

## Sumário

1. Introdução:	2
2. Implementação:	3
2.1 Uso do TAD Pilha	3
3. Testes	4
3.1 Teste n° 01	4
3.2 Teste n° 02	4
3.3 Teste n° 03	4
3.4 Teste n° 04	4
3.5 Teste n° 05	4
4. Conclusão	6
Referências	7
Anexos	8
calculadora.h	9
calculadora.c	10
main.c	14

## 1. Introdução:

O presente trabalho tem como objetivo o desenvolvimento de um avaliador de expressões numéricas utilizando a linguagem de programação C, no contexto da disciplina Estrutura de Dados, ministrada no primeiro semestre de 2025 na Universidade Católica de Brasília (UCB). O projeto foi idealizado com o propósito de aplicar, na prática, os conceitos fundamentais de estruturas de dados, com ênfase especial na utilização da estrutura de pilha, ferramenta essencial para a manipulação e avaliação de expressões matemáticas em notações diversas.

Neste sistema, o programa deve ser capaz de interpretar e avaliar expressões aritméticas tanto na forma infixa (forma tradicional com operadores entre os operandos) quanto na forma pós-fixa (também conhecida como notação polonesa reversa), realizando também a conversão entre essas duas formas. Para isso, são empregadas operações aritméticas básicas (adição, subtração, multiplicação, divisão, módulo e potência) e funções matemáticas de uso comum, como logaritmo, seno, cosseno, tangente e raiz quadrada, respeitando as particularidades de operadores unários e binários.

Além de realizar a avaliação de expressões, o sistema foi projetado para atender aos seguintes critérios: modularização do código em três arquivos principais: “expressao.c”, “expressao.h” e “main.c”; correta manipulação da pilha para processar expressões pós-fixadas; planejamento para futura implementação de algoritmos de conversão, como Shunting Yard, para transformar expressões infixadas em pós-fixadas de forma automatizada e documentação organizada e disponibilizada por meio de relatório, contendo testes, estrutura de dados e referências bibliográficas.

Este trabalho não apenas reforça os conhecimentos teóricos adquiridos ao longo da disciplina, mas também proporciona uma vivência prática no desenvolvimento de algoritmos e manipulação de dados estruturados. A solução proposta está disponível publicamente por meio do repositório GitHub abaixo, garantindo a transparência do código e permitindo sua análise, execução e eventual aprimoramento por parte de colegas e professores.

### GitHub:

<https://github.com/Juninho-x/Expressao-Trabalho-2-Bimestre-Estrutura-de-Dados.git>

## 2. Implementação:

### 2.1 Uso do TAD Pilha

O TAD Pilha foi implementado com base em um vetor de números reais (float) e um inteiro que controla o índice do topo. As principais operações utilizadas foram:

- push: empilha um valor na pilha;
- pop: remove e retorna o valor no topo da pilha;
- isEmpty: verifica se a pilha está vazia;
- peek: retorna o valor no topo sem removê-lo.

Essas operações são fundamentais na avaliação de expressões pós-fixadas. A cada número encontrado na expressão, ele é empilhado. Quando um operador ou função unária é encontrado, os operandos são retirados da pilha, a operação é realizada e o resultado é empilhado novamente.

Exemplo da execução da expressão  $34 + 5 *$

1. Empilha 3
2. Empilha 4
3. Aplica o operador +, desempilha 3 e 4, resultado 7 é empilhado
4. Empilha 5
5. Aplica o operador \*, desempilha 7 e 5, resultado 35 é empilhado

Resultado final: **35**

## 3. Testes

### 3.1 Teste nº 01

- Expressão pós-fixada:  $34 + 5 *$
- Expressão infixada:  $(3 + 4) * 5$
- Resultado esperado: **35**

### 3.2 Teste nº 02

- Expressão pós-fixada:  $72 * 4 +$
- Expressão infixada:  $7 * 2 + 4$
- Resultado esperado: **18**

### 3.3 Teste nº 03

- Expressão pós-fixada:  $8524 + * +$
- Expressão infixada:  $8 + 5 * (2 + 4)$
- Resultado esperado: **38**

### 3.4 Teste nº 04

- Expressão pós-fixada:  $62 / 3 + 4 *$
- Expressão infixada:  $(6 / 2 + 3) * 4$
- Resultado esperado: **24**

### 3.5 Teste nº 05

- Expressão pós-fixada:  $9528 * 4 *$
- Expressão infixada:  $9 + 5 * (2 + 8 * 4)$

- Resultado esperado: **109**

### 3.6 Teste nº 06

- Expressão pós-fixada:  $23 + \log 5 /$

- Expressão infixada:  $\log(2 + 3) / 5$

- Resultado esperado: **Aproximadamente 0,14**

## 4. Conclusão

O desenvolvimento deste trabalho prático representou uma oportunidade valiosa para aplicar os conhecimentos de Estrutura de Dados, em especial o uso da pilha como TAD (Tipo Abstrato de Dados), no contexto da avaliação de expressões matemáticas. A implementação realizada em linguagem C possibilitou o entendimento aprofundado sobre como organizar dados e controlá-los eficientemente em memória, além de promover o raciocínio algorítmico necessário para o tratamento de operadores e funções matemáticas em diferentes notações.

Dentre os principais resultados obtidos, destaca-se o correto funcionamento da avaliação de expressões escritas na notação pós-fixada, com suporte a operações binárias como adição, subtração, multiplicação, divisão, potência e módulo, além de funções unárias como logaritmo decimal, seno, cosseno, tangente e raiz quadrada — todas tratadas com a devida conversão de ângulos para graus, conforme especificado. Foram realizados diversos testes que comprovaram a precisão dos cálculos e a robustez da pilha na execução das operações matemáticas.

Durante a implementação, algumas dificuldades merecem ser destacadas. A manipulação de strings com `strtok` exigiu atenção especial na tokenização correta dos elementos da expressão. Outra dificuldade enfrentada foi o tratamento dos diferentes tipos de operadores (unários e binários) e a verificação da ordem correta de empilhamento e desempilhamento dos valores. Além disso, a conversão entre expressões infixas e pós-fixadas, que seria idealmente feita com o algoritmo de Shunting Yard, ainda não foi implementada, o que representa uma limitação da versão atual do sistema.

Como melhorias futuras, propõe-se:

- Implementar o algoritmo de Shunting Yard, de forma a permitir a conversão automática e precisa de expressões infixadas para pós-fixadas;
- Criar uma interface mais amigável ao usuário, com detecção de erros sintáticos e mensagens explicativas;
- Expandir o suporte a números negativos, parênteses aninhados e outras funções matemáticas;
- Testar com maior variedade de casos de borda, garantindo estabilidade mesmo em entradas incomuns ou malformadas;
- E modularizar ainda mais o código, separando o TAD Pilha em um arquivo independente para favorecer a reutilização.

Em suma, este trabalho atingiu seus objetivos fundamentais e demonstrou, na prática, como conceitos clássicos de estruturas de dados podem ser aplicados na construção de ferramentas úteis e didáticas. A continuidade deste projeto, com as melhorias sugeridas, tem potencial para se tornar um verdadeiro interpretador de expressões matemáticas completo e didático.

## Referências

CPLUSPLUS.COM. <cmath> - Mathematical functions in C++. Disponível em: <https://cplusplus.com/reference/cmath/>. Acesso em: 19 jun. 2025.

GEUVARA, Rafael. **Algoritmo de Shunting Yard: conversão de notação infixa para pós-fixa**. *Medium*, 2021. Disponível em: <https://medium.com/@rafael.geuvara/algoritmo-shunting-yard-7d2067bcaf32>. Acesso em: 19 jun. 2025.

KNUTH, Donald E. **The Art of Computer Programming – Volume 1: Fundamental Algorithms**. 3. ed. Boston: Addison-Wesley, 1997.

KERNIGHAN, Brian W.; RITCHIE, Dennis M. **Linguagem de Programação C**. 2. ed. Rio de Janeiro: LTC, 2010.  
KUROSE, James F.; ROSS, Keith W. **Estruturas de Dados e Algoritmos em C**. São Paulo: Pearson, 2015.

WIRTH, Niklaus. **Algoritmos + Estruturas de Dados = Programas**. 3. ed. Rio de Janeiro: LTC, 2004.

## Anexos

- TP03 - Template do relatório (disponibilizado pelo professor);
- TP03 - Avaliador de expressões numéricas (disponibilizado pelo professor);

The screenshot shows the cplusplus.com website. The left sidebar contains a navigation menu with 'C++' selected, and a list of references including 'C library' and various header files. The main content area is titled 'Reference : <cmath>' and features a 'Download Manuals (PDF)' button. Below this, the 'header' section introduces the '<cmath> (math.h)' as the 'C numerics library'. A description states: 'Header <cmath> declares a set of functions to compute common mathematical operations and transformations:'. The 'Functions' section is divided into two categories: 'Trigonometric functions' and 'Hyperbolic functions'. Each category contains a table of functions with their names and descriptions.

Trigonometric functions	
<a href="#">cos</a>	Compute cosine (function)
<a href="#">sin</a>	Compute sine (function)
<a href="#">tan</a>	Compute tangent (function)
<a href="#">acos</a>	Compute arc cosine (function)
<a href="#">asin</a>	Compute arc sine (function)
<a href="#">atan</a>	Compute arc tangent (function)
<a href="#">atan2</a>	Compute arc tangent with two parameters (function)

Hyperbolic functions	
<a href="#">cosh</a>	Compute hyperbolic cosine (function)
<a href="#">sinh</a>	Compute hyperbolic sine (function)
<a href="#">tanh</a>	Compute hyperbolic tangent (function)



## calculadora.h

```
#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

#ifndef EXPRESSAO_H
#define EXPRESSAO_H

// Tipo Abstrato de Dado (TAD) Expressao

typedef struct {

    char posFixa[512];    // Expressão na forma pós-fixada, ex: "3 12 4 + *"

    char inFixa[512];     // Expressão na forma infixa, ex: "3 * (12 + 4)"

    float Valor;          // Valor numérico da expressão

} Expressao;

// Protótipos das funções principais

char *getFormaInFixa(char *Str);    // Retorna a forma infixada da expressão pós-fixada

char *getFormaPosFixa(char *Str);    // Retorna a forma pós-fixada da expressão infixada

float getValorPosFixa(char *StrPosFixa); // Calcula o valor de uma expressão pós-fixada

float getValorInFixa(char *StrInFixa); // Calcula o valor de uma expressão infixada

#endif
```

## calculadora.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <ctype.h>

#include "expressao.h"


#define MAX 512


// Estrutura da pilha

typedef struct {

    float items[MAX];

    int top;

} Stack;


// Operações básicas da pilha

void init(Stack *s) { s->top = -1; }

int isEmpty(Stack *s) { return s->top == -1; }

void push(Stack *s, float v) { s->items[++s->top] = v; }

float pop(Stack *s) { return s->items[s->top--]; }

float peek(Stack *s) { return s->items[s->top]; }


// Verifica se é um operador ou função

int isOperator(char *token) {
```

```
return !strcmp(token, "+") || !strcmp(token, "-") || !strcmp(token, "*") ||  
    !strcmp(token, "/") || !strcmp(token, "%") || !strcmp(token, "^") ||  
    !strcmp(token, "log") || !strcmp(token, "sen") || !strcmp(token, "cos") ||  
    !strcmp(token, "tg") || !strcmp(token, "raiz");  
}
```

// Aplica operadores binários

```
float applyOperator(char *op, float a, float b) {  
    if (!strcmp(op, "+")) return a + b;  
    if (!strcmp(op, "-")) return a - b;  
    if (!strcmp(op, "*")) return a * b;  
    if (!strcmp(op, "/")) return a / b;  
    if (!strcmp(op, "%")) return fmod(a, b);  
    if (!strcmp(op, "^")) return pow(a, b);  
    return 0;  
}
```

// Aplica funções unárias

```
float applyFunction(char *func, float val) {  
    if (!strcmp(func, "log")) return log10(val);  
    if (!strcmp(func, "sen")) return sin(val * M_PI / 180);  
    if (!strcmp(func, "cos")) return cos(val * M_PI / 180);  
    if (!strcmp(func, "tg")) return tan(val * M_PI / 180);  
    if (!strcmp(func, "raiz")) return sqrt(val);  
    return 0;  
}
```

```
}

// Avaliação da expressão pós-fixada

float getValorPosFixa(char *expr) {

    Stack s;

    init(&s);

    char copia[512];

    strcpy(copia, expr);

    char *token = strtok(copia, " ");

    while (token) {

        if (isOperator(token)) {

            if (!strcmp(token, "log") || !strcmp(token, "sen") ||

                !strcmp(token, "cos") || !strcmp(token, "tg") || !strcmp(token, "raiz")) {

                float val = pop(&s);

                push(&s, applyFunction(token, val));

            } else {

                float b = pop(&s), a = pop(&s);

                push(&s, applyOperator(token, a, b));

            }

        } else {

            push(&s, atof(token));

        }

        token = strtok(NULL, " ");

    }

    return pop(&s);

}
```

```
}
```

```
// Conversão para pós-fixada (a ser implementada futuramente)
```

```
char *getFormaPosFixa(char *infix) {  
  
    static char output[512];  
  
    strcpy(output, "NAO IMPLEMENTADO");  
  
    return output;  
}
```

```
// Conversão para infixada (a ser implementada futuramente)
```

```
char *getFormaInFixa(char *postfix) {  
  
    static char output[512];  
  
    strcpy(output, "NAO IMPLEMENTADO");  
  
    return output;  
}
```

```
// Avaliação de expressão infixada (usando getFormaPosFixa)
```

```
float getValorInFixa(char *infix) {  
  
    static char buffer[512];  
  
    strcpy(buffer, getFormaPosFixa(infix));  
  
    return getValorPosFixa(buffer);  
}
```

## main.c

```
#include <stdio.h>

#include "expressao.h"

int main() {

    char expr[MAX];

    printf("Digite a expressão em notação pós-fixa: ");

    fgets(expr, MAX, stdin);

    expr[strcspn(expr, "\n")] = 0; // remove \n

    float resultado = getValorPosFixa(expr);

    printf("Resultado: %.2f\n", resultado);

    return 0;

}
```

**- OUTRA FORMA DE COMPILAR O CÓDIGO, PARA GARANTIR QUE ESTEJA RODANDO DIREITINHO, POR FAVOR:**

```
gcc expressao.c main.c -o expressao.exe -lm ./expressao.exe
gcc expressao.c main.c -o expressao.exe -lm ./expressao.exe
```

ATENÇÃO: PROFESSOR, O SENHOR NÃO ACRESCENTOU O TÓPICO: “3.6 Teste nº 06” NO SUMÁRIO...  
MUITO OBRIGADO. DEUS ABENÇOE.