Sumário

1. Introdução:

Neste projeto, implementaremos um programa para avaliar expressões matemáticas utilizando a notação pós-fixada, também conhecida como notação polonesa reversa (RPN - Reverse Polish Notation). Esta notação coloca os operadores após seus operandos, simplificando a avaliação de expressões ao eliminar a necessidade de parênteses para definir a ordem das operações. O objetivo é criar um avaliador de expressões que realize operações matemáticas básicas e funções especiais, garantindo eficiência e precisão.

GitHub:

O código-fonte completo deste projeto está disponível no seguinte repositório GitHub:

https://github.com/fiorellizz/Atividades-UCB/tree/main/EstruturaDados/Projetos/Calculadora

2. Implementação:

A implementação utiliza uma estrutura de dados do tipo pilha para avaliar as expressões em notação pós-fixada. Abaixo estão os detalhes das estruturas de dados utilizadas e os protótipos das funções:

Estrutura de dados

```c
#ifndef EXPRESSAO_H
#define EXPRESSAO_H

#define MAX_EXPR_SIZE 512
#define MAX_STACK_SIZE 100

typedef struct {
    char posFixa[MAX_EXPR_SIZE];     // Expressão na forma pós-fixada, como 3 12 4 + *
    char inFixa[MAX_EXPR_SIZE];      // Expressão na forma infixada, como 3 * (12 + 4)
    float Valor;                     // Valor numérico da expressão
} Expressao;

int is_operator(const char* token);
char *getFormaInFixa(char *Str);    // Retorna a forma infixa de Str (pós-fixada)
char *pop(char stack[][MAX_EXPR_SIZE], int *top);
float getValor(char *Str);          // Calcula o valor de Str (na forma pós-fixada)
float apply_operator(const char* token, float a, float b);
void push(char stack[][MAX_EXPR_SIZE], int *top, const char* value);
void executarTeste(char *expr, int numeroTeste);

#endif
```

Protótipos de Funções

```c
int is_operator(const char* token);
char *getFormaInFixa(char *Str);    // Retorna a forma infixa de Str (pós-fixada)
char *pop(char stack[][MAX_EXPR_SIZE], int *top);
float getValor(char *Str);          // Calcula o valor de Str (na forma pós-fixada)
float apply_operator(const char* token, float a, float b);
void push(char stack[][MAX_EXPR_SIZE], int *top, const char* value);
void executarTeste(char *expr, int numeroTeste);
```

Implementação das Funções em “calculadora.c”

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

#define ERRO 1
```

```c
void push(char stack[][MAX_EXPR_SIZE], int *top, const char* value) {
    if (*top < MAX_STACK_SIZE - 1) {
        strcpy(stack[++(*top)], value);
    } else {
        printf("ERRO: Limite de pilha\n");
        exit(ERRO);
    }
}

char* pop(char stack[][MAX_EXPR_SIZE], int *top) {
    if (*top >= 0) {
        return stack[(*top)--];
    } else {
        printf("ERRO: Subfluxo de pilha - Tentativa de desempilhar de uma pilha vazia\n");
        exit(ERRO);
    }
}

int is_operator(const char* token) {
    return strcmp(token, "+") == 0 || strcmp(token, "-") == 0 || strcmp(token, "*") == 0
|| strcmp(token, "/") == 0 || strcmp(token, "^") == 0;
}

float apply_operator(const char* token, float a, float b) {
    if (strcmp(token, "+") == 0) return a + b;
    if (strcmp(token, "-") == 0) return a - b;
    if (strcmp(token, "*") == 0) return a * b;
    if (strcmp(token, "/") == 0) return a / b;
    if (strcmp(token, "^") == 0) return pow(a, b);
    return 0.0;
}

float getValor(char *str) {
    char stack[MAX_STACK_SIZE][MAX_EXPR_SIZE];
    int top = -1;
    char expr_copy[MAX_EXPR_SIZE];
    strcpy(expr_copy, str);
    char* token = strtok(expr_copy, " ");

    while (token != NULL) {
        if (is_operator(token)) {
            float b = atof(pop(stack, &top));
            float a = atof(pop(stack, &top));
            float result = apply_operator(token, a, b);
            char result_str[MAX_EXPR_SIZE];
            snprintf(result_str, MAX_EXPR_SIZE, "%f", result);
            push(stack, &top, result_str);
        } else {
            push(stack, &top, token);
        }
```

```
        token = strtok(NULL, " ");
    }
    return atof(pop(stack, &top));
}

char *getFormaInFixa(char *str) {
    char stack[MAX_STACK_SIZE][MAX_EXPR_SIZE];
    int top = -1;
    char expr_copy[MAX_EXPR_SIZE];
    strcpy(expr_copy, str);
    char* token = strtok(expr_copy, " ");

    while (token != NULL) {
        if (is_operator(token)) {
            char b[MAX_EXPR_SIZE];
            strcpy(b, pop(stack, &top));
            char a[MAX_EXPR_SIZE];
            strcpy(a, pop(stack, &top));
            char result[MAX_EXPR_SIZE];
            snprintf(result, MAX_EXPR_SIZE, "(%s %s %s)", a, token, b);
            push(stack, &top, result);
        } else {
            push(stack, &top, token);
        }
        token = strtok(NULL, " ");
    }
    char* result = (char*) malloc(MAX_EXPR_SIZE * sizeof(char));
    strcpy(result, pop(stack, &top));
    return result;
}

void executarTeste(char *posFixa, int numeroTeste) {
    FILE *arquivo = fopen("Resultado.txt", "a");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        exit(ERRO);
    }

    fprintf(arquivo, "TESTE %d:\n", numeroTeste);
    fprintf(arquivo, "Forma posfixa: %s\n", posFixa);
    char *inFixa = getFormaInFixa(posFixa);
    fprintf(arquivo, "Forma infixa: %s\n", inFixa);
    float valor = getValor(posFixa);
    fprintf(arquivo, "Valor da expressão: %.2f\n\n", valor);

    fclose(arquivo);
}
```

3. Testes

3.1 Expressão 3 4 2 * 1 5 - 2 3 ^ ^ / +

O valor da expressão 3 4 2 * 1 5 - 2 3 ^ ^ / +, na forma infixa, é 3 + 4 * 2 / (1 - 5) ^ 2 ^ 3, tem valor igual a 3.00012207 e pode ser obtido a partir do detalhamento apresentado na tabela

		Pilha
1	Lê 3 e empilha.	`[3]`
2	Lê 4 e empilha.	[3, 4]
3	Lê 2 e empilha.	[3, 4, 2]
4	Lê *, desempilha os últimos valores, calcula 4 * 2 = 8, e empilha 8.	[3, 8]
5	Lê 1 e empilha.	[3, 8, 1]
6	Lê 5 e empilha.	[3, 8, 1, 5]
7	Lê -, desempilha os últimos valores, calcula 1 - 5 = -4, e empilha -4.	[3, 8, -4]
8	Lê 2 e empilha.	[3, 8, -4, 2]
9	Lê 3 e empilha.	[3, 8, -4, 2, 3]
10	Lê ^, desempilha os últimos valores, calcula 2 ^ 3 = 8, e empilha 8.	[3, 8, -4, 8]
11	Lê ^, desempilha os últimos valores, calcula (-4) ^ 8 = 65536, empilhando-o.	[3, 8, 65536]
12	Lê /, desempilha 8 e 65536, calcula 8 / 65536 = 0.00012207, empilhando-o.	[3, 0.00012207]
13	Lê +, desempilha os últimos operandos e efetua cálculos, empilhando o resultado.	[3.00012207]

3.2 Expressão 3 4 + 5 *

		Pilha
1	Lê 3 e empilha.	`[3]`
2	Lê 4 e empilha.	[3, 4]
3	Lê +, desempilha 3 e 4, calcula 3 + 4 = 7, e empilha.	[7]

4	Lê 5 e empilha.	[7, 5]
5	Lê *, desempilha 7 e 5, calcula 7 * 5 = 35, e empilha o resultado.	[35]

```
TESTE 3.2:
Forma posfixa: 3 4 + 5 *
Forma infixa: ((3 + 4) * 5)
Valor da expressão: 35.00
```

3.3 Expressão 7 2 * 4 +

		Pilha
1	Lê 7 e empilha.	`[7]`
2	Lê 2 e empilha.	[7, 2]
3	Lê *, desempilha 7 e 2, calcula 7 + 2 = 14, e empilha.	[14]
4	Lê 4 e empilha.	[14, 4]
5	Lê +, desempilha 14 e 4, calcula 14 + 4 = 18, e empilha o resultado.	[18]

```
TESTE 3.3:
Forma posfixa: 7 2 * 4 +
Forma infixa: ((7 * 2) + 4)
Valor da expressão: 18.00
```

3.4 Expressão 8 5 2 4 + * +

		Pilha
1	Lê 8 e empilha.	`[8]`
2	Lê 5 e empilha.	[8, 5]
3	Lê 2 e empilha.	[8, 5, 2]
4	Lê 4 e empilha.	[8, 5, 2, 4]
5	Lê +, desempilha 2 e 4, calcula 2 + 4 = 6, e empilha.	[8, 5, 6]
6	Lê *, desempilha 5 e 6, calcula 5 * 6 = 30, e empilha.	[8, 30]

7	Lê +, desempilha 8 e 30, calcula 8 + 30 = 38, e empilha o resultado.	[38]

```
TESTE 3.4:
Forma posfixa: 8 5 2 4 + * +
Forma infixa: (8 + (5 * (2 + 4)))
Valor da expressão: 38.00
```

3.5 Expressão 6 2 / 3 + 4 *

		Pilha
1	Lê 6 e empilha.	[6]
2	Lê 2 e empilha.	[6, 2]
3	Lê /, desempilha 6 e 2, calcula 6 / 2 = 3, e empilha.	[3]
4	Lê 3 e empilha.	[3, 3]
5	Lê +, desempilha 3 e 3, calcula 3 + 3 = 6, e empilha.	[6]
6	Lê 4 e empilha.	[6, 4]
7	Lê *, desempilha 6 e 4, calcula 6 * 4 = 24, e empilha o resultado.	[24]

```
TESTE 3.5:
Forma posfixa: 6 2 / 3 + 4 *
Forma infixa: (((6 / 2) + 3) * 4)
Valor da expressão: 24.00
```

4. Conclusão

O projeto demonstrou a eficiência da notação pós-fixada para avaliar expressões matemáticas complexas de forma direta e sem a necessidade de parênteses. A implementação das funções getValor e getFormaInFixa usando uma estrutura de pilha permitiu a conversão e avaliação corretas das expressões. Os testes realizados confirmaram a precisão dos resultados esperados. Possíveis melhorias futuras incluem a adição de mais funções matemáticas e a otimização do código para melhor desempenho.

Referências

Aulas ministradas pelo Prof. Marcelo na Universidade Católica de Brasília.

Anexos


main.c


calculadora.h


calculadora.c


main.exe


Resultado.txt

calculadora.h

```h
#ifndef EXPRESSAO_H
#define EXPRESSAO_H

#define MAX_EXPR_SIZE 512
#define MAX_STACK_SIZE 100

typedef struct {
    char posFixa[MAX_EXPR_SIZE];     // Expressão na forma pós-fixada, como 3 12 4 + *
    char inFixa[MAX_EXPR_SIZE];      // Expressão na forma infixada, como 3 * (12 + 4)
    float Valor;                     // Valor numérico da expressão
} Expressao;

int is_operator(const char* token);
char *getFormaInFixa(char *Str);    // Retorna a forma infixa de Str (pós-fixada)
char *pop(char stack[][MAX_EXPR_SIZE], int *top);
float getValor(char *Str);          // Calcula o valor de Str (na forma pós-fixada)
float apply_operator(const char* token, float a, float b);
void push(char stack[][MAX_EXPR_SIZE], int *top, const char* value);
void executarTeste(char *expr, int numeroTeste);

#endif
```

calculadora.c

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

#define ERRO 1

void push(char stack[][MAX_EXPR_SIZE], int *top, const char* value) {
    if (*top < MAX_STACK_SIZE - 1) {
        strcpy(stack[++(*top)], value);
    } else {
        printf("ERRO: Limite de pilha\n");
        exit(ERRO);
    }
```

```
}

char* pop(char stack[][MAX_EXPR_SIZE], int *top) {
    if (*top >= 0) {
        return stack[(*top)--];
    } else {
        printf("ERRO: Subfluxo de pilha - Tentativa de desempilhar de uma pilha vazia\n");
        exit(ERRO);
    }
}

int is_operator(const char* token) {
    return strcmp(token, "+") == 0 || strcmp(token, "-") == 0 || strcmp(token, "*") == 0
|| strcmp(token, "/") == 0 || strcmp(token, "^") == 0;
}

float apply_operator(const char* token, float a, float b) {
    if (strcmp(token, "+") == 0) return a + b;
    if (strcmp(token, "-") == 0) return a - b;
    if (strcmp(token, "*") == 0) return a * b;
    if (strcmp(token, "/") == 0) return a / b;
    if (strcmp(token, "^") == 0) return pow(a, b);
    return 0.0;
}

float getValor(char *str) {
    char stack[MAX_STACK_SIZE][MAX_EXPR_SIZE];
    int top = -1;
    char expr_copy[MAX_EXPR_SIZE];
    strcpy(expr_copy, str);
    char* token = strtok(expr_copy, " ");

    while (token != NULL) {
        if (is_operator(token)) {
            float b = atof(pop(stack, &top));
            float a = atof(pop(stack, &top));
            float result = apply_operator(token, a, b);
            char result_str[MAX_EXPR_SIZE];
            snprintf(result_str, MAX_EXPR_SIZE, "%f", result);
            push(stack, &top, result_str);
        } else {
            push(stack, &top, token);
        }
        token = strtok(NULL, " ");
    }
    return atof(pop(stack, &top));
}

char *getFormaInFixa(char *str) {
    char stack[MAX_STACK_SIZE][MAX_EXPR_SIZE];
```

```
    int top = -1;
    char expr_copy[MAX_EXPR_SIZE];
    strcpy(expr_copy, str);
    char* token = strtok(expr_copy, " ");

    while (token != NULL) {
        if (is_operator(token)) {
            char b[MAX_EXPR_SIZE];
            strcpy(b, pop(stack, &top));
            char a[MAX_EXPR_SIZE];
            strcpy(a, pop(stack, &top));
            char result[MAX_EXPR_SIZE];
            snprintf(result, MAX_EXPR_SIZE, "(%s %s %s)", a, token, b);
            push(stack, &top, result);
        } else {
            push(stack, &top, token);
        }
        token = strtok(NULL, " ");
    }
    char* result = (char*) malloc(MAX_EXPR_SIZE * sizeof(char));
    strcpy(result, pop(stack, &top));
    return result;
}

void executarTeste(char *posFixa, int numeroTeste) {
    FILE *arquivo = fopen("Resultado.txt", "a");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        exit(ERRO);
    }

    fprintf(arquivo, "TESTE 3.%d:\n", numeroTeste);
    fprintf(arquivo, "Forma posfixa: %s\n", posFixa);
    char *inFixa = getFormaInFixa(posFixa);
    fprintf(arquivo, "Forma infixa: %s\n", inFixa);
    float valor = getValor(posFixa);
    fprintf(arquivo, "Valor da expressão: %.2f\n\n", valor);

    fclose(arquivo);
}
```

main.c

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

```c
int main() {

    char posFixa[] = "53 23 + 8 2 - *";
    printf("Teste professor, forma posFixa: %s\n", posFixa);

    // Convertendo para a forma infixa
    char *inFixa = getFormaInFixa(posFixa);
    printf("Teste professor: %s\n", inFixa);

    // Calculando o valor da expressão
    float valor = getValor(posFixa);
    printf("Teste professor: %.2f\n", valor);

    char posFixa01[] = "3 4 + 5 *";
    executarTeste(posFixa01, 2);

    char posFixa02[] = "7 2 * 4 +";
    executarTeste(posFixa02, 3);

    char posFixa03[] = "8 5 2 4 + * +";
    executarTeste(posFixa03, 4);

    char posFixa04[] = "6 2 / 3 + 4 *";
    executarTeste(posFixa04, 5);

    char posFixa05[] = "9 5 2 8 * 4 + * +";
    executarTeste(posFixa05, 6);

    char posFixa06[] = "2 3 + log 5 /";
    executarTeste(posFixa06, 7);

    char posFixa07[] = "10 log 3 ^ 2 +";
    executarTeste(posFixa07, 8);

    char posFixa08[] = "45 60 + 30 cos *";
    executarTeste(posFixa08, 9);

    char posFixa09[] = "0.5 45 sen 2 ^ +";
    executarTeste(posFixa09, 10);

    return 0;
}
```