Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Estrutura de Dados - 1/2016
Aluno: Frederico Pinheiro Dib
Matricula: 15/0125925
Aluno: Amanda Oliveira Alves
Matricula: 15/0116276
Turma: A

Introdução

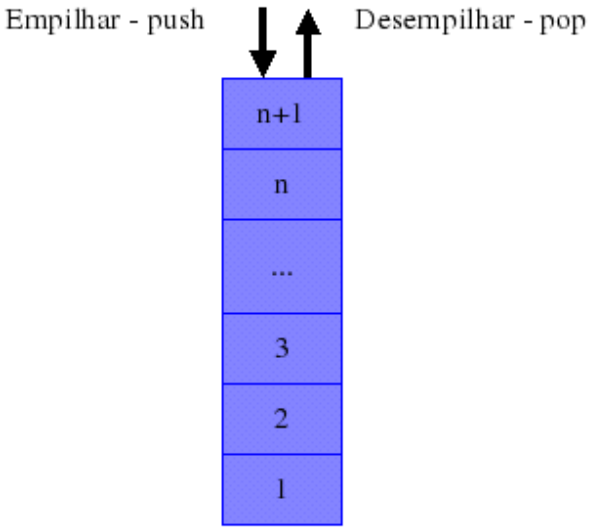
O algoritmo desenvolvido e implementado no trabalho proposto consiste em criar uma calculadora. O problema inicial é criar um programa que avalie uma expressão matemática lida do teclado e faça testes para confimar sua validez, se a expressão for inválida ao final dos testes, será exibida uma mensagem informando o usuário.

No caso em que a expressão for considerada válida, o programa deve transformá-la da forma infixa para a forma posfixa e efetuar as operações especificadas, por fim será mostrado na tela o valor final da expressão.

Implementação

A expressão infixa de entrada é declarada como um vetor de char (string) e recebida pelo comando scanf, o usuário pode digitar a expressão usando ou não espaços em branco entre os elementos, o programa é adaptado para as duas situações. O próximo passo é analisar os delimitadores da equação, ou seja, o programa deve contar o número de parênteses abertos e fechados presentes na equação. Ao final dos testes se quantidade for igual a expressão é válida, caso contrário a expressão é inválida e o programa exibe uma mensagem para o usuário, através do comando printf.

Para executar os testes primeiro cria-se uma pilha vazia, estrutura de dados que tem como principais operações as funções “empilhar” e “desempilhar”, que consistem respectivamente em inserir um elemento no início da pilha e remover um elemento do início da pilha. Exemplo:



Depois que a pilha estiver criada deve-se percorrer toda a string da esquerda para a direita conferindo se algum dos caracteres corresponde a um parêntese aberto. O programa irá fazer um teste em cada índice usando o comando if, caso o caracter seja um parêntese aberto, será empilhado. Quando for encontrado um parêntese fechado, o programa deve checar usando a função estaVaziapilha, se a pilha está vazia ou não. No caso de a pilha estar vazia a expressão já pode ser considerada inválida, se não estiver vazia desempilha-se e compara o que estava na pilha com o parêntese fechado, se os caracteres forem diferentes (um parêntese aberto e um fechado) então os testes podem continuar. Quando chegar ao final da string e não houverem mais testes a fazer a pilha deve estar vazia, então finalmente a equação será considerada inteiramente válida.

É preciso posteriormente transformar a expressão original que é representada na forma infixa para a forma posfixa, para esse processo é necessário o uso de uma pilha, começando vazia. Percorre-se novamente a expressão infixa, fazendo testes em cada índice, quando um elemento for diferente de um operador (os operadores permitidos são: +, -, * e /) ele será copiado diretamente para a nova string que futuramente será a expressão posfixa.

Ao encontrar um operador na string usa-se a função desempilhar para comparar o elemento da string com o obtido na pilha. Enquanto a pilha não estiver vazia e no topo houver um operador com prioridade maior ou igual ao encontrado na string, o programa vai iterativamente desempilhando e copiando os operadores contidos na pilha para a nova string, quando encerrado o loop de repetição empilha-se o operador que foi encontrado na string original.

Se o elemento encontrado na string for um parêntese de abertura basta usar a função empilhar. Se for um parêntese de fechamento é necessário um loop de repetição, enquanto o símbolo desempilhado for diferente do parêntese de abertura correspondente, o programa deve desempilhar e copiar os elementos na string de saída. Ao final da varredura esvazia-se a pilha movendo os operadores para a nova string e nela estará pronta a expressão posfixa desejada.

Do jeito que está implementado o programa só funciona para números de 0 a 9, o que seria ineficiente nesse caso, para corrigir esse problema foi preciso criar a função contaNumero. A função receberá um contador que armazena a quantidade de algarismos de cada operando, se o contador for igual a um o algarismo somente é convertido para real utilizando a função Transformar, se não o primeiro algarismo do número é multiplicado por 10 elevado a contador - 1, depois o segundo por contador - 2 e assim sucessivamente.

O processo se repetirá contador – 1 vezes e a soma dos valores obtidos nas multiplicações será igual ao número final desejado. Depois de finalizada a função o programa será capaz de efetuar operações com quaisquer números.

A única serventia da função Transformar é converter os elementos do tipo char para o tipo float. Ela recebe os caracteres que representam os números e retorna eles como números reais para serem operados normalmente.

A função Conta efetua todas as operações e retorna o resultado para ser exibido na tela. A string posf (variável que armazena a expressão posfixa) será percorrida da esquerda para a direita, enquanto o elemento testado for um operando chama-se a função Empilhar, assim os fatores de cada operação estarão organizados na pilha.

Quando o elemento for um operador duas variáveis vão ser inicializadas com os valores da função Desempilhar (é importante manter a ordem dos fatores). Testa-se o operador encontrado para saber qual das operações disponíveis efetuar e por último depois de executadas todas as operações requisitadas a função retorna o resultado final da equação.

Estudo de Complexidade

Nesse trabalho foram utilizadas diversas funções diferentes, cada uma com o seu próprio custo.

As funções de manuseio de pilha e lista como as funções *criaListaf, insereIniciof, insereFinalf, estaVaziaf, removeIniciof, criaPilhaf, Empilharf, Desempilharf, estaVaziaPilhaf, criaLista, insereInicio, insereFinal, estaVazia, removeInicio, criaPilha, Empilhar, Desempilhar e estaVaziaPilha* possuem custo constante de O(1), porem na pratica seus custos passam a ser de o(n), sendo "n" o numero de vezes q elas foram chamadas.

As funções auxiliares (funções criadas a partir de uma partição de uma função maior, com a finalidade de deixar o código mais legível e facilitar manutenções) como as funções *testeOperador*, *Prioridade e Tranformar* assim como as funções de manuseio de pilhas elas isoladas poossuem um custo constante de O(1), porem dentro do total o custo delas passa a ser O(n).

A função Conta *e validez* vão ter um custo único de O(n), sendo n o tamanho da string. Já as funções *contaNumero e Posfixa* possuem um custo único de $O(n^2)$.

O custo total do algoritmo consequentemente será $O(n^2)$.

Listagem de testes executados

2+3 => 5.00

2 + 3 => 5.00

144 – 14 => 130.00

(3 – 1) * 2 + 3 – 1 => 6.00

3 – 1 * 2 + 3 – 1 => 3.00

40 * (35 – 12) + 15 / 3 => 925.00

2 * (3 + 5 * (2 + 2)) => 46.00

5 + (2 * 3)) => A expressão não é valida!

15 / 2 => 7.50

Conclusão

O objetivo de concretizar os conceitos de pilha foi bem sucedido, trouxe ainda outras contribuições para o aprendizado na área de programação como um todo, colocando em prática grande parte do que aprendemos em sala de aula. Uma das principais dificuldades foi terminar a função Posfixa, surgiram algumas dúvidas no decorrer da implementação, mas no final funcionou perfeitamente. Exceto por algumas dúvidas pontuais em outras funções do programa, como a Contanumero, não houve grandes dificuldades para a realização do trabalho.

Bibliografia

Slides disponibilizados pelo professor – pilha, fila, estruturas fundamentais e algoritmos.