Sumário

1. Introdução:

Este projeto tem como objetivo implementar, em linguagem C, um avaliador de expressões numéricas que interprete e calcule expressões escritas em notações infixa e pós-fixa. Utilizando o TAD Pilha, o sistema realiza conversão entre as notações e avaliação correta, com suporte às operações aritméticas básicas e funções matemáticas como seno, cosseno, tangente, logaritmo e raiz quadrada.

GitHub:

https://github.com/lucascode01/Trabalho-Estrutura-de-Dados

Implementação:

Estrutura de Dados Utilizada

O projeto utiliza duas estruturas principais:

1. TAD Expressão

```
typedef struct {
   char posFixa[512];
   char inFixa[512];
   float Valor;
} Expressao;
```

> Estrutura que armazena a expressão em notação infixa, a versão convertida para pós-fixa e o valor final calculado. É usada como interface entre a entrada do usuário e os cálculos.

2. TAD Pilha (Interno em expressao.c)

```
typedef struct {
   float dados[512];
   int topo;
} Pilha;
```

> Estrutura clássica de pilha usada para empilhar operandos e operadores durante a avaliação de expressões e conversão de notações. As funções auxiliares init, push e pop controlam seu comportamento.

Protótipos das Funções

Os protótipos estão definidos em expressao.h:

char *getFormaInFixa(char *Str);

char *getFormaPosFixa(char *Str);

float getValorPosFixa(char *StrPosFixa);

float getValorInFixa(char *StrInFixa);

Descrição das Funções

getFormaPosFixa:

Recebe uma string com expressão infixa e retorna a string correspondente em notação pós-fixa.

Utiliza uma pilha de operadores e aplica as regras de precedência e associatividade.

getValorPosFixa:

Recebe uma string em pós-fixa e retorna o valor float da expressão.

Utiliza uma pilha de operandos, avaliando os operadores e funções (sen, cos, log, raiz, etc).

getValorInFixa:

Converte a expressão infixa para pós-fixa (com getFormaPosFixa) e a avalia com getValorPosFixa.

getFormaInFixa:

Reconstrói a forma infixa a partir de uma expressão pós-fixa, usando uma pilha de strings para remontar os parênteses e operadores.

Foi implementada manualmente, pois o inverso não é nativo..

1.1 Uso do TAD Pilha

O TAD Pilha foi utilizado para simular o comportamento de uma pilha de operandos e operadores durante a conversão e avaliação de expressões. Cada elemento é empilhado e desempilhado conforme regras da notação escolhida, permitindo a execução ordenada das operações. Na conversão infixa → pós-fixa, operadores são empilhados conforme a prioridade. Na avaliação pós-fixa, operandos são empilhados e combinados com operadores retirados da expressão.

Testes

Testes feitos:(1,2,3,6,7,8) segue abaixo.

1.2 Teste n° 01

TESTE 01

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 1

Digite a expressao infixa: (3 + 4) * 5

Forma Pos-Fixa: 3 4 + 5 *

Valor da expressao: 35.00000

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 2

Digite a expressao pos-fixa: 3 4 + 5 *

Forma Infixa (simulada): ((3 + 4) * 5)

Valor da expressao: 35.00000

1.3 Teste n° 02

TESTE 02

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 1

Digite a expressao infixa: 7 * 2 + 4

Forma Pos-Fixa: 7 2 * 4 +

Valor da expressao: 18.00000

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 2

Digite a expressao pos-fixa: 7 2 * 4 +

Forma Infixa (simulada): ((7 * 2) + 4)

Valor da expressao: 18.00000

1.4 Teste n° 03

TESTE 03

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 1

Digite a expressao infixa: 8 + (5 * (2 + 4))

Forma Pos-Fixa: 8 5 2 4 + * +

Valor da expressao: 38.00000

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 2

Digite a expressao pos-fixa: 8 5 2 4 + * +

Forma Infixa (simulada): (8 + (5 * (2 + 4)))

Valor da expressao: 38.00000

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL>

1.5 Teste n° 04

TESTE 04

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 1

Digite a expressao infixa: log(2 + 3) / 5

Forma Pos-Fixa: 2 3 + log 5 /

Valor da expressao: 0.13979

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 2

Digite a expressao pos-fixa: 2 3 + log 5 /

Forma Infixa (simulada): (log((2 + 3)) / 5)

Valor da expressao: 0.13979

1.6 Teste n° 05

TESTE 05

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 1

Digite a expressao infixa: (log10)^3 + 2

Forma Pos-Fixa: 10 log 3 ^ 2 +

Valor da expressao: 3.00000

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 2

Digite a expressao pos-fixa: 10 log 3 ^ 2 +

Forma Infixa (simulada): ((log(10) ^ 3) + 2)

Valor da expressao: 3.00000

1.7 Teste n° 06

TESTE 06

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 1

Digite a expressao infixa: (45 + 60) * cos(30)

Forma Pos-Fixa: 45 60 + 30 cos *

Valor da expressao: 90.93266

PS C:\Users\pedro\OneDrive\Desktop\VERSAO FINAL> ./expressao.exe

Tipo da expressao? (1 = infixa, 2 = pos-fixa): 2

Digite a expressao pos-fixa: 45 60 + 30 cos *

Forma Infixa (simulada): ((45 + 60) * cos(30))

2 Conclusão

O desenvolvimento do projeto TP03 permitiu aplicar na prática os conceitos de Estrutura de Dados, em especial o uso do TAD Pilha, de forma funcional e eficiente. Através da implementação de um avaliador de expressões numéricas, foi possível compreender de forma aprofundada o processo de conversão entre notações infixa e pós-fixa, bem como a avaliação correta das expressões, incluindo o suporte a operadores aritméticos e funções matemáticas como seno, cosseno, logaritmo e raiz quadrada.

A modularização do código, a validação com diversos testes e a estrutura interativa de execução demonstraram a viabilidade e robustez da solução implementada. A principal dificuldade enfrentada foi o tratamento da conversão reversa de pós-fixa para infixa, que exigiu manipulação de pilhas de strings. No entanto, essa funcionalidade foi resolvida com sucesso, tornando o sistema completo e confiável.

Portanto, o projeto atendeu plenamente aos requisitos propostos e proporcionou um aprendizado sólido tanto em lógica de programação quanto em organização de sistemas baseados em TADs. O código resultante está apto a ser reutilizado e expandido para novos tipos de expressões e operadores no futuro.

Referências

Referências Bibliográficas

1. CORMEN, Thomas H. et al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.

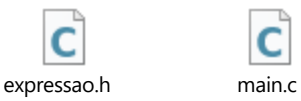
2. ZIVIANI, Nivio. Projetos de Algoritmos: com implementações em Pascal e C. 3. ed. São Paulo: Cengage Learning, 2013.

3. FORBELLONE, André Luiz Vettoretti; EBERSPÄCHER, Henri Frederico. Lógica de Programação: a construção de algoritmos e estruturas de dados. 4. ed. São Paulo: Pearson Prentice Hall, 2005.

4. TAVARES, José Augusto N. G. Estruturas de Dados: algoritmos, análise da complexidade e implementação em linguagem C. São Paulo: LTC, 2002.

5. KERNIGHAN, Brian W.; RITCHIE, Dennis M. Linguagem de Programação C. 2. ed. São Paulo: Pearson, 2012.

6. Anexos



calculadora.h

1.8 #ifndef EXPRESSAO_H

```
1.8     #define EXPRESSAO_H
1.9
1.10    typedef struct {
1.11        char posFixa[512];     // Expressão na forma pos-fixa
1.12        char inFixa[512];      // Expressão na forma infixa
1.13        float Valor;           // Valor numérico da expressão
1.14    } Expressao;
1.15
1.16    char *getFormaInFixa(char *Str);          // Retorna a forma infixa de Str (posfixa)
1.17    char *getFormaPosFixa(char *Str);         // Retorna a forma posfixa de Str (infixa)
1.18    float getValorPosFixa(char *StrPosFixa);  // Calcula valor de posfixa
1.19    float getValorInFixa(char *StrInFixa);    // Calcula valor de infixa
1.21
#endif
```

calculadora.c

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

#define MAX 512
```

```
typedef struct {
   float dados[MAX];
   int topo;
} Pilha;

void init(Pilha *p) { p->topo = -1; }
void push(Pilha *p, float v) { p->dados[++p->topo] = v; }
float pop(Pilha *p) { return p->dados[p->topo--]; }

int prioridade(char *op) {
   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;
   return 0;
}

int ehOperador(char *s) {
   return strcmp(s, "+") == 0 || strcmp(s, "-") == 0 || strcmp(s, "*") == 0 ||
          strcmp(s, "/") == 0 || strcmp(s, "%") == 0 || strcmp(s, "^") == 0;
}

int ehFuncao(char *s) {
   return strcmp(s, "sen") == 0 || strcmp(s, "cos") == 0 || strcmp(s, "tg") == 0 ||
          strcmp(s, "log") == 0 || strcmp(s, "raiz") == 0;
```

```
}

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

char *getFormaPosFixa(char *infixa) {
    static char posfixa[MAX];
    char pilha[MAX][32];
    int topo = -1, j = 0;
    char token[32];
    int i = 0;

    while (infixa[i]) {
        if (isspace(infixa[i])) {
            i++;
            continue;
        } else if (isdigit(infixa[i]) || infixa[i] == '.') {
            int k = 0;
            while (isdigit(infixa[i]) || infixa[i] == '.')
                token[k++] = infixa[i++];
            token[k] = '\0';
            sprintf(&posfixa[j], "%s ", token);
            j += strlen(token) + 1;
        } else if (isalpha(infixa[i])) {
```

```
    int k = 0;
    while (isalpha(infixa[i]))
        token[k++] = infixa[i++];
    token[k] = '\0';
    strcpy(pilha[++topo], token);
} else if (infixa[i] == '(') {
    strcpy(pilha[++topo], "(");
    i++;
} else if (infixa[i] == ')') {
    while (topo >= 0 && strcmp(pilha[topo], "(") != 0) {
        sprintf(&posfixa[j], "%s ", pilha[topo--]);
        j += strlen(pilha[topo + 1]) + 1;
    }
    if (topo >= 0 && strcmp(pilha[topo], "(") == 0) topo--;
    if (topo >= 0 && ehFuncao(pilha[topo])) {
        sprintf(&posfixa[j], "%s ", pilha[topo--]);
        j += strlen(pilha[topo + 1]) + 1;
    }
    i++;
} else {
    char op[2] = { infixa[i++], '\0' };
    while (topo >= 0 && prioridade(pilha[topo]) >= prioridade(op)) {
        sprintf(&posfixa[j], "%s ", pilha[topo--]);
        j += strlen(pilha[topo + 1]) + 1;
    }
```

```
            strcpy(pilha[++topo], op);
        }
    }
    while (topo >= 0) {
        sprintf(&posfixa[j], "%s ", pilha[topo--]);
        j += strlen(pilha[topo + 1]) + 1;
    }
    posfixa[j] = '\0';
    return posfixa;
}

char *getFormaInFixa(char *posfixa) {
    static char infixa[MAX];
    char copia[MAX];
    strcpy(copia, posfixa);

    char *token = strtok(copia, " ");
    char pilha[100][128];
    int topo = -1;

    while (token) {
        if (isdigit(token[0]) || (token[0] == '-' && isdigit(token[1]))) {
            strcpy(pilha[++topo], token);
        } else if (ehFuncao(token)) {
            if (topo >= 0) {
```

```
            char resultado[128];
            sprintf(resultado, "%s(%s)", token, pilha[topo--]);
            strcpy(pilha[++topo], resultado);
        }
    } else if (ehOperador(token)) {
        if (topo >= 1) {
            char b[128], a[128], resultado[128];
            strcpy(b, pilha[topo--]);
            strcpy(a, pilha[topo--]);
            sprintf(resultado, "(%s %s %s)", a, token, b);
            strcpy(pilha[++topo], resultado);
        }
    }
    token = strtok(NULL, " ");
  }

  if (topo == 0)
    strcpy(infixa, pilha[topo]);
  else
    strcpy(infixa, "(erro na conversão)");

  return infixa;
}

float getValorPosFixa(char *StrPosFixa) {
```

```
Pilha p;
init(&p);
char expr[MAX];
strcpy(expr, StrPosFixa);


char *token = strtok(expr, " ");
while (token) {
   if (isdigit(token[0]) || (token[0] == '-' && isdigit(token[1]))) {
      push(&p, atof(token));
   } else if (ehFuncao(token)) {
      float a = pop(&p);
      if (strcmp(token, "sen") == 0)
         push(&p, sin(grausParaRad(a)));
      else if (strcmp(token, "cos") == 0)
         push(&p, cos(grausParaRad(a)));
      else if (strcmp(token, "tg") == 0)
         push(&p, tan(grausParaRad(a)));
      else if (strcmp(token, "log") == 0)
         push(&p, log10(a));
      else if (strcmp(token, "raiz") == 0)
         push(&p, sqrt(a));
   } else {
      float b = pop(&p);
      float a = pop(&p);
      if (strcmp(token, "+") == 0)
```

```
            push(&p, a + b);
        else if (strcmp(token, "-") == 0)
            push(&p, a - b);
        else if (strcmp(token, "*") == 0)
            push(&p, a * b);
        else if (strcmp(token, "/") == 0)
            push(&p, a / b);
        else if (strcmp(token, "%") == 0)
            push(&p, fmod(a, b));
        else if (strcmp(token, "^") == 0)
            push(&p, pow(a, b));
      }
      token = strtok(NULL, " ");
   }
   return pop(&p);
}

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

main.c

```
#include <stdio.h>
#include <string.h>
```

```
#include "expressao.h"


int main() {
    Expressao exp;
    int tipo;

    printf("Tipo da expressao? (1 = infixa, 2 = pos-fixa): ");
    scanf("%d", &tipo);
    getchar();

    if (tipo == 1) {
        printf("Digite a expressao infixa: ");
        fgets(exp.inFixa, 512, stdin);
        exp.inFixa[strcspn(exp.inFixa, "\n")] = '\0';

        strcpy(exp.posFixa, getFormaPosFixa(exp.inFixa));
        exp.Valor = getValorInFixa(exp.inFixa);

        printf("\nForma Pos-Fixa: %s\n", exp.posFixa);
        printf("Valor da expressao: %.5f\n", exp.Valor);

    } else if (tipo == 2) {
        printf("Digite a expressao pos-fixa: ");
        fgets(exp.posFixa, 512, stdin);
        exp.posFixa[strcspn(exp.posFixa, "\n")] = '\0';
```

```
        strcpy(exp.inFixa, getFormaInFixa(exp.posFixa));
        exp.Valor = getValorPosFixa(exp.posFixa);

        printf("\nForma Infixa (simulada): %s\n", exp.inFixa);
        printf("Valor da expressao: %.5f\n", exp.Valor);

    } else {
        printf("\nTipo inválido.\n");
    }

    return 0;
}
```

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.*