Universidade Estadual de Campinas
Colégio Técnico de Campinas - DPD

Estruturas de Dados

Resolução de Expressões Aritméticas usando Pilhas e Filas
(2^o Ano do Curso 59)

O propósito deste trabalho é, usando pilhas e filas, construir em Java o programa de uma Calculadora de Expressões Aritméticas apropriadamente orientado a objetos. Nele, uma expressão algébrica deverá ser avaliada e seu resultado deverá ser calculado e exibido. Veja o exemplo abaixo:

Entrada: 10 + (2 * 3 - 4) ^ 2 / 4 + 6 * 2 Saída: 23

Para solucionar o problema devido a mudança de prioridade promovida pelos parênteses, o matemático polonês *Jan Lukasiewicz* elaborou uma saída para representarmos e avaliarmos expressões sem nos preocuparmos com as prioridades das operações e, até mesmo, abrir mão dos parênteses. Podemos considerar três formas para representar uma expressão:

- Infixa: operador está entre os operandos. (1 + 2)
- Pré-Fixa: Operador precede os operandos (+ 1 2)
- Pós-Fixa: Operador após os operandos (1 2 +)

Esta implementação utilizará pilhas e filas como estruturas de dados e a técnica utilizada será a de transformar a expressão fornecida da tradicional notação infixa para notação pós-fixa e, a partir desta última, calcular o valor da expressão

OBS: Expressões mal formadas também poderão ser entradas e a má formação deverá ser detectada e sinalizada.

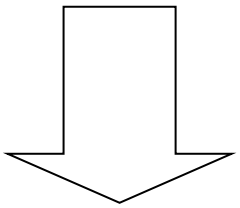
As formas pré-fixa e pós-fixa são conhecidas como notação polonesa (PN) e notação polonesa reversa (RPN), respectivamente, em que a última tem mostrado ser a mais eficiente para construção de algoritmos.

Para o funcionamento da nossa calculadora, os seguintes operadores poderão ser utilizados:

Símbolo	**Significado**
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
^	Exponenciação
( )	Parentetização

Essa calculadora considera a precedência dos operadores e os parênteses durante o cálculo da expressão. A tabela seguinte mostra a precedência dos operadores, da maior precedência no começo para os de menor precedência.

Símbolo	Significado
( )	Parentização
^	Exponenciação
* /	Multiplicação Divisão
+ -	Adição Subtração



Observação: Havendo operadores de mesma prioridade (como é o caso da multiplicação e da divisão, bem como da adição e da subtração), sua calculadora deverá resolver a expressão da esquerda para direita, ou seja, o que aparecer primeiro.

1. Uma *string* deverá ser lida e quebrada em pedaços para verificar se é válida ou não:

   1.1. Solicitar a digitação de uma expressão aritmética. Você deve ler a expressão digitada pelo usuário como uma *string* e remover todos os espaços em branco. Suponha que tenha sido:

   Veja como ficará a *string* do nosso exemplo:

   10+(2*3-4)^2/4+6*2

   1.2. Agora você deve implementar uma lógica para quebrar a string em partes. Na linguagem JAVA, você possui uma classe chamada StringTokenizer. Esta classe permite quebrar a string ("exp") em partes quando encontrar um delimitador ("**+-*/^()**"). O true faz com que pegue os delimitadores também como pedaços.

```
StringTokenizer quebrador = new StringTokenizer (exp, "+-
*/^()", true);
```

   1.3. Os seguintes métodos desta classe deverão ser utilizados para que você quebre a string em pedaços:

   - **quebrador.nextToken(); // Lê os pedaços**
   - **quebrador.hasMoreToken(); // Verifica se tem mais pedaços**

2. Você deverá nesta segunda etapa, construir **Conversor de Notação Infixa para Pós-fixa**. Nesta etapa, será utilizada uma pilha e uma fila para transformar a expressão da notação infixa para a notação pós-fixa:

2.1. Devemos pegar um pedaço da expressão que foi quebrada no passo anterior:

2.1.1. Se o pedaço for um **abre parênteses** (“(“), colocaremos este pedaço na Pilha de Operadores.

2.1.2. Se o pedaço for um **número**, colocaremos este pedaço na Fila de Saída.

2.1.3. Se o pedaço for um **operador aritmético** (+,-,*,/,^), deveremos realizar dois passos:

2.1.3.1. Analisar se não temos elementos para remover da pilha. Você deverá utilizar a tabela a seguir, para decidir se irá remover elementos da Pilha de Operadores:

2.1.3.1.1. Se o valor da tabela for T (true), o elemento da pilha deve ser desempilhado e colocado na Fila de Saída.

2.1.3.1.2. Se o valor da tabela for F (false), o elemento da pilha não deve ser desempilhado e você deve ir para o passo 2.1.3.2.

2.1.3.1.3. A remoção de elementos da pilha deve parar quando for encontrado o valor F, ou seja, o passo 2.1.3.1 deverá ser repetido enquanto for encontrado o valor T (true) na tabela.

		Símbolo pego da Sequência						
		(	**^**	*****	**/**	**+**	**-**	**)**
Símbolo que está no topo da Pilha	**(**	F	F	F	F	F	F	T
	^	F	F	T	T	T	T	T
	*****	F	F	T	T	T	T	T
	/	F	F	T	T	T	T	T
	+	F	F	F	F	T	T	T
	-	F	F	F	F	T	T	T
	)	F	F	F	F	F	F	F

2.1.3.2. Assim que o passo 2.1.3.1 for verificado e concluído, deveremos colocar o operador da expressão no topo da Pilha de Operadores.

2.1.4. Se o pedaço for um fecha parênteses (“)”), devemos desempilhar um elemento da pilha e colocá-lo na fila. Repetimos este passo até encontrar o “(”. Ambos os símbolos “(” como o “)” não irão para a fila. Eles simplesmente deverão ser descartados.

2.2. O passo 2.1 termina com a Expressão vazia; quando isso ocorrer, caso a Pilha de Operadores não estiver vazia, todo seu conteúdo deverá ser desempilhado e enfileirado na Fila de Saída.

2.3. Veja que ao final do passo 2.2, você terá obtido a expressão em notação pós-fixa. Veja abaixo a aplicação destes passos para o nosso exemplo:

Expressão	**Pilha de Operadores**	**Fila de Saída**
10+(2*3-4)^2/4+6*2	Vazia	Vazia
+(2*3-4)^2/4+6*2	Vazia	10
(2*3-4)^2/4+6*2	+	10
2*3-4)^2/4+6*2	(+	10
*3-4)^2/4+6*2	(+	10 2
3-4)^2/4+6*2	* (+	10 2
-4)^2/4+6*2	* (+	10 2 3

Expressão	Pilha de Operadores	Fila de Saída
-4)^2/4+6*2	(+	10 2 3 *

4)^2/4+6*2	- (+	10 | 2 | 3 | *
)^2/4+6*2	- (+	10 | 2 | 3 | * | 4
)^2/4+6*2	(+	10 | 2 | 3 | * | 4 | -
^2/4+6*2	+	10 | 2 | 3 | * | 4 | -
2/4+6*2	^ +	10 | 2 | 3 | * | 4 | -
/4+6*2	^ +	10 | 2 | 3 | * | 4 | - | 2
/4+6*2	+	10 | 2 | 3 | * | 4 | - | 2 | ^
4+6*2	/ +	10 | 2 | 3 | * | 4 | - | 2 | ^
+6*2	/	10 | 2 | 3 | * | 4 | - | 2 | ^ | 4

	+	
+6*2	+	10 2 3 * 4 - 2 ^ 4 /
+6*2	Vazia	10 2 3 * 4 - 2 ^ 4 / +
6*2	+	10 2 3 * 4 - 2 ^ 4 / +
*2	+	10 2 3 * 4 - 2 ^ 4 / + 6
2	* +	10 2 3 * 4 - 2 ^ 4 / + 6
Vazia	* +	10 2 3 * 4 - 2 ^ 4 / + 6 2
Vazia	Vazia	10 2 3 * 4 - 2 ^ 4 / + 6 2 * +

3. Você deverá nesta terceira etapa, construir **Calculadora de Expressão**:
   - 3.1. Devemos definir 3 variáveis: dois valores do tipo double (v1 e v2) e um char (op). Lembre-se, nossa expressão pós-fixa está na Fila de Saída.
   - 3.2. **Regra Geral** - Devemos remover um elemento da Fila de Saída e verificar:
     - 3.2.1. Se o elemento for um valor numérico, devemos empilhá-lo na Pilha Resultado. Devemos repetir este passo 1 até que o elemento removido da Fila de Saída seja um operador.
     - 3.2.2. Se o elemento removido da Fila de Saída for um operador, devemos armazená-lo na variável char definida como **op**.
       - **3.2.2.1.** Não pare por aí... ao encontrar um operador lógico e depois de armazená-lo em op, devemos

desempilhar um elemento (que deve ser um string numérico), transformá-lo em número e colocá-lo em v2 e, em seguida, devemos desempilhar mais um elemento (que deve ser um string numérico) transformá-lo em número e colocá-lo em v1. Em seguida, você deve calcular o resultado da expressão: **v1 opl v2.** Este resultado deve ser armazenado na Pilha Resultado.

3.3. Pronto! Caso haja ainda elementos na Fila de Saída, você deve aplicar a regra geral novamente (volte ao passo 3.2). Caso não haja elementos na Fila de Saída, deve ficar um único elemento na Pilha Resultado e este elemento será o resultado final da expressão.

3.4. Veja que ao final do passo 3.3, você terá obtido o resultado final da expressão que a saída esperada do seu programa. Veja abaixo a aplicação destes passos para o nosso exemplo:

Fila de Saída (expressão em notação pós-fixa)	**Pilha Resultado**	**Variáveis**
10 | 2 | 3 | * | 4 | - | 2 | ^ | 4 | / | + | 6 | 2 | * | +	Vazia	v1 = v2 = op =
2 | 3 | * | 4 | - | 2 | ^ | 4 | / | + | 6 | 2 | * | +	10	v1 = v2 = op =
3 | * | 4 | - | 2 | ^ | 4 | / | + | 6 | 2 | * | +	2 10	v1 = v2 = op =
* | 4 | - | 2 | ^ | 4 | / | + | 6 | 2 | * | +	3 2 10	v1 = v2 = op =
4 | - | 2 | ^ | 4 | / | + | 6 | 2 | * | +	3 2 10	v1 = v2 = op = ‘*’
4 | - | 2 | ^ | 4 | / | + | 6 | 2 | * | +	2 10	v1 = v2 = 3 op = ‘*’

Fila de Saída (expressão em notação pós-fixa)	Pilha Resultado	Variáveis
4 - 2 ^ 4 / + 6 2 * +	10	v1 = 2 v2 = 3 op = '*' (2*3 é 6)
4 - 2 ^ 4 / + 6 2 * +	6 10	v1 = v2 = op =
- 2 ^ 4 / + 6 2 * +	4 6 10	v1 = v2 = op =
2 ^ 4 / + 6 2 * +	4 6 10	v1 = v2 = op = '-'
2 ^ 4 / + 6 2 * +	6 10	v1 = v2 = 4 op = '-'
2 ^ 4 / + 6 2 * +	10	v1 = 6 v2 = 4 op = '-' (6-4 da 2)
2 ^ 4 / + 6 2 * +	2 10	v1 = v2 = op =
^ 4 / + 6 2 * +	2 2 10	v1 = v2 = op =
		v1 = v2 =

Fila de Saída (expressão em notação pós-fixa)	Pilha Resultado	Variáveis
4 / + 6 2 * +	2 2 10	op = ‘^’
4 / + 6 2 * +	2 10	v1 = v2 = 2 op = ‘^’
4 / + 6 2 * +	10	v1 = 2 v2 = 2 op = ‘^’ (2^2 dá 4)
- 4 / + 6 2 * +	4 10	v1 = v2 = op =
/ + 6 2 * +	4 4 10	v1 = v2 = op =
+ 6 2 * +	4 4 10	v1 = v2 = op = ‘/’
+ 6 2 * +	4 10	v1 = v2 = 4 op = ‘/’
+ 6 2 * +	10	v1 = 4 v2 = 4 op = ‘/’ (4/4 dá 1)
+ 6 2 * +	1 10	v1 = v2 = op =

Fila de Saída (expressão em notação pós-fixa)	Pilha Resultado	Variáveis
6 | 2 | * | +	1 10	v1 = v2 = op = ‘+’
6 | 2 | * | +	10	v1 = v2 = 1 op = ‘+’
6 | 2 | * | +	Vazia	v1 = 10 v2 = 1 op = ‘+’ (10+1 dá 11)
6 | 2 | * | +	11	v1 = v2 = op =
2 | * | +	6 11	v1 = v2 = op =
* | +	2 6 11	v1 = v2 = op =
+	2 6 11	v1 = v2 = op = ‘*’
+	6 11	v1 = v2 = 2 op = ‘*’

Fila de Saída (expressão em notação pós-fixa)	Pilha Resultado	Variáveis
+	11	v1 = 6 v2 = 2 op = '*' (6*2 dá 12)
+	12 11	v1 = v2 = op =
Vazia	12 11	v1 = v2 = op = '+'
Vazia	11	v1 = v2 = 12 op = '+'
Vazia	Vazia	v1 = 11 v2 = 12 op = '+' (11+12 dá 23)
Vazia	23	v1 = v2 = op =

Observações Finais:

- A pilha deve terminar com apenas um valor e este valor (23) é o resultado final da expressão inicialmente digitada.
- É imprescindível: (a) que o programa seja adequadamente dividido em classes; (b) que as classes Pilha e Fila sejam implementadas utilizando vetores (não podem ser utilizadas classes prontas da linguagem); (c) que todas as validações cabíveis sejam feitas por todos os métodos e que incorretudes sejam sinalizadas através de exceções que, posteriormente, sejam apropriadamente tratadas.
- O presente trabalho deve ser feito em grupos de até 3 alunos e deverá ser entregue, impreterivelmente no **dia 18 de Novembro de 2024.**

Bom Trabalho	Prof André Carvalho	04/nov/2024