GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: OPERADORES

1. Introducción

Los operadores son elementos que relacionan de forma diferente, los valores o variables con los que trabajamos en los lenguajes de programación. En los lenguajes de programación usamos los operadores para manipular los valores, variables y transformarlos, con el objetivo de realizar los objetivos de los programas.

Tanto C++ como Java son lenguajes ricos en operadores. Estos operadores se describen brevemente en los apartados siguientes de esta guía.

2. Desarrollo

Un operador es un carácter o grupo de caracteres que actúa sobre una, dos o más variables para realizar una determinada operación con un determinado resultado. Ejemplos típicos de operadores son la suma (+), la diferencia (-), el producto (*), etc. Los operadores pueden ser unarios, binarios y ternarios, según actúen sobre uno, dos o tres operandos, respectivamente.

2.1. Operadores Aritméticos

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y resto de la división (%).

Operador	Descripción
+	Suma los valores situados a su derecha y a su izquierda.
-	Resta el valor de su derecha del valor de su izquierda.
-	Como operador unario, cambia el signo del valor de su izquierda.
*	Multiplica el valor de su derecha por el valor de su izquierda.
/	Divide el valor situado a su izquierda por el valor situado a su derecha.
%	Proporciona el resto de la división del valor de la izquierda por el valor de la derecha (sólo enteros).

Todos estos operadores se pueden aplicar a constantes, variables y expresiones númericas. El resultado es el que se obtiene de aplicar la operación correspondiente entre los dos operandos.

El único operador que requiere una explicación adicional es el operador resto realidad su nombre completo es resto de la división entera. Este operador se aplica solamente a constantes, variables o expresiones de tipo int. Aclarado esto, su significado es evidente: 23 %4 es 3, puesto que el resto de dividir 23 por 4 es 3. Si a %b es cero, a es múltiplo de b.

El operador suma(+) se puede aplicar adicionalmente entre variables del tipo secuencia de caracteres (string) que arroja como resultado la concatenación de los valores de las variables.

2.1.1. Prioridad de los Operadores Aritméticos

Cuando encontramos varios operadores en una misma expresión los lenguajes de programación tendrán que evaluarlos en un orden determinado. Ese orden lo conocemos como prioridad o

precendencia de operadores.

Cuando se trata de operadores aritméticos es muy fácil imaginarse la prioridad de unos respecto a otros, dado que funcionan igual que en las matemáticas. Por ejemplo, siempre se evaluará una multiplicación antes que una suma.

Sin embargo, no siempre es tan fácil deducir cómo se va a resolver la asociatividad de los operadores, por lo que hay que aprenderse unas reglas de precedencia que vamos a resumir en este punto.

Dentro de una misma expresión los operadores se evalúan en el siguiente orden:

1. *, /, % (Multiplicación, división, resto de la división)
2. +, - (Suma y resta)

En el caso en el que en una misma expresión se asocien operadores con igual nivel de prioridad, éstos se evalúan de izquierda a derecha.

En el caso que quieras romper las reglas de precedencia de los operadores puedes usar los paréntesis. Funcionan con cualquier tipo de operadores y se comportan igual que en las matemáticas. Puedes definir mediante los paréntesis qué operadores se van a relacionar con qué operandos, independientemente de las reglas mencionadas anteriormente.

- Todas las expresiones entre paréntesis se evalúan primero
- Las expresiones con paréntesis anidados se evalúan de dentro a fuera
- El paréntesis más interno se evalúa primero.

2.2. Operadores de Asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación por excelencia es el operador igual (=). La forma general de las sentencias de asignación con este operador es:

```
variable = expression;
```

Cuyo funcionamiento es como sigue: se evalúa expresion y el resultado se deposita en variable, sustituyendo cualquier otro valor que hubiera en esa posición de en esa posición de memoria anteriormente.

C++ y Java dispone de otros operadores de otros operadores de asignación. asignación. Se trata de versiones abreviadas del operador (=) que realizan operaciones acumulativas sobre una variable.

Operador	Utilización	Expresión equivalente
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

Desde el punto de vista matemático no tiene sentido (¡Equivale a 0 = 1!), pero sí lo tiene considerando que en realidad el operador de asignación (=) representa una sustitución; en efecto, se toma el valor de variable contenido en la memoria, se le suma una unidad y el valor resultante vuelve a depositarse en memoria en la zona correspondiente al identificador variable, sustituyendo al valor que había anteriormente.

Así pues, una variable puede aparecer a la izquierda y a la derecha del operador (=). Sin embargo, a la izquierda del operador de asignación (=) no puede haber nunca una expresión: tiene que ser necesariamente el nombre de una variable

2.3. Operadores unarios

Los operadores más (+) y menos (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica.

2.4. Operadores incrementales

Los operadores incrementales (++) y (- -) son operadores unarios que incrementan o disminuyen en una unidad el valor de la variable a la que afectan. Estos operadores se pueden utilizar de dos formas:

- Precediendo a la variable (por ejemplo: ++i). En este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.
- Siguiendo a la variable (por ejemplo: i++). En este caso primero se utiliza la variable en la expresión (con el valor anterior) y luego se incrementa.

2.5. Operadores de Relación

Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor. El resultado de estos operadores es siempre un valor boolean (true o false) según se cumpla o no la relación considerada.

Operador	Utilización	El resultado es verdadero (true)
>	op1 >op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 <op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes

Todos los operadores relacionales son operadores binarios (tienen dos operandos), y su forma general es la siguiente:

```
expresion1 op expresion2
```

donde **op** es uno de los operadores (==, <, >, <=, >=, !=). El funcionamiento de estos operadores es el siguiente: se evalúan **expresion1** y **expresion2**, y se comparan los valores resultantes. Si la condición representada por el operador relacional se cumple, el resultado es verdadero (true,1); si la condición no se cumple, el resultado es falso (false,0).

2.5.1. Prioridad de los Operadores Relacionales

Todos los operadores relacionales tienen el mismo nivel de prioridad en su evaluación. En general, los operadores relacionales tienen menor prioridad que los aritméticos.

2.6. Operadores Lógicos

Los operadores lógicos son operadores binarios que permiten combinar los resultados de los operadores relacionales, comprobando que se cumplen simultáneamente varias condiciones, que se cumple una u otra, etc. Estos operadores se utilizan para establecer relaciones entre valores lógicos. Los valores lógicos son los valores boleanos:

- True (verdadero)
- False (falso)

Operador	Nombre	Utilizacion	El resultado es verdadero (true)
&&	AND	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
||	OR	op1 || op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	NOT	!op1	true si op es false y false si op es true
&	AND	op1 & op2	true si op1 y op2 son true. Siempre se evalúa op2
|	OR	op1 | op2	true si op1 u op2 son true. Siempre se evalúa op2

Además, dado que los operadores relacionales tienen como resultado un operador lógico, que se deduce mediante la comparación de los operandos, los operadores lógicos pueden tener como operandos el resultado de una expresión relacional.

2.6.1. Prioridad de los Operadores Lógicos

El orden de precedencia de los operadores lógicos entre ellos es el siguiente, de más precedente a menos:

1. NOT
2. AND

3. OR

2.7. Operadores Relacionados con punteros

Estos operadores son propiamente de C++ y no están presentes en Java.

Operador	Nombre	Descripción
&	Dirección	Cuando va seguido por el nombre de una variable, entrega la dirección de dicha variable &abc es la dirección de la variable abc.
*	Indirección	Cuando va seguido por un puntero, entrega el valor almacenado en la dirección apuntada por él abc.

2.8. Operadores de Estructuras y Uniones

Estos operadores son propiamente de C++ y no están presentes en Java.

Operador	Nombre	Descripción
.	Pertenecia directa	El operador de pertenencia (punto) se utiliza junto con el nombre de la estructura o unión, para especificar un miembro de las mismas. Si tenemos una estructura cuyo nombre es nombre, y miembro es un miembro especificado por el patrón de la estructura, nombre.miembro identifica dicho miembro de la estructura. El operador de pertenencia puede utilizarse de la misma forma en uniones.
->	Pertenecia indirecta	El operador de pertenencia indirecto: se usa con un puntero estructura o unión para identificar un miembro de las mismas. Supongo que ptrstr es un puntero a una estructura que contiene un miembro especificado en el patrón de estructura con el nombre miembro. En este caso identifica al miembro correspondiente de la estructura apuntada. El operador de pertenencia indirecto puede utilizarse de igual forma con uniones.

2.9. Operadores que actúan a nivel de bits

C++ y Java disponen también de un conjunto de operadores que actúan a nivel de bits. Las operaciones de bits se utilizan con frecuencia para definir señales o flags, esto es, variables de tipo entero en las que cada uno de sus bits indican si una opción está activada o no.

Estos operadores sólo pueden usarse con los tipos int y char y funcionan bit a bit. El operador de desplazamiento se puede utilizar para realizar multiplicaciones o divisiones rápidas, pues cada desplazamiento a la izquierda multiplica por 2, y cada desplazamiento a la derecha divide por 2.

Operador	Uso	Resultado

>>	op1 >> op2	Desplaza los bits de op1 a la derecha una distancia op2. El desplazamiento a la derecha es un operador que desplaza los bits del operando situado a su izquierda hacia la derecha el número de sitios marcado por el operando situado a su derecha. Los bits que superan el extremo derecho del byte se pierden. En tipos unsigned, los lugares vacantes a la izquierda se rellenan con ceros. En tipos con signo el resultado depende del ordenador utilizado; los lugares vacantes se pueden rellenar con ceros o bien con copias del signo (bit extremo izquierdo). En un valor sin signo tendremos (10001010)>>2 == (00100010) en el que cada bit se ha movido dos lugares hacia la derecha.
<<	op1 << op2	Desplaza los bits de op1 a la izquierda una distancia op2. El desplazamiento a la izquierda es un operador que desplaza los bits del operando izquierdo a la izquierda el número de sitios indicando por el operados de su derecha. Las posiciones vacantes se rellenan con ceros, y los bits que superan el límite del byte se pierden. Así: (10001010)<<2 == (1000101000) cada uno de los bits se ha movido dos lugares hacia la izquierda
>>>	op1 >>> op2	Desplaza los bits de op1 a la derecha una distancia op2 (positiva)
&	op1 & op2	Operador AND a nivel de bits. El and de bits es un operador que hace la comparación bit por bit entre dos operandos. Para cada posición de bit, el bit resultante es 1 únicamente cuando ambos bits de los operandos sean 1. En terminología lógica diríamos que el resultado es cierto si, y sólo si, los dos bit que actúan como operandos lo son también. Por tanto (10010011) & (00111101) == (00010001) ya que únicamente en los bits 4 y 0 existe un 1 en ambos operandos.
|	op1 | op2	Operador OR a nivel de bits. El or para bits es un operador binario realiza una comparación bit por bit entre dos operandos. En cada posición de bit, el bit resultante es 1 si alguno de los operandos o ambos contienen un 1. En terminología lógica, el resultado es cierto si uno de los bits de los operandos es cierto, o ambos lo son. Así: (10010011) | (00111101) == (10111111) porque todas las posiciones de bits con excepción del bit número 6 tenían un valor 1 en, por lo menos, uno de los operandos.

∧	op1 ∧ op2	Operador XOR a nivel de bits (1 si sólo uno de los operandos es 1). El or exclusivo de bits es un operador binario que realiza una comparación bit por bit entre dos operandos. Para cada posición de bit, el resultante es 1 si alguno de los operandos contiene un 1; pero no cuando lo contienen ambos a la vez. En terminología, el resultado es cierto si lo es el bit u otro operando, pero no si lo son ambos. Por ejemplo: (10010011) ∧ (00111101) == (10101110) Observe que el bit de posición 0 tenía valor 1 en ambos operandos; por tanto, el bit resultante ha sido 0.
∼	∼op2	Operador complemento (invierte el valor de cada bit). El complemento a uno o negación en bits es un operador unario, cambia todos los 1 a 0 y los 0 a 1. Así: ∼(10011010) == 01100101

En binario, las potencias de dos se representan con un único bit activado. Por ejemplo, los números (1, 2, 4, 8, 16, 32, 64, 128) se representan respectivamente de modo binario en la forma (00000001, 00000010, 00000100, 00001000, 00010000, 00100000, 01000000, 10000000), utilizando sólo 8 bits. La suma de estos números permite construir una variable flags con los bits activados que se deseen. Por ejemplo, para construir una variable flags que sea 00010010 bastaría hacer flags=2+16.

Operador	Utilización	Expresión equivalente
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
∧ =	op1 ∧ = op2	op1 = op1 ∧ op2
>>=	op1 >>= op2	op1 = op1 >> op2
<<=	op1 <<= op2	op1 = op1 << op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

2.10. Operador ternario

Este operador permite realizar bifurcaciones condicionales sencillas. Su forma general es la siguiente:

```
booleanExpression ? res1 : res2
```

donde se evalúa **booleanExpression** y se devuelve **res1** si el resultado es true y **res2** si el resultado es false. Es el único operador ternario. Como todo operador que devuelve un valor puede ser utilizado en una expresión.

2.11. Misceláneas

Operador	Descripción
sizeof	Devuelve el tamaño, en bytes, del operando situado a su derecha. El operando puede ser un especificador de tipo, en cuyo caso se emplean paréntesis; por ejemplo, sizeof(float). Puede ser también el nombre de una variable con concreta o de un array, en cuyo caso no se emplean paréntesis: sizeof foto. Solo presente en C++
(tipo)	Operador de moldeado, convierte el valor que vaya a continuación en el tipo especifi- cado por la palabra clave encerrada entre los paréntesis. Por ejemplo, (float)9 convierte el entero 9 en el número de punto flotante 9.0.
,	El operador coma une dos expresiones en una, garantizando que se evalúa en primer lugar la expresión situada a la izquierda, una aplicación típica es la inclusión de más información de más información en la expresión de control de un bucle for:

2.12. Precedencia de operadores

El orden en que se realizan las operaciones es fundamental para determinar el resultado de una expresión.La siguiente lista muestra el orden en que se ejecutan los distintos operadores en un sentencia, de mayor a menor precedencia:

1. expr++ expr - -
2. ++expr - -expr +expr -expr ∼ !
3. * / %
4. + -
5. << >> >>>
6. < > <= >=
7. == !=
8. &
9. ∧
10. |
11. &&
12. ? :
13. = += -= *= /= %= &= ∧= |= <<= >>= >>>=

Para priorizar una operación con operador de menor precedencia que otro se debe encerrar dicha operación entre parentisis.

Todos los operadores binarios, excepto los operadores de asignación, se evalúan de izquierda a derecha. Los operadores de asignación se evalúan de derecha a izquierda, lo que significa que el valor de la derecha se copia sobre la variable de la izquierda.

3. Implementación

3.1. C++

```
struct {
   int codigo;
   float precio;
} articulo, *ptrstr; //Operador Indireccion

int main() {
   //Operador asignacion
   int abc = 22;
   //Operador de direccion
   int def = &abc ;
   //Operador Indireccion
   int val = *def;
   //Operador pertenencia directa
   articulo.codigo = 1265;
   //Operador de direccion
   ptrstr = &articulo;
   // Operador pertenencia indirecta,Operador unitario
   ptrstr->codigo = -3451;
   int sum =0;
   //Operador asignacion, operador coma, operarador asigancion,
   // operador relacional,operador logico ,operador relacional, operador
      asignacion
   for(int chatos=2,ronda=0;ronda<1000 && chatos<=50000;chatos*=2){
      //Operador asignacion, operador aritmetico
      ronda = ronda + chatos;
      //Operador a nivel bits
      sum =sum | chatos;
   }
   //Operador asignacion
   int x=1 ;
   //Operador asignacion
   int y=10;
   //Operador incremental
   y++;
  //Operador relacional, Operador ternario,Operador aritmetico,Operador
     aritmetico
  int z = (x<y)?x-3:y/8;
  return 0;
}
```

3.2. Java

```java
public static void main(String[] args){
   //Operador asignacion
   int abc = 22;
   int sum =0;
   //Operador asignacion, operador coma, operarador asigancion,
   // operador relacional,operador logico ,operador relacional, operador
      asignacion
   for(int chatos=2,ronda=0;ronda<1000 && chatos<=50000;chatos*=2){
      //Operador asignacion, operador aritmetico
      ronda = ronda + chatos;
      //Operador a nivel bits
      sum =sum | chatos;
   }
   //Operador asignacion
   int x=1 ;
   //Operador asignacion
   int y=10;
   //Operador incremental
   y++;
   //Operador relacional, Operador ternario,Operador aritmetico,Operador
      aritmetico
   int z = (x<y)?x-3:y/8;
}
```

4. Complejidad

La complejidad de trabajar con los operadores en las sentencias se consideran como operaciones elementales por tanto su complejidad es $O(1)$.

5. Aplicaciones

Es indudable que el uso de operadores con variables y constantes nos permiten elaborar las instrucciones que conforman el algoritmo para solucionar un determinado problema. Aunque cabe destacar que existen un grupo de operadores que están un tanto especializados o son mas usados en determinadas instrucciones como por ejemplo :

- Los operadores relacionales se utilizan con mucha frecuencia en las bifurcaciones y en los bucles.
- Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false) o los resultados de los operadores relacionales.