1) Uma utilização prática de pilhas poderia ser para solucionar o problema de **examinar expressões matemáticas** que contém vários conjuntos de parênteses agrupados. Exemplo:

7 – ((x * ((x + y) / (j – 3)) + y) / (4 – 2.5))

É necessário garantir que os parênteses estejam corretamente agrupados, ou seja, deseja-se verificar que:

a) Existe um número igual de parênteses esquerdos e direitos?

b) Todo parênteses da direita está precedido por um parênteses da esquerda correspondente?

Assim, expressões como: “**((a + b) ou a + b(**“, violam o critério 1, e expressões como: “**)a + b(-c ou (a + b)) – (c + d**”, violam o critério 2.

<u>Para solucionar esse problema, pode-se imaginar cada parênteses da esquerda como uma abertura de um escopo, e cada parênteses da direita como um fechamento de escopo.</u> **A profundidade do agrupamento em determinado ponto numa expressão é o número de escopos abertos, mas ainda não fechados nesse ponto.** Isto corresponderá ao número de parênteses da esquerda encontrados cujos correspondentes parênteses da direita ainda não foram encontrados. Determina-se a contagem de parênteses em determinado ponto numa expressão como o número de parênteses da esquerda menos o número de parênteses da direita encontrado ao rastrear a expressão a partir do início até o ponto em questão. Se a contagem de parênteses for não-negativa, ela equivalerá a profundidade de aninhamento. As duas condições que devem vigorar caso os parênteses de uma expressão formem um padrão admissível são:

a) **A contagem de parênteses no final da expressão é 0**. Isso implica que nenhum escopo ficou aberto ou que foi encontrada a mesma quantidade de parênteses da direita e da esquerda.

b) **A contagem de parênteses em cada ponto da expressão é não-negativa.** Isso implica que não foi encontrado um parênteses da direita para o qual não exista um correspondente parênteses da esquerda.

Alterando ligeiramente o problema e supondo a existência de três tipos diferentes de delimitadores de escopo. Esses tipos são indicados por **parênteses** (e), **colchetes** [e] e **chaves** {e}. Um finalizador de escopo deve ser do mesmo tipo de seu iniciador. Sendo assim, expressões como **(a + b], [(a + b]), {a – (b]}**, são inválidas.

É necessário rastrear não somente quantos escopos foram abertos como também seus tipos. Estas informações são importantes porque, quando um finalizador de escopo é encontrado, precisa-se conhecer o símbolo com o qual o escopo foi aberto para assegurar que ele seja corretamente fechado.

Uma pilha pode ser usada para rastrear os tipos de escopos encontrados. Sempre que um iniciador de escopo for encontrado, ele será **empilhado**. Sempre que um finalizador de escopo for encontrado, a pilha será examinada. Se a pilha estiver vazia, o finalizador de escopo não terá um iniciador correspondente e a *string* será, consequentemente, inválida. Entretanto se a pilha não estiver vazia, **desempilha** e verifica se o item desempilhado corresponde ao finalizador de escopo. Se ocorrer uma coincidência, o processo continua, caso contrária, a expressão é inválida. Quando o final da expressão for encontrada, a pilha deverá estar vazia. **Caso contrário, existem um ou mais escopos abertos que ainda não foram fechados, e a expressão será inválida**.

2) Simule a ação do algoritmo apresentado no exercício 1 para cada uma das seguintes *strings*, apresentando o conteúdo da pilha em cada ponto.

a) (A + B}) = Inválida

b) {[A + B] - [(C - D)] = Inválida

c) (A + B)-{C + D}-[F+ G] = Válida

d) ((H) * {([J + K])}) = Válida

e) (((A)))) = Inválida

Todos esses testes estão comentados na main() do arquivo Expressões_matematicas.c++;