PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
PUCRS – CIÊNCIAS DA COMPUTAÇÃO

ÉRICK CONDE, GUSTAVO WILLIAN

TRABALHO 2 – MANIPULANDO ARQUIVOS E EXPRESSÕES

Algoritmos e Estrutura de dados I

Porto Alegre, RS - Brasil
2022

1. INTRODUÇÃO

Este trabalho consiste no desenvolvimento de uma calculadora para resolver expressões aritméticas simples, levando em conta parênteses, colchetes e chaves, além de só utilizar números inteiros. Apenas as seguintes operações serão consideradas: soma, subtração, multiplicação, divisão e potenciação, respectivamente representadas pelos símbolos +, -, *, / e ^.

Com esse objetivo em vista, utilizamos a o conceito de pilha como um controlador para a ordem em que as operações e expressões serão executadas. E para isso decidimos que, assim que o programa encontrar um símbolo referente a algum fechamento de expressão (símbolos como “)”, “]” ou “}”), começa a calcular os valores até que a mesma expressão “abra”, ou seja, encontre seu par de abertura (respectivamente “(”, “[” e “{”), e assim conseguimos identificar os erros da expressão e extrair a resposta da operação.

2. DESENVOLVIMENTO

Através do código disponibilizado pela professora, chamamos a classe Resolução, que roda o código escrito afim de executar as operações a serem feitas. Os conceitos a seguir foram utilizados para construção do código:

PILHA ENCADEADA

Primeiramente foi criado a Classe Pilha Encadeada que funciona com o conceito de stack LIFO, “Last In, First Out” (o último que entra é o primeiro que sai).

- Os elementos da pilha são nodos, que armazenam a informação e um ponteiro para o próximo nodo;
- Elementos: no caso vamos utilizar objetos do tipo String como elementos;
- **Métodos:**
- Push: é usado para adicionar novos elementos na pilha (sempre sendo colocada no início da pilha), assim ela deve manter uma referência para o nodo que está uma posição à frente;
- Pop: remove o elemento do topo da pilha, retornando-o;
- Size: retorna o número de nodos na pilha, ou seja, seu tamanho;
- Clear: esvazia a pilha;

LISTA DE ARRANJOS

Foi utilizada para guardar a operação que está sendo executada pelo programa, para termos uma visibilidade de qual operação estamos trabalhando com. (ArrayList)

CLASSE RESOLUÇÃO

Sendo nossa classe que contém o método que irá resolver as operações e testar possíveis erros na expressão.

Primeiramente foi criado o método “fechado” (notação O(n)), que é utilizado como conversão da String para o seu par (assim podendo ser comparada o elemento da pilha). Toda vez, antes de diminuir a pilha, verifica se o tamanho da pilha naquele momento é maior que o de antes, se for atualizar o tamanho máximo alcançado pela pilha. Usamos o ArrayList com as Strings da operação e passamos adicionando na pilha até que apareça um símbolo de fechamento (“)” , “]”, “}”) , quando isso acontece, pega-se os últimos 4 elementos que entraram na pilha, utilizando o método “pop”, para então calcular as operações:

1. O primeiro elemento é convertido para double; (Se não for um valor numérico, mostra uma mensagem de erro e bloqueia as operações até a próxima expressão)
2. No segundo elemento, verificamos qual o sinal da operação;
3. O terceiro elemento é convertido para double; (Se não for um valor numérico, mostra uma mensagem de erro e bloqueia as operações até a próxima expressão)
4. Já no quarto elemento, verificamos se faz par com o que fecha; (Se não mostra uma mensagem de erro)

Se o código não for interrompido, nem retornar uma mensagem de erro, é calculado o resultado da operação. Quando a expressão acabar, e todas as operações tenham sido executadas, a expressão inteira é impressa no terminal, juntamente com o resultado final e o tamanho máximo alcançado pela a pilha.

MENSAGENS DE ERROS

As mensagens de erros foram feitas individualmente para cada caso; quando ocorre um erro na expressão, mostra uma mensagem de erro de acordo com o problema ocorrido, além de excluir aquela operação nas conversões. Quando a expressão acaba, zeramos todos os valores e começamos a nova expressão normalmente.

Casos:

1. **Erro de Conversão:** a primeira ou terceira String não é um valor numérico;
2. **Erro Abre/Fecha** (“( )” , “[ ]”, “{ }”)**:** verifica até onde o erro acontece, e quando o problema é alcançado, há duas situações:
    - Ou o erro aconteceu por uma síntaxe abriu mas não fechou;
    - Ou ela abriu e fechou com outra de outro par;
3. **Erro sem Operador:** caso a expressão não contenha um operador;

3. RESULTADOS

```
--- Inicio expressao

{(5+12)+[(10-8)+2]}
Resultado: 21.0
Tamanho máximo da pilha: 8

--- Fim expressao


--- Inicio expressao

{(2+3)*[3/(1-3)]}
Resultado: -7.5
Tamanho máximo da pilha: 10

--- Fim expressao


--- Inicio expressao

Erro de sintaxe: ] no lugar de )
{(12+34)*[(47-17/(60-20)]}

--- Fim expressao


--- Inicio expressao

{[((27-18)*3)-((58+33)-((108-79)+2))]+[(5+12)+((10-8)+2)]}
Resultado: -12.0
Tamanho máximo da pilha: 12

--- Fim expressao
```

```
--- Inicio expressao

Erro de sintaxe: ] no lugar de )
{[[(27-18)*3]-[(58+33)-[(108-79]+2)]]+[(5+12)+((10-8)+2)]}

--- Fim expressao


--- Inicio expressao

{[((2^5)-(3*15))+((102+379)*(468-248))]-[((3^6)-(54*11))+((175/5)/(100-117))]}
Resultado: 105674.05882352941
Tamanho máximo da pilha: 13

--- Fim expressao


--- Inicio expressao

Erro de sintaxe: * no lugar de um inteiro
{[((2^5)-(3*15))+((102+379)*(468-248))]-[((3^6)-(54*))+((175/5)/(100-117))]}

--- Fim expressao


--- Inicio expressao

{[[[(4^4)-(13*15))+((123+456)*(987-654)))+(((3^6)-(2*34))+((242+353)*(468-248)))]-[[((2^5)-(3*15))+((102+379)*(468-248)))+(((2^5)-(3*
15))+((102+379)*(468/2)))]]}
Resultado: 106081.0
Tamanho máximo da pilha: 16

--- Fim expressao
```

```
--- Inicio expressao

Erro de sintaxe: abriu ( mas não fechou
{[[[(4^4)-(13*15))+((123+456)*(987-654)))+(((3^6)-(2*34))+((242+353)*(468-248)))]-[[((2^5)-(3*15))+((102+379)*(468-248)))+(((2^5)-(3*
15)+((102+379)*(468/2)))]}

--- Fim expressao
```

4. DIFICULDADES

- Especificar os problemas sem afetar operações futuras: para isso tivemos que fazer um booleano para especificar as entradas e saídas de dados no código principal.
- Utilizar a pilha de forma eficiente controlando a entrada e saída de dados, sendo feita sob medida para o devido problema.
- A quantidade de exceções fez com que problemas desencadeassem outros problemas, que foram corrigidos, retornando a resposta esperada

5. CONCLUSÃO

Esse trabalho foi, de fato, muito desafiador, mas por essa dificuldade, exercitamos nossa habilidade de encontrar e solucionar problemas. Com esse trabalho, pudemos observar de forma prática e interessante, o uso prático do conceito pilha e uma de suas possíveis utilizações, apresentando novas ideias e conceitos para futuras aplicações.

Outro ponto fundamental para esse trabalho, foi perceber a importância de criar um plano antes de começar a resolução, analisando os possíveis erros e assim obter a melhor eficiência possível na hora de programar.