Sumário

[1. Introdução: 2](#_Toc181025009)

[2. Implementação: 2](#_Toc181025010)

[3. Testes 2](#_Toc181025011)

[3.1 Teste n° 01 2](#_Toc181025012)

[3.2 Teste n° 02 2](#_Toc181025013)

[3.3 Teste n° 03 2](#_Toc181025014)

[3.4 Teste n° 04 2](#_Toc181025015)

[3.5 Teste n° 05 3](#_Toc181025016)

[4. Conclusão 3](#_Toc181025017)

[Referências 3](#_Toc181025018)

[Anexos 3](#_Toc181025019)

[calculadora.h 4](#_Toc181025020)

[calculadora.c 4](#_Toc181025021)

[main.c 4](#_Toc181025022)

## Introdução:

Este trabalho aborda a implementação de operações matemáticas usando a notação polonesa inversa (RPN), ou notação pós-fixa, que facilita o cálculo de expressões sem a necessidade de parênteses para definir a precedência de operações. O problema proposto consiste em processar e calcular o valor de expressões matemáticas em RPN, convertê-las para notação infixa e verificar se os resultados calculados correspondem aos valores de referência fornecidos. O principal objetivo é desenvolver estruturas de dados e algoritmos eficientes para avaliar expressões matemáticas complexas, incluindo operações aritméticas básicas e funções trigonométricas.

#### GitHub:­

https://github.com/lucasdaher/edd-calculadora

## Implementação:

Para a resolução deste problema, foram implementadas estrutura de dados e funções auxiliares que auxiliam na manipulação de expressões matemáticas:

1. **Estrutura de Dados Utilizadas:**
   1. **Pilha**: Esta estrutura, implementada como uma pilha de números (**float**), é usada para armazenar os operandos durante o cálculo da expressão em notação pós-fixa.
   2. **PilhaString:** Similar à pilha numérica, esta estrutura armazena strings e é utilizada para manipular os operandos e operadores ao converter uma expressão da forma pós-fixa para a forma infixa.
   3. **Expressao:** Estrutura principal que armazena a expressão na forma pós-fixa (**posFixa**), sua forma infixa (**inFixa**) e o valor de referência (**valor**).
2. **Protótipos de Funções e Suas Funções:**
   1. **void empilhar(Pilha \*pilha, float valor):** Função que insere um valor na pilha numérica, controlando o topo.
   2. **float desempilhar(Pilha \*pilha):** Remove e retorna o valor no topo da pilha numérica, decrementando o índice do topo.
   3. **void empilharString(PilhaString \*pilha, const char \*valor):** Adiciona uma string ao topo da pilha de strings.
   4. **char \*desempilharString(PilhaString \*pilha):** Retira e retorna a string do topo da pilha de strings.
   5. **int isOperador(char ch):** Verifica se o caractere é um operador aritmético.
   6. **int isFuncao(char \*token):** Verifica se o token é uma função matemática, como **sen**, **cos** ou **log**.
   7. **float verificarOperador(char operador, float primeiroValor, float segundoValor):** Executa a operação especificada entre dois operadores.
   8. **float verificarFuncao(char \*operador, float numero):** Calcula o valor de uma função matemática com base no operador e número fornecidos.
   9. **float getValor(char \*str):** Avalia a expressão em RPN e retorna o valor calculado.
   10. **char \*getFormaInFixa(char \*str):** Converte uma expressão de RPN para notação infixa.

## Testes

*(Devem ser descritos os testes realizados, mostrando a saída do programa, além de eventuais análises que sejam solicitadas no enunciado. Aqui recomenda-se colar o recorte, usando a ferramenta de captura do Windows, do campo “terminal” do VSCode*

*A seguir, deve ser estruturado um teste, com tamanho similar ao que foi indicado acima, para cada aluno do grupo, incluindo conversão para a notação infixa, resultado da expressão e detalhamento na tabela. No final, atualize o índice.*

### Teste n° 01

**Expressão: 6 3 \* 4 +**

|  |  |  |
| --- | --- | --- |
| **Passo** | **Ação** | **Pilha** |
| 1 | Lê 6 e empilha. | [6] |
| 2 | Lê 3 e empilha. | [6, 3] |
| 3 | Lê \*, desempilha os últimos valores, calcula 6 \* 3 = 18 e empilha 18. | [18] |
| 4 | Lê 4 e empilha. | [18, 4] |
| 5 | Lê +, desempilha os últimos valores, calcula 18 + 4 = 22 e empilha 22. | [22] |

### Teste n° 02

**Expressão: 10 3 4 \* + 5 -**

|  |  |  |
| --- | --- | --- |
| Passo | Ação | Pilha |
| 1 | Lê 10 e empilha. | [10] |
| 2 | Lê 3 e empilha. | [10, 3] |
| 3 | Lê 4 e empilha. | [10, 3, 4] |
| 4 | Lê \*, desempilha os últimos valores, calcula 3 \* 4 = 12 e empilha 12. | [10, 12] |
| 5 | Lê +, desempilha os últimos valores, calcula 10 + 12 = 22 e empilha 22. | [22] |
| 6 | Lê 5 e empilha. | [22, 5] |
| 7 | Lê -, desempilha os últimos valores, calcula 22 - 5 e empilha 17. | [17] |

### Teste n° 03

**Expressão: 8 5 6 2 \* + +**

|  |  |  |
| --- | --- | --- |
| **Passo** | **Ação** | **Pilha** |
| 1 | Lê 8 e empilha. | [8] |
| 2 | Lê 5 e empilha. | [8, 5] |
| 3 | Lê 6 e empilha. | [8, 5, 6] |
| 4 | Lê 2 e empilha. | [8, 5, 6, 2] |
| 5 | Lê \*, desempilha os últimos valores, calcula 6 \* 2 = 12 e empilha 12. | [8, 5, 12] |
| 6 | Lê +, desempilha os últimos valores, calcula 5 + 12 = 17 e empilha 17. | [8, 17] |
| 7 | Lê +, desempilha os últimos valores, calcula 8 + 17 = 25 e empilha. | [25] |

### 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)*

## Conclusão

O trabalho de implementação de um sistema para avaliação de expressões matemáticas em notação pós-fixa utilizando estrutura de dados em C proporcionou uma visão prática dos conceitos fundamentais de manipulação de pilhas e da lógica de cálculo em notação polonesa inversa. Os resultados obtidos confirmaram a precisão dos cálculos e das conversões entre as notações pós-fixa e infixa, além de evidenciarem a eficiência do uso de pilhas para gerenciar a ordem de operações matemáticas. Durante a implementação, uma das principais dificuldades enfrentadas foi a gestão adequada das operações envolvendo funções matemáticas e a conversão precisa das expressões para a forma infixa. Além disso, a manipulação de strings para representar operações complexas exigiu cuidado adicional para evitar erros de pilha, como estouro ou esvaziamento incorreto. Para futuras melhorias, poderiam ser adicionadas otimizações na gestão de memória e no tratamento de erros, além de uma implementação mais robusta para suportar um conjunto maior de operações e funções matemáticas. Outras possíveis extensões incluem a adaptação do programa para trabalhar com expressões em outras notações, como a prefixa, e a introdução de uma interface mais interativa para o usuário. Essas melhorias tornariam o sistema ainda mais eficiente e versátil para o processamento de expressões matemáticas complexas.

## Referências

ISO/IEC 9899:2018. *Programming Languages – C*. Padrão internacional para a linguagem C, que fornece especificações para uso de estruturas, ponteiros e manipulação de strings.

GNU C Library, disponível em: https://www.gnu.org/software/libc/. Acesso em: 13 nov. 2024.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. **Algoritmos: Teoria e Prática. 3ª edição**. Elsevier, 2012.

FOROUZAN, Behrouz A.; GILBERG, Richard F. **Estrutura de Dados: Uma Abordagem Prática com Algoritmos em C**. Cengage Learning, 2008.

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

## 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 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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

#include "expressao.h"

#ifndef M\_PI

#define M\_PI 3.14159265358979323846

#endif

#define MAXIMO\_PILHA 512

typedef struct

{

  float itens[MAXIMO\_PILHA];

  int topo;

} Pilha;

typedef struct

{

  char itens[MAXIMO\_PILHA][512];

  int topo;

} PilhaString;

void empilhar(Pilha \*pilha, float valor)

{

  if (pilha->topo < MAXIMO\_PILHA - 1)

  {

    pilha->itens[++pilha->topo] = valor;

  }

  else

  {

    perror("Erro encontrado: A pilha esta cheia.\n");

  }

}

float desempilhar(Pilha \*pilha)

{

  if (pilha->topo >= 0)

  {

    return pilha->itens[pilha->topo--];

  }

  else

  {

    perror("Erro encontrado: A pilha esta vazia.\n");

    return 0;

  }

}

void empilharString(PilhaString \*pilha, const char \*valor)

{

  if (pilha->topo < MAXIMO\_PILHA - 1)

  {

    strcpy(pilha->itens[++pilha->topo], valor);

  }

  else

  {

    perror("Erro encontrado: A pilha de string esta cheia.\n");

  }

}

char \*desempilharString(PilhaString \*pilha)

{

  if (pilha->topo >= 0)

  {

    return pilha->itens[pilha->topo--];

  }

  else

  {

    perror("Erro encontrado: A pilha de string esta vazia.\n");

    return NULL;

  }

}

int isOperador(char ch)

{

  return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^');

}

int isFuncao(char \*token)

{

  return (!strcmp(token, "sen") || !strcmp(token, "cos") || !strcmp(token, "tg") ||

          !strcmp(token, "log") || !strcmp(token, "raiz"));

}

float verificarOperador(char operador, float primeiroValor, float segundoValor)

{

  switch (operador)

  {

  case '+':

    return primeiroValor + segundoValor;

  case '-':

    return primeiroValor - segundoValor;

  case '\*':

    return primeiroValor \* segundoValor;

  case '/':

    return primeiroValor / segundoValor;

  case '^':

    return pow(primeiroValor, segundoValor);

  default:

    return 0;

  }

}

float verificarFuncao(char \*operador, float numero)

{

  if (strcmp(operador, "sen") == 0)

    return sin(numero \* M\_PI / 180);

  if (strcmp(operador, "cos") == 0)

    return cos(numero \* M\_PI / 180);

  if (strcmp(operador, "tg") == 0)

    return tan(numero \* M\_PI / 180);

  if (strcmp(operador, "log") == 0)

    return log10(numero);

  if (strcmp(operador, "raiz") == 0)

    return sqrt(numero);

  return 0;

}

float getValor(char \*str)

{

  Pilha pilha;

  pilha.topo = -1;

  char \*token = strtok(str, " ");

  while (token != NULL)

  {

    if (isdigit(token[0]) || (token[0] == '-' && isdigit(token[1])))

    {

      empilhar(&pilha, atof(token));

    }

    else if (isOperador(token[0]))

    {

      float b = desempilhar(&pilha);

      float a = desempilhar(&pilha);

      empilhar(&pilha, verificarOperador(token[0], a, b));

    }

    else if (isFuncao(token))

    {

      float numero = desempilhar(&pilha);

      empilhar(&pilha, verificarFuncao(token, numero));

    }

    token = strtok(NULL, " ");

  }

  return desempilhar(&pilha);

}

char \*getFormaInFixa(char \*str)

{

  PilhaString pilha;

  pilha.topo = -1;

  char \*token = strtok(str, " ");

  static char resultado[512];

  while (token != NULL)

  {

    if (isdigit(token[0]) || (token[0] == '-' && isdigit(token[1])))

    {

      empilharString(&pilha, token);

    }

    else if (isOperador(token[0]))

    {

      char b[512], a[512];

      strcpy(b, desempilharString(&pilha));

      strcpy(a, desempilharString(&pilha));

      sprintf(resultado, "(%s %s %s)", a, token, b);

      empilharString(&pilha, resultado);

    }

    else if (isFuncao(token))

    {

      char a[512];

      strcpy(a, desempilharString(&pilha));

      sprintf(resultado, "%s(%s)", token, a);

      empilharString(&pilha, resultado);

    }

    token = strtok(NULL, " ");

  }

  strcpy(resultado, desempilharString(&pilha));

  return resultado;

}

### main.c

int main() {

    Expressao expressoes[] = {

      {"3 4 + 5 \*", "(3 + 4) \* 5", 35},

      {"7 2 \* 4 +", "7 \* 2 + 4", 18},

      {"8 5 2 4 + \* +", "8 + 5 \* (2 + 4)", 38},

      {"6 2 / 3 + 4 \*", "(6 / 2 + 3) \* 4", 24},

      {"9 5 2 8 \* 4 + \* +", "9 + 5 \* (2 + 8 \* 4)", 109},

      {"2 3 + log 5 /", "log(2 + 3) / 5", 0.14},

      {"10 log 3 ^ 2 +", "(log 10)^3 + 2", 3},

      {"45 60 + 30 cos \*", "(45 + 60) \* cos(30)", 90.93},

      {"0.5 45 sen 2 ^ +", "sen(45)^2 + 0.5", 1}};

  for (int i = 0; i < 9; i++)

  {

    float resultado = getValor(expressoes[i].posFixa);

printf("Expressao %s\nResultado - %.2f\nGabarito - %.2f\n\n", expressoes[i].posFixa, resultado, expressoes[i].Valor);

  }

  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. *O trabalho 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 deve ser indicado no rodapé.* 9. *As notas serão disponibilizadas em área específica do AVA.* |