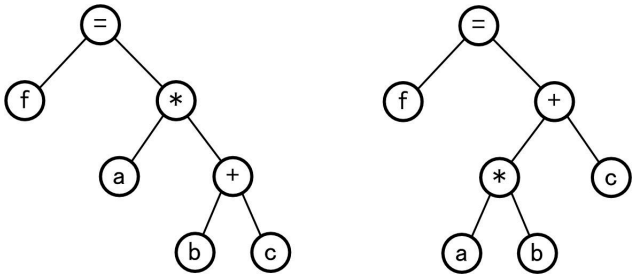
V プログラムの翻訳技術（第2回　構文解析）

前回は構文解析における文法チェック機能であるパージングについて述べた。今回は、構文解析の処理の中で、解析木を作る処理について説明する。
なお、説明や処理を理解しやすくするため、本講義では数式を対象として扱う。

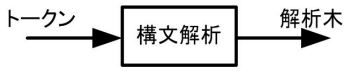
解析木
f = a * b + c
を解析木にすると、二通りできる。これは計算の優先順位を考慮していない場合で、考慮すると右側が正しい。



式が f = a * (b + c)ならば、左が正しい

これらを区別して最終的に解析木に変換する。解析木において括弧‘(‘ , ’)’は無くなる。つまり
f = a * b + c　ならば右側の解析木
f = a * (b + c)　なら左側の解析木
を作るような処理を考える。

2．2　構文解析（解析木の作成）

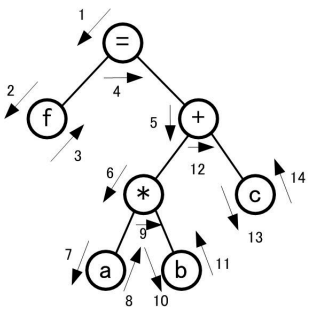


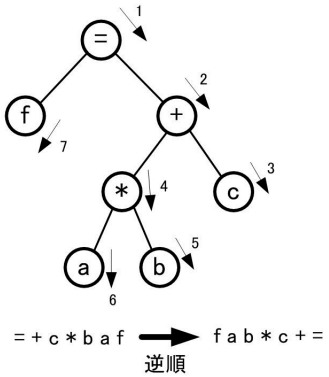
構文解析部はトークンを入力し、構文を解析して解析木を出力する処理を行う。

2．2．1　逆ポーランド式(Inverse Polish Notation)

四則演算式の構文解析を行ってみる。
この計算では、＋,－,＊,／のみからなる計算で優先順位が存在し　＋,－よりも＊,／が優先される。

解析木の特徴
f = a * b + c の解析木に対して





解析木を上から左優先にたどっていき、ノードを2回通る順に書き出すと

f = a * b + c

のように、与えられた演算式が出力される。

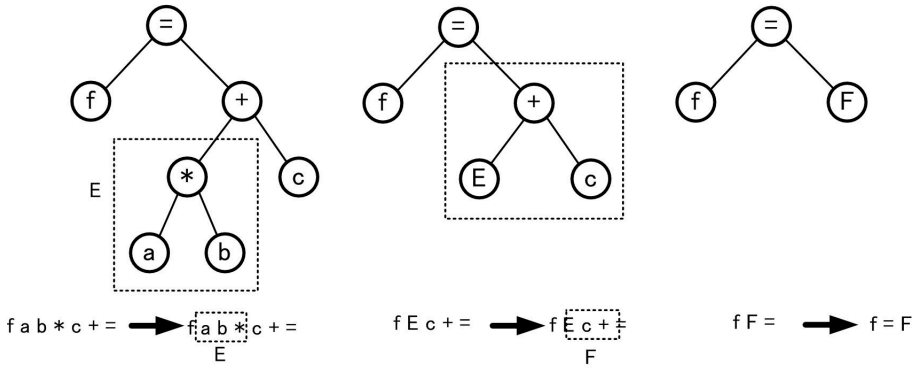
一方、解析木を上から右優先にたどっていき、ノードを通る順に書き出すと

= * c + b a f

となる。これを逆順にすると

f a b * c + =

となる。実はこの式は計算の手順を示していることになる。左から順に呼んでいき演算子が出てきたら直前の二つの値について演算し、その値を格納する。さらに読み進めこの処理を繰り返す事により計算が完了する。



つまり、この式は計算手順を示しており、この構造が解析木と同様の役割を果たしていることがわかる。

この表現方式は、逆ポーランド式とよばれ、構文解析でよく利用される方式である。もちろん、数式以外の構文解析にも利用されている。

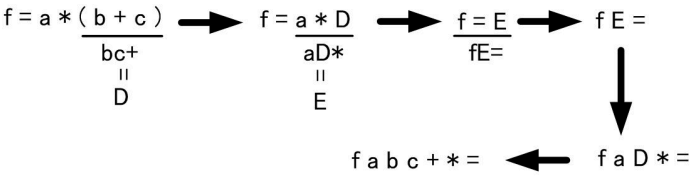
逆ポーランド式は後置記法ともよばれる。それに対して通常我々が使う式を挿入記法（中置記法）と呼ぶ。逆ポーランド式では人間が用いる挿入表記に比べ、①括弧が不要となる②式の計算は左から右に順に処理を行うことで計算できる。という利点を持っている。この利点はコンピュータが処理するのに都合がよい。

ちなみに前置記法もあり、これを考えた一がポーランド人だったので、ポーランド法と名付けられた。これの全く逆なので逆ポーランド法と名付けられた。 +ab(前置記法)　a+b(中期記法)　ab+(後置記法)

逆ポーランド式変換の仕方

例

f = a * (b + c)



練習1

次の式を逆ポーランド式で表現し、解析木にせよ。

(1)　f = a + b - c

(2)　f = a * b + c * d

(3)　f = a – b * c + d

(4)　f = (a – b) / (c + d)

(5)　f = (a + b) – c * (d + e)

cf. 具体的な数値が入った式でも同じ扱いができる。ここでは表現が曖昧になるため、全て変数として扱う

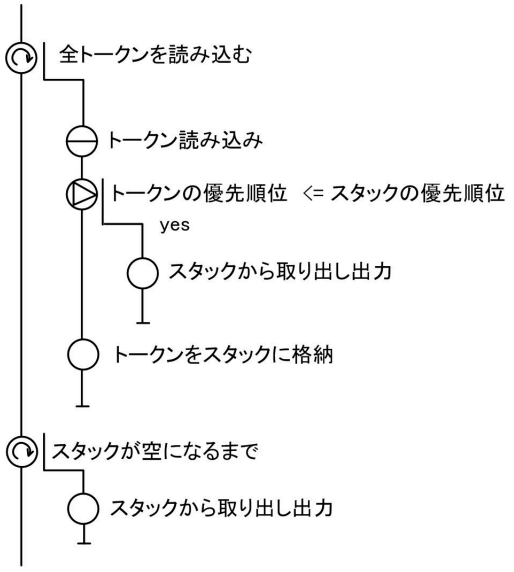
f = 52 * a + c　----> f52a*c+=

2．2．2 逆ポーランド式の算出プログラム（構文解析プログラム）

人間が行うと比較的簡単に逆ポーランド式を作成することができるが、これをコンピュータにやらせるにはアルゴリズムを考える必要がある。

[四則演算対応]

構文解析（逆ポーランド式作成）



優先順位表

オペランド	3
*, /	2
+, −	1

ここで、オペランドとは四則演算以外の変数や数値を示す。

実際に a + b * c を上述のアルゴリズムに従って実行すると、

a + b * c

リードトークン	比較	スタック	出力
a	a(3) > NULL(0)	a	
+	+(2) < a(3)		a
	+(2) > NULL(0)	+	a
b	b(3) > +(1)	+, b	a
*	*(2) < b(3)	+	ab
	*(2) > +(1)	+, *	ab
c	c(3) > *(2)	+, *, c	ab
			abc
			abc*
			abc*+

cf. スタック

スタックとはプログラミングで用いられるデータ構造のひとつで Last In First Out の構造を持つ。

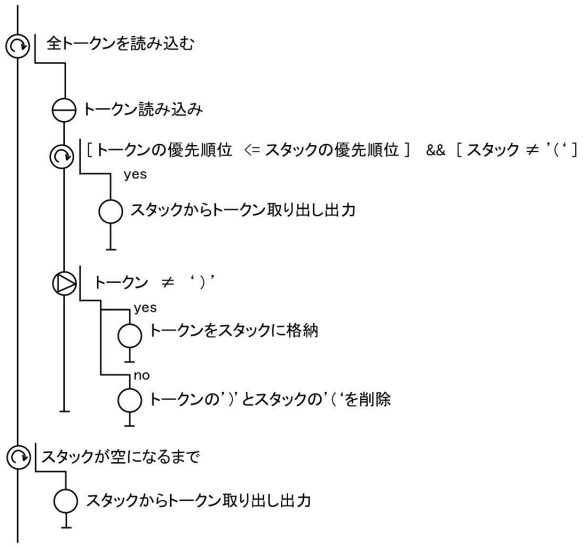
つまり、机の上に本を積むようにデータを積み上げ、取り出すときは最も上の本からのみ取り出せる機構を持ったデータ構造である。データを積み上げることを push データを取り出すことを pop と呼ぶ。

[括弧付き演算対応(代入を含む)]

次に括弧付きの計算を可能にするアルゴリズムを示す。括弧が有効になった場合には「優先順位表」を改訂する

必要がある。さらに、代入演算子'='も扱えるようにする。

構文解析（逆ポーランド式作成）



優先順位表

オペランド	6
(	5
*, /	4
+, −	3
)	2
=	1

実際に f = a * (b + c)を実行してみると

f = a * (b + c)

リードトークン	比較	スタック	出力
f	f(6) > NULL(0)		f
=	=(2) < f(6) && stack≠'('		f
	=(2) > NULL(0)	=	f
a	a(6) > =(1)	=, a	f
*	*(4) < a(6) && stack≠'('	=	fa
	*(4) > =(1) && stack≠'('	=, *	fa
(	((5) > *(4) && stack= '('	=, *, (	fa
b	b(6) > ((5) && stack≠'('	=, *, (, b	fa
+	+(3) < b(6) && stack≠'('	=, *, (	fab
	+(3) < ((5) && stack = '('	=, *, (, +	fab
c	c(6) > +(3) && stack ≠ '('	=, *, (, +, c	fab
)	)(2) < c(6) && stack ≠ '('	=, *, (, +	fabc
	)(2) < +(3) && stack ≠ '('	=, *, (	fabc+
	)(2) < ((5) && stack = '('	=, *,	fabc+
			fabc+*
			fabc+*=