Trabalho sobre Avaliações de Expressões Algébricas

Estruturas de Dados I
Prof. Roney Pignaton da Silva
Data de Entrega: 07/05/2019
Meio: enviar arquivos para roney.silva@ufes.br

1. Introdução

As notações infixa, posfixa e prefixa são três notações diferentes, mas equivalentes, de escrita de expressões algébrica. Dessas notações, estamos mais familiarizados com a notação infixa na escrita de expressões algébricas. Ao escrever uma expressão aritmética usando a notação infixa, o operador é colocado entre os operandos. Por exemplo, A + B; aqui, o operador MAIS (+) é colocado entre os dois operandos A e B.

Embora seja fácil escrever expressões usando notação infixa, os computadores acham difícil analisar tais expressões, uma vez que são necessárias muitas informações para avaliar tais expressões (p.e., regras de precedência e associatividade do operador, e sobre como os colchetes substituem essas regras).

Assim, os computadores trabalham de maneira mais eficiente com expressões escritas usando as notações prefixa e posfixa.

A notação posfixa foi desenvolvida pelo matemático e filósofo polonês Jan Łukasiewicz. Seu objetivo era desenvolver uma notação prefixa sem parênteses (também conhecida como Reverse Polish Notation ou RPN – Notação Polonesa Reversa). Na notação posfixa, como o nome sugere, o operador é colocado após os operandos. Por exemplo, se uma expressão é escrita como A + B em notação infixa, a mesma expressão pode ser escrita como AB + em notação posfixa. A ordem de avaliação de uma expressão posfixa é sempre da esquerda para a direita. Até a presença de colchetes não podem alterar a ordem de avaliação. A expressão (A + B) * C pode ser escrita como:

a) [AB+]*C

ou AB+C* na notação posfixa

Uma operação posfixa nem sequer segue as regras de precedência do operador. O operador que ocorre primeiro na expressão é operada primeiro. Por exemplo, dada uma notação posfixa AB + C *. Ao avaliarmos essa expressão, a adição será executada antes da multiplicação. Assim, vemos que na uma notação posfixa, os operadores são aplicados aos operandos que estão a sua esquerda. No exemplo, AB + C *, + é aplicado em A e B, então * é aplicado sobre o resultado da adição e C.

Na notação prefixa a avaliação também é realizada da esquerda para direita, e a única diferença entre uma notação posfixa e uma notação de prefixa é que, em uma notação de prefixa, o operador é colocado antes dos operandos. Por exemplo, se A + B for uma expressão na notação infixa, então a expressão correspondente na notação de prefixo é dada por + AB.

Ao avaliar uma expressão prefixa, os operadores são aplicados aos operandos que estão presentes imediatamente à direita do operador. Essa notação também não segue as regras de precedência do operador e associatividade, e mesmo os colchetes não podem alterar a ordem de avaliação.

2. Conversão de expressões Infixa para Posfixa

Uma expressão algébrica infixa pode conter parênteses, operandos e operadores. Por simplicidade, considere apenas os operandos +, -, *, /,% (módulo). A ordem de precedência destes operadores pode ser dada conforme abaixo:

- Maior prioridade *, /,%
- Prioridade inferior +, -

A ordem de avaliação desses operadores pode ser alterado fazendo uso de parênteses. Por exemplo, se temos uma expressão A + B * C, então primeiro B * C será avaliado e o resultado será adicionado a A. Mas a mesma expressão, se escrito como (A + B) * C, irá avaliar A + B primeiro e depois o resultado será multiplicado por C.

O algoritmo abaixo transforma uma expressão infixa na expressão posfixa. Esse algoritmo aceita uma expressão infixa que pode conter operadores, operandos, e parênteses. Para simplificar, assumimos que a operação infixa contém apenas módulo (%), operadores de multiplicação (*), divisão (/), adição (+) e subtração (-) e operadores com mesma ordem de precedência são executados da esquerda para a direita.

O algoritmo usa uma pilha para manter temporariamente os operadores. A expressão posfixa é obtida da esquerda para a direita usando os operandos da expressão infixa e os operadores que são removidos da pilha.

Algorítmo

Passo 1) O primeiro passo neste algoritmo colocar (push) um um parêntese esquerdo na pilha e adicionar um parêntese direito correspondente no final da expressão infixa. O algoritmo é repetido até que a pilha esteja vazia.

```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
       IF a "(" is encountered, push it on the stack
       IF an operand (whether a digit or a character) is encountered, add it to the
       postfix expression.
       IF a ")" is encountered, then
         a. Repeatedly pop from stack and add it to the postfix expression until a
            "(" is encountered.
         b. Discard the "(". That is, remove the "(" from stack and do not
            add it to the postfix expression
       IF an operator O is encountered, then
         a. Repeatedly pop from stack and add each operator (popped from the stack) to the
            postfix expression which has the same precedence or a higher precedence than O
         b. Push the operator O to the stack
       [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```

Abaixo, podemos observar como o algoritmo se comporta ao converter a expressão abaixo:

(a) A – (B / C + (D % E * F) / G)* H

(Passo 1) (A – (B / C + (D % E * F) / G)* H)

Solução

Infix Character Scanned	Stack	Postfix Expression
	(	
A	(	A
–	(–	A
(	(– (	A
B	(– (	A B
/	(– (/	A B
C	(– (/	A B C
+	(– (+	A B C /
(	(– (+ (	A B C /
D	(– (+ (	A B C / D
%	(– (+ (%	A B C / D
E	(– (+ (%	A B C / D E
*	(– (+ (% *	A B C / D E
F	(– (+ (% *	A B C / D E F
)	(– (+	A B C / D E F * %
/	(– (+ /	A B C / D E F * %
G	(– (+ /	A B C / D E F * % G
)	(–	A B C / D E F * % G / +
*	(– *	A B C / D E F * % G / +
H	(– *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * –

Considerando o exposto acima, como primeira parte deste trabalho, desenvolver o especificado abaixo.

PARTE 1:

Escrever um programa em C que transforma uma expressão INFIXA em uma expressão POSFIXA. Para tal, utilize o esquema mostrado acima, sendo a conversão suportada por um TAD Stack (Pilha).

Além das funções pop, push e auxiliares de pilhas, o programa deverá implementar a função principal de conversão, conforme abaixo:

*void InfixtoPostfix(Stack *stk, char source[], char target[])*: Essa função recebe como argumentos de entrada:
1) um ponteiro para uma pilha (Stack) que será utilizada como suporte para o processo de conversão;
2) Uma string source contendo a expressão infixa a ser convertida;
3) Uma string target a qual conterá a expressão posfixa final resultado da conversão;

Obs: o elemento de informação armazenado na pilha (stack) é sempre um caracter.

Exemplo de Saída do programa

```
Entre com uma expressão INFIXA: A+B-C*D
A Correspondente expressão POSFIXA é: AB+CD*-
```

3. Avaliação de uma expressão Posfixa

A facilidade de avaliação das expressões na notação posfixa é um motivador para o processo de conversão mostrado anteriormente. Assim, dada uma expressão algébrica escrita em notação infixa, o computador primeiro converte a expressão na notação posfixa equivalente e, em seguida, avalia a expressão posfixa. Ambas tarefas (conversão da notação infixa para postfix e avaliação da posfixa), fazem uso de pilhas (Stack). Usando pilhas, qualquer expressão posfixa pode ser avaliada com muita facilidade. Todos os caracteres da expressão posfixa serão lidos da esquerda para a direita. Se o caractere encontrado for um operando, ele será enviado (PUSH) para a pilha. Por outro lado, se um operador for encontrado, os dois valores que estiverem no top da pilha serão retirados (POP) e o operador será aplicado sobre eles. O resultado é então empilhado (PUSH) na pilha.

Veja o exemplo a seguir. Considere a expressão infixa dado por:

INFIXA: 9 - ((3 * 4) + 8) / 4.

Essa expressão infixa pode ser escrita como:

POSFIXA: 9 3 4 * 8 + 4 / -

O procedimento de avaliação é mostrado abaixo:

Character Scanned	Stack
9	9
3	9, 3
4	9, 3, 4
*	9, 12
8	9, 12, 8
+	9, 20
4	9, 20, 4
/	9, 5
-	4

Algoritmo

```
Step 1: Add a ")" at the end of the
        postfix expression
Step 2: Scan every character of the
        postfix expression and repeat
        Steps 3 and 4 until ")"is encountered
Step 3: IF an operand is encountered,
        push it on the stack
        IF an operator O is encountered, then
        a. Pop the top two elements from the
           stack as A and B as A and B
        b. Evaluate B O A, where A is the
           topmost element and B
           is the element below A.
        c. Push the result of evaluation
           on the stack
        [END OF IF]
Step 4: SET RESULT equal to the topmost element
        of the stack
Step 5: EXIT
```

PARTE 2:

Escrever um programa em C que transforma uma expressão INFIXA em uma expressão POSFIXA e, a partir da expressão POSFIXA realiza a sua avaliação mostrando o resultado. Para tal, utilize o esquema mostrado acima, sendo a conversão de INFIXA para POSFIXA resultado da implementação da PARTE 1 deste trabalho e a avaliação também suportada por um TAD Stack (Pilha).

Além das funções pop, push, InfixtoPostfix e auxiliares de pilhas, o programa deverá implementar a função principal da avaliação da expressão, conforme abaixo:

*float evaluatePostfixExp(Stack *stc,char exp[])*: Essa função recebe como argumentos de entrada:
1) um ponteiro para uma pilha (Stack) que será utilizada como suporte para o processo de avaliação;
2) Uma string exp contendo a expressão posfixa a ser avaliada;

Exemplo de Saída do programa

```
Entre com uma expressão INFIXA: 9 – ((3 * 4) + 8) / 4
A Correspondente expressão POSFIXA é: 9 3 4 * 8 + 4 / –
Valor da Avaliação para: 4.00
```

FINAL (Parte 1 + Parte 2)

O que deve ser apresentado (entregue):

1) Arquivo stack.h, com a definição da estrutura do TAD Stack, bem como as assinaturas das funções sobre esse TAD;

2) Arquivo stack.c, com a implementação das funções do TAD Stack;

3) Arquivo mainExpressoes.c com o programa que testa as funções de conversão de notação INFIXA para POSFIXA e a avaliação da expressão POSFIXA. Obviamente que o resultado da parte 1 e parte 2 do trabalho devem ser testados neste mesmo arquivo.

É obrigatório o uso de alocação dinâmica de memória para implementar a pilha (stack).