GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: EXPRESIONES Y SENTENCIAS

1. Introducción

Ya han aparecido algunos ejemplos de expresiones o sentencias de los lenguaje C++ y Java en las guías precedentes. Una expresión es una combinación de variables y/o constantes, y operadores. La expresión es equivalente al resultado que proporciona al aplicar sus operadores a sus operandos. Por ejemplo, 1+5 es una expresión formada por dos operandos (1 y 5) y un operador (el +); esta expresión es equivalente al valor 6, lo cual quiere decir que allí donde esta expresión aparece en el programa, en el momento de la ejecución es evaluada y sustituida por su resultado. Una expresión puede estar formada por otras expresiones más sencillas, y puede contener paréntesis de varios niveles agrupando distintos términos. En C++ y Java existen distintos tipos de expresiones.

2. Conocimientos previos

2.1. Variable

Una variable es un nombre que hace referencia a un lugar en memoria que contiene un valor que puede cambiar a lo largo del programa.

2.2. Operador

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.

3. Desarrollo

3.1. Sentencias y expresiones

Una **expresión** es un conjunto variables unidos por operadores. Son órdenes que se le dan al computador para que realice una tarea determinada.

Una **sentencia** es una **expresión** que acaba en **punto y coma** (;). Se permite incluir varias sentencias en una línea, aunque lo habitual es utilizar una línea para cada sentencia.

Las **expresiones** de C++ y Java son unidades o componentes elementales de unas entidades de rango superior que son las **sentencias**. Las **sentencias** son unidades completas, ejecutables en sí mismas. Ya se verá que muchos tipos de **sentencias** incorporan **expresiones** aritméticas, lógicas o generales como componentes de dichas **sentencias**.

3.2. Expresiones Aritméticas

Están formadas por variables y/o constantes, y distintos operadores aritméticos e incrementales (+, -, *, /, %, ++, - -). Como se ha dicho, también se pueden emplear paréntesis de tantos niveles como se desee, y su interpretación sigue las normas aritméticas convencionales. Por ejemplo, la solución de la ecuación de segundo grado:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

se escribe, en C++ en la forma:

```
x=(-b+sqrt((b*b)-(4*a*c)))/(2*a);
```

donde, estrictamente hablando, sólo lo que está a la derecha del operador de asignación (=) es una expresión aritmética. El conjunto de la variable que está a la izquierda del signo (=), el operador de asignación, la expresión aritmética y el carácter (;) constituyen una sentencia. En la expresión anterior aparece la llamada a la función de librería sqrt(), que tiene como valor de retorno la raíz cuadrada de su único argumento. En las expresiones se pueden introducir espacios en blanco entre operandos y operadores; por ejemplo, la expresión anterior se puede escribir también de la forma:

```
x = (-b + sqrt((b * b) - (4 * a * c)))/(2 * a);
```

3.3. Expresiones Lógicas

Los elementos con los que se forman estas expresiones son **valores lógicos**; verdaderos (**true**, o distintos de 0) y falsos (**false**, o iguales a 0), y los operadores lógicos ||, && y !. También se pueden emplear los **operadores relacionales** (<, >, <=, >=, ==, !=) para producir estos valores lógicos a partir de valores numéricos. Estas expresiones equivalen siempre a un valor 1 (**true**) o a un valor 0 (**false**). Por ejemplo:

```
a = ((b>c)&&(c>d))||((c==e)||(e==b));
```

donde de nuevo la expresión lógica es lo que está entre el operador de asignación (=) y el (;). La variable **a** valdrá (1,true) si **b** es mayor que **c** y **c** mayor que **d**, ó si **c** es igual a **e** ó **e** es igual a **b**.

3.4. Expresiones generales

Una de las características más importantes (y en ocasiones más difíciles de manejar) del C++ o Java es su flexibilidad para combinar expresiones y operadores de distintos tipos en una expresión que se podría llamar general, aunque es una expresión absolutamente ordinaria.

Recuérdese que el resultado de una expresión lógica es siempre un valor numérico (un 1 ó un 0); esto permite que cualquier expresión lógica pueda aparecer como sub-expresión en una expresión aritmética. Recíprocamente, cualquier valor numérico puede ser considerado como un valor lógico: **true** si es distinto de 0 y false si es igual a 0. Esto permite introducir cualquier expresión aritmética como sub-expresión de una expresión lógica. Por ejemplo:

```
(a - b*2.0) && (c != d)
```

A su vez, el operador de asignación (=), además de introducir un nuevo valor en la variable que figura a su izda, deja también este valor disponible para ser utilizado en una expresión más general. Por ejemplo, supóngase el siguiente código que inicializa a 1 las tres variables **a**, **b** y **c**:

```
a = b = c = 1;
```

que equivale a:

```
a = (b = (c = 1));
```

En realidad, lo que se ha hecho ha sido lo siguiente. En primer lugar se ha asignado un valor unidad a **c**; el resultado de esta asignación es también un valor unidad, que está disponible para ser asignado a **b**; a su vez el resultado de esta segunda asignación vuelve a quedar disponible y se puede asignar a la variable **a**.

3.5. Reglas de precedencia y asociatividad

El resultado de una expresión depende del orden en que se ejecutan las operaciones. El siguiente ejemplo ilustra claramente la importancia del orden. Considérese la expresión:

```
3 + 4 * 2
```

Si se realiza primero la suma (3+4) y después el producto (7*2), el resultado es 14; si se realiza primero el producto (4*2) y luego la suma (3+8), el resultado es 11. Con objeto de que el resultado de cada expresión quede claro e inequívoco, es necesario definir las reglas que definen el orden con el que se ejecutan las expresiones de C++ y Java. Existe dos tipos de reglas para determinar este orden de evaluación: las reglas de **precedencia** y de **asociatividad**. Además, el orden de evaluación puede modificarse por medio de paréntesis, pues siempre se realizan primero las operaciones encerradas en los paréntesis más interiores. Los distintos operadores de C++ y Java se ordenan según su distinta precedencia o prioridad; para operadores de la misma precedencia o prioridad, en algunos el orden de ejecución es de izquierda a derecha, y otros de derecha a izquierda (se dice que se asocian de izda a dcha, o de dcha a izda). A este orden se le llama **asociatividad**.

3.6. Sentencias simples

Una sentencia simple es una expresión de algún tipo terminada con un carácter (;). Un caso típico son las declaraciones o las sentencias aritméticas. Por ejemplo:

```
float real;
espacio = espacio_inicial + velocidad * tiempo;
```

3.7. Sentencias nulas o vacías

En algunas ocasiones es necesario introducir en el programa una sentencia que ocupe un lugar, pero que no realice ninguna tarea. A esta sentencia se le denomina sentencia vacía y consta de un simple carácter (;). Por ejemplo:

```
;
```

3.8. Sentencias compuestas o bloques

Muchas veces es necesario poner varias sentencias en un lugar del programa donde debería haber una sola. Esto se realiza por medio de sentencias compuestas. Una sentencia compuesta es un conjunto de declaraciones y de sentencias agrupadas dentro de llaves .... También se conocen con el nombre de bloques. Una sentencia compuesta puede incluir otras sentencias, simples y compuestas. Un ejemplo de sentencia compuesta es el siguiente:

```
{
   int i = 1, j = 3, k;
   double masa;
   masa = 3.0;
   k = y + j;
}
```

Las sentencias compuestas se utilizarán con mucha frecuencia en al introducir las sentencias que permiten modificar el flujo de control del programa o cuando se programa de forma modular o funcional.

3.9. Comentarios

Existen dos formas diferentes de introducir comentarios entre el código de Java o C++. Los comentarios son tremendamente útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. Además permite que cualquier persona distinta al programador original pueda comprender el código escrito de una forma más rápida. Se recomienda acostumbrarse a comentar el código desarrollado. De esta forma se simplifica también la tarea de estudio y revisión posteriores.

Java y C++ interpreta que todo lo que aparece a la derecha de dos barras "//" en una línea cualquiera del código es un comentario del programador y no lo tiene en cuenta. El comentario puede empezar al comienzo de la línea o a continuación de una instrucción que debe ser ejecutada. La segunda forma de incluir comentarios consiste en escribir el texto entre los símbolos /*...*/. Este segundo método es válido para comentar más de una línea de código. Por ejemplo:

```
// Esta linea es un comentario
int a=1; // Comentario a la derecha de una sentencia

// Esta es la forma de comentar mas de una linea utilizando
```

```
// las dos barras. Requiere incluir dos barras al comienzo de cada linea

/* Esta segunda forma es mucho mas comoda para comentar un numero elevado de
   lineas ya que solo requiere modificar el comienzo y el final. */
```

4. Complejidad

La complejidad de una sentencia no va estar dada por el largo que tengan sino por los operadores u operaciones que ella realice en el caso que sea una sentencia compuesta.

5. Aplicaciones

Esta claro que las sentencias nos vas a representar las intrucciones que definidos en nuestro algoritmos (independientemente que si utilizamos diagrama de flujo o seudocódigo) en un determinado lenguaje de programación. Es la estructura organizativa básica para la conformación de programas computacionales.