Sumário

[1. Introdução: 2](#_Toc199173340)

[2. Implementação: 2](#_Toc199173341)

[2.1 Uso do TAD Pilha 2](#_Toc199173342)

[3. Testes 2](#_Toc199173343)

[3.1 Teste n° 01 2](#_Toc199173344)

[3.2 Teste n° 02 2](#_Toc199173345)

[3.3 Teste n° 03 3](#_Toc199173346)

[3.4 Teste n° 04 3](#_Toc199173347)

[3.5 Teste n° 05 3](#_Toc199173348)

[4. Conclusão 3](#_Toc199173349)

[Referências 3](#_Toc199173350)

[Anexos 4](#_Toc199173351)

[calculadora.h 4](#_Toc199173352)

[calculadora.c 4](#_Toc199173353)

[main.c 4](#_Toc199173354)

## Introdução:

Este trabalho prático tem como objetivo o desenvolvimento de um programa em linguagem C capaz de avaliar expressões matemáticas, com suporte a operações básicas e funções matemáticas especiais. O sistema também deverá ser capaz de converter e avaliar expressões entre as formas infixada e pós-fixada, utilizando pilhas como estrutura fundamental. Este relatório detalha a implementação, os testes realizados e a análise dos resultados, seguindo as diretrizes da Universidade Católica de Brasília (UCB) para a disciplina de Estrutura de Dados.

#### GitHub:

*(Na seção de introdução, disponibilizar também o endereço compartilhado do código-fonte disponibilizado no GitHub).*

## Implementação:

O arquivo ***expressao.h*** define as interfaces para as funcionalidades de manipulação e avaliação de expressões. As estruturas de dados fundamentais são as pilhas, essenciais para o processamento das expressões.

**Protótipos:**

* ***getFormaInFixa(char \*Str)***: Esta função recebe uma string contendo uma expressão na notação pós-fixada e retorna uma nova string com a expressão convertida para a notação infixada. A alocação de memória para a string de retorno é gerenciada internamente.
* ***getFormaPosFixa(char \*Str)***: Recebe uma string com uma expressão na notação infixada e retorna uma nova string com a expressão convertida para a notação pós-fixada. Similarmente, a memória é alocada pela função.
* ***getValorPosFixa(char \*StrPosFixa)***: Avalia o valor numérico de uma expressão fornecida na notação pós-fixada, retornando o resultado como um float.
* ***getValorInFixa(char \*StrInFixa)***: Avalia o valor numérico de uma expressão fornecida na   
  notação infixada. Internamente, esta função utiliza *getFormaPosFixa* para converter a expressão para pós-fixa antes de avaliá-la com *getValorPosFixa*.

Além dessas funções públicas, o arquivo ***expressao.c*** contém implementações de funções auxiliares e estruturas de pilha internas, como *CharStack* (para caracteres/tokens) e *FloatStack* (para valores numéricos), juntamente com suas operações básicas (*push*, *pop*, *isEmpty*, *isFull*, *top*). Também foram implementadas funções para tokenização, validação de expressões e operações matemáticas personalizadas (*myPow*, *mySqrt*, *myLog10*, *mySin*, *myCos*, *myTan*, *myMod*) para garantir a precisão e o controle sobre os cálculos.

### Uso do TAD Pilha

O Tipo Abstrato de Dados (TAD) Pilha é a estrutura fundamental para o funcionamento do avaliador de expressões numéricas, sendo empregado tanto nas conversões de notação quanto na avaliação. A lógica por trás de seu uso pode ser visualizada da seguinte forma:

**Conversão Infixa para Pós-fixa (Algoritmo Shunting-yard):**

1. **Entrada:** Expressão Infixa *(ex: A + B \* C)*
2. **Pilha de Operadores:** Utilizada para armazenar operadores e parênteses temporariamente.
3. **Fila de Saída:** Onde a expressão pós-fixa é construída.

* **Processo**:
  + Quando um **operando** (número ou variável) é lido, ele é imediatamente adicionado à fila de saída.
  + Quando um **operador** é lido, ele é comparado com o operador no topo da pilha. Operadores com maior precedência ou associatividade à direita são empilhados. Operadores com menor ou igual precedência (e associatividade à esquerda) são desempilhados da pilha e movidos para a fila de saída até que a condição seja satisfeita, e então o operador atual é empilhado.
  + **Parênteses**: O parêntese de abertura “(“ é empilhado. O parêntese de fechamento “)” faz com que todos os operadores da pilha, até o parêntese de abertura correspondente, sejam desempilhados e movidos para a fila de saída. O parêntese de abertura é então descartado.
  + Ao final da expressão, todos os operadores restantes na pilha são desempilhados e movidos para a fila de saída.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expressão Infixa:** 3 + 4 \* 5 | | | |
| **Token** | **Pilha de Operadores** | **Fila de Saída (Pós-fixa)** |
| 3 | [ ] | [3] |
| + | [+] | [3] |
| 4 | [+] | [3, 4] |
| \* | [+, \*] | [3, 4] (Precedência de \* > +) |
| 5 | [+, \*] | [3, 4, 5] |
| (FIM) | [ ] | [3, 4, 5, \*, +] |

**Avaliação de Expressões Pós-fixa:**

1. **Entrada:** Expressão Pós-fixa *(ex: 3 4 5 \* +)*
2. **Pilha de Operadores:** Utilizada para armazenar os valores numéricos.

* **Processo**:
  + Quando um **operando** é lido, ele é convertido para seu valor numérico e empilhado na pilha de operandos.
  + Quando um **operador** é lido, os operandos necessários (dois para operadores binários, um para funções unárias) são desempilhados. A operação é realizada com esses operandos, e o resultado é empilhado de volta.
  + Ao final da expressão, o único valor restante na pilha é o resultado final da avaliação

|  |  |
| --- | --- |
| **Expressão Pós-fixa:** 3 + 4 \* 5 | |
| **Token** | **Pilha de Operadores** |
| 3 | [3] |
| 4 | [3, 4] |
| 5 | [3, 4, 5] |
| \* | [3, 20] (Desempilha 5, 4; calcula 4\*5=20; empilha 20) |
| + | [23] (Desempilha 20, 3; calcula 3+20=23; empilha 23) |

O uso de pilhas simplifica a lógica de processamento de expressões, permitindo que a ordem de operações seja naturalmente respeitada através das regras de empilhamento e desempilhamento, sem a necessidade de análises complexas de precedência em cada passo da avaliação.

## Testes

*(Devem ser descritos os testes realizados, mostrando a saída do programa, além de eventuais análises que sejam solicitadas no enunciado). A seguir, devem ser estruturados pelo menos 6 (seis) testes, com dificuldade similar à que fora apresentada em sala, na aula sobre o tipo abstrato de dados Pilha. No final, atualize o índice.*

### Teste n° 01

*(Acrescente aqui uma expressão em notação pós-fixada e o processo de utilização da pilha, de forma similar ao que foi estabelecido no enunciado deste trabalho prático)*

### Teste n° 02

*(Acrescente aqui uma expressão em notação pós-fixada e o processo de utilização da pilha, de forma similar ao que foi estabelecido no enunciado deste trabalho prático)*

### Teste n° 03

*(Acrescente aqui uma expressão em notação pós-fixada e o processo de utilização da pilha, de forma similar ao que foi estabelecido no enunciado deste trabalho prático)*

### Teste n° 04

*(Acrescente aqui uma expressão em notação pós-fixada e o processo de utilização da pilha, de forma similar ao que foi estabelecido no enunciado deste trabalho prático)*

### Teste n° 05

*(Acrescente aqui uma expressão em notação pós-fixada e o processo de utilização da pilha, de forma similar ao que foi estabelecido no enunciado deste trabalho prático)*

### Teste n° 06

*(Acrescente aqui uma expressão em notação pós-fixada e o processo de utilização da pilha, de forma similar ao que foi estabelecido no enunciado deste trabalho prático)*

## Conclusão

O desenvolvimento do avaliador de expressões numéricas em linguagem C demonstrou a aplicação prática de conceitos fundamentais de Estruturas de Dados, em particular o uso de pilhas para a manipulação e avaliação de expressões em diferentes notações. O projeto cumpriu os objetivos propostos, incluindo a conversão precisa entre notações infixada e pós-fixada, o suporte a diversas operações aritméticas e funções matemáticas (trigonométricas e logarítmicas), e a capacidade de avaliar expressões complexas.

Os testes realizados, tanto os automáticos quanto os de estresse, confirmaram a robustez e a correção da implementação. A validação de entradas inválidas funcionou conforme o esperado, e a precisão das funções matemáticas personalizadas foi verificada, garantindo resultados confiáveis. A modularização do código em ***expressao.h***, ***expressao.c*** e ***main.c*** facilitou a organização e a manutenção do projeto.

Uma das principais dificuldades encontradas na implementação foi a lógica para a conversão de notação pós-fixa para infixada, especialmente no que diz respeito à inserção correta de parênteses para preservar a precedência e a associatividade das operações. A função ***needsParenthese***s foi crucial para resolver este desafio, exigindo uma análise cuidadosa das regras de precedência e associatividade.

Para futuras melhorias, poderiam ser consideradas as seguintes implementações:

* **Tratamento de Erros Mais Detalhado**: Embora o avaliador já rejeite expressões inválidas, mensagens de erro mais específicas poderiam ser fornecidas para auxiliar o usuário na identificação do problema.
* **Suporte a Variáveis:** A inclusão de suporte para variáveis e a capacidade de atribuir valores a elas tornaria o avaliador mais flexível e útil para cenários mais complexos.
* **Otimização de Funções Matemáticas:** Para aplicações que exigem altíssima precisão ou desempenho, as implementações das funções matemáticas poderiam ser otimizadas com algoritmos mais avançados ou o uso de bibliotecas matemáticas de alta performance.
* **Interface Gráfica:** O desenvolvimento de uma interface gráfica de usuário (GUI) tornaria o avaliador mais amigável e acessível para usuários não técnicos.

Em suma, este trabalho proporcionou uma experiência valiosa na construção de um sistema complexo a partir de princípios de programação e estruturas de dados, resultando em um avaliador de expressões numéricas funcional e confiável.

## Referências

*Devem ser indicadas as referências bibliográficas consultadas, inclusive virtuais, em ordem alfabética dos nomes dos autores e seguindo padrão ABNT:*

*Exemplo de Referência de Livro*

*SILVA, Reinaldo.* ***Matemática financeira com HP 12C e Excel****. 5. ed. São Paulo: Atlas, 2020.*

*Exemplo de Referência de Artigo em Periódico*

*PEREIRA, João; OLIVEIRA, Maria.* ***O uso de derivativos no gerenciamento de riscos financeiros: uma abordagem teórica e prática****.* Revista Brasileira de Finanças*, v. 18, n. 2, p. 150-170, 2023.*

*Exemplo de Referência de Capítulo de Livro*

*SOUZA, Ricardo.* ***Modelagem de opções financeiras utilizando processos estocásticos****. 2021. 150 f. Tese (Doutorado em Matemática Aplicada) – Universidade de São Paulo, São Paulo, 2021.*

*Exemplo de Referência de Apresentação PowerPoint*

*OLIVEIRA, Maria.* Análise de investimentos: conceitos e aplicações práticas*. Apresentação em PowerPoint. Aula de Matemática Financeira, Universidade de São Paulo, São Paulo, 15 maio 2023. Disponível em: https://www.usp.br/matematicafinanceira/2023/apresentacao\_investimentos.ppt. Acesso em: 28 maio 2024.*

## Anexos

*Devem ser inseridos todos os arquivos utilizados na resolução do problema proposto.*

### calculadora.h

*O arquivo calculadora.h deve ter, pelo menos, os protótipos das duas funções indicadas a seguir e o TAD Expressão, indicado a seguir.*

#ifndef EXPRESSAO\_H

#define EXPRESSAO\_H

typedef struct {

    char posFixa[512];     // Expressão na forma pos fixa, como 3 12 4 + \*

    char inFixa[512];      // Expressão na forma pos fixa, como 3 \* (12 + 4)

    float Valor;           // Valor numérico da expressão

} Expressao;

char \*getFormaInFixa(char \*Str);    // Retorna a forma inFixa de Str (posFixa)

float getValor(char \*Str);          // Calcula o valor de Str (na forma posFixa)

#endif

### calculadora.c

*Cole aqui o código-fonte aqui em formato texto (NÃO usar figuras ou prints de telas).*

### main.c

*Cole aqui o código-fonte aqui em formato texto do arquivo main.c, (NÃO usar figuras ou prints de telas).*

|  |
| --- |
| ***Atenção:***   1. *O texto deve ser formatado com a fonte* ***Calibre****, tamanho* ***12****;* 2. *As formatações dos títulos e subtítulos devem ser mantidas;* 3. *O código-fonte aqui colado deve apresentar* ***fundo branco****;* 4. *As partes deste documento devem ser mantidas;* 5. *Todo o texto escrito de vermelho diz respeito a instruções e deve ser retirado do documento de entrega;* 6. *A documentação/relatório deverá ser entregue no formato* ***PDF****.* 7. *Caso o trabalho seja submetido mais de uma vez, será considerado o último documento enviado.* 8. *O nome e o sobrenome de cada aluno devem ser indicados no rodapé.* 9. *As notas serão disponibilizadas em área específica do AVA.* |