relacionaremos con los demás operadores de asignación que ya fueron presentados en un documento anterior.

El operador de asignación =

Recordemos que este operador se utiliza para asignar un valor a una variable de cualquier tipo. Es un operador binario, ya que requiere de dos operandos, uno a su izquierda y otro a su derecha. La sintaxis de uso de este operador es la siguiente:

Variable **=** Expresión

En donde:

- Variable, que es el operando izquierdo, es un depósito de datos de cualquiera de los tipos primitivos vistos hasta ahora y de cualquier otro tipo definido por el programador.

- Expresión, que es el operando derecho, puede consistir de un valor literal (específico), otra variable del mismo tipo o una expresión que es necesario evaluar para obtener un resultado.

En todo caso, el resultado de la evaluación del operando derecho se le asigna a la variable de la izquierda.

Hay que recalcar que la asignación es destructiva, lo que quiere decir que, si la variable de la izquierda tenía almacenado un valor previo, éste se pierde cuando se le asigna un valor nuevo. Esto es importante tomarlo en cuenta, ya que no habrá forma de recuperar el valor anterior, a menos que se haya guardado en otra variable distinta antes de adquirir su nuevo valor. Pero, en todo caso, el valor que antes tenía esta variable, lo ha perdido pues debe almacenar, en el mismo espacio de memoria, un valor distinto.

Por ejemplo, supongamos que la variable x tiene almacenado el valor 5, el cual se le pudo asignar de cualquier manera: al momento de la declaración, introduciéndolo desde teclado o en una instrucción de asignación recién realizada. Podemos imaginarnos que en la memoria de la computadora existe una casilla con nombre y un valor, así:

x: | *5* |

Siendo esta la situación, si eventualmente se realiza una instrucción de asignación, por ejemplo:

```
x = 12;
```

El nuevo valor ocupará esa casilla o espacio de memoria, de la siguiente manera:

x: | *12* |

Por tanto, al desplegar la variable *x*, o utilizarla en una expresión más adelante, el valor que aportará es el 12.

El proceso de acumulación

Recordemos que en un documento anterior se dijo que uno de los usos de una variable era como acumulador, y que el conteo es un caso particular de acumulación.

Si una variable se utiliza como acumulador, entonces adquirirá un nuevo valor, incrementándose su contenido a partir del valor que almacena previamente. Podemos hacer un símil con un cofre de monedas: si se introduce un nuevo montoncito de monedas, a partir de ese momento el cofre tiene más monedas, las que ya tenía más las nuevas. Es decir, al introducir las nuevas monedas, "las antiguas no desaparecen", lo cuál sería una tragedia para cualquiera.

Pero, ¿cómo se logra eso en un programa?, o sea, ¿cómo asigno un nuevo valor, para ser acumulado y que ello no se convierta una simple asignación de tipo destructivo?, o podemos también hacernos la pregunta: ¿cómo llego a un nuevo valor en mi variable, a partir de un valor que ya contiene? Estas preguntas, que en esencia son la misma, se resuelven mediante la siguiente operación:

Variable **=** Variable **+** Expresión;

La interpretación de esta instrucción es la siguiente: se toma el valor actual de la variable, se suma con un nuevo valor, proveniente de la expresión, y el resultado se asigna a la misma variable.

Por ejemplo, si mi variable tiene almacenado el valor 5:

x: | *5* |

Y luego se realiza la instrucción:

x = x + 12;

Dado que en esta instrucción se realizará primero lo de la derecha, se sumará 5 + 12, y eso se le asignará a *x*. Así que, luego de ejecutar esta instrucción, *x* tendrá almacenado el valor 17:

x: | *17* |

Recordemos que en las instrucciones de asignación se realiza primero lo que está a la derecha y el resultado se guarda en la variable que está a la izquierda del signo =.

Así que, si antes tenías 5 moneditas en tu cofre, y luego depositás 12 moneditas más, vas a tener 17 en total, y no 12 como en el caso más arriba expuesto.

Por último es necesario decir que esto funciona para cualquier tipo de proceso de acumulación, incluso utilizando producto, *. De la misma forma, podríamos necesitar ir decreciendo, a partir de una cantidad inicial, así utilizaríamos el operador -,o el operador /.

Realización de conteos

Es muy común que surja la necesidad de realizar conteos. Para ello, requerimos partir de un valor inicial, por ejemplo cero, e ir incrementando de uno en uno, así:

```
i = 0;
.
.
.
i = i + 1;
```

Como resultado de la inicialización, tenemos una variable, i, cuyo valor es cero. Luego se realiza un proceso de acumulación de una unidad, que dará como resultado que la variable i contendrá 1. A continuación se realiza otro proceso de acumulación de una unidad, que dará como resultado que la variable i contendrá 2; y así sucesivamente.

El conteo también puede ser decreciente, lo que implica partir de una cantidad e ir decreciendo con el uso del operador de resta, -.

Otros operadores de asignación

Los operadores de asignación: +=, -=, *=, /=, %=, ++ y --, se aplican a una variable para cambiar o actualizar su contenido a partir del valor que posee.

Los operadores: +=, -=, *=, /= y %= son operadores binarios, cuyos operandos se colocan así:

Variable Operador Expresión

Donde:

Expresión puede ser una literal numérica, una variable o una expresión más compleja.

Por ejemplo:

```
a += 5;

b *= 10;
```

k %= 4;

Todas ellas, a su vez, son la forma abreviada de expresiones de la forma:

Variable = Variable Operador Expresión

Así, cada uno de los tres ejemplos anteriores se puede reescribir de la siguiente manera:

a = a + 5;

*b = b * 10;*

k = k % 4;

La ventaja de escribir estas asignaciones en la forma abreviada no estriba en que se escriben, de hecho, abreviadamente. Las ventajas son computacionales: a) la operación se aplica directamente sobre la celda de memoria que contiene la variable, sin necesidad de trasladar su valor a los registros del microprocesador, ahorrando tiempo de cpu, y b) estas asignaciones pueden formar parte de expresiones más complejas, realizando varias evaluaciones y asignaciones en una sola instrucción.

Por ejemplo, dada la variable *m*, cuyo contenido es 10:

int m;

m = 10;

Luego de incrementarla en 5, sabemos que su nuevo valor será 15. El incremento puede hacerse de la forma:

m = m + 5;

O de la forma:

m +=5;

El cuerpo de la función *main*, en ambas versiones de programa, será:

```
int m;

m = 10;
m = m + 5;

cout << "Valor de m: " << m << endl;
```

```
int m;

m = 10;
m += 5;

cout << "Valor de m: " << m << endl;
```

Escriba y ejecute ambas versiones de programa, verificando que se obtiene el mismo resultado.

Estos operadores pueden formar parte de expresiones más complejas, como por ejemplo:

Si *m* tiene un valor inicial de 3, la secuencia de instrucciones:

```
m *= 10;
k = 25 + m;
```

Pueden combinarse en una sola instrucción, así:

```
k = 25 + (m *= 10);
```

En el primer caso, primero *m* toma el valor de 30 y luego *k* toma el valor de 55.

Lo mismo sucede en el segundo caso. Para lograr el mismo efecto cuando se combinan ambas instrucciones, note que el incremento de *m* se ha colocado entre paréntesis. Esto se hace para asegurarse que el operador *= se aplique a *m*, antes de sumar el nuevo valor de *m* a *k*. Por otro lado, si se omitieran los paréntesis, dado que la suma tiene mayor prioridad que el operador *=, el compilador interpretaría que se quiere multiplicar por 10 el resultado de haber sumado 25 con *m*, lo cual no es una celda de memoria sino un resultado intermedio de la operación a realizar. Esto provocaría un error de compilación.

Se logra el mismo resultado de obtener 55 en *k* y 30 en *m*, al escribir la expresión conmutada:

```
k = (m *= 10) + 25;
```

Note que si se omitieran los paréntesis en esta expresión, el resultado no sería el esperado, ya que, por la precedencia de operadores, a *k* se le asignaría 105.

Ejercicios:

Para cada problema, escriba su propuesta en una sola instrucción abreviada. Luego verifique el resultado escribiendo el programa y realice las correcciones necesarias, si se requiere, tanto a su programa como a su propuesta.

1) Valor inicial de a: 50.

```
a = a / 10;
```

La instrucción abreviada es:

2) Valor inicial de n: 75.

```
n = n % 12;
```

La instrucción abreviada es:

3) Valor inicial de k: 10.

```
k = k * 2;
c = k + 3;
```

La instrucción abreviada es:

```
c = (k *= 2) + 3
```

4) Valor inicial de k: 10 y de n: 8.

```
k = k * 2;
n = n /4;
c = k - n;
```

La instrucción abreviada es:

5) Valor inicial de k: 10 y de n: 8.

```
k = k * 2;
n = n /4;
c = 10 * k * n;
```

La instrucción abreviada es:

Los operadores ++ y --

Los operadores ++ y -- son unarios, es decir, se aplican solamente a un operando. El operando siempre debe ser una variable, no puede ser una literal numérica ni una expresión aritmética más compleja.

Se aplican de acuerdo a la siguiente sintaxis:

Operador Variable

O también:

Variable Operador

El efecto que producen sobre la variable a la que se le aplican es: en el caso de ++, incrementar la variable en 1; en el caso de --, decrementar la variable en 1. Es decir, el uso de ++ es equivalente a escribir:

Variable = Variable + 1

```
a = a + 1;
a += 1;
a++;
++a;
```

Y el uso de -- es equivalente a escribir:

Variable = Variable - 1

```
a = a - 1;
a -= 1;
a--;
--a;
```

Por ejemplo:

Si *m* almacena el valor 7, y luego se ejecuta la instrucción:

++m;

O la instrucción:

m++;

El valor almacenado en *m* será 8. Lo que habrá sido equivalente a escribir la instrucción:

m = m + 1;

Si *f* almacena el valor 12, y luego se ejecuta la instrucción:

--f;

O la instrucción:

f--;

El valor almacenado en *f* será 11. Lo que habrá sido equivalente a escribir la instrucción:

f = f - 1;

Nuevamente, las ventajas de escribir estas asignaciones (incrementos) en la forma abreviada son computacionales: a) la operación se aplica directamente sobre la celda de memoria que contiene la variable, ahorrando tiempo de cpu, y b) estas asignaciones pueden formar parte de expresiones más complejas, realizando varias evaluaciones y asignaciones en una sola instrucción.

Es importante mencionar que sí hay una diferencia importante entre colocar los operadores de incremento, ++, o decremento, --, a la izquierda o a la derecha de su operando, pero solo cuando forman parte de una expresión más compleja: si el operador está a la izquierda de la variable, este se aplica a la variable antes de evaluar el resto de la expresión; si el operador está a la derecha de la variable, se le aplica a la variable después de evaluar el resto de la expresión. Esto tiene enorme importancia en los cálculos que se están realizando, pues, en el primer caso, la variable cambia su valor antes de que la expresión en que está inmersa se efectúe y en el segundo caso la expresión se evalúa con el valor que ya tenía la variable y esta cambia su valor después.

Por ejemplo, dada la variable *m*, cuyo contenido es 10, y la variable *j*, cuyo contenido es 5:

Luego de ejecutar la instrucción:

```
x = m * ++j;
```

El valor almacenado en *x* será 60 y el valor de *j* será 6.

Pero si se ejecuta la instrucción:

```
x = m * j++;
```

Después de ejecutarla, el valor almacenado en *x* será 50 y el valor de *j* será 6.

En el primer caso, la expresión corresponde a la secuencia de instrucciones:

```
j = j + 1;
x = m * j;
```

En el segundo caso, la expresión corresponde a la secuencia de instrucciones:

```
x = m * j;
j = j + 1;
```

Ejercicios:

Para cada problema, escriba su propuesta en una sola instrucción abreviada. Luego verifique el resultado escribiendo el programa y realice las correcciones necesarias, si se requiere, tanto a su programa como a su propuesta.

1) Valor inicial de a: 20 y de b: 5.

a = a / 10;
b = b + 1;
m = a + b;

La instrucción abreviada es:

2) Valor inicial de a: 20 y de b: 5.

a = a / 10;
m = a + b;
b = b + 1;

La instrucción abreviada es:

3) Valor inicial de a: 20, de b: 5 y de c: 3.

a = a / 10;
b = b + 1;
c = c - 1;
*m = a + b * c;*

La instrucción abreviada es:

4) Valor inicial de a: 20, de b: 5 y de c: 3.

a = a / 10;
c = c - 1;
*m = a + b * c;*
b = b + 1;

La instrucción abreviada es:

A continuación, para cada problema desglose la instrucción abreviada en la cantidad de instrucciones de asignación necesarias que eliminan todos los operadores de asignación utilizados, excepto el =.

1) Valor inicial de a: 20 y de j: 10.

m = a * 10 + ++j;

La secuencia de instrucciones equivalentes es:

2) Valor inicial de a: 20 y de j: 10.

m = a * 10 + j++;

La secuencia de instrucciones equivalentes es:

3) Valor inicial de a: 20, de k: 5 y de j: 10.

m = k++ + (a *= 10) + --j;

La secuencia de instrucciones equivalentes es: