**Introdução**

Neste laboratório, você vai programar uma mini-calculadora usando a notação polonesa reversa (*Reverse Polish Notation* - RPN), também denominada notação pós-fixa, como uma representação interna. RPN é uma maneira de escrever uma expressão que é particularmente útil quando se avalia a expressão. O processo de avaliação de uma expressão em RPN ilustra a utilidade da pilha como uma estrutura de dados.

Uma expressão é uma sequência de símbolos. Neste laboratório, estaremos lidando com expressões aritméticas. Assim, os símbolos utilizados nas expressões são dígitos (de 0 a 9), os operadores (+, -, *, /, %, ^, este último indicando exponenciação) bem como parênteses esquerdos e direitos (ou parênteses de abertura e de fechamento, respectivamente).

Na notação infixa, a qual estamos acostumados, os operadores são colocados entre seus operandos e os parênteses, quando utilizados, podem mudar a ordem como os operadores são aplicados. Neste formato, há uma relação de precedência especificada, onde ^ tem a maior precedência; em seguida, *, / e % têm mesma precedência; por fim + e -. No caso de operadores com mesma precedência, os operadores são aplicados da esquerda para a direita.

Observe na tabela seguinte que na notação RPN:

- Os operadores são colocados após os seus operandos
- Parênteses são desnecessários
- Os operandos encontram-se na mesma ordem que na notação infixa

| Notação Infixa | Notação Pós-fixa (RPN) |
|---|---|
| 2 + 3 | 2 3 + |
| 4 * (2 + 3) | 4 2 3 + * |
| ((3 + 5 * 1) / 8) * 7 | 3 5 1 * + 8 / 7 * |

**Algoritmo 1: Conversão da notação infixa para pós-fixa**

Dada uma expressão infixa representada pelos caracteres $U_1$, $U_2$, ..., $U_m$, onde $U_i$ é um operando (neste caso um valor inteiro), um operador ou um parêntese, o algoritmo seguinte gera a expressão em RPN (pós-fixa), fazendo uso de uma pilha de caracteres.

```
para i de 1 até m
    se Ui é um operando:              Transferir Ui para pós-fixa

    se Ui é um parêntese esquerdo:    Coloque Ui na pilha

    se Ui é um parêntese direito:     Retire caracteres da pilha e transfira-os para pós-
                                      fixa até encontrar um parêntese esquerdo. Retire o
                                      parêntese esquerdo da pilha e descarte-o.

    se Ui é um operador:              Assuma t como o elemento no topo da pilha.
                                      Transfira os elementos da pilha para pós-fixa até
                                      que:
                                         • precedência de t seja menor do que a
                                           precedência de Ui ou
                                         • t seja um parêntese esquerdo ou
                                         • a pilha esteja vazia
                                      Insira Ui na pilha
Transferir os símbolos restantes na pilha para pós-fixa
```

Exemplo:

| Infixa | Pilha (topo à direita) | Pós-fixa |
|---|---|---|
| ((3+5*1)/8)*7 | | |
| (3+5*1)/8)*7 | ( | |
| 3+5*1)/8)*7 | (( | |
| +5*1)/8)*7 | (( | 3 |
| 5*1)/8)*7 | ((+ | 3 |
| *1)/8)*7 | ((+ | 35 |
| 1)/8)*7 | ((+* | 35 |
| )/8)*7 | ((+* | 351 |
| /8)*7 | ( | 351*+ |
| 8)*7 | (/ | 351*+ |
| )*7 | (/ | 351*+8 |
| *7 | | 351*+8/ |
| 7 | * | 351*+8/ |
| | * | 351*+8/7 |
| | | 351*+8/7* |

**Algoritmo 2: Avaliação de uma expressão em RPN**

Dada uma expressão na notação pós-fixa $V_1$, $V_2$, ..., $V_n$, onde Vi é um operando ou um operador o seguinte algoritmo avalia a expressão, fazendo o uso de uma pilha de inteiros:

```
para i de 1 até n
```

| | |
|---|---|
| se $V_i$ é um operando: | Coloque Vi na pilha |
| se $V_i$ é um operador: | Remova os dois elementos superiores da pilha e aplique o operador $V_i$ a ambos. Insira o resultado da operação na pilha |

```
Ao final, o resultado da avaliação da expressão encontra-se no topo da pilha
```

Caso a expressão pós-fixa seja vazia, assuma como zero o valor da expressão.

Exemplo: 351*+8/7*

| Pilha |
|---|
| 3 |
| 3 5 |
| 3 5 1 |
| 3 5 |
| 8 |
| 8 8 |
| 1 |
| 1 7 |
| 7 |

**Implementação**

O Algoritmo 1 deve ser implementado dentro do método **infixToPostfix()** da classe *Expression* escrita na linguagem C++, tendo o seguinte protótipo:

**void Expression::infixToPostfix()**

Este método deve, a partir do campo *infix* (da classe *Expression*) contendo uma expressão aritmética válida (ou seja, operadores e dígitos na notação infixa) convertê-lo na notação pós-fixa, colocando o resultado no campo *postfix* da classe. Espaços em branco no campo *infix* devem ser ignorados por este método e não devem ser colocados no campo *postfix*.

O Algoritmo 2 deve ser implementado dentro do método **evalPostfix()** da classe Expression escrita na linguagem C++, tendo o seguinte protótipo:

**void Expression::evalPostfix()**

Este método deve, a partir do campo *postfix* contendo uma expressão na notação pós-fixa, calcular seu valor, colocando-o no campo *value* da classe.

Sua implementação não deve fazer uso do comando "go to" (ou seja, utilize apenas programação estruturada). Juntamente com este documento, estão sendo disponibilizados os seguintes arquivos:

- StackTemplate.h (interface e implementação da classe Stack usando Templates)
- Expression.h, Expression.cpp (interface e implementação parcial da classe Expression)
- StudentEmptyTest.h (arquivo de teste, no formato da plataforma CxxTest)

Você pode incluir quaisquer subalgoritmos (funções, procedimentos ou métodos) que se fizerem necessários nestes arquivos, porém não remova ou altere os métodos já fornecidos (caso contrário, é possível que a plataforma CxxTest atribua pontuação menor que o máximo admissível ao seu trabalho, mesmo que ele esteja correto).

Exemplo de uso da classe *Expression*:

```
Expression e("4 * (2 + 3)");
cout << "Infix..: |" << e.getInfix()   << "|" << endl
     << "Postfix: |" << e.getPostfix() << "|" << endl
     << "Value..:  " << e.getValue()   << endl;
```

Saída:

```
Infix..: |4 * (2 + 3)|
Postfix: |423+*|
Value..: 20
```

**Submissão**

Submeta sua implementação no sistema Web-CAT, disponível em http://143.107.137.90:8080/WebCat. Este laboratório pode ser submetido individualmente ou em equipes de, no máximo, 2 alunos. Neste caso, informe o nome do seu companheiro de equipe ao submeter o arquivo compactado no Web-CAT.

Antes de submeter os arquivos necessários, coloque seu nome completo (no caso de equipes, os nomes dos 2 integrantes) em todos os arquivos sendo submetidos, na forma de comentário no início de cada arquivo (.h ou .cpp).

Compacte os seguintes arquivos em um único arquivo .zip (não utilize espaços no nome do arquivo compactado, nem adicione pastas/diretórios no arquivo compactado):

- StackTemplate.h (interface e implementação da classe Stack usando Templates)
- Expression.h, Expression.cpp (interface e implementação da classe Expression)
- StudentEmptyTest.h (arquivo de teste, no formato da plataforma CxxTest[1])

Respeite os nomes de arquivos e das classes. Submeta o arquivo compactado ao Web-CAT. Em caso de dúvida, procure o professor.

**Avaliação**

[1] Você pode inserir mais casos de teste neste arquivo, caso queira tenha interesse em testar com mais detalhes seu código. Consulte http://cxxtest.com/guide.html#testAssertions.

Embora esse seja um laboratório no qual não há nota associada na disciplina, observe que nos trabalhos futuros serão considerados os seguintes critérios (além dos já mencionados nas Disposições Gerais entregues no início do semestre):

- **Correção:** O programa faz o que foi solicitado? Faz tudo o que foi solicitado? Utiliza encapsulamento de informação? (i.e., acessa adequadamente os ADTs definidos?)
- **Eficiência: A**s operações são executadas da maneira mais eficiente para cada estrutura de dados? Evita código duplicado/redundante/não atingível?
- **Interface:** É simples de usar, genérico, prático, tolera os erros mais óbvios? O trabalho foi entregue dentro das especificações (um arquivo *.h* para cada *.cpp* implementando um ADT? Os arquivos estão em formato ZIP, com os nomes de arquivos solicitados)?
    - interface do programa;
    - implementação dos métodos;
- **Código fonte:** é claro (*layout*, espaçamento, organização em geral), nomes de variáveis são sugestivos, e há documentação/comentários apropriados no código? Faz uso de pré- e pós-condições? Quando aplicável, faz uso de sublagoritmos (funções, procedimentos ou métodos) adicionais que melhoram a legibilidade do(s) método(s) solicitado(s) sem comprometer sua eficiência (por exemplo, na notação assintótica **O(*n*),** onde ***n*** representa o tamanho da entrada?)