Algoritmos e Estruturas de Dados I

Expressões aritméticas e estrutura de dados linear

Vitor Nascimento Aguiar, William Massami Costa Harada, Yago de Oliveira Feitoza

Universidade do Estado do Amazonas
Escola Superior de Tecnologia

July 3, 2023

Problema proposto

Existem duas alternativas à representação tradicional de expressões aritméticas, a notação posfixa e prefixa. Na posfixa o operador é expresso após seus operandos e na prefixa antes. Alguns exemplos são:

Padrão: a+b.
Posfixa = a b +.
Prefixa = + a b

Padrão: (a+b)*c = a b + c *.
Posfixa = a b + c *.
Prefixa = + a b * c

Problema proposto

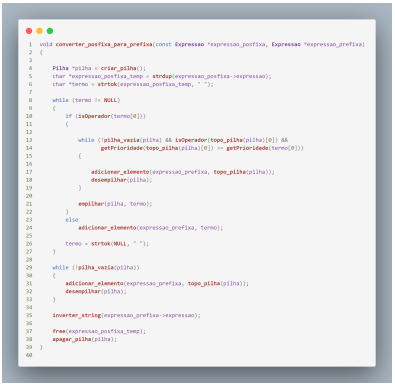
a) Criar um TDA que represente expressão aritmética na forma posfixa. Deve ter pelo menos as operações: Criar expressão, apagar expressão, imprimir expressão.

b) Criar uma operação no TDA da questão a) para converter e armazenar uma expressão na notação posfixa para a notação prefixa e vice-versa (Utilizar estrutura de dados tipo Pilha).

c) Criar uma operação para obter o resultado de avaliar uma expressão posfixa.

d) Criar uma operação para obter o resultado de avaliar uma expressão prefixa.

Código



```c
void converter_posfixa_para_prefixa(const Expressao *expressao_posfixa, Expressao *expressao_prefixa)
{

    Pilha *pilha = criar_pilha();
    char *expressao_posfixa_temp = strdup(expressao_posfixa->expressao);
    char *termo = strtok(expressao_posfixa_temp, " ");

    while (termo != NULL)
    {
        if (isOperador(termo[0]))
        {

            while (!pilha_vazia(pilha) && isOperador(topo_pilha(pilha)[0]) &&
                   getPrioridade(topo_pilha(pilha)[0]) >= getPrioridade(termo[0]))
            {

                adicionar_elemento(expressao_prefixa, topo_pilha(pilha));
                desempilhar(pilha);
            }

            empilhar(pilha, termo);
        }
        else
            adicionar_elemento(expressao_prefixa, termo);

        termo = strtok(NULL, " ");
    }

    while (!pilha_vazia(pilha))
    {
        adicionar_elemento(expressao_prefixa, topo_pilha(pilha));
        desempilhar(pilha);
    }

    inverter_string(expressao_prefixa->expressao);

    free(expressao_posfixa_temp);
    apagar_pilha(pilha);
}

```

Figure 1: Conversão Posfixa -> Prefixa

```c
void converter_prefixa_para_posfixa(const Expressao *expressao_prefixa, Expressao *expressao_posfixa)
{

    Pilha *pilha = criar_pilha();
    char *expressao_prefixa_temp = strdup(expressao_prefixa->expressao);
    char *termo = strtok(expressao_prefixa_temp, " ");

    while (termo != NULL)
    {
        if (isOperador(termo[0]))
        {
            while (!pilha_vazia(pilha) && isOperador(topo_pilha(pilha)[0]) &&
                   getPrioridade(topo_pilha(pilha)[0]) >= getPrioridade(termo[0]))
            {

                adicionar_elemento(expressao_posfixa, topo_pilha(pilha));
                desempilhar(pilha);
            }

            empilhar(pilha, termo);
        }
        else
            adicionar_elemento(expressao_posfixa, termo);

        termo = strtok(NULL, " ");
    }

    while (!pilha_vazia(pilha))
    {
        adicionar_elemento(expressao_posfixa, topo_pilha(pilha));
        desempilhar(pilha);
    }

    apagar_pilha(pilha);
    free(expressao_prefixa_temp);
}
```

Figure 2: Conversão Prefixa -> Posfixa

Código

```c
typedef struct
{
    char expressao[200];
    int tamanho;
} Expressao;

typedef struct no
{
    char elemento[200];
    struct no *proximo;
} No;

typedef struct
{
    No *topo;
} Pilha;

/* Funções relacionadas ao TDA Expressao */
Expressao criar_expressao();
void apagar_expressao(Expressao *expressao);
void imprimir_expressao(const Expressao *expressao);
void adicionar_elemento(Expressao *expressao, const char *elemento);

void converter_posfixa_para_prefixa(const Expressao *expressao_posfixa,
                                    Expressao *expressao_prefixa);
void converter_prefixa_para_posfixa(const Expressao *expressao_prefixa,
                                    Expressao *expressao_posfixa);
float resultado_expressao(const Expressao *expressao);

/* Funções relacionadas ao TDA Pilha */
Pilha *criar_pilha();
void apagar_pilha(Pilha *pilha);
int pilha_vazia(const Pilha *pilha);
void empilhar(Pilha *pilha, const char *elemento);
void desempilhar(Pilha *pilha);
char *topo_pilha(const Pilha *pilha);

/* Funções auxiliares */
int isOperador(char c);
int getPrioridade(char c);
void inverter_string(char *str);
```

Arquivo de cabeçalho

Código

```c
#include <stdio.h>
#include "main.h"

int main()
{
    printf("[Criando expressao posfixa]: ");
    Expressao expressao = criar_expressao();
    adicionar_elemento(&expressao, "5");
    adicionar_elemento(&expressao, "3");
    adicionar_elemento(&expressao, "*");

    imprimir_expressao(&expressao);

    float resultado = resultado_expressao(&expressao);
    printf("Resultado: %.2f\n", resultado);

    printf("\nRealizando conversao para prefixa... \n");
    Expressao expressao_prefixa = criar_expressao();
    converter_posfixa_para_prefixa(&expressao, &expressao_prefixa);
    imprimir_expressao(&expressao_prefixa);

    printf("Convertendo de volta para posfixa... \n");
    Expressao expressao_posfixa = criar_expressao();
    converter_prefixa_para_posfixa(&expressao_prefixa, &expressao_posfixa);
    imprimir_expressao(&expressao_posfixa);

    apagar_expressao(&expressao);
    apagar_expressao(&expressao_prefixa);
    apagar_expressao(&expressao_posfixa);

    return 0;
}
```

Programa principal