Faculdade de Computação e Informática
Ciência da Computação
Estrutura de Dados I – 3ª etapa – 2024.1
Professor André Kishimoto

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 *

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.

O programa deve apresentar 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.

Algoritmo para conversão de expressão infixa para posfixa

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.

Um exemplo de execução do algoritmo de avaliação de uma expressão na forma posfixa é apresentado abaixo, assumindo a expressão infixa **(A+B)/(C-D)*E** que foi convertida para a expressão posfixa **AB+CD-/E*** e que A=7, B=3, C=6, D=4 e E=9.

Expressão	Elemento	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]

Desenvolvimento e vídeo explicativo

Grupo:
A atividade deve ser realizada em **grupo de, no máximo, 3 pessoas**.

Código:

A solução deve ser implementada em linguagem Java e deve usar uma versão adaptada da implementação do TAD Pilha estática/sequencial realizada durante as aulas.

- A solução não deve usar as estruturas de dados oferecidas pela linguagem Java (projetos que usarem tais estruturas, como a classe Stack do Java, serão desconsiderados – zero).
- Inclua a identificação do grupo (nome completo e RA de cada integrante) no início de cada arquivo de código, como comentário.
- Inclua todas as referências (livros, artigos, sites, vídeos, entre outros) consultadas para solucionar a atividade como comentário no arquivo .java que contém a main(). Caso use ChatGPT ou similar, inclua, em um documento PDF, o histórico da interação (prompt + respostas)

Vídeo:

- Além da solução escrita em Java, o grupo deverá gravar um vídeo de, no máximo, 7 (sete) minutos, explicando como a expressão matemática em notação infixa é convertida para a notação posfixa e como a expressão em notação posfixa é avaliada.
- Inclua a identificação do grupo (nome completo e RA de cada integrante) no vídeo.
- Use a sua solução na explicação do vídeo (ex. mostrar conversão e avaliação da expressão).
- Você pode usar quaisquer recursos que achar necessário para o vídeo, tais como animações, slides, captura do projeto rodando etc.
- Certifique-se que o vídeo esteja em qualidade alta, que todo o conteúdo seja legível e que todos os integrantes do grupo participem no vídeo (não é obrigatório aparecer no vídeo, mas cada pessoa deve se apresentar – ainda que somente por áudio – e explicar uma parte da solução).

Entrega

- **Código**: Compacte o código-fonte (somente arquivos *.java) no **formato zip**.
  No arquivo zip, inclua um **arquivo texto** (.txt) contendo a identificação do grupo e o link do vídeo disponibilizado em algum serviço online (veja o item a seguir).

- **Vídeo**: O vídeo pode ser enviado para o Youtube ou algum outro serviço online e não precisa ser público (pode ficar como não listado) OU, se o arquivo de vídeo tiver até 30MB, pode ser anexado no Moodle.

Atenção: O arquivo zip não deve conter arquivos intermediários e/ou pastas geradas pelo compilador/IDE (ex. arquivos *.class, etc.).

Prazo de entrega: via link do Moodle até 05/04/2024 23:59.

Critérios de avaliação

A nota da atividade é calculada de acordo com os critérios da tabela a seguir.

Item avaliado	Pontuação máxima
Implementação: Leitura da expressão matemática na notação infixa.	até 0,5 pontos
Implementação: Leitura dos valores numéricos de cada variável.	até 0,5 pontos
Implementação: Conversão da expressão para notação posfixa.	até 2,0 pontos
Implementação: Avaliação da expressão (cálculo e apresentação do resultado da expressão).	até 2,0 pontos
Implementação: Validação das entradas do usuário.	até 1,0 ponto
Vídeo explicando a solução do problema.	até 4,0 pontos

Tabela 1 - Critérios de avaliação.

A tabela a seguir contém critérios de avaliação que podem **reduzir** a nota final da atividade.

Item indesejável	Redução de nota
O projeto é cópia de outro projeto.	Projeto é zerado
O projeto usa a classe Stack oferecida pela linguagem Java.	Projeto é zerado
Há erros de compilação e/ou o programa trava durante a execução.[1]	-1,0 ponto
Não há identificação do grupo (código-fonte e vídeo). Não há indicação de referências (código-fonte e vídeo). Arquivos enviados em formatos incorretos. Arquivos e/ou pastas intermediárias que são criadas no processo de compilação foram enviadas junto com o código-fonte. O vídeo possui mais de 7 (sete) minutos.	-1,0 ponto

Tabela 2 - Critérios de avaliação (redução de nota).

O código-fonte será compilado com o compilador javac (21.0.2) na plataforma Windows da seguinte forma:
> javac *.java -encoding utf8

O código compilado será executado com java (21.0.2) na plataforma Windows da seguinte forma:
> java <Classe>

Sendo que **<Classe>** deve ser substituído pelo nome da classe que contém o método *public static void main(String[] args)*.

[1] Sobre erros de compilação: considere apenas erros. Não há problema se o projeto tiver *warnings* (embora *warnings* podem avisar sobre possíveis travamentos em tempo de execução, como loop infinito, divisão por zero, etc.).