Sumário

1. Introdução:

Este trabalho tem como objetivo desenvolver um avaliador de expressões numéricas em linguagem C, capaz de converter e avaliar expressões nas notações infixa e pós-fixa, incluindo operações matemáticas básicas e funções especiais como seno, cosseno, tangente, logaritmo e raiz quadrada.
O sistema utiliza o TAD Pilha como estrutura fundamental para manipulação das expressões, garantindo modularidade e clareza na implementação.

GitHub:

https://github.com/luizunc/Expressoes_numericas_n2

2. Implementação:

O projeto foi implementado em C, utilizando três arquivos principais: expressao.h, expressao.c e main.c.

O TAD Pilha foi utilizado para armazenar operadores e operandos durante a conversão e avaliação das expr essões.

As principais funções implementadas foram:

-getFormaPosFixa: converte expressão infixa para pós-fixa.
-getFormaInFixa: converte expressão pós-fixa para infixa.
-getValorPosFixa: avalia expressão pós-fixa.
- getValorInFixa: avalia expressão infixa.

As funções matemáticas especiais (sen, cos, tg, log, raiz) foram implementadas considerando argumentos em graus para as funções trigonométricas.

2.1 Uso do TAD Pilha

O TAD Pilha foi utilizado para:
*- **Conversão infixa → pós-fixa:** armazenar operadores e garantir a precedência correta.*

*- **Avaliação pós-fixa:** armazenar operandos e calcular o resultado conforme os operadores e funções são encontrados.*

Exemplo de uso da pilha na avaliação pós-fixa:

*Para a expressão 3 4 + 5 *, a pilha evolui assim:*

PASSO	*TOKEN*	*PILHA*
1	*3*	*3*
2	*4*	*3, 4*
3	*+*	*7*
4	*5*	*7, 5*

5	***	*35*

3. Testes

Foram realizados diversos testes para validar o funcionamento do avaliador, incluindo os nove exemplos das instruções e outros 3 de complexidade igualitária ou superior aos instruídos, criados para verificar casos especiais.

3.1 Teste n° 01

Expressão em notação pós-fixada:

3 4 + 5 *

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*3*	*Empilha 3*	*3*
2	*4*	*Empilha 4*	*3, 4*
3	*+*	*Desempilha 3 e 4, soma(3+4=7), empilha resultado*	*7*
4	*5*	*Empilha 5*	*7, 5*
5	***	*Desempilha 7 e 5, multiplica(7.5=35), empilha [35]* *Valor Final: 35*	

3.2 Teste n° 02

Expressão em notação pós-fixada:

7 2 * 4 +

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*7*	*Empilha 7*	*7*
2	*2*	*Empilha 2*	*7, 2*

3	***	*Desempilha 7 e 2, multiplica(7.2=14), empilha[14]* *4 | 4 | empilha 4 | [14, 4] | 5 | + | Desempilha 14 e 4, soma(14+4=18), empilha[18]* *Valor final: 18*	

3.3 Teste n° 03

Expressão em notação pós-fixada:

8 5 2 4 + * +

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*8*	*Empilha 8*	*8*
2	*5*	*Empilha 5*	*8, 5*
3	*2*	*Empilha 2*	*8, 5, 2*
4	*4*	*Empilha 4*	*8, 5, 2, 4*
5	*+*	*Desempilha 2 e 4, soma (2+4=6), empilha*	*8, 5, 6*
6	***	*Desempilha 5 e 6, multiplica (5.6=30), empilha [8, 30] | 7 | + | desempilha 8 e 30, soma (8+30=38) empilha [38]* *Valor final: 38*	

3.4 Teste n° 04

Expressão em notação pós-fixada:

6 2 / 3 + 4 *

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*

1	*6*	*Empilha 6*	*6*
2	*2*	*Empilha 2*	*6, 2*
3	*/*	*Desempilha 6 e 2, divide(6/2=3), empilha*	*3*
4	*3*	*Empilha 3*	*3, 3*
5	*+*	*Desempilha 3 e 3, soma(3+3=6), empilha*	*6*
6	*4*	*Empilha 4*	*6, 4*
7	***	*Desempilha 6 e 4, multiplica(6.4=24), empilha[24]* *Valor final: 24*	

3.5 Teste n° 05

Expressão em notação pós-fixada:

*9 5 2 8 * 4 + * +*

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*9*	*Empilha 9*	*9*
2	*5*	*Empilha 5*	*9, 5*
3	*2*	*Empilha 2*	*9, 5, 2*
4	*8*	*Empilha 8*	*9, 5, 2, 8*
5	***	*Desempilha 2 e 8, multiplica(2.8=16), empilha [9, 5, 16]| 6 | 4 |empilha 4 [9, 5, 16, 4] | 7 | + | desempilha 16 e 4, soma(16+4=20), empilha [9, 5, 20] | 8 |desempilha 5 e 20, multiplica 5 e 20(5.20=100), empilha |[9, 100] | 9 | + | desempilha 9 e 100, soma(9+100=109), empilha [109]* *Valor final: 109*	

3.6 Teste n° 06

Expressão em notação pós-fixada:

2 3 + log 5 /

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*2*	*Empilha 2*	*2*
2	*3*	*Empilha 3*	*2, 3*
3	*+*	*Desempilha 2 e 3, soma(2+3=5), empilha*	*5*
4	*Log*	*Desempilha 5, aplica log10(5) aproximadamente 0.69897, empilha*	*0.69897*
5	*5*	*Empilha 5*	*0.69897, 5*
6	*/*	*Desempilha 0.69897 e 5, divide(0.69897/5 aproximadamente 0.14), empilha* *Valor final= 0.14*	*0.14*

3.7 Teste n° 07

Expressão em notação pós-fixada:

10 log 3 ^ 2 +

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*10*	*Empilha 10*	*10*
2	*Log*	*Desempilha 10, aplica log 10(10)=1, empilha*	*1*
3	*3*	*Empilha 3*	*1, 3*
4	*^*	*Desempilha 1 e 3, potência(1^3=1), empilha*	*1*
5	*2*	*Empilha 2*	*1, 2*
6	*+*	*Desempilha 1 e 2, soma(1+2=3), empilha* *Valor final= 3*	*3*

3.8 Teste n° 08

Expressão em notação pós-fixada:

*45 60 + 30 cos **

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*45*	*Empilha 45*	*45*
2	*60*	*Empilha 60*	*45, 60*
3	*+*	*Desempilha 45 e 60, soma(45+60=105), empilha*	*105*
4	*30*	*Empilha 30*	*105, 30*
5	*Cos*	*Desempilha, aplica cos(30º) aproximadamente 0.8660, empilha*	*105, 0.8660*
6	*	*Desempilha 105 e 0.8660, multiplica(105x0.8660=90.93), empilha* *Valor final= 90,93*	

3.9 Teste n° 09

Expressão em notação pós-fixada:

0.5 45 sen 2 ^ +

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*0.5*	*Empilha 0.5*	*0.5*
2	*45*	*Empilha 45*	*0.5, 45*
3	*Sem*	*Desempilha 45, aplica sem(45º) aproximadamente 0.7071, empilha*	*0.5, 0.7071*
4	*2*	*Empilha 2*	*0.5, 0.7071, 2*
5	*^*	*Desempilha 0.7071 e 2, potência(0.7071^2 aproximadamente 0.5), empilha*	*0.5, 0.5*

6	*+*	*Desempilha 0.5 e 0.5, soma(0.5+0.5=1), empilha* *Valor final= 1*	*1*

3.10 Teste n° 10

Expressão em notação pós-fixada:

*30 sen 8 * raiz 1 +*

Expressão infixa correspondente: *raiz(sen(30) * 8) + 1*

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*30*	*Empilha 30*	*30*
2	*Sen*	*Desempilha 30, aplica sem(30º)=0.5, empilha*	*0.5*
3	*8*	*Empilha 8*	*0.5, 8*
4	***	*Desempilha 0.5 e 8, multiplica (0.58=4), empilha | [4] |* *| 5 | raiz | Desempilha 4, aplica raiz(4)=2, empilha | [2] |* *| 6 | 1 | Empilha 1 | [2, 1] |* *| 7 | + | Desempilha 2 e 1, soma (2+1=3), empilha | [3]* *Valor final= 3*	

3.11 Teste n° 11

Expressão em notação pós-fixada:

*100 log 3 + 60 cos 1 - **

Expressão infixa correspondente: *(log(100) + 3) * (cos(60) - 1)*

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*100*	*Empilha 100*	*100*
2	*Log*	*Desempilha 100, aplica log10(100)=2, empilha*	*2*
3	*3*	*Empilha 3*	*2, 3*
4	*+*	*Desempilha 2 e 3, soma(2+3=5), empilha*	*5*
5	*60*	*Empilha 60*	*5, 60*
6	*Cos*	*Desempilha 60, aplica cos(60º)=0.5, empilha*	*5, 0.5*
7	*1*	*Empilha 1*	*5, 0.5, 1*
8	*-*	*Desempilha 0.5 e 1, subtrai(0.5-1=-0.5), empilha*	*5, 0.5*
9	***	*Desempilha 5 e -0.5, multiplica(5(-0.5)=-2.5), empilha[-2.5]* *Valor final= -2.5*	

3.12Teste n° 12

Expressão em notação pós-fixada:

*45 tg 2 ^ 90 sen 1 + / 0.5 **

Expressão infixa correspondente: *(tg(45)^2 / (sen(90) + 1)) * 0.5*

Processo de utilização da pilha:

PASSO	*TOKEN*	*AÇÃO*	*PILHA APÓS A AÇÃO*
1	*45*	*Empilha 45*	*45*
2	*Tg*	*Desempilha 45, aplica tg(45°)=1, empilha*	*1*
3	*2*	*Empilha 2*	*1, 2*
4	*^*	*Desempilha 1 e 2, potencia (1^2=1), empilha*	*1*
5	*90*	*Empilha 90*	*1, 90*
6	*Sen*	*Desempilha 90, aplica sen(90°)=1, empilha*	*1, 1*
7	*1*	*Empilha 1*	*1, 1, 1*

8	*+*	*Desempilha 1 e 1, soma (1+1=2), empilha*	*1, 2*
9	*/*	*Desempilha 1 e 2, divide (1/2=0.5), empilha*	*0.5*
10	*0.5*	*Empilha 0.5*	*0.5, 0.5*
11	***	*Desempilha 0.5 e 0.5, multiplica (0.50.5=0.25), empilha [0.25]* *Valor final= 0.25*	

4. Conclusão

O trabalho permitiu consolidar o uso do TAD Pilha na manipulação de expressões matemáticas, tanto para conversão quanto para avaliação. O avaliador implementado mostrou-se eficiente e flexível, suportando operações básicas e funções matemáticas especiais. As principais dificuldades estiveram na manipulação correta das funções de um operando e no tratamento de precedência de operadores. Como melhoria futura, sugere-se aprimorar o tratamento de erros e adicionar suporte a mais funções matemáticas.

Referências

EUSTÁQUIO, M. *Aulas e códigos-fonte. Ambiente Virtual de Aprendizagem – Brightspace, Universidade Católica. Disponível apenas para alunos e professores em: https://ava.catolica.edu.br. Acesso em: 6 jun. 2025.*

EUSTÁQUIO, M. *Encadeamentos Duplos. Ambiente Virtual de Aprendizagem – Brightspace, Universidade Católica. Disponível apenas para alunos e professores em: https://ava.catolica.edu.br. Acesso em: 31 mai. 2025.*

EUSTÁQUIO, M. *Filas. Ambiente Virtual de Aprendizagem – Brightspace, Universidade Católica. Disponível apenas para alunos e professores em: https://ava.catolica.edu.br. Acesso em: 23 mai. 2025.*

EUSTÁQUIO, M. *Pilhas. Ambiente Virtual de Aprendizagem – Brightspace, Universidade Católica. Disponível apenas para alunos e professores em: https://ava.catolica.edu.br. Acesso em: 15 jun. 2025.*

FARIAS, R. *Pilhas. Disponível em: https://www.cos.ufrj.br/~rfarias/cos121/pilhas.html. Acesso em: 12 jun. 2025.*

FORSTER, P. *Pilhas. Disponível em: https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html. Acesso em: 10 jun. 2025.*

YOUTUBE. *Dominando os Requisitos de Software com Engenharia de Requisitos - Parte 1. [S.l.]: Engenharia de Requisitos, 2021. Disponível em: https://www.youtube.com/watch?v=s0dGZHdPNKc. Acesso em: 03 jun. 2025.*

YOUTUBE. *Engenharia de Requisitos - Série completa. [S.l.]: Engenharia de Requisitos, 2021. Disponível em: https://www.youtube.com/playlist?list=PLqJK4Oyr5WSj_Ngpezsatdu5s2G74g9sf. Acesso em: 03 jun. 2025.*

Anexos

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 infixa, como 3 * (12 + 4)

float Valor;                // Valor numérico da expressão


} Expressao;

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

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

float getValorPosFixa(char *StrPosFixa); // Calcula o valor de Str (na forma posFixa)

float getValorInFixa(char *StrInFixa); // Calcula o valor de Str (na forma inFixa)

#endif
```

calculadora.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "expressao.h"
```

```
#define MAX 512

//

// Função auxiliar para verificar precedência dos operadores

//

int precedencia(char op) {

   switch(op) {

      case '+':

      case '-': return 1;

      case '*':

      case '/':

      case '%': return 2;

      case '^': return 3;

      default: return 0;

   }

}


// Função para converter infixa para pós-fixa

char *getFormaPosFixa(char *Str) {

   static char saida[MAX];

   char pilha[MAX];

   int topo = -1, j = 0;

   int i = 0;

   memset(saida, 0, sizeof(saida));

   while (Str[i] != '\0') {
```

```
if (Str[i] == ' ') { i++; continue; }

if ((Str[i] >= '0' && Str[i] <= '9') || Str[i] == '.') {

   //

   // Número: copiar para saída

   //

   while ((Str[i] >= '0' && Str[i] <= '9') || Str[i] == '.') {

      saida[j++] = Str[i++];

   }

   saida[j++] = ' ';

   continue;

}

if (Str[i] == '(') {

   pilha[++topo] = Str[i];

} else if (Str[i] == ')') {

   while (topo >= 0 && pilha[topo] != '(') {

      saida[j++] = pilha[topo--];

      saida[j++] = ' ';

   }

   if (topo >= 0 && pilha[topo] == '(') topo--;

} else if (strchr("+-*/%^", Str[i])) {

   while (topo >= 0 && precedencia(pilha[topo]) >= precedencia(Str[i])) {

      saida[j++] = pilha[topo--];

      saida[j++] = ' ';

   }

   pilha[++topo] = Str[i];
```

```
        }
        //
        // TODO: Funções matemáticas (sen, cos, etc)
        //
        i++;
    }
    while (topo >= 0) {
        if (pilha[topo] != '(')
            saida[j++] = pilha[topo];
        topo--;
        saida[j++] = ' ';
    }
    saida[j] = '\0';
    return saida;
}
//
// Função para converter pós-fixa para infixa
//
char *getFormaInFixa(char *Str) {
    static char saida[MAX * 2];
    char *pilha[MAX];
    int topo = -1;
    char token[64];
    int i = 0, j = 0;
    while (Str[i] != '\0') {
```

```
if (Str[i] == ' ') { i++; continue; }

//

// Se for número

//

if ((Str[i] >= '0' && Str[i] <= '9') || Str[i] == '.') {

   j = 0;

   while ((Str[i] >= '0' && Str[i] <= '9') || Str[i] == '.') {

      token[j++] = Str[i++];

   }

   token[j] = '\0';

   pilha[++topo] = strdup(token);

   continue;

}

//

// Operadores binários

//

if (strchr("+-*/%^", Str[i])) {

   if (topo < 1) return NULL;

   char *b = pilha[topo--];

   char *a = pilha[topo--];

   char *expr = malloc(strlen(a) + strlen(b) + 8);

   sprintf(expr, "(%s %c %s)", a, Str[i], b);

   free(a); free(b);

   pilha[++topo] = expr;

   i++;
```

```
    continue;
}
//
// Funções de um operando
//
if (strncmp(&Str[i], "sen", 3) == 0) {
    if (topo < 0) return NULL;
    char *a = pilha[topo--];
    char *expr = malloc(strlen(a) + 8);
    sprintf(expr, "sen(%s)", a);
    free(a);
    pilha[++topo] = expr;
    i += 3;
    continue;
}
if (strncmp(&Str[i], "cos", 3) == 0) {
    if (topo < 0) return NULL;
    char *a = pilha[topo--];
    char *expr = malloc(strlen(a) + 8);
    sprintf(expr, "cos(%s)", a);
    free(a);
    pilha[++topo] = expr;
    i += 3;
    continue;
}
```

```
if (strncmp(&Str[i], "tg", 2) == 0) {
    if (topo < 0) return NULL;
    char *a = pilha[topo--];
    char *expr = malloc(strlen(a) + 8);
    sprintf(expr, "tg(%s)", a);
    free(a);
    pilha[++topo] = expr;
    i += 2;
    continue;
}
if (strncmp(&Str[i], "log", 3) == 0) {
    if (topo < 0) return NULL;
    char *a = pilha[topo--];
    char *expr = malloc(strlen(a) + 8);
    sprintf(expr, "log(%s)", a);
    free(a);
    pilha[++topo] = expr;
    i += 3;
    continue;
}
if (strncmp(&Str[i], "raiz", 4) == 0) {
    if (topo < 0) return NULL;
    char *a = pilha[topo--];
    char *expr = malloc(strlen(a) + 8);
    sprintf(expr, "raiz(%s)", a);
```

```
            free(a);
            pilha[++topo] = expr;
            i += 4;
            continue;
        }
        i++;
    }
    if (topo == 0) {
        strncpy(saida, pilha[0], MAX * 2);
        free(pilha[0]);
        return saida;
    }
    return NULL;
}
//
// Função para avaliar expressão pós-fixa
//
float getValorPosFixa(char *StrPosFixa) {
    float pilha[MAX];
    int topo = -1;
    char token[64];
    int i = 0, j = 0;
    while (StrPosFixa[i] != '\0') {
        if (StrPosFixa[i] == ' ') { i++; continue; }
        //
```

```
// Se for número
//
if ((StrPosFixa[i] >= '0' && StrPosFixa[i] <= '9') || StrPosFixa[i] == '.') {
   j = 0;
   while ((StrPosFixa[i] >= '0' && StrPosFixa[i] <= '9') || StrPosFixa[i] == '.') {
      token[j++] = StrPosFixa[i++];
   }
   token[j] = '\0';
   pilha[++topo] = atof(token);
   continue;
}
//
// Se for operador
//
if (strchr("+-*/%^", StrPosFixa[i])) {
   if (topo < 1) return 0.0; // erro: operandos insuficientes
   float b = pilha[topo--];
   float a = pilha[topo--];
   switch (StrPosFixa[i]) {
      case '+': pilha[++topo] = a + b; break;
      case '-': pilha[++topo] = a - b; break;
      case '*': pilha[++topo] = a * b; break;
      case '/': pilha[++topo] = a / b; break;
      case '%': pilha[++topo] = fmod(a, b); break;
      case '^': pilha[++topo] = pow(a, b); break;
```

```
    }
    i++;
    continue;
}
//
// Funções matemáticas
//
if (strncmp(&StrPosFixa[i], "sen", 3) == 0) {
    if (topo < 0) return 0.0;
    float a = pilha[topo--];
    pilha[++topo] = sin(a * M_PI / 180.0); // graus para radianos
    i += 3;
    continue;
}
if (strncmp(&StrPosFixa[i], "cos", 3) == 0) {
    if (topo < 0) return 0.0;
    float a = pilha[topo--];
    pilha[++topo] = cos(a * M_PI / 180.0);
    i += 3;
    continue;
}
if (strncmp(&StrPosFixa[i], "tg", 2) == 0) {
    if (topo < 0) return 0.0;
    float a = pilha[topo--];
    pilha[++topo] = tan(a * M_PI / 180.0);
```

```
            i += 2;
            continue;
        }
        if (strncmp(&StrPosFixa[i], "log", 3) == 0) {
            if (topo < 0) return 0.0;
            float a = pilha[topo--];
            pilha[++topo] = log10(a);
            i += 3;
            continue;
        }
        if (strncmp(&StrPosFixa[i], "raiz", 4) == 0) {
            if (topo < 0) return 0.0;
            float a = pilha[topo--];
            pilha[++topo] = sqrt(a);
            i += 4;
            continue;
        }
        //
        // Se não reconhecido, avança
        //
        i++;
    }
    if (topo == 0) return pilha[0];
    return 0.0; // Log: caso de erro
}
```

```
//
// Função para avaliar expressão infixa
//
float getValorInFixa(char *StrInFixa) {
   char *posfixa = getFormaPosFixa(StrInFixa);
   return getValorPosFixa(posfixa);
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "expressao.h"

int main() {
   char entrada[512];
   int opcao;
   printf("Avaliador de Expressoes Numericas\n");
   printf("1. Infixa para Pos-fixa\n");
   printf("2. Pos-fixa para Infixa\n");
   printf("3. Avaliar Pos-fixa\n");
   printf("4. Avaliar Infixa\n");
   printf("0. Sair\n");
   do {
      printf("\nEscolha uma opcao: ");
      scanf("%d", &opcao);
      getchar(); //limpeza de buffer
```

```
switch(opcao) {
    case 1:
        printf("Digite a expressao infixa: ");
        fgets(entrada, 512, stdin);
        entrada[strcspn(entrada, "\n")] = 0;
        printf("Pos-fixa: %s\n", getFormaPosFixa(entrada));
        break;
    case 2:
        printf("Digite a expressao pos-fixa: ");
        fgets(entrada, 512, stdin);
        entrada[strcspn(entrada, "\n")] = 0;
        printf("Infixa: %s\n", getFormaInFixa(entrada));
        break;
    case 3:
        printf("Digite a expressao pos-fixa: ");
        fgets(entrada, 512, stdin);
        entrada[strcspn(entrada, "\n")] = 0;
        printf("Valor: %.2f\n", getValorPosFixa(entrada));
        break;
    case 4:
        printf("Digite a expressao infixa: ");
        fgets(entrada, 512, stdin);
        entrada[strcspn(entrada, "\n")] = 0;
        printf("Valor: %.2f\n", getValorInFixa(entrada));
        break;
```

```
        case 0:
            printf("Saindo...\n");
            break;
        default:
            printf("Opcao invalida!\n");
    }
  } while(opcao != 0);
  return 0;
}
```