Centro de Ciências Exatas, Ambientais e de Tecnologias

Faculdade de Análise de Sistemas

Curso Sistemas de Informação

Projeto Integrado C

1º Trabalho do 1º Semestre de 2018

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

Entrada: ((T ∨ F → T) ∧ (T → F)) → (~F → T) Saída: T

Para solucionar o problema devido a mudança de prioridade pelo parêntese, 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. (A + B)
- Pré-Fixa: Operador precede os operandos (+ AB)
- Pós-Fixa: Operador após os operandos (AB +)

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. Veja o exemplo:

Entrada (Notação Infix): 3 + 4

Passos:

1) Leve 3 para a fila de saída (sempre que um número é lido é empurrado para a fila de saída)

Fila de Saída

Início | 3 | Fim

2) Leve + (ou sua identificação) na pilha do operador

Pilha de Operador

| + | ß Top

3) Leve 4 para a fila de saída

Fila de Saída

Início | 3 | 4 | Fim

4) Depois de ler a expressão, retire os operadores da pilha e adicione-os à fila de saída. Neste caso, existe apenas um, "+".

Fila de Saída

Início | 3 | 4 | + | Fim

Saída (Notação Posfix): 3 4 +

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.

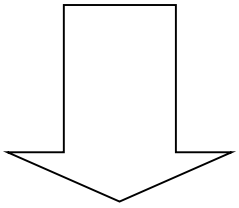
O programa que vamos implementar não uma calculadora aritmética, e sim uma **calculadora lógica**. Para você entender melhor como funciona uma calculadora lógica, sugiro que você visite o site http://www.calculadoraonline.com.br/tabela-verdade . Neste site você encontra uma calculadora lógica online. Inclusive, você poderá utilizar esta calculadora lógica para testar a sua calculadora.

Para o funcionamento da nossa calculadora, os seguintes símbolos poderão ser utilizados como entrada:

Símbolo	Significado	Funcionamento
T	Verdadeiro	---
F	Falso	---
^	Conjunção (operador AND)	A conjunção é verdadeira se e somente se os operandos são verdadeiros.
v	Disjunção (operador OR)	A disjunção é falsa se, e somente se ambos os operandos forem falsos.
~	Negação (operador NOT)	A negação da proposição "V" é a proposição "~V", ou seja, é falsa, e vice-versa.
->	Implicação	A conjunção é falsa se, e somente se, o primeiro operando é verdadeiro e o segundo operando é falso.
<->	Equivalência	A conjunção é verdadeira se, e somente se, ambos operandos forem falsos ou ambos verdadeiros
( )	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
~	Negação (operador NOT)
^	Conjunção (operador AND)
v	Disjunção (operador OR)
-> <->	Implicação Equivalência



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

Para implementar esta calculadora, seu programa deve realizar uma sequência de passos, a saber:

1. Você deverá nesta primeira etapa, construir o **Validador de Expressão**. Uma *string* deverá ser lida e quebrada em pedaços para verificar se é válida ou não:
   1.1. Solicitar a digitação de uma expressão lógica. Você deve ler a expressão digitada pelo usuário como uma *string* e remover todos os espaços em branco. Suponha que tenha sido:

   ((T ∨ F -> T) ∧ (T -> F)) -> (~F <-> T)

   1.2. Os símbolos de operadores da string que são compostos por mais de um caracter, você deverá substituir por um único símbolo, conforme tabela abaixo:

	DE	PARA
Implicação	->	-
Equivalência	<->	<

   Veja como ficará a *string* do nosso exemplo:

   ((T∨F-T)∧(T-F))-(~F<T)

   1.3. Agora que você já tem cada operador lógico representado por um único caracter, 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 (“~^v-<”). O true faz com que pegue os delimitadores também como pedaços.

```
StringTokenizer quebrador = new StringTokenizer (exp, "~^v-<", true);
```

1.4. 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 tokens`**

1.5. Por fim, você deverá verificar se a expressão lógica é válida. Caso não seja uma expressão válida, notificar o usuário. Caso seja válida, prosseguir para a segunda etapa.

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 **valor lógico** (T ou F), colocaremos este pedaço na Fila de Saída.

2.1.3. Se o pedaço for um **operador lógico** (~, ^, v, -, <), 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 1 (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 0 (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 0, ou seja, o passo 2.1.3.1 deverá ser repetido enquanto for encontrado o valor 1 (true) na tabela.

		Símbolo pego na Expressão						
		(	~	^	v	-	<	)
Símbolo que está no topo da Pilha de Operadores	(	0	0	0	0	0	0	1
	~	0	0	1	1	1	1	1
	^	0	0	1	1	1	1	1
	v	0	0	0	1	1	1	1
	-	0	0	0	0	1	1	1
	<	0	0	0	0	1	1	1
	)	0	0	0	0	0	0	0

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. Veja que ao final do passo 2.1, 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
((TvF-T)∧(T-F))-(~F<T)	Vazia	Vazia
(TvF-T)∧(T-F))-(~F<T)	(	Vazia
TvF-T)∧(T-F))-(~F<T)	((	Vazia
vF-T)∧(T-F))-(~F<T)	((	T
F-T)∧(T-F))-(~F<T)	v ((	T
-T)∧(T-F))-(~F<T)	v ((	T F
-T)∧(T-F))-(~F<T)	((Valor na tabela 1, remove v da Pilha de Operadores e coloca na Fila de Saída.	T F v
T)∧(T-F))-(~F<T)	- ((Não há mais elementos da Pilha de Operadores para remover. Empilha-se o operador que estava na expressão.	T F v
)∧(T-F))-(~F<T)	- ((	T F v T
)∧(T-F))-(~F<T)	((Desempilhamos os elementos da Pilha de Operadores e colocamos na Fila de Saída até encontrar o “(”.	T F v T -
∧(T-F))-(~F<T)	(Ambos os símbolos “(” como o “)” não foram para a fila - foram descartados.	T F v T -

Expressão	Pilha de Operadores	Fila de Saída
(T-F))-(~F<T)	^ (	T F v T -
T-F))-(~F<T)	(^ (	T F v T -
-F))-(~F<T)	(^ (	T F v T - T
F))-(~F<T)	- (^ (	T F v T - T
))-(~F<T)	- (^ (	T F v T - T F
))-(~F<T)	(^ (Desempilhamos os elementos da Pilha de Operadores e colocamos na Fila de Saída até encontrar o "(".	T F v T - T F -
)-(~F<T)	^ (Ambos os símbolos "(" como o ")" não foram para a fila - foram descartados.	T F v T - T F -
)-(~F<T)	(Desempilhamos os elementos da Pilha de Operadores e colocamos na Fila de Saída até encontrar o "(".	T F v T - T F - ^
-(~F<T)	Vazia Ambos os símbolos "(" como o ")" não foram para a fila - foram descartados.	T F v T - T F - ^
(~F<T)	-	T F v T - T F - ^

Expressão	Pilha de Operadores	Fila de Saída
~F<T)	(-	T F v T - T F - ^
F<T)	~ (-	T F v T - T F - ^
<T)	~ (-	T F v T - T F - ^ F
<T)	(- Valor na tabela 1, remove ~ da Pilha de Operadores e coloca na Fila de Saída.	T F v T - T F - ^ F ~
T)	< (- Não há mais elementos da Pilha de Operadores para remover. Empilha-se o operador que estava na expressão.	T F v T - T F - ^ F ~
)	< (-	T F v T - T F - ^ F ~ T
)	(- Desempilhamos os elementos da Pilha de Operadores e colocamos na Fila de Saída até encontrar o “(”.	T F v T - T F - ^ F ~ T <
Vazia	- Ambos os símbolos “(” como o “)” não foram para a fila - foram descartados.	T F v T - T F - ^ F ~ T <
Vazia	Vazia	T F v T - T F - ^ F ~ T < - Resultado Final: Expressão em notação pós-fixa.

3. Você deverá nesta terceira etapa, construir **Calculadora de Expressão Lógica**:

   3.1. Devemos definir 3 variáveis: dois valores lógicos (v1 e v2) e um char (opl). 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 lógico (T ou F), devemos empilhá-lo na Pilha Resultado. Devemos repetir este passo 1 até que o elemento removido da Fila de Saída seja um operador lógico.

      3.2.2. Se o elemento removido da Fila de Saída for um operador lógico, devemos armazená-lo na variável char definida como **opl**.

      3.2.3. Não pare por aí...ao encontrar um operador lógico e depois de armazená-lo em opl, devemos desempilhar um elemento e colocá-lo em v2. Em seguida, devemos fazer mais uma verificação:

         3.2.3.1. Se o operador lógico opl não for do tipo negação (~), devemos desempilhar mais um elemento 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.

         3.2.3.2. Se o operador lógico opl for do tipo negação (~), você deve calcular o resultado da expressão: **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**
T F v T - T F - ^ F ~ T < -	Vazia	v1 = v2 = opl =
F v T - T F - ^ F ~ T < -	T	v1 = v2 = opl =
v T - T F - ^ F ~ T < -	F T	v1 = v2 = opl =
T - T F - ^ F ~ T < -	F T	v1 = v2 = opl = v
T - T F - ^ F ~ T < -	Vazia	v1 = T v2 = F opl = v Calcula-se a expressão T v F = T e seu resultado empilha na Pilha Resultado.
T - T F - ^ F ~ T < -	T	v1 = v2 = opl =

Fila de Saída (expressão em notação pós-fixa)	Pilha Resultado	Variáveis
- T F - ^ F ~ T < -	T T	v1 = v2 = opl =
T F - ^ F ~ T < -	T T	v1 = v2 = opl = -
T F - ^ F ~ T < -	Vazia	v1 = T v2 = T opl = -
T F - ^ F ~ T < -	Vazia	v1 = T v2 = T opl = - Calcula-se a expressão T -> T = T e seu resultado empilha na Pilha Resultado.
T F - ^ F ~ T < -	T	v1 = v2 = opl =
F - ^ F ~ T < -	T T	v1 = v2 = opl =
- ^ F ~ T < -	F T T	v1 = v2 = opl =
^ F ~ T < -	F T T	v1 = v2 = opl = -
^ F ~ T < -	T	v1 = T v2 = F opl = -
^ F ~ T < -	T	v1 = T v2 = F opl = - Calcula-se a expressão T -> F = F e seu resultado empilha na Pilha Resultado.
^ F ~ T < -	F T	v1 = v2 = opl =
F ~ T < -	F T	v1 = v2 = opl = ^

Fila de Saída (expressão em notação pós-fixa)	Pilha Resultado	Variáveis
F ~ T < -	Vazia	v1 = T v2 = F opl = ^
F ~ T < -	Vazia	v1 = T v2 = F opl = ^ Calcula-se a expressão T ^ F = F e seu resultado empilha na Pilha Resultado
F ~ T < -	F	v1 = v2 = opl =
~ T < -	F F	v1 = v2 = opl =
T < -	F F	v1 = v2 = opl = ~
T < -	F	v1 = v2 = F opl = ~
T < -	F	v1 = v2 = F opl = ~ Calcula-se a expressão ~F = T e seu resultado empilha na Pilha Resultado
T < -	T F	v1 = v2 = opl =
< -	T T F	v1 = v2 = opl =
-	T T F	v1 = v2 = opl = <

Fila de Saída (expressão em notação pós-fixa)	Pilha Resultado	Variáveis
-	F	v1 = T v2 = T opl = <
-	F	v1 = T v2 = T opl = < Calcula-se a expressão T <-> T = T e seu resultado empilha na Pilha Resultado.
-	T F	v1 = v2 = opl =
Vazia	T F	v1 = v2 = opl = -
Vazia	Vazia	v1 = F v2 = T opl = -
Vazia	Vazia	v1 = F v2 = T opl = - Calcula-se a expressão F -> T = T e seu resultado empilha na Pilha Resultado.
Vazia	T Este é o resultado da expressão que deverá ser apresentado ao usuário como saída do programa.	v1 = v2 = opl =

Observações Finais:

- 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 27 de Março de 2018 (esta é a data de entrega para ambas turmas 0101 e 0102).** Somente o representante do grupo é que poderá postar no seu escaninho o projeto.

Bom trabalho! **Profs André Carvalho, Daniele Frosoni & Patrícia Nogueira** **Campinas, 27/Fev/2018**