Problem J: いにしえの数式

問題作成・解説: 北村
解答作成協力: 八森

問題

- 演算子の優先順位と結合方向が与えられる
- どの項とどの項が結合しているかを正しく表現せよ
- 例
  - ＋ より * の方が優先順位が高い
  - * は左から右に結合する
  - このとき、a ＋ b * c * d は (a+((b*c)*d)) となる

解法

- 入力形式がややこしい！
  - 頑張って読みましょう
- 入力文字列の長さがたかだか100文字なので、効率の悪い方法でも正しくパーズさえできれば解ける
  - 入力文字列の長さ L に対して、$O(L^2)$-time のアルゴリズムでも可
  - $O(L)$-time で解くアルゴリズムも存在する

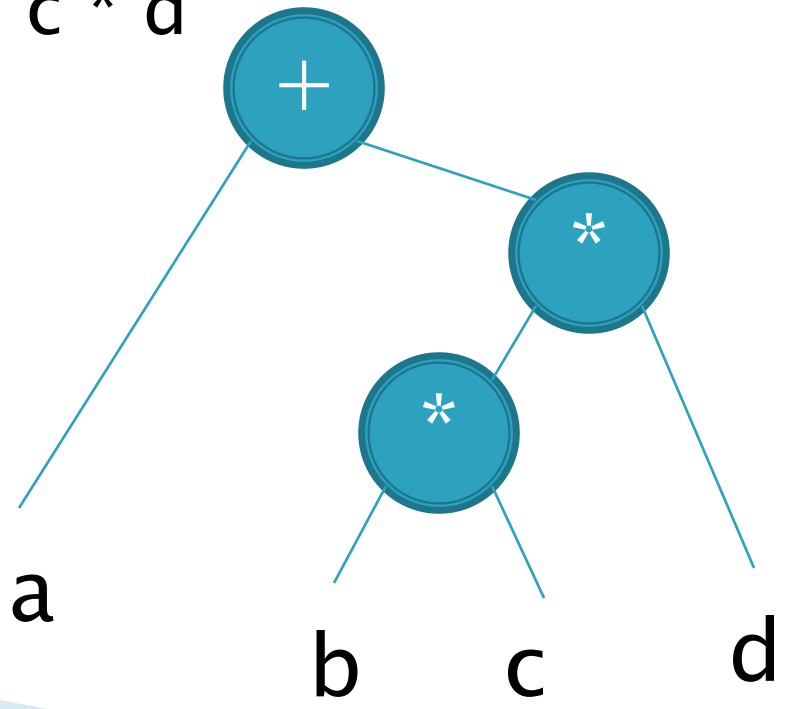
素朴な方法 $O(L^2)$-time

- 変数と演算子をそれぞれ独立したノードとする
- 一番優先順位の高い演算子を選ぶ
  - 優先順位が同じ演算子が複数ある場合は、左結合のときは左から、右結合のときは右から優先して選ぶ
- 演算子とその両隣のノードをまとめて1つのノードにする
- ノードが1つになるまで繰り返す
- 括弧は前もって処理
- 例: a + b * c * d
  - a + b * c * d
  - a + (b*c) * d
  - a + ((b*c)*d)
  - (a + ((b*c)*d))

他にも、優先順位の低い演算子から
処理していくなど、
いろいろなバリエーションはありうる

構文木

- 演算子を節点、変数やリテラル項などを葉ノードとしたツリー
- 処理を再帰的に書くことができる
- 例: a + b * c * d



再帰降下法

- 文字列を効率よくパーズする手法 O(L)-time
- やり方
  - 左再帰のない文脈自由文法を作る。例:
    - Expr ::= Term ('+' Term | '-' Term)*
    - Term ::= Factor ('*' Factor | '/' Factor)*
    - Factor ::= Var | '(' Expr ')'
  - 非終端記号それぞれについてパーザ関数を作る
  - 必要なら先読みをして入力文字列の読み取りが逆戻りしないようにする
    - Expression* expr() {
      - t1 = term();
      - if (nextchar() == '+') then ...
      - ... return new Expression(...);
    - }
- 詳しくはコンパイラの教科書を参考のこと

この問題でも再帰降下法が使えるか？

- 使えます
- それぞれの結合の強さをレベルとして、以下のように文法を定義する
  - $\mathrm{Expr} ::= \mathrm{Expr}_0$
  - $\mathrm{Expr}_i ::= \mathrm{Expr}_{i+1}\ (\ op_i\ \mathrm{Expr}_{i+1}\ )^*$
  - $\mathrm{Expr}_L ::= \mathrm{Var} \mid$ ‘(‘ Expr ‘)’
  - なんと関数1個でパーズできる
- 同一レベル内の項が揃ってから結合方向に従って構文木を作ればよい

解答状況

- 14 accepts / 24 submissions
- 14 users tried
  - 挑戦した方は皆さん解けたようです
- 最初の正答は高橋周平さん（171分）