Sumário

1. Introdução: 2

2. Implementação: 2

2.1 Uso do TAD Pilha 3

3. Testes 3

3.1 Teste n° 01 4

3.2 Teste n° 02 4

3.3 Teste n° 03 5

3.4 Teste n° 04 5

3.5 Teste n° 05 6

3.6 Teste n° 06 6

4. Conclusão 6

Referências 7

Anexos 7

calculadora.h 7

calculadora.c 7

main.c 13

1. Introdução:

Este é um projeto da disciplina Estrutura de Dados, ministrado pelo professor Marcelo Eustáquio para alunos do terceiro semestre de Engenharia de Software, na Universidade Católica de Brasília.

O intuito deste projeto é construir uma aplicação, em linguagem C, capaz de ler, converter e calcular expressões matemáticas na ordem infixa e pós-fixa. Para isso, usaremos uma *pilha* como estrutura fundamental do projeto.

A aplicação é dividida em três arquivos, “calculadora.h”, o cabeçalho com funções fornecidas pelo professor, “calculadora.c”, que contém o restante das funções que utilizamos, criadas pelos alunos, e “main.c”, que contém a parte do código que o usuário interage.

GitHub:

https://github.com/LuisFernandoV14/TPF---Avaliacao-de-expressoes-numericas

2. Implementação:

As funções que servem como base para os cálculos realizados pelo programa são as funções presentes em “calculadora.h”:

- *char *gerFormaInFixa(char *Str)* , *char *getFormaPosFixa(char *Str)*, que, respectivamente, convertem uma expressão para a forma infixa ou pós-fixa.
- *float getValorPosFixa(char *StrPosFixa)* e *float getValorInFixa(char *StrInFixa)*, que, respectivamente, calculam o valor de uma expressão pós-fixa ou infixa.

Além disso, o programa funciona utilizando duas pilhas distintas, uma pilha de valores do tipo *float* e outra usando valores do tipo *string* (vetor de caracteres). Destrinchando essas estruturas, temos:

- *PilhaFloat*, uma pilha para armazenar valores numéricos. Uma estrutura com um array chamado *itens[]* – Que serve para armazenar os elementos da pilha – e um ponteiro para o topo, chamado *topo*. Para o funcionamento e criação dessa pilha, existem as funções: *inicializaPilhaFloat(PilhaFloat *p)*, *empilhaFloat(PilhaFloat *p, float val)* e *desempilhaFloat(PilhaFloat *p)*.
- *PilhaString*, uma pilha para armazenar operandos e operadores. Também contém um array chamado *itens[]* para armazenar os elementos da pilha e um ponteiro para o topo, chamado *topo*. Para o funcionamento e criação dessa pilha, existem as funções: *inicializaPilhaStr(PilhaString *p)*, *empilhaStr(PilhaString *p, char *val)*, *desempilhaStr(PilhaString *p)*.

Também temos as funções que interagem com o usuário, presente no arquivo “main.c”. A *main()* é um menu interativo, que roda em loop e chama as funções que calculam o valor de uma expressão matemática infixa (*testarExpressaoInfixa()*) ou pós-fixa (*testarExpressaoPosfixa()*), ou roda testes automáticos fornecidos pelo professor (*testesAutomaticos()*).

Por fim, temos as funções que trabalham para o funcionamento correto de outras funções, como *grausParaRadianos(float graus)* e *prioridade(char *op)*, que, respectivamente, converte graus para radianos e retorna um inteiro para definir a prioridade de um operador na pilha de strings.

2.1 Uso do TAD Pilha

Assim como foi explicado na parte de *Implementação*, nós utilizamos duas pilhas para a aplicação. Cada pilha tem um array de valores, chamado de *itens*, com tamanho *MAX*, uma constante definida em "calculadora.c" que equivale ao inteiro 512. Ambas as pilhas também possuem um ponteiro para o valor que está no topo na pilha, esse ponteiro serve para garantir que valores adicionados na pilha serão adicionados no topo, assim como os elementos removidos também serão removidos do topo, o que caracteriza uma pilha.

A diferença entre as pilhas está no tipo que o vetor *itens[]* pode armazenar, sendo um para strings (vetor de caracteres) e outro para floats.

A pilha de strings serve para converter operações infixas em pós-fixas. É usada na função "*char *getFormaPosFixa(char *StrInFixa)*". Ao ler uma operação infixa, sempre que o compilador detectar um inteiro ele o converte em char e concatena na string *resultado*. Sempre que o compilador detectar um operador ele o empilha na pilha de strings e a cada dois inteiros lidos ele desempilha e concatena o operador no topo na string *resultado*. No final da função, *resultado* é retornado.

A pilha de floats serve para fazer o cálculo das expressões. É usada na função "*float getValorPosFixa(char *StrPosFixa)*". Ao ler uma operação pós-fixa, sempre que o compilador detectar um número ele o converte de char para inteiro e o empilha. Sempre que o compilador detectar um operador ele desempilha os últimos dois números, realiza o cálculo numérico usando esses valores e o operador e empilha o resultado. No final da função, só haverá um número na pilha, o resultado da operação inteira, que será retornado.

3. Testes

Para comprovar a eficácia da nossa aplicação, realizamos diversos testes. Por isso, os próximos tópicos deste documento servirão para mostrar esses testes, servindo como feedback e aprendizado.

Os testes usando notação pós-fixa funcionam da seguinte forma: assim que o programa lê um número ele o joga para a pilha e faz isso até achar um operador. Ao achar um operador o programa desempilha os últimos dois números e realiza a operação do operador lido, depois, empilha o resultado.

Os testes usando notação infixa funcionam da mesma forma, para isso, o programa primeiro converte a notação para pós-fixa antes de calcular a expressão. A conversão segue a seguinte lógica: números lidos vão direto para a *string* que vai ser retornada, operadores são empilhados. Ao ler dois números, o operador na pilha é desempilhado e jogado para a *string* que vai ser retornada.

Para não ficar maçante a leitura deste documento, serão explicados em detalhes apenas os primeiros exemplos de cada teste, todos os outros seguem a mesma lógica.

3.1 Teste n° 01

```
========== AVALIADOR DE EXPRESSOES ==========
1. Avaliar expressao INFIXADA
2. Avaliar expressao POS-FIXADA
3. Rodar testes automaticos
0. Sair
=============================================
Digite o numero da opcao: 1

Digite a expressao INFIXADA (ex: (3 + 4) * 5):
> (10 + 5) * 2

=== Resultado ===
Forma pos-fixada: 10 5 + 2 *
Resultado: 30.0000
=================
```

1 - Conversão: Primeiro, a expressão infixa é convertida para a pós-fixa 10 5 + 2 *.
2 - Avaliação Pós-fixa:
    2.1 - Lê 10. Pilha: [10]
    2.2 - Lê 5. Pilha: [10, 5]
    2.3 - Lê +. Desempilha 5 e 10, calcula 10 + 5 = 15, empilha o resultado. Pilha: [15]
    2.4 - Lê 2. Pilha: [15, 2]
    2.5 - Lê *. Desempilha 2 e 15, calcula 15 * 2 = 30, empilha o resultado. Pilha: [30]
3 - Fim da expressão. O resultado final é 30.

3.2 Teste n° 02

```
========== AVALIADOR DE EXPRESSOES ==========
1. Avaliar expressao INFIXADA
2. Avaliar expressao POS-FIXADA
3. Rodar testes automaticos
0. Sair
=============================================
Digite o numero da opcao: 2

Digite a expressao POS-FIXADA (ex: 3 4 + 5 *):
> 8 4 / 2 + 5 *

=== Resultado ===
Forma infixa: (((8 / 4) + 2) * 5)
Resultado: 20.0000
=================
```

1 - Conversão: A expressão já está na forma pós-fixa.
2 - Avaliação Pós-fixa:
    2.1 - Lê 8. Pilha: [8]
    2.2 - Lê 4. Pilha: [8, 4]

2.3 - Lê /. Desempilha 4 e 8, calcula 8 / 4 = 2, empilha o resultado. Pilha: [2]
2.4 - Lê 2. Pilha: [2, 2]
2.5 - Lê +. Desempilha 2 e 2, calcula 2 + 2 = 4, empilha o resultado. Pilha: [4]
2.6 - Lê 5. Pilha: [4, 5]
2.7 - Lê *. Desempilha 5 e 4, calcula 4 * 5 = 20, empilha o resultado. Pilha: [20]
3 - Fim da expressão. O resultado final é 20

3.3 Teste n° 03

```
========== AVALIADOR DE EXPRESSOES ==========
1. Avaliar expressao INFIXADA
2. Avaliar expressao POS-FIXADA
3. Rodar testes automaticos
0. Sair
=============================================
Digite o numero da opcao: 1

Digite a expressao INFIXADA (ex: (3 + 4) * 5):
> (30 sen + 60 cos) * 2

=== Resultado ===
Forma pos-fixada: 30 sen 60 cos + 2 *
Resultado: 2.0000
=================
```

3.4 Teste n° 04

```
========== AVALIADOR DE EXPRESSOES ==========
1. Avaliar expressao INFIXADA
2. Avaliar expressao POS-FIXADA
3. Rodar testes automaticos
0. Sair
=============================================
Digite o numero da opcao: 2

Digite a expressao POS-FIXADA (ex: 3 4 + 5 *):
> 100 log 2 ^

=== Resultado ===
Forma infixa: (log(100) ^ 2)
Resultado: 4.0000
=================
```

3.5 Teste n° 05

```
========== AVALIADOR DE EXPRESSOES ==========
1. Avaliar expressao INFIXADA
2. Avaliar expressao POS-FIXADA
3. Rodar testes automaticos
0. Sair
=============================================
Digite o numero da opcao: 1

Digite a expressao INFIXADA (ex: (3 + 4) * 5):
> (9 raiz + 45 tg) / 2

=== Resultado ===
Forma pos-fixada: 9 raiz 45 tg + 2 /
Resultado: 2.0000
=================
```

3.6 Teste n° 06

```
========== AVALIADOR DE EXPRESSOES ==========
1. Avaliar expressao INFIXADA
2. Avaliar expressao POS-FIXADA
3. Rodar testes automaticos
0. Sair
=============================================
Digite o numero da opcao: 2

Digite a expressao POS-FIXADA (ex: 3 4 + 5 *):
> 5 3 2 * + 9 - 3 /

=== Resultado ===
Forma infixa: (((5 + (3 * 2)) - 9) / 3)
Resultado: 0.6667
=================
```

4. Conclusão

Conforme nos aprofundamos na área de programação, vemos que existem diversas formas de calcular uma expressão matemática, e a forma pós-fixa foi um desses novos conhecimentos adquiridos com o estudo da programação.

Dito isso, ressalto que este projeto foi de crucial importância para nosso aprendizado sobre o tipo abstrato de dados “pilha”.

Acerca de dificuldades, notamos que a parte mais difícil do projeto foi organizar as pilhas para que pudessem ler parênteses, tanto na forma infixa quanto na pós-fixa, mas é com prazer que digo que passamos por essa dificuldade, após muito quebrar a cabeça.

Sobre melhorias, acredito que uma interface gráfica seria uma melhoria bem-vinda, para aprimorar a experiência do usuário, entretanto, teríamos que refazer o projeto em outra linguagem, porque o suporte que a linguagem de programação C tem para interfaces gráficas é quase nulo.

Em conclusão, agradeço novamente ao professor Marcelo Eustáquio por nos proporcionar essa jornada de aprendizado e aos integrantes do grupo.

Referências

Códigos disponibilizados pelo professor Marcelo Eustáquio no Ambiente Virtual do Aluno.

Anexos

calculadora.h

```
#ifndef CALCULADORA_H
#define CALCULADORA_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 "calculadora.h"
#include <ctype.h>

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
#define MAX 512

typedef struct {
   float itens[MAX];
   int topo;
} PilhaFloat;

void inicializaPilhaFloat(PilhaFloat *p) {
   p->topo = -1;
}

int pilhaFloatVazia(PilhaFloat *p) {
```

```
    return p->topo == -1;
}

int empilhaFloat(PilhaFloat *p, float val) {
    if (p->topo >= MAX - 1) return 0;
    p->itens[++p->topo] = val;
    return 1;
}

float desempilhaFloat(PilhaFloat *p) {
    if (pilhaFloatVazia(p)) {
        printf("Erro: pilha vazia!\n");
        return 0;
    }
    return p->itens[p->topo--];
}

typedef struct {
    char itens[MAX][MAX];
    int topo;
} PilhaString;

void inicializaPilhaStr(PilhaString *p) {
    p->topo = -1;
}

int empilhaStr(PilhaString *p, char *val) {
    if (p->topo >= MAX - 1) return 0;
    strcpy(p->itens[++p->topo], val);
    return 1;
}

char *desempilhaStr(PilhaString *p) {
    if (p->topo < 0) {
        printf("Erro: pilha vazia!\n");
        return NULL;
    }
    return p->itens[p->topo--];
}

float grausParaRadianos(float graus) {
    return graus * M_PI / 180.0;
}

float getValorPosFixa(char *StrPosFixa) {
    PilhaFloat p;
    inicializaPilhaFloat(&p);
```

```
char copia[MAX];
strcpy(copia, StrPosFixa);

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

while (token != NULL) {
   if (strcmp(token, "+") == 0) {
      float b = desempilhaFloat(&p);
      float a = desempilhaFloat(&p);
      empilhaFloat(&p, a + b);
   } else if (strcmp(token, "-") == 0) {
      float b = desempilhaFloat(&p);
      float a = desempilhaFloat(&p);
      empilhaFloat(&p, a - b);
   } else if (strcmp(token, "*") == 0) {
      float b = desempilhaFloat(&p);
      float a = desempilhaFloat(&p);
      empilhaFloat(&p, a * b);
   } else if (strcmp(token, "/") == 0) {
      float b = desempilhaFloat(&p);
      float a = desempilhaFloat(&p);
      if (b == 0) {
         printf("Erro: divisao por zero!\n");
         return INFINITY;
      }
      empilhaFloat(&p, a / b);
   } else if (strcmp(token, "%") == 0) {
      float b = desempilhaFloat(&p);
      float a = desempilhaFloat(&p);
      if (b == 0) {
         printf("Erro: modulo por zero!\n");
         return INFINITY;
      }
      empilhaFloat(&p, fmod(a, b));
   } else if (strcmp(token, "^") == 0) {
      float b = desempilhaFloat(&p);
      float a = desempilhaFloat(&p);
      empilhaFloat(&p, pow(a, b));
   } else if (strcmp(token, "raiz") == 0) {
      float a = desempilhaFloat(&p);
      if (a < 0) {
         printf("Erro: raiz de numero negativo!\n");
         return NAN;
      }
      empilhaFloat(&p, sqrt(a));
   } else if (strcmp(token, "log") == 0) {
      float a = desempilhaFloat(&p);
      if (a <= 0) {
```

```
            printf("Erro: logaritmo de numero nao positivo!\n");
            return NAN;
        }
        empilhaFloat(&p, log10(a));
    } else if (strcmp(token, "sen") == 0) {
        float a = desempilhaFloat(&p);
        empilhaFloat(&p, sin(grausParaRadianos(a)));
    } else if (strcmp(token, "cos") == 0) {
        float a = desempilhaFloat(&p);
        empilhaFloat(&p, cos(grausParaRadianos(a)));
    } else if (strcmp(token, "tg") == 0) {
        float a = desempilhaFloat(&p);
        double cosValor = cos(grausParaRadianos(a));
        if (fabs(cosValor) < 1e-10) {
            printf("Erro: tangente indefinida para %.2f graus!\n", a);
            return INFINITY;
        }
        empilhaFloat(&p, tan(grausParaRadianos(a)));
    } else {
        empilhaFloat(&p, atof(token));
    }

    token = strtok(NULL, " ");
  }

  if (p.topo != 0) {
    printf("Erro: expressao mal formada!\n");
    return NAN;
  }

  return desempilhaFloat(&p);
}

float getValorInFixa(char *StrInFixa) {
  char posfixa[MAX];
  strcpy(posfixa, getFormaPosFixa(StrInFixa));
  return getValorPosFixa(posfixa);
}

int prioridade(char *op) {
  if (strcmp(op, "sen") == 0 || strcmp(op, "cos") == 0 || strcmp(op, "tg") == 0 ||
      strcmp(op, "log") == 0 || strcmp(op, "raiz") == 0)
      return 5;
  if (strcmp(op, "^") == 0) return 4;
  if (strcmp(op, "*") == 0 || strcmp(op, "/") == 0 || strcmp(op, "%") == 0) return 3;
  if (strcmp(op, "+") == 0 || strcmp(op, "-") == 0) return 2;
  if (strcmp(op, "(") == 0) return 1;
  return 0;
```

```
}

// Coloque esta função no lugar da getFormaPosFixa() original em expressao.c

char *getFormaPosFixa(char *StrInFixa) {
   static char resultado[MAX] = "";
   resultado[0] = '\0';

   PilhaString operadores;
   inicializaPilhaStr(&operadores);

   int i = 0;
   while (StrInFixa[i] != '\0') {
      // Ignora espaços em branco
      if (isspace((unsigned char)StrInFixa[i])) {
         i++;
         continue;
      }

      // Se for um número (pode ter vários dígitos ou ser decimal)
      if (isdigit((unsigned char)StrInFixa[i]) || (StrInFixa[i] == '.' && isdigit((unsigned char)StrInFixa[i+1]))) {
         char numero[MAX];
         int k = 0;
         while (isdigit((unsigned char)StrInFixa[i]) || StrInFixa[i] == '.') {
            numero[k++] = StrInFixa[i++];
         }
         numero[k] = '\0';
         strcat(resultado, numero);
         strcat(resultado, " ");
         continue; // Continua a análise da string de entrada
      }

      char token[MAX];

      // Se for uma função (log, sen, cos, etc.)
      if (isalpha((unsigned char)StrInFixa[i])) {
         int k = 0;
         while (isalpha((unsigned char)StrInFixa[i])) {
            token[k++] = StrInFixa[i++];
         }
         token[k] = '\0';
      } else { // Se for um operador de um caractere ou parêntese
         token[0] = StrInFixa[i++];
         token[1] = '\0';
      }

      // Agora, 'token' contém um operador ou parêntese. Vamos processá-lo.
      if (strcmp(token, "(") == 0) {
```

```
            empilhaStr(&operadores, token);
        } else if (strcmp(token, ")") == 0) {
            while (operadores.topo >= 0 && strcmp(operadores.itens[operadores.topo], "(") != 0) {
                strcat(resultado, desempilhaStr(&operadores));
                strcat(resultado, " ");
            }
            if (operadores.topo < 0) { // Erro de parênteses
                printf("Erro: parenteses desalinhados na expressao!\n");
                resultado[0] = '\0';
                return resultado;
            }
            desempilhaStr(&operadores); // Descarta o "("
        } else { // É um operador
            while (operadores.topo >= 0 && strcmp(operadores.itens[operadores.topo], "(") != 0 &&
                   prioridade(token) <= prioridade(operadores.itens[operadores.topo])) {
                strcat(resultado, desempilhaStr(&operadores));
                strcat(resultado, " ");
            }
            empilhaStr(&operadores, token);
        }
    }

    // Desempilha os operadores restantes
    while (operadores.topo >= 0) {
        if (strcmp(operadores.itens[operadores.topo], "(") == 0) {
            printf("Erro: parenteses desalinhados na expressao!\n");
            resultado[0] = '\0';
            return resultado;
        }
        strcat(resultado, desempilhaStr(&operadores));
        strcat(resultado, " ");
    }

    // Remove o espaço extra no final, se houver
    int len = strlen(resultado);
    if (len > 0 && resultado[len - 1] == ' ') {
        resultado[len - 1] = '\0';
    }

    return resultado;
}

char *getFormaInFixa(char *StrPosFixa) {
    static char resultado[MAX] = "";
    resultado[0] = '\0';

    PilhaString pilha;
    inicializaPilhaStr(&pilha);
```

```
    char copia[MAX];
    strcpy(copia, StrPosFixa);

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

    while (token != NULL) {
        if (strcmp(token, "+") == 0 || strcmp(token, "-") == 0 ||
            strcmp(token, "*") == 0 || strcmp(token, "/") == 0 ||
            strcmp(token, "%") == 0 || strcmp(token, "^") == 0) {

            char *b = desempilhaStr(&pilha);
            char *a = desempilhaStr(&pilha);
            char temp[MAX];
            snprintf(temp, MAX, "(%s %s %s)", a, token, b);
            empilhaStr(&pilha, temp);

        } else if (strcmp(token, "sen") == 0 || strcmp(token, "cos") == 0 ||
                   strcmp(token, "tg") == 0 || strcmp(token, "log") == 0 ||
                   strcmp(token, "raiz") == 0) {

            char *a = desempilhaStr(&pilha);
            char temp[MAX];
            snprintf(temp, MAX, "%s(%s)", token, a);
            empilhaStr(&pilha, temp);

        } else {
            empilhaStr(&pilha, token);
        }

        token = strtok(NULL, " ");
    }

    strcpy(resultado, desempilhaStr(&pilha));
    return resultado;
}
```

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "calculadora.h"

void menu() {
    printf("\n========== AVALIADOR DE EXPRESSOES ==========\n");
    printf("1. Avaliar expressao INFIXADA\n");
```

```
    printf("2. Avaliar expressao POS-FIXADA\n");
    printf("3. Rodar testes automaticos\n");
    printf("0. Sair\n");
    printf("=============================================\n");
    printf("Digite o numero da opcao: ");
}

void testarExpressaoInfixa() {
    char entrada[512];
    printf("\nDigite a expressao INFIXADA (ex: (3 + 4) * 5):\n> ");
    fgets(entrada, sizeof(entrada), stdin);
    entrada[strcspn(entrada, "\n")] = '\0'; // remove o \n do final

    char posfixa[512];
    strcpy(posfixa, getFormaPosFixa(entrada));

    printf("\n=== Resultado ===\n");
    printf("Forma pos-fixada: %s\n", posfixa);
    printf("Resultado: %.4f\n", getValorInFixa(entrada));
    printf("=================\n");
}

void testarExpressaoPosfixa() {
    char entrada[512];
    printf("\nDigite a expressao POS-FIXADA (ex: 3 4 + 5 *):\n> ");
    fgets(entrada, sizeof(entrada), stdin);
    entrada[strcspn(entrada, "\n")] = '\0';

    printf("\n=== Resultado ===\n");
    printf("Forma infixa: %s\n", getFormaInFixa(entrada));
    printf("Resultado: %.4f\n", getValorPosFixa(entrada));
    printf("=================\n");
}

void testesAutomaticos() {
    printf("\n============ TESTAR AUTOMATICAMENTE ===========\n");

    char *testes[][3] = {
        {"3 4 + 5 *", "(3 + 4) * 5", "35.00"},
        {"7 2 * 4 +", "(7 * 2) + 4", "18.00"},
        {"8 5 2 4 + * +", "8 + 5 * (2 + 4)", "38.00"},
        {"6 2 / 3 + 4 *", "(6 / 2 + 3) * 4", "24.00"},
        {"9 5 2 8 * 4 + * +", "9 + 5 * (2 + 8 * 4)", "109.00"},
        {"2 3 + log 5 /", "log(2 + 3) / 5", "0.14"},
        {"10 log 3 ^ 2 +", "(log10)^3 + 2", "3.00"},
        {"45 60 + 30 cos *", "(45 + 60) * cos(30)", "90.93"},
        {"0.5 45 sen 2 ^ +", "0.5 + sen(45)^2", "1.00"}
    };
```

```
    const double margemErro = 0.01;

    for (int i = 0; i < 9; i++) {
        printf("\n------ Teste %d ------\n", i + 1);
        printf("Expressao pos-fixada: %s\n", testes[i][0]);

        char *infixa = getFormaInFixa(testes[i][0]);
        float valorCalculado = getValorPosFixa(testes[i][0]);
        float valorEsperado = atof(testes[i][2]);

        printf("Forma infixa esperada: %s\n", testes[i][1]);
        printf("Forma infixa obtida   : %s\n", infixa);
        printf("Valor esperado        : %.4f\n", valorEsperado);
        printf("Valor calculado       : %.4f\n", valorCalculado);

        if (fabs(valorCalculado - valorEsperado) < margemErro) {
            printf("? Resultado correto!\n");
        } else {
            printf("? Resultado incorreto!\n");
        }
    }
    printf("\n=========================================\n");
}

int main() {
    int opcao;
    do {
        menu();
        if (scanf("%d", &opcao) != 1) {
            printf("Entrada invalida. Encerrando.\n");
            break;
        }
        getchar();

        switch (opcao) {
            case 1:
                testarExpressaoInfixa();
                break;
            case 2:
                testarExpressaoPosfixa();
                break;
            case 3:
                testesAutomaticos();
                break;
            case 0:
                printf("\nEncerrando...\n");
                break;
```

```
            default:
                printf("\n? Opcao invalida. Tente novamente.\n");
        }
    } while (opcao != 0);

    return 0;
}
```