MC202GH - ESTRUTURAS DE DADOS

2º Semestre de 2017

Professor: Rafael C. S. Schouery

Monitores: Yulle Glebbyo Felipe Borges (PED)

Caio Vinícius Piologo Véras Fernandes (PAD)

Lab. 05 - Calculadora Pós-fixa

Peso 2

1. O Problema

Nas expressões em notação infixa temos operações definidas por um operador e um conjunto de operandos. Por simplicidade, consideraremos aqui apenas operações binárias (com exatamente dois operandos). Neste caso, a notação infixa tem a forma operando-operador-operando, onde cada operando pode ser uma outra expressão com seu próprio operador. Em uma expressão em notação infixa completamente parentizada, todos os operadores e seus argumentos estão cercados por parênteses. Como exemplo de expressão em notação infixa, temos `1 + ( 2 * 3 ) + 4`. Por outro lado, um exemplo válido de expressão em notação infixa completamente parentizada é `( 1 + ( ( 2 * 3 ) + 4 ) )`. Em uma expressão totalmente parentizada, é dispensada a necessidade de se ter prioridade entre os operadores, já que as ordens de processamento são sempre dadas de acordo com a hierarquia formada pelos parênteses.

Já expressões em notação pós-fixa (também conhecida como notação polonesa reversa), temos uma organização da forma operando-operando-operador, onde cada operando pode ser uma outra expressão com seu próprio operador. Esta notação dispensa o uso de parênteses já que, uma vez que ela é sempre processada da esquerda para direita, os operadores são processados conforme a ordem que eles aparecem na expressão.

Notação Infixa	Notação Infixa Completamente Parentizada	Notação Pós-fixa
`1 + 2 * 3`	`( 1 + ( 2 * 3 ) )`	`1 2 3 * +`
`( 2 * 4 ) / 3`	`( ( 2 * 4 ) / 3 )`	`2 4 * 3 /`

`1 * 2 + 2 / 1`	`( ( 1 * 2 ) + ( 2 / 1 ) )`	`1 2 * 2 1 / +`

Neste trabalho, você é responsável por desenvolver um programa com as 3 seguintes funcionalidades básicas:

- Processa uma expressão em notação pós-fixa e retorna seu resultado;
- Converte uma expressão de notação infixa completamente parentizada para notação pós-fixa;
- Converte uma expressão de notação pós-fixa para notação infixa completamente parentizada.

Estas operações devem ser implementadas utilizando **pilhas** e **sem o uso de chamadas recursivas.** Você poderá usar pilhas de floats para avaliar uma expressão e retornar seu resultado e pilhas de vetores de caracteres para fazer as conversões entre as notações.

1.1 Avaliação de uma expressão pós-fixa

Para esta funcionalidade, você deve implementar uma **pilha de floats usando um vetor**. Você deve implementar o algoritmo visto em classe para expressões pós-fixas com as adaptações necessárias.

A função `atof`, do cabeçalho `stdlib.h` pode ser útil para converter um vetor de caracteres para um `float`. Veja mais informações sobre esta função em sua manpage do Linux, digitando `man atof` no terminal.

1.2 Conversão de infixa para pós-fixa

Para esta funcionalidade, você deve implementar uma **pilha de vetores de caracteres usando lista ligada**. Você deve implementar o algoritmo visto em classe conversão de infixa para pós-fixa com as adaptações necessárias para lidar com parênteses.

As funções de comparação e concatenação de cadeias de caracteres podem ser úteis. Estas funções são `strcmp` e `strcat`, e elas são definidas no cabeçalho `string.h`. Veja mais informações sobre estas funções em suas manpages do Linux, digitando `man strcmp` ou `man strcat` no terminal.

1.3 Conversão de pós-fixa para infixa

Para esta funcionalidade, você pode utilizar a mesma pilha da seção anterior, ou seja, uma **pilha de vetores de caracteres usando lista ligada**. É possível estender o raciocínio utilizado anteriormente para fazer esta conversão.

Disponibilizamos abaixo um esboço de pseudocódigo da conversão de notação pós-fixa para notação infixa completamente parentizada:

```
1 . Enquanto houver elementos para serem lidos da entrada
   -  Leia o elemento e
         -  Se e é um número
               -  Empilhe e
         -  Se e é um operador: e ∈ {+,-,*,/}
               -  Desempilhe op2
               -  Desempilhe op1
               -  Faça uma string da forma op1 e op2
               -  Encapsule esta string com parênteses: ( op1 e op2 )
               -  Empilhe a string resultante
2 . Imprima o único elemento restante na pilha.
```

Você pode utilizar as funções de concatenação e comparação de vetores de caracteres mencionadas anteriormente para implementar este algoritmo.

2. Entrada

Por simplicidade, assumimos que as expressões atuam apenas sob valores inteiros, e suportam os operadores de soma (+), subtração (-), multiplicação (*) e divisão (/), além dos parênteses (e). Por convenção, assumimos que ao fim de qualquer expressão terá um sinal de igual, sinalizando o fim da expressão.

Assume-se também que todos os símbolos que compõem uma expressão estarão separados por um espaço em branco, para que possam ser lidos individualmente com a chamada `scanf("%s",s)`, onde s é um vetor de caracteres a ser lido.

Mais detalhes sobre as entradas de cada funcionalidade a ser implementada seguem na tabela abaixo.

ID	Nome	Descrição
1	Avaliação de expressão pós-fixa	Dada uma **expressão em notação pós-fixa** seguida de um sinal = indicando o fim da expressão, imprime o seu resultado. O tamanho máximo da expressão é 141 caracteres, contando espaços brancos e o `\0`
2	Conversão de notação infixa para pós-fixa	Dada uma **expressão em notação infixa completamente parentizada** seguida de um sinal = indicando o fim da expressão, imprime a expressão em notação pós-fixa equivalente. O tamanho máximo da expressão é 141 caracteres, contando espaços brancos e o `\0`

3	Conversão de notação pós-fixa para infixa	Dada uma **expressão em notação pós-fixa** seguida de um sinal = indicando o fim da expressão, imprime a expressão equivalente em notação infixa completamente parentizada. O tamanho máximo da expressão é 141 caracteres, contando espaços brancos e o `\0`
0	Finalização	Saia do sistema, certificando-se que toda a memória alocada foi liberada

Exemplo de entrada:

```
1
1 2 3 / + =
2
( 1 + ( 2 / 3 ) =
3
1 2 3 4 + + + =
0
```

3. Saída

As saídas devem obedecer o mesmo padrão das entradas, ou seja, nas expressões, deve-se ter todos os símbolos separados por um espaço branco.

ID	Saída	Comentário
1	X	**X** é um número real com precisão de 4 casas, representando o resultado da expressão dada como entrada. Utilize a chamada `printf("%.4f", num)`, onde `num` é uma variável do tipo float, para imprimir exatamente 4 casas decimais
2	EXPR_POS	**EXPR_POS** é uma expressão em notação pós-fixa com cada símbolo separada por um espaço branco. Pode-se assumir que a expressão resultante terá no máximo 140 caracteres, contando os espaços brancos. Obs: deve existir um espaço branco após o último símbolo, logo antes do `\n`

3	EXPR_IN	**EXPR_IN** é uma expressão em notação infixa completamente parentizada, com cada símbolo separada por um espaço branco. Pode-se assumir que a expressão resultante terá no máximo 140 caracteres, contando os espaços brancos. Obs: deve existir um espaço branco após o último símbolo, logo antes do `\n`
0	Sistema encerrado.	Antes de encerrar o sistema, é necessário liberar toda a memória alocada

Exemplo de saída:

```
1.6666
1 2 3 / +
( 1 + ( 2 + ( 3 + 4 ) ) )
Sistema encerrado.
```

4. Informações

- Este laboratório possui **peso 2**.
- **Não** há um número máximo de submissões.
- No início de cada arquivo a ser submetido, insira um comentário com seu nome, RA e uma breve descrição do conteúdo do arquivo.
- Apenas comentários no formato `/* comentário */` serão aceitos. Comentários com `//` serão acusados como erros pelo SuSy.
- Os cabeçalhos permitidos são:
  - `stdio.h`
  - `stdlib.h`
  - `string.h`
  - `math.h`
- A submissão da sua solução deverá conter múltiplos arquivos:
  - **pilha_string.h**: interface da pilha de vetores de caracteres
  - **pilha_string.c**: implementação da pilha de vetores de caracteres utilizando uma lista ligada
  - **pilha_float.h**: interface da pilha de floats
  - **pilha_float.c**: implementação da pilha de floats utilizando vetores

    - **lab05.c**: cliente que utiliza a estrutura
- Será disponibilizado um **Makefile** para este trabalho na página do laboratório:
    - Para compilar seu projeto, basta navegar até o diretório do projeto e utilizar o comando ‘make’ no terminal do Linux.
    - Veja mais instruções na página da disciplina.
- Toda a memória alocada deve ser liberada adequadamente ao fim do programa. Isso pode te ajudar a evitar a ocorrência de falhas relacionadas a memória.
- Indente corretamente todo o seu código. Escolha entre espaços ou tabs para identação e seja coerente em todos os arquivos. Indentação é fundamental para a legibilidade do seu código, e ajuda no entendimento do fluxo de controle do seu programa.

5. Critérios de Avaliação

- Seu código deverá passar por todos os casos de teste do SuSy definidos para este laboratório. Caso positivo, utilizaremos os seguintes critérios adicionais:
    - A **não utilização da estrutura de pilha implementada usando uma lista ligada** para a solução dos problemas de conversão resultará em uma penalização na nota
    - A **não utilização da estrutura de pilha implementada usando um vetor** para a solução do problema de avaliação da expressão resultará em uma penalização na nota
    - O uso de recursão em conjunto com pilhas para resolução de qualquer um dos problemas propostos será penalizado
    - Falha na implementação do comportamento esperado para as pilhas acarretará em uma penalização
    - Utilização de bibliotecas ou funções não permitidas pelo enunciado resultará em uma **penalização severa** na nota