Determinação de expressões bem formadas contendo delimitadores aritméticos (parênteses, colchetes e chaves)

Neste laboratório você deve implementar o algoritmo fornecido a seguir para determinar se uma expressão está bem formada ou não em relação ao número e posição de delimitadores aritméticos (parênteses, colchetes e chaves), um uso clássico no qual pilhas são aplicadas. O objetivo é determinar se um conjunto de delimitadores de expressões aritméticas (incluem-se na definição os parênteses, colchetes e chaves) é "bem formado" ("equilibrado" ou "balanceado"). Isso significa que para uma expressão estar bem formada, um símbolo (delimitador) de fechamento deve coincidir com o último símbolo (delimitador) de abertura e todos os símbolos de delimitadores devem estar combinados quando a expressão de entrada for toda processada.

Considere a tabela a seguir mostrando expressões bem formadas *versus* expressões mal formadas:

Expressão bem formada	Expressão mal formada
`( x+y*z [h] )`	`( x a b [ )`
`( ) [ ] { }`	`( (`
`{ ( x+a*b/c-d ) ( ) }`	`{ a-b*c ) ( ) }`

Observe que, nesta definição, apenas são considerados os símbolos de delimitadores (os demais símbolos, tais como variáveis, operadores e espaços são ignorados). Por simplicidade, você pode assumir que as variáveis e operadores são representados por meio de um único caractere; entretanto, você notará que essa restrição não se aplica, de fato, utilizando o algoritmo dado a seguir.

Por que nos importamos sobre o equilíbrio de delimitadores? Porque o compilador de uma linguagem de programação procura delimitadores equilibrados em situações como:

- componentes aninhados (como comandos *while*, *if-then* e *for*). Por exemplo, os compiladores C/C++/Java verificam por correspondências de pares de {} para designar blocos de código. Se os pares não coincidem, um erro do compilador ocorrerá.
- expressões matemáticas, por exemplo: $a = b - \{(x + y) \times z + 3\}$
- notação de vetores e matrizes []
- chamadas de funções e definições de funções. Especificamente, os parâmetros devem ser cercados por "(" e ")" equilibrados na maioria das linguagens de programação.

O Algoritmo

Dada uma expressão contendo os delimitadores "()", "[]", e "{}" — como também símbolos representando variáveis, operadores, espaços, etc — determinar se essa expressão é bem formada no uso de delimitadores, utilizando o seguinte algoritmo em conjunto com uma pilha:

Analise a expressão, caractere por caractere. A cada passagem deste laço:

- Se o caractere é um símbolo de abertura (caracteres "(", "{", ou "["), coloque-o na pilha
- Se o caractere é um símbolo de fechamento (caracteres ")", "}", ou "]") então:
    - Verifique se a pilha está vazia. Se estiver, então não há nenhum símbolo de abertura correspondente. A expressão não é bem formada.
    - Caso contrário, retire o caractere do topo da pilha (que deve ser um símbolo de abertura) e compare-o com o caractere sob análise da expressão.
        - Se eles são complementares (um é de abertura e outro de fechamento) então a expressão ainda é equilibrada e laço deve ser continuado (o próximo caractere da expressão deve ser analisado).
        - Caso contrário, a expressão não é bem formada

Se o fim da expressão é atingido e a pilha está vazia então a expressão é bem formada; caso contrário será mal formada.

Implementação

O algoritmo deve ser implementado dentro do método **`checkBalancedBrackets()`** da classe Brackets escrita na linguagem C++, tendo o seguinte protótipo:

```
bool Brackets::checkBalancedBrackets()
```

Este método deve retornar *true* caso a expressão seja bem formada no uso de delimitadores; caso contrário, o método deve retornar *false*. 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:

- Stack.h, Stack.cpp (interface e implementação da classe Stack)
- Brackets.h, Brackets.cpp (interface e implementação parcial da classe Brackets)
- 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 qualquer dos 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).

Submissão

Submeta sua implementação no sistema Web-CAT, disponível em http://kode.ffclrp.usp.br:8080/WebCat. Compacte os seguintes arquivos em um único arquivo .zip (não utilize espaços no nome do arquivo compactado, nem adicione pastas/diretórios no arquivo compactado):

- Stack.h, Stack.cpp (interface e implementação da classe Stack)
- Brackets.h, Brackets.cpp (interface e implementação da classe Brackets)
- 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

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?)

[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.