1 Avaliação de expressões utilizando pilhas

Existem três tipos de notação para representar uma expressão matemática. A representação mais conhecida é denominada **notação infixa**, onde o operador aparece entre os dois operandos. Por exemplo, considere o operador de adição + e os operandos A e B. A notação infixa da expressão, que aplica o operador + sobre os operandos A e B, é $A+B$. Essa mesma expressão pode ser representada ainda por outras duas notações alternativas. São elas:

- **notação prefixa:** $+AB$
- **notação posfixa:** $AB+$

Os prefixos **in**, **pre** e **pos** referem-se a posição relativa do operador em relação aos dois operandos. As regras para transformação de uma expressão infixa com parênteses, para prefixa e posfixa, são:

- converter primeiro as operações de precedência mais alta;
- tratar a parte convertida como único operando.

Considere a expressão $A + B * C$, em notação infixa, e a precedência natural do operador $*$ sobre o operador +. Logo, esta expressão é interpretada como $A + (B * C)$ (com parênteses). Convertendo a expressão infixa para as notações prefixa e posfixa temos:

- **conversão para prefixa:** $A + (B * C) \quad \Rightarrow \quad +A(*BC) \quad \Rightarrow \quad +A * BC$
- **conversão para posfixa:** $A + (B * C) \quad \Rightarrow \quad A(BC*)+ \quad \Rightarrow \quad ABC * +$

Considere agora a seguinte ordem de precedência dos operadores:

- **exponenciação:** \$
- **multiplicação e divisão:** $*$ e /
- **adição e subtração:** + e −

onde o operador \$ tem maior precedência e os operadores + e − tem menor precedência.

Usando as regras de precedência acima, é possível realizar a transformação de expressões na notação infixa para as notações posfixa e prefixa. As expressões convertidas podem ser avaliadas por algoritmos mais simples se comparados a algoritmos para avaliação de expressões na notação infixa, no entanto, os algoritmos para conversão tendem a ser mais complicados. Nas seções seguintes são apresentados os algoritmos para conversão e avaliação de expressões utilizando o conceito de pilhas.

1.1 Conversão da notação infixa para posfixa

O procedimento converterInfixaPosfixa() utiliza o conceito de pilhas para realizar a conversão de uma expressão infixa para a notação posfixa. O algoritmo mantém uma pilha para os operadores e parênteses (`PilhaA`) e um vetor ou lista de símbolos para armazenar a expressão convertida para a notação posfixa (`ListaB`).

Considere a expressão abaixo, na notação infixa, e execute os passos definidos pelo algoritmo converterInfixaPosfixa() para realizar a conversão para posfixa. Em seguida, verifique se o resultado obtido está correto.

- **Infixa:** $A\$B * C - D + E/F/(G + H)$
- **Posfixa:** $AB\$C * D - EF/GH + /+$

Observe que os operandos na expressão posfixa correspondente aparecem na mesma ordem que na expressão infixa. Observe também que a precedência dos operadores tem papel importante nestas transformações.

Algoritmo 1: converterInfixaPosfixa

```
1  início
2      repita
3          Obter o próximo símbolo da string de entrada, a partir da esquerda.
4          se símbolo for operando então
5              Inserir o símbolo na ListaB (string posfixa), a partir da esquerda.
6          senão
7              se símbolo for abre parênteses então
8                  Empilhar o símbolo na PilhaA.
9              senão
10                 se símbolo for operador então
11                     se precedência do operador ≤ operador no topo da PilhaA então
12                         repita
13                             Desempilhar o símbolo da PilhaA (operador).
14                             Inserir na ListaB.
15                         até PilhaA estar vazia ou precedência ser maior
16                     Empilhar o símbolo na PilhaA.
17                 senão
18                     se símbolo for fecha parênteses então
19                         repita
20                             Desempilhar o símbolo da PilhaA (operador).
21                             Inserir na ListaB.
22                         até símbolo ser um abre parênteses
23                         Descartar os símbolos abre e fecha parênteses.
24     até não haver mais símbolos na string
25     repita
26         Desempilhar o símbolo da PilhaA (operador)
27         Inserir na ListaB
28     até PilhaA estar vazia
```

1.2 Conversão da notação infixa para prefixa

O procedimento converterInfixaPrefixa() utiliza o conceito de pilhas para realizar a conversão de uma expressão infixa para a notação prefixa. O algoritmo mantém uma pilha para os operadores e parênteses (`PilhaA`) e um vetor ou lista de símbolos para armazenar a expressão convertida para a notação prefixa (`ListaB`).

Considere a expressão abaixo, na notação infixa, e execute os passos definidos pelo algoritmo converterInfixaPrefixa() para realizar a conversão para prefixa. Em seguida, verifique se o resultado obtido está correto.

- **Infixa:** $A\$B * C - D + E/F/(G + H)$
- **Prefixa:** $+ - *\$ABCD//EF + GH$

Observe que os operandos na expressão prefixa correspondente aparecem na mesma ordem que na expressão infixa. Observe também que a precedência dos operadores tem papel importante nestas transformações.

Algoritmo 2: converterInfixaPrefixa

```
1  início
2      repita
3          Obter o próximo símbolo da string de entrada, a partir da direita.
4          se símbolo for operando então
5              Inserir o símbolo na ListaB (string prefixa), a partir da direita.
6          senão
7              se símbolo for fecha parênteses então
8                  Empilhar o símbolo na PilhaA.
9              senão
10                 se símbolo for operador então
11                     se precedência do operador < operador no topo da PilhaA então
12                         repita
13                             Desempilhar o símbolo da PilhaA (operador).
14                             Inserir na ListaB.
15                         até PilhaA estar vazia ou precedência ser maior ou igual
16                     Empilhar o símbolo na PilhaA.
17                 senão
18                     se símbolo for abre parênteses então
19                         repita
20                             Desempilhar o símbolo da PilhaA (operador).
21                             Inserir na ListaB.
22                         até símbolo ser um fecha parênteses
23                         Descartar os símbolos abre e fecha parênteses.
24     até não haver mais símbolos na string
25     repita
26         Desempilhar o símbolo da PilhaA (operador)
27         Inserir na ListaB
28     até PilhaA estar vazia
```

1.3 Avaliação de expressões na notação posfixa

Cada operador em uma *string* posfixa refere-se aos dois operandos anteriores. Na avaliação de uma expressão posfixa, o operando no topo da pilha é o segundo operando (op2) na operação.

Considere a expressão $623+-382/+*2\$3+$, na notação posfixa, e execute os passos do algoritmo avaliarExpressaoPosfixa() para avaliar a expressão. O resultado é 52.

Algoritmo 3: avaliarExpressaoPosfixa

```
1  início
2      repita
3          Obter o próximo símbolo da string de entrada, a partir da esquerda.
4          se símbolo for operando então
5              Empilhar o símbolo em PilhaA.
6          senão
7              se símbolo for operador então
8                  Desempilhar o operador do topo da PilhaA (op2).
9                  Desempilhar o operador do topo da PilhaA (op1).
10                 Efetuar operação: op1 <operador> op2
11                 Empilhar o resultado da operação na PilhaA.
12     até não haver mais símbolos na string
13     Imprimir o resultado da expressão.
```

1.4 Avaliação de expressões na notação prefixa

Cada operador em um *string* prefixa refere-se aos dois operandos posteriores. Na avaliação de uma expressão prefixa, o operando no topo da pilha é o primeiro operando (op1) na operação.

Considere a expressão $+*+34+5*237$, na notação prefixa, e execute os passos do algoritmo avaliarExpressaoPrefixa() para avaliar a expressão. O resultado é 84.

Algoritmo 4: avaliarExpressaoPrefixa

```
1  início
2      repita
3          Obter o próximo símbolo da string de entrada, a partir da direita.
4          se símbolo for operando então
5              Empilhar o símbolo em PilhaA.
6          senão
7              se símbolo for operador então
8                  Desempilhar o operador do topo da PilhaA (op1).
9                  Desempilhar o operador do topo da PilhaA (op2).
10                 Efetuar operação: op1 <operador> op2
11                 Empilhar o resultado da operação na PilhaA.
12     até não haver mais símbolos na string
13     Imprimir o resultado da expressão.
```

Referências

[1] Sdnei de Brito Alves, *Estruturas de Dados e Algoritmos com Aplicações em C*, 2004.