EXPRESSÃO PÓS -FIXA

Participantes:

Jorge Afonso Rabelo de Araujo

Otávio de Queiroz Franco

João Antônio Coelho dos Reis

SUMÁRIO

INTRODUÇÃO

- Em matemática e programação, expressões aritméticas podem ser escritas de diferentes formas.
- A notação infixa (ex: A + B) é a mais comum, mas requer o uso de regras de precedência e parênteses para definir a ordem das operações.
- A notação pós-fixa (ex: AB+), ou Notação Polonesa Reversa (RPN), elimina a necessidade de parênteses, tornando a avaliação mais direta e simples.
- Neste conteúdo, vamos entender como converter uma expressão infixa para pós-fixa e como resolver essa expressão pós-fixa usando pilhas.

EXPRESSÕES

INFIXA

- Operadores entre operandos (ex: A + B)
- Ordem das operações depende da precedência e parênteses
- Ex: (1+2)*3

PÓSFIXA

- Operadores vêm após operandos (ex: AB+)
- Ordem de execução definida pela posição dos operadores
- Não usa parênteses ou precedência explícita
- Ex: 1 2 + 3 *

CONVERSÃO DE INFIXA PARA PÓS-FIXA

1. Inicialize uma pilha (para operadores).
2. Varra cada caractere da expressão infixa:
    a. Operando → adicione ao resultado.
    b. Operador → desempilhe operadores de maior/igual precedência, depois empilhe o atual.
    c. ‘(’ → empilhe.
    d. ‘)’ → desempilhe até ‘(’.
3. Desempilhe operadores restantes ao final.

EXEMPLO

Infixa: 5+6*7

PosFixa: 567*+

EXEMPLO

Infixa: (4+5)/6

PosFixa: 45+6*

AVALIAÇÃO DE EXPRESSÃO PÓS-FIXA

1. Inicialize uma pilha (para valores).
2. Varre cada caractere da expressão pós-fixa:
    a. Operando → empilhe.
    b. Operador → desempilhe dois operandos, aplique a operação e empilhe o resultado.
3. Resultado final está no topo da pilha.

EXEMPLO

1. Pós-fixa: 34+2*
2. Avaliação:
    a. 3 + 4 → empilha 7
    b. 7 * 2 → empilha 14
    c. Resultado: 14

CÓDIGO

```java
// Classe que representa uma pilha de operadores para a conversão infixa para pós-fixa
static class PilhaDeOperadores {  2 usages  new *
    int topo;  5 usages
    char[] elementos;  4 usages

    PilhaDeOperadores(int tamanho) {  1 usage  new *
        elementos = new char[tamanho];
        topo = -1;
    }

    boolean estaVazia() {  3 usages  new *
        return topo == -1;
    }

    void empilhar(char elemento) {  2 usages  new *
        elementos[++topo] = elemento;
    }

    char desempilhar() {  4 usages  new *
        return elementos[topo--];
    }

    char espiar() {  2 usages  new *
        return elementos[topo];
    }
}
```

CÓDIGO

```java
// Classe que representa uma pilha de números para a avaliação pós-fixa
static class PilhaDeNumeros {  2 usages  new *
    int topo;  4 usages
    int[] elementos;  3 usages

    PilhaDeNumeros(int tamanho) {  1 usage  new *
        elementos = new int[tamanho];
        topo = -1;
    }

    boolean estaVazia() {  no usages  new *
        return topo == -1;
    }

    void empilhar(int elemento) {  5 usages  new *
        elementos[++topo] = elemento;
    }

    int desempilhar() {  3 usages  new *
        return elementos[topo--];
    }
}
```

CÓDIGO

```java
// Função que converte expressão infixa para pós-fixa
static String converterParaPosfixa(String expressaoInfixa) {  1 usage  new *
    StringBuilder expressaoPosfixa = new StringBuilder();
    PilhaDeOperadores pilhaDeOperadores = new PilhaDeOperadores(expressaoInfixa.length());

    // Percorre cada caractere da expressão
    for (int i = 0; i < expressaoInfixa.length(); i++) {
        char caractereAtual = expressaoInfixa.charAt(i);

        // Ignora espaços em branco
        if (caractereAtual == ' ') continue;

        // Se for um número ou letra, adiciona diretamente ao resultado
        if (Character.isLetterOrDigit(caractereAtual)) {
            expressaoPosfixa.append(caractereAtual);
        }
        // Se for um parêntese de abertura, empilha
        else if (caractereAtual == '(') {
            pilhaDeOperadores.empilhar(caractereAtual);
        }
        // Se for um parêntese de fechamento, desempilha até encontrar o parêntese de abertura
        else if (caractereAtual == ')') {
            while (!pilhaDeOperadores.estaVazia() && pilhaDeOperadores.espiar() != '(') {
                expressaoPosfixa.append(pilhaDeOperadores.desempilhar());
            }
            pilhaDeOperadores.desempilhar(); // Remove o '(' da pilha
        }
        // Se for um operador
        else {
            while (!pilhaDeOperadores.estaVazia() &&
                    obterPrecedencia(caractereAtual) <= obterPrecedencia(pilhaDeOperadores.espiar())) {
                expressaoPosfixa.append(pilhaDeOperadores.desempilhar());
            }
            pilhaDeOperadores.empilhar(caractereAtual); // Empilha o operador atual
        }
    }
```

CÓDIGO

```java
// Função para avaliar uma expressão pós-fixa
static int avaliarPosfixa(String expressaoPosfixa) {  1 usage  new *
    PilhaDeNumeros pilhaDeNumeros = new PilhaDeNumeros(expressaoPosfixa.length());

    // Percorre cada caractere da expressão pós-fixa
    for (int i = 0; i < expressaoPosfixa.length(); i++) {
        char caractereAtual = expressaoPosfixa.charAt(i);

        // Se for um número, empilha
        if (Character.isDigit(caractereAtual)) {
            pilhaDeNumeros.empilhar( elemento: caractereAtual - '0'); // Converte o caractere para número
        } else {
            // Se for um operador, desempilha dois números e realiza a operação
            int valor2 = pilhaDeNumeros.desempilhar();
            int valor1 = pilhaDeNumeros.desempilhar();

            switch (caractereAtual) {
                case '+':
                    pilhaDeNumeros.empilhar( elemento: valor1 + valor2);
                    break;
                case '-':
                    pilhaDeNumeros.empilhar( elemento: valor1 - valor2);
                    break;
                case '*':
                    pilhaDeNumeros.empilhar( elemento: valor1 * valor2);
                    break;
                case '/':
                    pilhaDeNumeros.empilhar( elemento: valor1 / valor2);
                    break;
            }
        }
    }
```

CÓDIGO

```java
// Função que define a precedência dos operadores
static int obterPrecedencia(char operador) {  2 usages  new *
    switch (operador) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '(':
            return 0;
    }
    return -1; // Retorna -1 se o caractere não for um operador conhecido
}
```

CONCLUSÃO

Conversão de infixa para pós-fixa organiza a ordem das operações.

Avaliação da pós-fixa elimina a necessidade de precedência e parênteses.

Simples e eficiente com uso de pilhas!


OBRIGADO