Sumário

[1. Introdução: 2](#_Toc152192803)

[2. Implementação: 2](#_Toc152192804)

[3. Testes 2](#_Toc152192805)

[4. Conclusão 3](#_Toc152192806)

[Referências 3](#_Toc152192807)

[Anexos 3](#_Toc152192808)

[Calculadora2.0.c 3](#_Toc152192809)

## Introdução:

O código em C realizado pelo nosso grupo implementa uma calculadora simples que é capaz de avaliar expressões matemáticas básicas, incluindo operações aritméticas (+, -, \*, /), potenciação (^) e funções trigonométricas (seno, cosseno, tangente), logaritmo (log) e algumas verificações de erro.

#### GitHub:

https://github.com/Andrade098/ManipulandoArquivos/blob/main/TP07%20-%20Calculadora/calculadora2\_0.c

## Implementação:

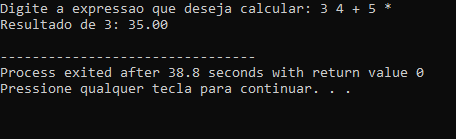
O código em C implementa uma calculadora que avalia expressões matemáticas básicas, incluindo operações aritméticas, potenciação, logaritmo, seno, cosseno e tangente. Abaixo estão alguns pontos-chave:

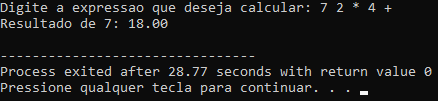
1. Estrutura de Dados: Utiliza uma estrutura chamada No para representar os elementos de uma pilha, contendo um valor (do tipo float) e um ponteiro para o próximo nó.
2. Funções Principais:   
     
   empilhar: Adiciona um novo nó à pilha.   
     
   Desempilhar: Remove o nó do topo da pilha.   
     
   operacao: Realiza operações matemáticas e trigonométricas com base nos parâmetros passados.   
     
   resolver\_expressao: Analisa a expressão, realiza as operações e retorna o resultado.
3. Decisões de Implementação:

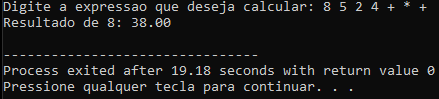
Usa uma pilha para armazenar temporariamente os valores durante a avaliação da expressão.

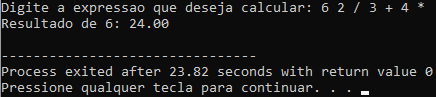
Trata operações unárias (log, sen, cos, tan) e binárias (soma, subtração, multiplicação, divisão, potenciação).

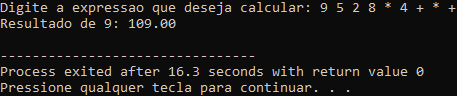
## Testes

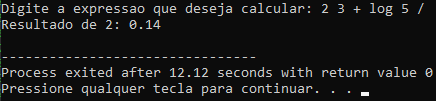


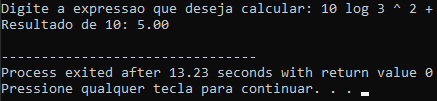


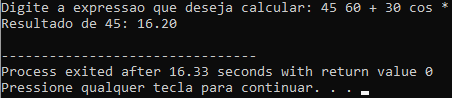


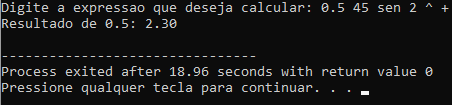


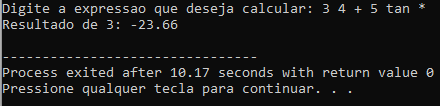












## Conclusão

O código que foi feito oferece uma implementação funcional de uma calculadora para expressões matemáticas básicas, com tratamento de erros para expressões mau formadas ou operadores sem operandos suficientes. O uso de uma pilha é fundamental para avaliar a expressão de maneira eficiente, mantendo o controle sobre os valores e operadores durante o processo de avaliação.

## Referências

https://cplusplus.com/reference/cmath/log10/

https://cplusplus.com/reference/cmath/sin/

https://cplusplus.com/reference/cmath/cos/

https://cplusplus.com/reference/cmath/tan/

## Anexos



### Calculadora2.0.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

typedef struct no {

float valor;

struct no \*proximo;

} No;

No \*empilhar(No \*pilha, float num) {

No \*novo = malloc(sizeof(No));

if (novo) {

novo->valor = num;

novo->proximo = pilha;

return novo;

} else {

printf("\tErro ao alocar memoria!\n");

exit(EXIT\_FAILURE);

}

}

No \*desempilhar(No \*\*pilha) {

No \*remover = NULL;

if (\*pilha) {

remover = \*pilha;

\*pilha = remover->proximo;

} else {

printf("\tPilha vazia\n");

exit(EXIT\_FAILURE);

}

return remover;

}

float operacao(float a, float b, char x) {

switch (x) {

case '+':

return a + b;

case '-':

return a - b;

case '/':

return a / b;

case '\*':

return a \* b;

case '^':

return pow(b, a);

case 'l': // log

return log10(a);

case 's': // sin

return sin(a);

case 'c': // cos

return cos(a);

case 't': // tan

return tan(a);

default:

return 0.0;

}

}

float resolver\_expressao(char x[]) {

char \*pt;

float num;

No \*n1, \*pilha = NULL;

pt = strtok(x, " ");

while (pt) {

if (pt[0] == '+' || pt[0] == '-' || pt[0] == '/' || pt[0] == '\*' || pt[0] == '^' ||

(pt[0] == 'l' && pt[1] == 'o' && pt[2] == 'g') ||

(pt[0] == 's' && pt[1] == 'e' && pt[2] == 'n') ||

(pt[0] == 'c' && pt[1] == 'o' && pt[2] == 's') ||

(pt[0] == 't' && pt[1] == 'a' && pt[2] == 'n')) {

if (pilha == NULL) {

printf("\tErro: operador sem operandos suficientes.\n");

exit(EXIT\_FAILURE);

}

n1 = desempilhar(&pilha);

// Se o operador for log, sen, cos ou tan, eles esperam apenas um operando

if ((pt[0] == 'l' && pt[1] == 'o' && pt[2] == 'g') ||

(pt[0] == 's' && pt[1] == 'e' && pt[2] == 'n') ||

(pt[0] == 'c' && pt[1] == 'o' && pt[2] == 's') ||

(pt[0] == 't' && pt[1] == 'a' && pt[2] == 'n')) {

num = operacao(n1->valor, 0, pt[0]);

} else {

// Para outros operadores, desempilhar outro operando

if (pilha == NULL) {

printf("\tErro: operador sem operandos suficientes.\n");

exit(EXIT\_FAILURE);

}

No \*n2 = desempilhar(&pilha);

num = operacao(n2->valor, n1->valor, pt[0]);

free(n2);

}

pilha = empilhar(pilha, num);

free(n1);

} else {

num = strtof(pt, NULL);

pilha = empilhar(pilha, num);

}

pt = strtok(NULL, " ");

}

if (pilha == NULL || pilha->proximo != NULL) {

printf("\tErro: expressao mal formada.\n");

exit(EXIT\_FAILURE);

}

n1 = desempilhar(&pilha);

num = n1->valor;

free(n1);

return num;

}

int main() {

char exp[50];

printf("Digite a expressao que deseja calcular: ");

fgets(exp, sizeof(exp), stdin);

exp[strcspn(exp, "\n")] = '\0'; // Remover a quebra de linha do final

printf("Resultado de %s: %.2f\n", exp, resolver\_expressao(exp));

return 0;

}