Algoritmos e Estruturas de Dados LC 2015/2

# Exercício Programa 3: Interpretador LEP

Rodrigo de Souza

Entrega: 06/12/2015

## 1 Enunciado

Nosso objetivo neste Exercício-Programa é muito interessante: vamos implementar um interpretador para uma linguagem de programação rudimentar, a LEP – Linguagem de Expressões Posfixas.

A LEP não é na verdade uma linguagem de programação completa, no sentido de que não permite escrever programas para resolver qualquer problema abordável em uma linguagem tradicional como C ou Java.[1] A LEP é uma calculadora de expressões na notação posfixa (que estudamos na aula), mas com a vantagem de permitir chamadas recursivas dentro de expressões. Com isso, com a LEP podemos calcular valores não-triviais, como os números de Fibonacci.[2]

Formalmente, uma expressão na linguagem LEP envolve um parâmetro $n$, cujo valor é inteiro e deve ser fornecido pelo usuário. Vamos denotar uma tal expressão por $E(n)$, onde $E$ é um identificador para a expressão e $n$ é o parâmetro. Então, $E(n)$ é uma expressão na notação posfixa usando os operadores aritméticos usuais de soma, subtração, produto e divisão, e onde os operandos podem ser:

- Números inteiros (vamos supor que positivos)
- O parâmetro $n$
- Chamadas recursivas $E(n-c)$, onde $c > 0$ é um inteiro, e o valor desse operando é o valor da expressão $E$ para o parâmetro $n-c$ (podemos supor que se $n < c$, o parâmetro a ser considerado é 0).

Ademais, toda expressão deve incluir valores para a base da recursão: a base consiste de uma faixa de números de 0 a um $b$ fixo, e para cada $i$ nessa faixa, especifica-se explicitamente um valor $E(i)$.

Sua tarefa é escrever um interpretador de expressões LEP que, ao ler uma expressão de um arquivo de entrada `lep.in`, executa as chamadas recursivas e imprime na tela o valor da expressão. O arquivo de entrada tem o seguinte formato:

[1] Diz-se que LEP não é Turing completa

[2] Essas expressões são o que se chama de recorrências em Análise de Algoritmos.

- a primeira linha tem o número $b$ representando a faixa de valores para a base da recursão;
- nas próximas $b+1$ linhas, temos os valores $E(0), E(1), \ldots, E(b)$ da base da recursão, um por linha (todos números inteiros positivos);
- na linha seguinte temos a expressão, que como já dissemos é uma expressão na notação posfixa (pode supor que bem formada) onde cada operando pode ser a variável $n$, números inteiros ou chamadas recursivas. Uma chamada da forma $E(n-c)$ é representda na expressão simplesmente por $-c$. Suponha que os componentes da expressão são separados por um ou mais espaços em branco;
- a última linha do arquivo tem o valor do parâmetro $n$, que é um número inteiro positivo.

Você não sabe de antemão o tamanho da expressão, e por isso ela deve ser armazenada em uma lista ligada $L$, onde cada nó armazena um componente da expressão (valor numérico, chamada recursiva, variável, operador). Você deve portanto implementar funções para manipular listas ligadas (e não usar uma biblioteca pronta para esse fim).

O cálculo do valor da expressão deve ser feito usando uma pilha de números inteiros. Seu algoritmo percorre a expressão da esquerda para a direita; sempre que encontra um operando numérico ou a variável $n$, empilha; sempre que encontra um operador, desempilha dois operandos, calcula o resultado da operação com esses valores, e o empilha. Você pode encontrar um exemplo de como se faz isso aqui:

`http://www.ime.usp.br/~pf/mac0122-2002/aulas/stacks.html`

Mas as chamadas recursivas complicam esse cálculo, porque sempre que uma chamada recursiva é encontrada, seu programa deve calcular o valor devolvido por essa chamada e empilhá-lo. Se $n-c$ é um valor na base da recursão, ou seja, $n-c \leq b$, basta devolver o valor correspondente de $E(n-c)$ (pode supor que caso $c > n$, $E(n-c) = E(0)$). Caso contrário, seu programa deve suspender o processamento da expressão (deixá-la em espera, digamos assim) e calcular o valor $E(n-c)$; somente após obter esse valor ele pode retomar o processamento. Para isso, você deverá implementar o que os sistemas operacionais fazem ao executar uma função recursiva: criam na memória uma pilha de execução (uma nova pilha, com outra finalidade, diferente das pilhas para o cálculo das expressões que comentamos acima).

O princípio da pilha de execução é o seguinte. Cada chamada recursiva ainda não finalizada (em espera) é um elemento na pilha; o topo da pilha representa a chamada recursiva atualmente em execução. Mais formalmente, cada chamada recursiva leva ao empilhamento de uma tripla $(m, L, p)$ onde $m$ é o valor do parâmetro da expressão na chamada recursiva, $L$ é (um ponteiro para) a lista representando a expressão posfixa, e $p$ é o nó na lista onde foi feita a atual chamada recursiva. Essa tripla é o que se chama *contexto* na área de Sistemas Operacionais. Então, a cada chamada recursiva, seu programa empilha um novo contexto, e processa a expressão $L$ com o parâmetro $m$; após o término dessa chamada recursiva, ou seja, após o término do cálculo de $E(m)$, o contexto é desempilhado, e a execução continua no contexto imediatamente abaixo (que

agora é o novo topo da pilha) a partir do nó $p$ da expressão. Observe que a cada chamada recursiva uma nova pilha de inteiros (para o cálculo da expressão) deve ser alocada, e essas pilhas também devem ser armazenadas nos contextos. Você pode na verdade elaborar o contexto como achar necessário, desde que deixe claro sua estrutura.

Como você está projetando um interpretador, que é uma peça de *software* elaborada, convém observar seu funcionamento na memória do computador para se certificar de que ele está funcionando corretamente, antes de entregar a versão de produção para o usuário. Então, seu programa deve, sempre que um novo contexto é empilhado, imprimir na tela a pilha de execução completamente.[3] Ao sair de uma chamada recursiva, deve imprimir o valor calculado para a expressão. Você também deverá definir um valor fixo para o tamanho máximo da pilha de execução, e caso esse tamanho seja excedido, imprimir a mensagem "ESTOURO DE PILHA"e parar a execução.

## 2 Instruções gerais

- Seu programa deve ser feito em C, Java ou Python.
- Não tente embelezar a saída do seu programa com mensagens, formatações, etc. Seu programa será julgado segundo a adequação de sua saída à descrição do exercício.
- Seu programa deve consistir de um único arquivo, e deve ser razoavelmente modular, ou seja: deve consistir de diversas funções que, juntas, realizam a tarefa descrita.
- Documente cada função dizendo o quê ela faz. Se o seu código não está legível, fica muito difícil adivinhar o que a função faz e isso prejudica a correção. Esse comentário deve especificar o que a função recebe (os parâmetros) e o que devolve.
- Capriche. Cuidado com a indentação. Deixei seu programa suficientemente modularizado para que cada tarefa específica esteja implementada em uma função (documentada explicando o que recebe e o que faz). Faça isso em particular para as estruturas de dados, implementando funções que realizam as operações pertinentes. Modularização e capricho serão considerados na nota.
- Escreva no início do código um cabeçalho com comentários, indicando nome, número do EP, data, nome da disciplina.
- Entregue também um mini-relatório, pode ser em formato txt, explicando como resolveu o exercício, ou seja, como funcionam as diversas partes do seu programa.
- A entrega será eletrônica no ambiente Moodle da disciplina (não receberei exercícios impressos ou via email).

[3] Essa observação da pilha de execução é uma das funcionalidades dos depuradores de programação, que permitem visualizar o estado da memória durante a execução de um programa.