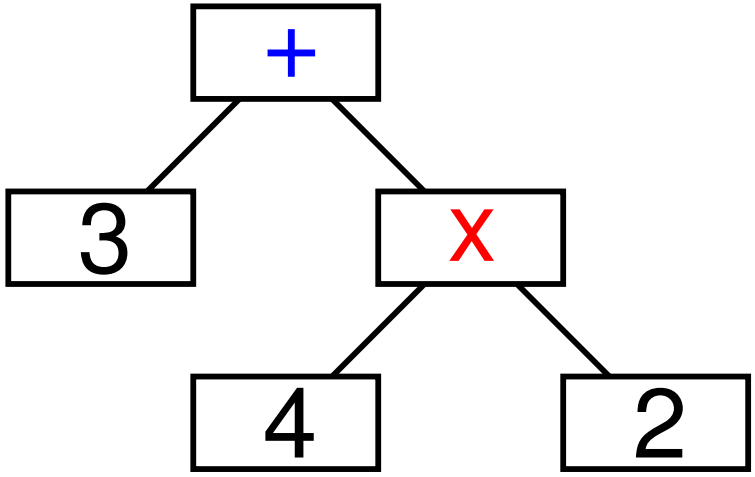
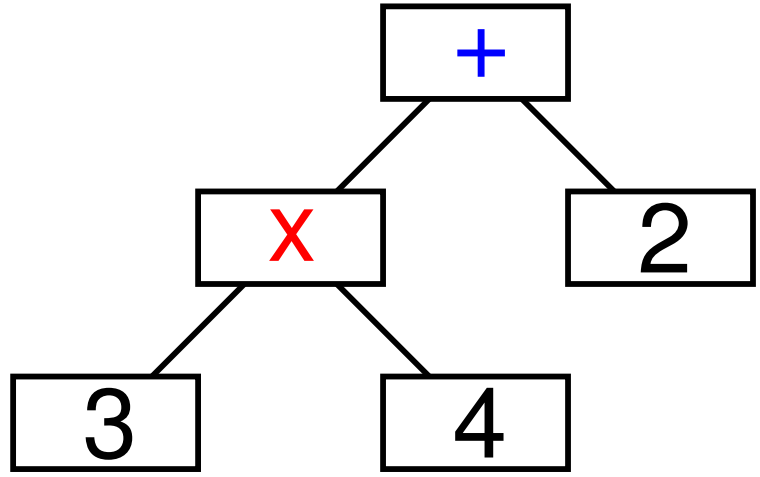
演算木



3+4x2



3x4+2

数式処理 (計算機代数) ソフトウェアでは、

通常、内部的に数式の木構造を保持

$\longrightarrow$ **Mathematica で見てみよう**

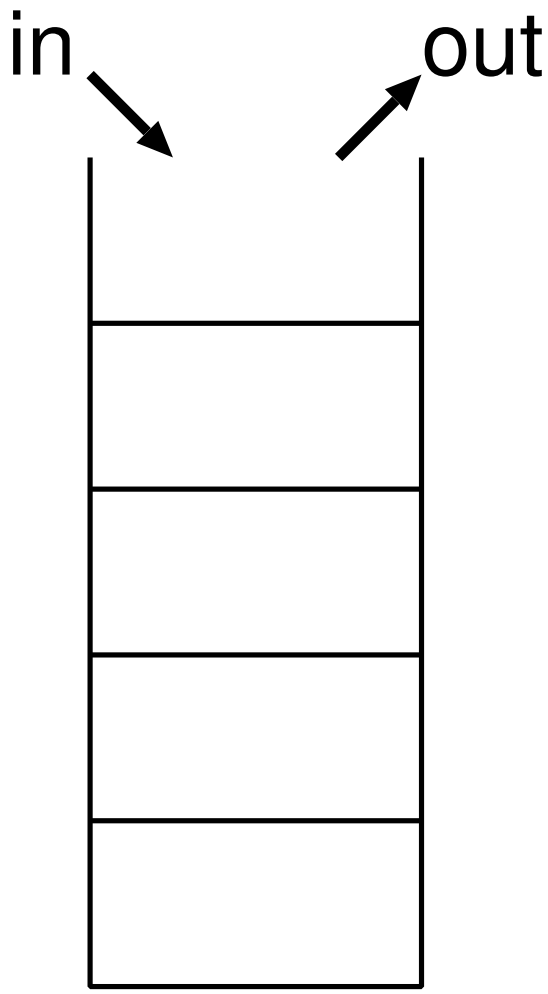
中置	前置	後置
$3+4*2$	$+\ 3\ *\ 4\ 2$	$3\ 4\ 2\ *\ +$
$(3+4)*2$	$*\ +\ 3\ 4\ 2$	$3\ 4\ +\ 2\ *$
$3*4+2$	$+\ *\ 3\ 4\ 2$	$3\ 4\ *\ 2\ +$

後置	日本語
$3\ 4\ 2\ *\ +$	3 に 4 に 2 を**掛**けたものを**足**したもの
$3\ 4\ +\ 2\ *$	3 に 4 を**足**したものに 2 を**掛**けたもの
$3\ 4\ *\ 2\ +$	3 に 4 を**掛**けたものに 2 を**足**したもの

$\Downarrow$

スタックを用いた計算に便利

プッシュダウンスタック (pushdown stack)



- 新しい要素を先頭に挿入
  (プッシュ (push))
- 先頭の要素を取り出して削除
  (ポップ (pop))

Last-In First-Out (LIFO)

後置記法の演算式のスタックを用いた計算

- **数値**

  $\Longrightarrow$ **push**

- **演算子**

  $\Longrightarrow$ **被演算子を (所定の個数だけ) pop**

  $\longrightarrow$ **演算を施し、結果を push**

- **入力終了**

  $\Longrightarrow$ **pop**

  $\longrightarrow$ **スタックが丁度空になったら**
  その値が答え

演習 10

後置記法 (逆ポーランド記法) の式に対し

スタックを用いて値を計算する、

アルゴリズムを実装せよ。

(任意課題だが提出を強く推奨する)

演算記法の補足

後置記法の演算式が簡明に計算できるのは、

(各演算子に対して
被演算子の個数が決まっていれば)

括弧が必要ない (優先順位を考慮しなくてよい)

ことが大きく効いている。

- 式 :: 定数 || 変数 || 式 式 二項演算子
(+ も × も区別なし)

演算記法の補足

後置記法の演算式が簡明に計算できるのは、

(各演算子に対して
被演算子の個数が決まっていれば)

括弧が必要ない (優先順位を考慮しなくてよい)

ことが大きく効いている。

- **式 :: 定数 || 変数 || 式 式 二項演算子**
  (+ も × も区別なし)

演算記法の補足

中置記法の演算式には括弧が必要
(演算子の優先順位を定めておく必要あり)

$$3 \times 4 + 2 \qquad\qquad 3 + 4 \times 2$$

計算する際には優先順位を考慮する必要がある

- **式 :: 項 || 項 + 式**
- **項 :: 因子 || 因子 × 項**
- **因子 :: 定数 || 変数 || (式)**

(+ と × とで純然たる区別あり)

演算記法の補足

中置記法の演算式には括弧が必要
(演算子の優先順位を定めておく必要あり)

$$3 \times 4 + 2 \qquad\qquad 3 + 4 \times 2$$

計算する際には優先順位を考慮する必要がある

- **式** :: **項** || **項** $+$ **式**
- **項** :: **因子** || **因子** $\times$ **項**
- **因子** :: **定数** || **変数** || (**式**)

($+$ と $\times$ とで純然たる区別あり)

さて、本授業の大きめの目標の一つ、

多項式の計算の実装

を考えよう。

仕様: 簡単の為、

- **整数係数の**
- **一変数多項式で**
- **変数名は X で固定**

としよう。

出来るようにしたいこと:

- **入力・表示**
- **定数倍・加減算・乗算・整除**

「多項式型」の設計
(= 「多項式」というデータをどう保持するか)

不定長・可変長のデータ
$\longrightarrow$ リストで実装しよう

例えば、

- 係数のリスト
  (抜けている次数は係数 0 とする)
- (係数, 次数) の組のリスト

「多項式型」の設計

例: $f(X) = 2 - X + 4X^3 + X^5$ **の場合**

- **係数のリスト:**
  → 2 →-1 → 0 → 4 → 0 → 1 → NULL

- **(係数, 次数) の組のリスト:**
  → (2,0) → (-1,1) → (4,3) → (1,5) → NULL

いづれにせよ、データ型の設計が決まったら、やることは決まるので、後はコーディング

← データ型の設計までが一番の考え所

データ型の設計を決めないと書けないので、
暫定的に決めて書いてみよう。

- データ型の設計
- 関数プロトタイプの設計
- 関数の実装

→ データ型の設計からやり直すこともある

いづれにせよ、データ型の設計が決まったら、
やることは決まるので、後はコーディング

⟵ データ型の設計までが一番の考え所

データ型の設計を決めないと書けないので、
暫定的に決めて書いてみよう。

- **データ型の設計**
- **関数プロトタイプの設計**
- **関数の実装**

⟶ データ型の設計からやり直すこともある

いづれにせよ、データ型の設計が決まったら、
やることは決まるので、後はコーディング

⟵ データ型の設計までが一番の考え所

データ型の設計を決めないと書けないので、
暫定的に決めて書いてみよう。

- **データ型の設計**
- **関数プロトタイプの設計**
- **関数の実装**

⟶ データ型の設計からやり直すこともある

いづれにせよ、データ型の設計が決まったら、
やることは決まるので、後はコーディング

⟵ データ型の設計までが一番の考え所

データ型の設計を決めないと書けないので、
暫定的に決めて書いてみよう。

- **データ型の設計**
- **関数プロトタイプの設計**
- **関数の実装**

⟶ データ型の設計からやり直すこともある

いづれにせよ、データ型の設計が決まったら、
やることは決まるので、後はコーディング

$\longleftarrow$ データ型の設計までが一番の考え所

データ型の設計を決めないと書けないので、
暫定的に決めて書いてみよう。

- **データ型の設計**
- **関数プロトタイプの設計**
- **関数の実装**

$\longrightarrow$ データ型の設計からやり直すこともある

以下、

今回は実習時間とするので、

今迄の課題の未提出分や
実習課題・演習課題に取り組もう。

質問歓迎