Frage **1**

Richtig

Erreichte Punkte 9,00 von 9,00

Leeroy der Listenrechner

Programmieren Sie einen einfachen Rechner, der zuerst einen Reihe von Operationen, Klammern und Werte einliest, diese in einer verketteten Liste speichert und sie dann auswertet!

Aufgabe

Ihr Rechenprogramm liest zeilenweise Zahlen, Operatoren und Klammern ein, speichert diese in einer verketteten Liste und wertet diese dann als einen einzigen Ausdruck aus.

Eingabe

Als Eingabe kommen Integer-Zahlen, Operatoren und Klammern (`'+'`, `'-'`, `'*'`, `'/'`, `'('`, `')'`). Sie können mit der Funktion `inputToNode`, die bereits implementiert ist, Eingaben in entsprechende Knoten umwandeln, implementieren Sie dazu noch die fehlenden Funktionen `createParenthesisNode`, `createOperatorNode` und `createNumberNode` (siehe auch Vorausfüllung in Antwort).
Die Eingaben sollen in einer verketteten Liste gespeichert werden. Jeder Knoten der Liste kann sowohl einen char für den Operator als auch einen Integer für eine Zahl speichern. Zusätzlich enthält der Knoten einen enum-Typen der angibt, ob es sich um einen Operator oder eine Zahl handelt.
Jeder neue Knoten wird am Ende der Liste eingefügt und nach jedem Einfügen soll die Liste ausgegeben werden.
Die Eingabe wird durch ein `'='` beendet. Dieser Operator wird nicht mehr in die Liste eingefügt.

Auswertung

Wurde die Eingabe beendet soll die Liste in drei Schritten ausgewertet werden. Dabei wird sie vom Anfang zum Ende hin ausgewertet. Es gelten folgende Regeln:

- Es werden nur Integerrechnungen durchgeführt.
- Divisionen durch 0 werden zu Divisionen durch 1.
- Im ersten Schritt werden Klammern aufgelöst und berechnet (Hinweis: Suchen Sie nach der letzten öffnenden Klammer und lösen Sie diese zuerst auf)
- Im zweiten Schritt werden nur Punktoperationen durchgeführt (Multiplikation und Division).
- Im dritten Schritt werden nur Strichoperationen durchgeführt (Addition und Subtraktion).
- Immer wenn ein Operator ausgewertet wird und eine Berechnung stattfindet soll die Liste danach neu ausgegeben werden. Zum Beispiel:
  - (2 + 8) / 5 * 4 - 3
  - (10) / 5 * 4 - 3
  - 2 * 4 - 3
  - 8 - 3
  - 5

Bei jedem Zwischenergebnis reduziert sich somit die Anzahl der Knoten und am Ende der Auswertung sollte die Liste nur noch einen Knoten enthalten - der mit dem Endergebnis (siehe vorhergehendes Beispiel).

Hier ein Beispiel des Programmablaufs:

```
Input: 4

Term: 4
Input: *

Term: 4*
Input: (

Term: 4*(
Input: 5

Term: 4*(5
Input: +

Term: 4*(5+
Input: 2

Term: 4*(5+2
Input: )

Term: 4*(5+2)
Input: /

Term: 4*(5+2)/
Input: 2

Term: 4*(5+2)/2
Input: =

Resulting term: 4*(7)/2
Resulting term: 28/2
Resulting term: 14
```

Die Struktur der Knoten ist bereits definiert:

```
struct node { char operator;
    int number;
    enum node_type type;
    struct node* next;
};
```

Das enum type bestimmt dabei ob es sich um einen Knoten für einen Operator, eine Zahl oder eine Klammer handelt. Das entsprechende Enum ist ebenfalls bereits definiert. Wurde eine Zahl eingegeben wird also ein Knoten erstellt, bei dem dann number gesetzt wird und der type auf number_type. Wurde ein Operator eingegeben wird nur operator gesetzt und der type auf operator_type. Wurde eine Klammer eingegeben wird operator gesetzt und der type auf parenthesis_type. Es werden nie beide Felder belegt (also number und operator).

Folgende Funktionen könnten hilfreich sein, sind aber nicht vorgeschrieben:

- Das Einfügen eines Knotens am Ende der Liste:
  `struct node *addLast(struct node *head, struct node *newNode);`
- Das Finden des Knotens der die letzte öffnende Klammer enthält '(', um von dort aus die Klammer auswerten zu können (bis zu ')'):
  `struct node *findLastParenthesisOpen(struct node *head);`
- Das Finden des Knotens, der den nächsten Operator enthält (Achtung: Punkt vor Strich!):
  `struct node *findFirstPointOperator(struct node *startNode);`
  `struct node *findFirstDashOperator(struct node *startNode);`
- Das Finden des Knotens, der in der Liste vor einem bestimmten Knoten liegt (um den ersten/linken Operanden zu erhalten):
  `struct node *findPrevious(struct node *head, struct node *node);`
- Das Löschen eines bestimmten Knotens innerhalb der Liste (um bereits verarbeitete Knoten zu entfernen).

Wichtige Hinweise

- Verwenden Sie dieselbe Ein- und Ausgabestruktur wie im obenstehenden Beispiel, damit Sie die automatisierten Tests erfolgreich bestehen können. Der rote Text dient nur zur Hervorhebung des User Inputs im Beispiel, Sie müssen diese Färbung nicht im Programm abbilden!
- Geben Sie **Zeilenumbrüche immer vor einer neuen Zeile aus und nicht am Ende**. Wenn Sie sich daran halten, sollte die Formatierung leichter zur Übereinstimmung mit der erwarteten Ausgabe gebracht werden können. Das Programm startet daher auch mit einem

Zeilenumbruch.

- In CodeRunner wird jede einzelne Eingabe von einem Zeilenumbruch bestätigt, der in der Ausgabe aber nicht sichtbar ist. Wenn Sie außerhalb von CodeRunner das Programm starten, selber Werte eingeben (siehe Beispielausgabe) und dabei Zeilenumbrüche produzieren, wirkt sich das auf die Ausgabe aus (mehr Zeilenumbrüche als beim automatischen Testen in CodeRunner).
- Achten Sie darauf den Speicher korrekt zu verwalten.
- Es werden **immer** gültige Rechnungen eingegeben, Fälle wie fehlende schließende Klammern werden nicht getestet.