CALCULADORA POLACA Y CALCULADORA POLACA INVERSA

Sandra Castro

Sandyc47@hotmail.com

Bryan Tualle

Brayscout96@hotmail.com

Universidad de las Fuerzas Armadas "ESPE"

RESUMEN: El artículo presenta la evaluación de expresiones algebraicas empleando algoritmos fundamentales para la construcción de compiladores, pasando por la conversión de expresiones de infija a postfija, y de expresiones de infija a prefija, la evaluación de expresiones en notación postfija y prefija, el algoritmo de evaluación por precedencia de operadores, y el algoritmo de construcción de funciones de precedencia. El objetivo del artículo es escribir un programa en un lenguaje convencional de programación de sistemas, utilizando las posibilidades de entrada y salida del lenguaje c++ para leer las expresiones a evaluar desde la entrada; procesarlas y enviar los resultados a la salida.

PALABRAS CLAVE: Evaluación de expresiones postfijas prefija, algoritmo de precedencia de operadores.

ABSTRACT: The article presents the evaluation of algebraic expressions using fundamental algorithms for the construction of compilers, through the conversion of expressions from infix to postfix, and from expressions of infix to prefix, the evaluation of expressions in postfix and prefix notation, the evaluation algorithm by precedence of operators, and the algorithm of construction of precedence functions. The objective of the article is to write a program in a conventional system programming language, using the possibilities of input and output of the c ++ language to read the expressions to evaluate from the input; process them and send the results to the exit.

INTRODUCCIÓN

Nuestro objetivo es construir un programa evaluador de expresiones aritméticas simples, para ello veremos que es necesario utilizar pilas. Algunas de las técnicas utilizadas en el diseño de compiladores pueden utilizarse a pequeña escala en la implementación de una calculadora típica. Las calculadoras evalúan lo que se conoce como expresiones infijas (o notación infija), como por ejemplo 1 + 2, que consisten en un operador binario (la suma, +) junto con operandos (argumentos) a su izquierda y su derecha (el 1 y el 2). Este formato, bastante fácil de evaluar en un principio, puede hacerse más complejo. Considérese la expresión:

$$1 + 2 * 3$$

Matemáticamente, la expresión se evalúa a 7, porque el operador de multiplicación tiene mayor precedencia que la suma. Sin embargo, algunas calculadoras con otro esquema de evaluación podrían dar como respuesta 9. Esto ilustra que una simple evaluación de izquierda a derecha no es suficiente, ya que no podemos empezar evaluando 1 + 2. Considérese ahora la expresión:

$$10 - 4 - 3$$

¿Qué resta debe evaluarse primero? Las restas se evaluarán de izquierda a derecha, dando como resultado 3. Es decir, la resta, a igualdad de precedencia, asocia de izquierda a derecha; supondremos que con el resto de operadores que vamos a usar ocurrirá lo mismo, es decir, a igualdad de precedencia entre operadores se asocia de izquierda a derecha. Con todo esto, la evaluación de la siguiente expresión se vuelve bastante compleja:

$$1 - 2 - 4/5 * (3 * 6)/7$$

Para eliminar la posible ambigüedad de la expresión anterior podemos introducir paréntesis para ilustrar el orden correcto de los cálculos:

$$(1 - 2) - (((4/5) * (3 * 6))/7)$$

Aunque los paréntesis desambiguan el orden de evaluación, resulta difícil decir que hacen más claro el mecanismo de evaluación. Sin embargo, existe una notación para expresiones diferente, denominada notación postfija, que proporciona un mecanismo directo de evaluación. Las siguientes secciones muestran como funciona. En la sección 1. estudiaremos la notación postfija, mostrando cómo las expresiones escritas con esta notación pueden evaluarse con un simple recorrido de izquierda a derecha. La sección 2. muestra cómo las expresiones

originales, que utilizan la notación habitual infija, se pueden convertir a notación postfija. De esta manera podremos construir un sistema de evaluación de expresiones aritméticas infijas en dos pasos. En primer lugar, transformaremos la expresión infija a la expresión postfija correspondiente y en segundo lugar evaluaremos esta última expresión. A continuación, se presenta un ejemplo completo en la sección 3.

DESARROLLO

1. Expresiones postfijas

Una expresión postfija está formada por una serie de operadores y operandos, donde un operador va precedido por sus operadores. Por ejemplo, para una de las expresiones infijas mostradas en la sección anterior:

$$1 + 2 * 3$$

la expresión postfija equivalente sería

$$1\ 2\ 3 * +$$

donde el 2 y el 3 preceden al operador *, puesto que son sus operandos, y el 1 y el resultado de 2 3 $*$ preceden al operador+.

Se evalúa de la siguiente forma: cuando se encuentra un operando, se apila en la pila; cuando se encuentra un operador, el número apropiado de operandos son desapilados de la pila, se evalúa la operación indicada por el operador, y el resultado se apila de nuevo en la pila.

Para operadores binarios, que son los más comunes (y los ´únicos que vamos a tratar en esta práctica), se desapilan dos operandos. Ejemplos de operadores binarios son la suma (+), la resta (−), la multiplicación (*) y la división (/).

Cuando la expresión postfija ha sido procesada completamente, el ´único elemento que quede en la pila será el resultado de la evaluación, es decir, el valor equivalente a evaluar la expresión postfija. La natación postfija es una forma natural de evaluar expresiones, pues con ella no son necesarias reglas de precedencia ni existen ambigüedades en la evaluación. Un pequeño ejemplo de cómo se evalúa una expresión postfija.

Ejemplo de evaluación de la expresión postfija 1 2 3 $*$ +, correspondiente a la expresión infija 1 + 2 $*$ 3. Se muestra cómo se va realizando el análisis de la expresión paso a paso. Para cada símbolo que se analiza se muestra la acción a realizar, así como el estado de la pila.

La evaluación procede de la siguiente manera: el 1, el 2 y el 3 son apilados, en ese orden en la pila. Para procesar el *, se desapilan los dos elementos superiores de la pila: esto es, el 3 y después el 2. Obsérvese que el primer elemento desapilado se convierte en el operando derecho del operador, y el segundo elemento desapilado en el operando izquierdo; por tanto, los parámetros se obtienen de la pila en orden inverso al natural. Para la multiplicación, esto no importa, pero para la resta y la división, desde luego que sí. El resultado de la multiplicación es 6, que es apilado en la pila. En este momento, la cima de la pila es un 6, y debajo hay un 1. Para procesar el +, se desapilan el 6 y el 1, y su suma, 7, se apila. En este punto, la expresión se ha leído completamente y la pila sólo tiene un elemento. Por tanto, la respuesta final a la evaluación de la expresión es 7.

Como hay 3 operandos y 2 operadores, habrá 5 pasos y 5 apilamientos para evaluar la expresión. Así pues, la evaluación de una expresión postfija requiere un tiempo lineal. El punto que falta por estudiar para realizar la evaluación de expresiones infijas es un algoritmo que convierta notación infija en notación postfija. Una vez tengamos uno, tendremos un algoritmo para evaluar expresiones infijas.

2. Conversión de notación infija a postfija

El principio básico para convertir una expresión infija a una postfija es el siguiente: cuando se encuentra un operando en la cadena de entrada podemos pasarlo directamente a la cadena de salida; sin embargo, cuando encontramos un operador, no podemos pasarlo todavía a la salida, pues debemos esperar a encontrar su segundo operando. Por consiguiente, debemos guardarlo temporalmente en una estructura adecuada. Si consideramos una expresión como:

$$1 + 2 * 3/4$$

la expresión correspondiente en notación postfija es:

$$1\ 2\ 3 * 4/+$$

observamos que en ocasiones los operadores pueden aparecer en orden inverso a como aparecían en la expresión infija. Por supuesto, esto sólo es cierto si la precedencia de los operadores involucrados crece al ir de izquierda a derecha. Aun así, el hecho comentado sugiere que una pila es la estructura adecuada para almacenar los operadores pendientes. Siguiendo esta lógica, cuando encontremos un

operador, debe ser colocado de alguna manera en la pila.

Consideramos ahora como ejemplo otra expresión infija más simple:

$$2 * 5 - 1$$

Ejemplo de transformación de la expresión infija 2 ∗ 5 − 1, a la expresión postfija correspondiente 2 5 ∗ 1−. Se muestra cómo se va realizando la transformación de la expresión paso a paso. Para cada símbolo de la entrada que se procesa se muestra cómo va quedando la salida y el estado de la pila.

Su transformación de forma infija a postfija se muestra en la figura 1. Cuando alcanzamos el operador −, el 2 y el 5 se han incorporado ya a la salida, y * está en la pila. Ya que − tiene menor precedencia que *, * tiene que aplicarse al 2 y al 5. Por lo que debemos desapilar * y, en general, cualquier otro operador que hubiese en la cima de la pila con mayor precedencia que −. En consecuencia, la expresión resultante postfija es:

$$2\ 5 * 1-$$

En general, cuando procesamos un operador de la entrada, debemos sacar de la pila aquellos operadores que deban ser procesados atendiendo a las reglas de precedencia y asociatividad. Antes de resumir el algoritmo, deben responderse algunas preguntas. En primer lugar, si el símbolo actual es un + y la cima de la pila es un +, ¿debería desapilarse el + de la pila, o debería quedarse ahí? La respuesta se obtiene decidiendo si el + encontrado en la entrada nos indica que el + de la pila cuenta ya con sus operandos. Ya que el + asocia de izquierda a derecha, la respuesta es afirmativa. ¿Qué ocurre con los paréntesis? un paréntesis izquierdo puede considerarse como un operador de máxima precedencia cuando está en la entrada, pero de precedencia mínima cuando está en la pila. En consecuencia, el paréntesis izquierdo de la entrada se apilaría siempre sin más. Cuando encontremos un paréntesis derecho en la entrada, desapilaremos hasta encontrar el correspondiente paréntesis izquierdo. Por otra parte, en la cadena de salida no aparecen paréntesis. En la sección 5.4 se muestra un ejemplo de transformación que ilustra todos los aspectos expuestos. Presentamos ahora un resumen de los diferentes casos del algoritmo de transformación de expresiones infijas a expresiones postfijas. En cada paso se supone que estamos consultando el primer elemento de la expresión infija de entrada. Todo lo que se desapila se concatena a la expresión postfija de salida, a excepción de los paréntesis. Los casos son:

- **Operandos:** pasan directamente a la salida.
- **Paréntesis derecho:** desapilar símbolos hasta encontrar un paréntesis izquierdo.
- **Operador:** Desapilar todos los símbolos hasta que encontremos un símbolo de menor precedencia. Apilar entonces el operador encontrado.
- **Fin de la entrada:** desapilar el resto de símbolos en la pila.

3. Un ejemplo completo

Ahora ya conocemos el mecanismo de transformación de una expresión infija a forma postfija (visto en la sección 2) y el mecanismo de evaluación de una expresión postfija (visto en la sección 1). Así pues, ya estamos en disposición de crear un sistema que haciendo uso de las dos técnicas pueda evaluar expresiones aritméticas sencillas. En esta práctica solo trataremos los operadores de suma (+), resta (−), multiplicación (*) y división (/), así como el uso de paréntesis.

Pero antes de pasar a la implementación del sistema ofrecemos a continuación un ejemplo que ilustra los algoritmos mostrados en las secciones anteriores.

El objetivo es evaluar la expresión infija:

$$1 - 2 - 4/5 * (3 * 6)/7$$

Para ello, el primer paso será transformarla a la expresión postfija correspondiente:

$$1\ 2 - 4\ 5\ /\ 3\ 6 * *\ 7\ / -$$

Para cada símbolo de la entrada analizado se muestra lo que queda por escanear de la cadena de entrada, el estado de la pila de operadores y la cadena que se ha obtenido hasta este momento en la salida.

Una vez hemos obtenido la expresión postfija

$$1\ 2 - 4\ 5/3\ 6 * *7/-$$

queda evaluarla conforme al algoritmo descrito en la sección 1. Los pasos seguidos en este algoritmo se muestran a continuación. Para cada símbolo de la expresión postfija procesado se muestra el estado de la pila. Cuando se haya terminado de procesar toda la expresión, el número que permanezca en la pila será el valor correspondiente a la expresión original.

4. Código QR

Los códigos QR son códigos bidimensionales formados por una matriz de propósito general diseñados para un escaneo rápido de información. Su nombre procede de “Quick Response” (respuesta rápida) ya que se diseñó para ser decodificado a alta velocidad. El código QR es un código abierto, de ahí su extensa utilización. Presentan una forma cuadrada y está caracterizados por tres cuadros más pequeños, ubicados en la parte superior e inferior izquierda, que permiten identificar la posición del código al lector (ver Figura 1).

Figura 1. Ejemplo de código QR

5. Backup

Si está escribiendo un programa de copia de seguridad, debe poder leer y acceder a los archivos que están siendo utilizados por otros programas. La forma correcta de hacerlo es usar el Servicio de instantáneas de volumen. Las páginas de MSDN cubren una gran cantidad de material que no es necesario para un programa de copia de seguridad simple, así que aquí hay algunos pasos simples para comenzar.

CONCLUSIONES

Para concluir tenemos que las pilas las listas enlazadas, todos sus métodos y las diferentes expresiones son una de las herramientas utilizadas para programar y procesar de una manera automatizada todos los sistemas que usamos día tras día en relación a técnicas o métodos para agilizar operaciones se refiere, es su principio fundamental dar soluciones a procesos que ameritan de largos pasos que dan como resultado el control de un asusto. Estas herramientas son uno de los instrumentos más usados en la programación para estructurar y organizar información.

REFRERENCIAS

[1] Selección, desarrollo y evaluación de competencias en DLL (Didáctica de la Lengua y la Literatura), Universidad de Sevilla, Secretariado de Recursos Audiovisuales y Nuevas Tecnologías, 2012

[2] msdn.microsoft.com, visual C++, IDE y herramientas de desarrollo, complilar programas de C/C++, archivos dll en visual c++

[3] QR-Codes, http://www.denso-dave.com/qrcode/qrstandard-e.hmtl

[4] https://wj32.org/wp/2012/12/13/how-to-backup-files-in-c-using-the-volume-shadow-copy-service-vss/

[5] http://www.wetcom.com/content/como-desarrollar-un-servicio-web-rest-con-java-y-spring/

[6] Hurtado Albir, A., *Traducción y Traductología*, p. 25