Faculdade de Computação e Informática
Ciência da Computação
Estrutura de Dados I – 3ª etapa – 2023.1 – 3G
Prof. Dr. Ivan Carlos Alcântara de Oliveira

Atividade Apl1 – Avaliador de expressões matemáticas

Uma aplicação clássica do TAD Pilha é a avaliação de expressões matemáticas.

Podemos encontrar algumas dificuldades na avaliação de uma expressão matemática, tais como:

- A existência de prioridades diferentes para os operadores, que não nos permite efetuá-los na ordem em que os encontramos na expressão;
- A existência de parênteses, que alteram a prioridade das operações.

Uma solução para os itens descritos no parágrafo anterior foi criada pelo polonês Jan Łukasiewicz e consiste em usar uma representação de expressões matemáticas onde não existam prioridades e nem a necessidade do uso de parênteses. Essa solução é conhecida como Notação Polonesa Reversa (*Reverse Polish Notation* ou RPN).

Na notação infixa, a mais "tradicional" na aritmética, o operador aparece entre os operandos. Por exemplo: **A + B**.

Além da notação infixa, temos:

- A notação prefixa: o operador precede os operandos. Por exemplo: **+ A B**.
- A notação posfixa: o operador segue os operandos. Por exemplo: **A B +**.

As formas prefixa e posfixa são denominadas notação polonesa e notação polonesa reversa, respectivamente.

A notação polonesa reversa tem algumas vantagens em relação à notação infixa:

- Cada operador aparece imediatamente após os valores que ele deve operar;
- Não existe a necessidade de usar parênteses.

A tabela a seguir contém alguns exemplos de notação infixa e o equivalente em notação posfixa.

Notação infixa	**Notação posfixa**
A + B * C	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 *

Para converter uma expressão de infixa para posfixa, deve-se:

- Parentizar completamente a expressão (definir a ordem de avaliação);
- Varrer a expressão da esquerda para a direita e, para cada símbolo:
  a) Se for parêntese de abertura, ignorar;
  b) Se for operando, copiá-lo para a expressão posfixa (saída);
  c) Se for operador, fazê-lo aguardar;

d) Se for parêntese de fechamento, copiar o último operador acessado.

Supondo a conversão da expressão A+B*C-D:

1ª Etapa: Parentização

A + (B * C) - D →
(A + (B * C)) – D →
((A + (B * C)) – D)

2a. Etapa: Varredura, conforme quadro ilustrativo desse processo a seguir:

Símbolo	Ação	Pilha	Saída
(	ignora	P:[]	
(	ignora	P:[]	
A	copia	P:[]	A
+	aguarda	P:[+]	A
(	ignora	P:[+]	A
B	copia	P:[+]	AB
*****	aguarda	P:[*, +]	AB
C	copia	P:[*, +]	ABC
)	remove	P:[+]	ABC*
)	remove	P:[]	ABC*+
-	aguarda	P:[-]	ABC*+
D	copia	P:[-]	ABC*+D
)	remove	P:[]	ABC*+D-

Um algoritmo para conversão de uma expressão infixa qualquer para posfixa seria:

- Inicie com uma pilha vazia;
- Realize uma varredura na expressão infixa, copiando todos os identificadores encontrados diretamente para a expressão de saída.

a) Ao encontrar um operador:

1. Enquanto a pilha não estiver vazia e houver no seu topo um operador com prioridade maior ou igual ao encontrado, desempilhe o operador e copie-o na saída;
2. Empilhe o operador encontrado;

b) Ao encontrar um parêntese de abertura, empilhe-o;

c) Ao encontrar um parêntese de fechamento, remova um símbolo da pilha e copie-o na saída, até que seja desempilhado o parêntese de abertura correspondente.

- Ao final da varredura, esvazie a pilha, movendo os símbolos desempilhados para a saída.

Um exemplo de execução do algoritmo de Conversão de infixa para posfixa, supondo a expressão A*(B+C)/D, é apresentado abaixo:

Símbolo	Ação	Pilha	Saída
A	copia para a saída	P:[]	A
*****	pilha vazia, empilha	P:[*]	A
(	sempre deve ser empilhado	P:[(, *]	A
B	copia para a saída	P:[(,*]	AB
+	prioridade maior, empilha	P:[+, (, *]	AB
C	copia para a saída	P:[+, (, *]	ABC
)	desempilha até achar ‘)’	P:[*]	ABC+
/	prioridade igual, desempilha	P:[/]	ABC+*
D	copia para a saída	P:[/]	ABC+*D
	final, esvazia pilha	P:[]	ABC+*D/

Algoritmo de avaliação de uma expressão na forma posfixa:

- Primeiramente, atribui-se valores numéricos às variáveis da expressão a ser avaliada
- Inicia-se com uma pilha vazia;
- Varre-se a expressão e, para cada elemento encontrado:

a) Se for operando, então empilhar seu valor;
b) Se for operador, então desempilhar os dois últimos valores, efetuar a operação com eles e empilhar de volta o resultado obtido;

- No final do processo, o resultado da avaliação estará no topo da pilha.

Avaliando a expressão (A+B)/(C-D)*E → AB+CD-/E*

Expressão	Elem.	Ação	Pilha
AB+CD-/E*			P:[ ]
B+CD-/E*	A	empilha valor de A = 7	P:[7]
+CD-/E*	B	empilha valor de B = 3	P:[3, 7]
CD-/E*	+	desempilha 3	P:[7]
		desempilha 7	P:[]
		empilha 3 + 7	P:[10]
D-/E*	C	empilha valor de C = 6	P:[6, 10]
-/E*	D	empilha valor de D = 4	P:[4, 6, 10]
/E*	-	desempilha 4	P:[6, 10]
		desempilha 6	P:[10]
		empilha 6 - 4	P:[2, 10]
E*	/	desempilha 2	P:[10]
		desempilha 10	P:[ ]
		empilha 10/2	P:[5]
*	E	empilha valor de E = 9	P:[9, 5]
		desempilha 9	P:[5]
		desempilha 5	P:[ ]
	*	empilha 5 * 9	P:[45]

(continua na próxima página...)

Implementação

Implementar um programa em Java que avalia expressões aritméticas que podem conter os seguintes operadores:
+ (adição)
- (subtração)
***** (multiplicação)
/ (divisão)
^ (exponenciação)

O programa deve:

- Realizar a leitura de uma expressão matemática na notação infixa, sendo que as variáveis possuem uma única letra, por exemplo, (A + B) * A + B – C;
- Após inserida a expressão, deve-se realizar a leitura dos valores numéricos de cada variável (no exemplo do item anterior, as variáveis A, B e C);
- Converter a expressão para notação polonesa reversa (notação posfixa);
- Realizar o cálculo e apresentar o resultado da expressão.

O programa também deve validar a expressão matemática, isto é:

- Aceitar somente os cinco operadores (adição, subtração, multiplicação, divisão e exponenciação).
- Aceitar somente variáveis como operandos, sendo que as variáveis possuem uma única letra;
- Considerar que uma expressão matemática na notação infixa pode conter parênteses que definem a prioridade das operações.

Caso a expressão inserida seja inválida (por exemplo, a expressão contém algum operador que não seja um dos cinco indicados ou possui uma quantidade incorreta de parênteses, como ((((A * (B – C)), o programa deve exibir uma mensagem informando o erro.

Na implementação desta Aplicação 1 deve ser utilizada a classe “TAD pilha” apesentada em aula contendo as devidas alterações de tamanho e tipo de dado.

Deve ser implementado um menu de opções contendo 5 opções:

1. Entrada da expressão aritmética na notação infixa.
2. Entrada dos valores numéricos associados às variáveis.
3. Conversão da expressão, da notação infixa para a notação posfixa, e exibição da expressão convertida para posfixa.
4. Avaliação da expressão (apresentação do resultado do cálculo, mostrando a expressão e os valores das variáveis).
5. Encerramento do programa.

(continua na próxima página...)

Observações

1. O trabalho pode ser feito por grupos de até 5 pessoas.

2. Um único aluno do grupo deverá publicar o trabalho no Moodle.

3. Deverá ser entregue um relatório com os resultados da **"Atividade Aplicação 1"** com base no Template disponibilizado contendo:

- Dados dos integrantes do grupo (nome e TIA).
- *Printscreen* de testes de execução das opções do Menu. Ao menos 2 testes de cada opção, se for permitido, caso contrário basta um único teste da opção.
- O relatório deve conter ao seu final um Apêndice contendo o código fonte desenvolvido (separado por arquivos, se for o caso). Em cada arquivo inserir:
    - um cabeçalho (comentário) com as identificações completas de todos os membros do grupo.
    - Documentação adequada e inclusão de comentários úteis e informativos em cada bloco de código.

4. Junto ao relatório também devem ser entregues os códigos fontes em Java.

5. **A entrega pelo Moodle (relatório + fontes em um arquivo compactado no formato "zip") da Aplicação 1 deve ser realizada no dia 26/03 (domingo) até as 23h59min**.

6. **Deverá ser realizada uma apresentação pelo grupo para a Aplicação 1 no dia 27/03 no horário da aula**:

- O grupo deverá apresentar o processo de construção da solução do seu projeto, resultados obtidos e testes do menu de opções no tempo máximo de 6 (seis) minutos.

O projeto será avaliado de acordo com os seguintes critérios:

- Completude, clareza e ausência de erros de linguagem no relatório;
- Funcionamento correto da Aplicação;
- O trabalho deve ser desenvolvido na **linguagem Java** e será testado usando o compilador do **Eclipse**
- O quão fiel é o programa quanto à descrição do enunciado;
- Indentação, comentários e legibilidade do código;
- Clareza na nomenclatura de variáveis e funções;
- Apresentação realizada com clareza, conhecimento e cumprimento do tempo estabelecido.

Para auxiliar na documentação do código e entendimento do que é um programa com boa legibilidade siga as dicas apresentadas nas páginas abaixo:

- http://www.ime.usp.br/~pf/algoritmos/aulas/layout.html
- http://www.ime.usp.br/~pf/algoritmos/aulas/docu.html

Como este trabalho pode ser feito em **grupo**, evidentemente um grupo pode "*discutir*" o problema dado com outros **grupos**, inclusive as "*dicas*" para chegar às soluções, mas cada grupo deve ser responsável pela solução final e pelo desenvolvimento da sua aplicação.

(continua na próxima página...)

Informações importantes sobre critérios de avaliação

- Será descontado 1,0 (um) ponto caso a entrega não respeite o enunciado. Exemplos:
  - O enunciado pede para enviar um arquivo compactado no formato zip, mas o arquivo enviado está no formato rar.
  - Não há identificação nem referências nos arquivos de código.

- Será descontado 1,0 (um) ponto caso o arquivo zip contenha pastas e arquivos desnecessários.
  Exemplo:
  - Pastas intermediárias criadas no processo de compilação (Debug, obj, bin, ...).

- O projeto deve ser desenvolvido em linguagem Java. Caso a solução apresentada use outra linguagem, será descontado 2,0 (dois) pontos.

- Projeto que possui erros de compilação ou que trava durante a execução automaticamente perde 50% da nota máxima.
  Sobre erros de compilação: considere apenas erros, não há problema se o projeto tiver *warnings* (embora *warnings* podem avisar possíveis travamentos em tempo de execução, como loop infinito, divisão por zero etc.).
  Quando há necessidade de entrada de dados por parte do usuário, assuma que o usuário vai inserir as informações corretas (ex. tipos de dados corretos), a menos que o enunciado explicite que você deve garantir que os dados de entrada estejam corretos.

- Entregas que são cópias de outros projetos serão automaticamente zeradas. Tenha em mente que códigos em que a pessoa só alterou nomes de variáveis, funções etc. são considerados cópia.