Estrutura de Dados: Trabalho 1
Avaliação de Expressões Aritméticas (forma pós-fixa)

Gabriel Correia de Vasconcelos, 16/0120781
Pedro Augusto Coelho Nunes, 16/0141044

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 116319 - Estrutura de Dados - Turma A

{gcvasconcelos98,pedropacn}@gmail.com

Abstract. *The following project has as objective the concretization of the concepts of stacks, developed in C programming language, through the evaluation of arithmetic expressions in the postfix form.*
Keywords*: data structure, stack, postfix.*

Resumo. *O presente projeto tem como objetivo a concretização dos conceitos de pilhas,desenvolvidas na linguagem de programação C, através da avaliação de expressões aritméticas na forma pós-fixa.*
Palavras-chave*: estrutura de dados, pilha, pós-fixa.*

1. Introdução

Equações algébricas, quando compostas por vários operandos, números e parênteses, podem ter uma resolução complexa na forma infixa, ou seja, quando o operador matemático se encontra no meio dos números. Uma forma de simplificar essas equações é escrevê-la na forma pós-fixa. A forma pós-fixa é formada quando o operador se encontra depois dos números.

O que faz da forma pós-fixa mais simples de ser calculada do que a forma infixa é o fato de que na primeira, não há parênteses para definir prioridade das operações, ao contrário da segunda. A forma pós-fixa já está na ordem correta para a resolução do problema.

O programa feito neste trabalho visa ter como entrada uma equação na forma infixa, transformá-la para a forma pós-fixa, que é mais simples quando se trata de interpretação, e então calcular o resultado.

2. Implementação

Para facilitar o desenvolvimento do algoritmo e do projeto como um todo, foi utilizado o git. Assim, foi possível modularizar as etapas de desenvolvimento do código fonte, além do controle de versões do trabalho, bem como quanto cada integrante do grupo contribuiu para sua conclusão.

No programa descrito, foi usada a estrutura de dados chamada de pilha. A pilha é formada por uma lista interligada, estrutura essa que é composta por duas variáveis que armazenam os endereços do primeiro e do último elemento da lista. Cada elemento possui um ponteiro para o próximo da lista e uma variável chamada de dado. No algoritmo, foram usadas duas pilhas diferentes, uma para variáveis tipo char e outra para variáveis tipo ponto flutuante (o motivo será explicado mais a frente).

```c
typedef struct elemento {
    char dado; //ou float
    struct elemento *proximo;
}t_elemento;
typedef struct lista {
    t_elemento *primeiro;
    t_elemento *ultimo;
}t_lista;
typedef struct pilha {
    t_lista *l;
}t_pilha;
```

Pilha é uma estrutura de dados do tipo FIFO (First In, First Out), ou seja, aonde o primeiro elemento que entra é o último que sai. As pilhas tem como principais funções para alterar sua quantidade de elementos o empilhamento e o desempilhamento. Para empilhar, um elemento é inserido no início da pilha, modificando assim o endereço do primeiro elemento da estrutura e armazenado o endereço do elemento inicial anterior neste novo elemento. Quando se desempilha, o endereço inicial da pilha torna-se o do segundo elemento e o dado do elemento removido é retornado pela função. No algoritmo, para se criar essas funções da pilha, foram utilizadas as funções básicas da lista interligada.

```c
t_lista *criaLista() {...
}
void insereInicio(char valor, t_lista *l) {...
}
int estaVazia(t_lista *l) {...
}
int removeInicio(t_lista *l) {...
}
t_pilha * criaPilha() {
    t_pilha * p =  (t_pilha *)malloc(sizeof(t_pilha));
    p->l = criaLista();
    return p;
}
void empilhar(char valor,t_pilha *p) {
    insereInicio(valor, p->l);
}
char desempilhar(t_pilha *p) {
    return removeInicio(p->l);
```

Outra função utilizada foi a função que verifica se uma pilha está vazia. Esta função lê o endereço do primeiro elemento de uma pilha e verifica se ele existe ou não. Se sim, ele retorna verdadeiro. Caso contrário, falso.

```c
int estaVaziaPilha(t_pilha *p) {
    return estaVazia(p->l);
}
```

Funções auxiliares também foram criadas para simplificar a leitura do algoritmo. Essas funções verificam se uma variável do tipo char é um número, um operador matemático ou um escopo. Foi feita também uma função que retorna a prioridade de um operador em relação a outro. Por exemplo, em uma equação algébrica a multiplicação e a divisão tem prioridade igual entre si e maior em relação à soma e subtração.

```
int ehNumero(char ch) {
    if (ch == '0' || ch == '1' || ch == '2' || ch == '3' || ch ==
        '4' || ch == '5' || ch == '6' || ch == '7' || ch == '8'
       || ch == '9')
        return 1;
    return 0;
}
int ehOperador(char ch) {
    if (ch == '+' || ch == '-' || ch == '/' || ch == '*')
        return 1;
    return 0;
}
int ehEscopo(char ch) {
    if (ch == '(')
        return 1;
    else if (ch == ')')
        return -1;
    return 0;
}
int prioridade(char ch) {
    if (ch == '*' || ch == '/')
        return 2;
    else if (ch == '+' || ch == '-')
        return 1;
    return 0;
}
}
```

Na função principal, foram declarados 3 vetores de char. Infixa, que receberá a equação na forma infixa, pós-fixa para armazenar a forma pós-fixa e temp, uma string para ser utilizada na função strcat().Também foi declarado um inteiro i para iterações e um char t para ajudar nas comparações, empilhamentos e desempilhamentos. Foram criados dois ponteiros para pilhas. Um para pilhas de char e ouro para pilhas de ponto flutuante. Isso foi necessário para que fosse possível receber inicialmente os caracteres da forma infixa, que são do tipo char e depois usar uma pilha para calcular a equações com casas decimais. Como variáveis do tipo char não possuem casas decimais, foi preciso criar um outro tipo de pilha com dados do tipo float que aceitam números racionais. Devido a este fato, praticamente todas as funções tiveram de ter duas versões, uma para o tipo char e outra para o tipo float. Isso teve de acontecer devido ao fato de que char e float possuem tamanhos diferentes na memória. Logo, as operações e parâmetros eram diferentes dependendo do tipo de variável utilizado.

```
int i;
char infixa[50] = "\0", posfixa[50] = "\0", temp[2], t;
float a, b, c;
t_pilha *p = criaPilha();
t_pilhaf *pf = criaPilhaf();
```

Primeiramente, o usuário entra com a equação algébrica. Em seguida, esta é verificada para garantir que não existem parênteses em excesso ou em falta.

```
while (1) {
    printf("Digite a expressao na forma infixa:");
    scanf("%s", infixa);

    for (i = 0; i < strlen(infixa); i++) {
        if (ehEscopo(infixa[i] == 1))
            empilhar(infixa[i], p);
        else if (ehEscopo(infixa[i] == -1)) {
            if (estaVaziaPilha(p)) {
                printf("Formato invalido. Tente novamente.\n");
                break;
            }
            if (p->l->primeiro->dado == '(' && infixa[i] == ')')
                desempilhar(p);
            else
                printf("Formato invalido. Tente novamente.\n");
        }
    }
    if (estaVaziaPilha(p))
        break;
    while(!estaVaziaPilha(p))
        desempilhar(p);
}
```

Em seguida, os elementos da string infixa são analisados um a um para serem passados para a string pós-fixa. Se o termo em análise for um algarismo, ele é colocado na string pós-fixa. Caso o termo seja um operador, é verificado se a pilha de char está vazia e se este termo da pilha possui uma prioridade matemática maior do que a do termo em análise no vetor infixa. Se sim, todos os termos da pilha são desempilhados e colocados na string pós-fixa. Em seguida o termo em análise é empilhado. Ainda se o char analisado for uma abertura de parênteses, este é empilhado e se for um fechamento de parênteses, a pilha é completamente esvaziada. Se ainda assim a pilha não estiver vazia, seus termos remanescentes são desempilhados e escritos em pós-fixa.

```
for (i = 0; i < strlen(infixa); i++) {
    temp[0] = ' ';
    temp[1] = ' ';
    if (ehNumero(infixa[i])) {
        if (ehNumero(infixa[i+1])) {
            temp[0] = infixa[i];
            temp[1] = '\0';
            strcat(posfixa, temp);
        }
        else {
            temp[0] = infixa[i];
            strcat(posfixa, temp);
        }
    } else if (ehOperador(infixa[i])) {
        if (!estaVaziaPilha(p) && prioridade(p->l->primeiro->dado
            ) >= prioridade(infixa[i])) {
            while(!estaVaziaPilha(p) && prioridade(p->l->primeiro
                ->dado) >= prioridade(infixa[i])) {
                temp[0] = desempilhar(p);
                if (!ehEscopo(temp[0]))
                    strcat(posfixa, temp);
```

```
            }
        }
        empilhar(infixa[i], p);
    } else if (infixa[i] == '(') {
        empilhar(infixa[i], p);
    } else if (infixa[i] == ')') {
        do {
            t = desempilhar(p);
            if (t != '('){
                temp[0] = t;
                strcat(posfixa, temp);
            }
        } while (t != '(');
    }
}
while(!estaVaziaPilha(p)) {
    temp[0] = desempilhar(p);
    temp[1] = ' ';
    strcat(posfixa, temp);
}
```

Agora, será analisada a array pós-fixa. Se for um número, este é convertido de char para float e empilhado em uma pilha de float. Se não for um número, é um operador, logo deverão ser desempilhados os dois primeiros elementos da pilha para realizar a operação encontrada no vetor. Após o cálculo, o resultado é empilhado para ser utilizado na próxima operação. Quando a string estiver analisada por completo, há uma verificação se os dados do primeiro e do último termos da pilha são iguais. Se não, houve um erro. Se sim, este dado é desempilhado e impresso na tela.

```
for (i = 0; i < strlen(posfixa); i++) {
    if (ehNumero(posfixa[i])) {
        i = strToInt(posfixa, i);
        empilharf((float)i_num, pf);
    }
    else if (ehOperador(posfixa[i])) {
        a = desempilharf(pf);
        b = desempilharf(pf);
        switch (posfixa[i]) {
            case '+':
                c = b + a;
                break;
            case '-':
                c = b - a;
                break;
            case '*':
                c = b * a;
                break;
            case '/':
                c = b / a;
                break;
        }
        empilharf(c, pf);
    }
}
```

3. Estudo de Complexidade

Estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O).

4. Testes Executados

Após a finalização do código-fonte, foram realizados os testes de validação da expressão escrita, ou seja, se todos os escopos "("são fechados com seus correspondentes ")". Os próximos testes tiveram o objetivo de checar se a conversão para forma pós-fixa foi realizada com sucesso e se seus resultados após o cálculo batem com o esperado. As expressões usadas foram as seguintes.

$$1 + 2 * 3 - 4/2 = 5 \tag{1}$$

1. Checa se as operações estão na ordem certa de prioridade.

$$(1 + 2) * 3) = 9 \tag{2}$$

2. Exemplo de expressão inválida, e que depois é novamente testada de forma válida (sem o último ")") e que checa se os escopos estão sendo respeitados.

$$(40 + 80) * (60 - 50)/((30 - 20) * (20 + 20)) = 3 \tag{3}$$

3. Este último teste, além de ser uma expressão mais complexa, checa também se o algoritmo consegue fazer os cálculos quando os números analisados tem mais de um algarismo.

5. Conclusão

Conclusão e referências: [Vasconcelos and Nunes 2017], [Wikipédia 2017], [IME 2016], [TutorialPoint 2017b], [TutorialPoint 2017a], [Cormen et al. 2002].

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2002). Algoritmos: teoria e prática. *Editora Campus*, 2.

IME (2016). Pilhas. `https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html`. Acessado: 2017-05-07.

TutorialPoint (2017a). C library function - atoi(). `https://www.tutorialspoint.com/c_standard_library/c_function_atoi.htm`. Acessado: 2017-05-07.

TutorialPoint (2017b). C library function - strcat(). `https://www.tutorialspoint.com/c_standard_library/c_function_strcat.htm`. Acessado: 2017-05-07.

Vasconcelos, G. C. d. and Nunes, P. A. C. (2017). Repo: ed-trabalho1. `https://github.com/gcvasconcelos/ed-trabalho1`. Acessado: 2017-05-07.

Wikipédia (2017). Pilha (informática) — wikipédia, a enciclopédia livre. [Online; Acessado: 11-abril-2017].