T-201-GSKI, GAGNASKIPAN

Vor 2017

RECURSION

Assignment grading. For full marks your solution needs to be accepted by Mooshak. Points may be deducted for

- not implementing the solution according to the specifications,
- redundant or repeated code,
- messy code,
- code that is difficult to understand and not explained in comments.

If your solution is not accepted by Mooshak, it will receive a maximum grade of 7.

PREFIX EXPRESSION PARSER 50%

You are to implement a prefix expression parser using recursion. A prefix parser is a program which takes input in the form of prefix notation, also called polish notation, i.e. / 10 5 and returns the outcome which is 2. Prefix notation doesn't need any parenthesis do always be unambiguous, for example:
+ 5 - + 3 4 2 = (+ 5 (- (+ 3 4) 2)) = 5 + ((3 + 4) - 2) = 10
Another example of prefix notation is - * / 15 - 7 + 1 1 3 + 2 + 1 1 = 5.

THE IMPLEMENTATION

Implement the function parseExpression which takes an input stream as a parameter and returns an integer. The function should read the input from the stream, one token at a time (valid tokens being '+', '-', '*', '/' and any integer), and evaluate the calculations recursively. The function should be able to compute the value of any expression written in polish notation.
If the program would ever have to divide by zero it should instead throw a DivideByZeroException.

EXTRA

For more info about prefix notation see: https://en.wikipedia.org/wiki/Polish_notation

REVERSE LIST 50%

You are given a fully functioning linked list program. You are to implement the function reverse() which reverses the list using recursion.

THE IMPLEMENTATION

Implement the **public** function reverse(). It is recommended to make another, **private**, reverse function that uses recursion to reverse the list. The **public** function would then call the **private** one (It may do a bit more than that as well). The **private** function can use its parameters and return value to send information, for example which node is being worked on, through the process. The program should throw an EmptyException if a list is about to be reversed when it is empty.

TEMPLATE

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

#include <cstddef>
#include <iostream>

#include "listnode.h"

using namespace std;

class EmptyException { };

template <class T>
class LinkedList
{
    public:
        LinkedList() {
            head = NULL;
            tail = NULL;
        }

        virtual ~LinkedList() {
            ListNode<T> *tmp = head;
            while(head != NULL) {
                head = head->next;
                delete tmp;
                tmp = head;
            }
            head = NULL;
            tail = NULL;
        }
```

```
32.
33.         void push_front(T value) {
34.             head = new ListNode<T>(value, head);
35.             if(tail == NULL) {
36.                 tail = head;
37.             }
38.         }
39.
40.         void push_back(T value) {
41.             if(head == NULL) {
42.                 push_front(value);
43.             }
44.             else {
45.                 ListNode<T> *newNode = new ListNode<T>(value, NULL);
46.                 tail->next = newNode;
47.                 tail = newNode;
48.             }
49.         }
50.
51.         T pop_front() {
52.             if(head == NULL) {
53.                 throw EmptyException();
54.             }
55.             T data = head->data;
56.             ListNode<T> *tmpNode = head;
57.             head = head->next;
58.             if(head == NULL) {
59.                 tail = head;
60.             }
61.             delete tmpNode;
62.             return data;
63.         }
64.
65.         // This is the public reverse function, called from the outside
66.         void reverse(){
67.             // TODO: Implement this!
68.             // Note that you can make another, private, reverse function
69.             // that does the actual RECURSION.  It may take a parameter
70.             // and/or return a value/reference/pointer to aid the process.
71.             // This function then calls the private function once
72.             // while the private function does all the actual work.
73.         }
74.
75.         friend ostream& operator <<(ostream& outs, const LinkedList<T> &lis) {
76.             ListNode<T> *tmpNode = lis.head;
77.             while(tmpNode != NULL) {
78.                 outs << tmpNode->data << " ";
79.                 tmpNode = tmpNode->next;
80.             }
81.             return outs;
82.         }
83.
84.     private:
85.
86.         ListNode<T> *head;
87.         ListNode<T> *tail;
88.
89. };
90.
91.
```

```
92. #endif // LINKEDLIST_H
```

PREFIX EXPRESSION PARSER 50%

Þú átt að útfæra prefix expression parser með endurkvæmni. Prefix parser er forrit sem tekur inntak á formi sem kallað er prefix notation eða polish notation og lítur svona út: / 10 5. Þessi segð myndi skila 2. Prefix notation þarf enga sviga en er samt alltaf laus við tvíræðni, t.d.:

+ 5 - + 3 4 2 = (+ 5 (- (+ 3 4) 2)) = 5 + ((3 + 4) - 2) = 10

Annað dæmi um prefix notation er - * / 15 - 7 + 1 1 3 + 2 + 1 1 = 5.

ÚTFÆRSLA

Útfærið fallið parseExpression sem tekur inn inntaksstraum og skilar heiltölu. Fallið á að lesa inntak frá straumnum, einn tóka í einu (gildir tókar eru '+', '-', '*', '/' og allar heiltölur), og framkvæma útreikningana með endurkvæmni. Fallið á að geta reiknað segðir sem skrifaðar eru í polish notation. Ef að forritið stendur frammi fyrir því að þurfa að deila með núlli skal það þess í stað kasta DivideByZeroException.

EXTRA

Frekari upplýsingar um prefix notation: https://en.wikipedia.org/wiki/Polish_notation

REVERSE LIST 50%

Þér er gefið fullgild útfærsla á eintengdum lista. Þá átt að útfæra fallið reverse() sem notar endurkvæmni til að snúa listanum við.

ÚTFÆRSLA

Útfærið **public** fallið reverse(). Mælt er með því að útfæra annað, **private**, reverse fall sem notar endurkvæmni til að snúa listanum. **Public** fallið myndi þá kalla á private fallið einu sinni (það gæti gert aðeins meir en bara það). **Private** fallið getur síðan notað breytur og úttak til að koma skilaboðum áfram, t.d. Hvaða nóðu er verið að nota, gegnum ferlið. Ef að það er reynt að snúa lista við þegar hann er tómur skal kasta EmptyException.

SNIÐMÁT

Sjá sniðmát að ofan.