コンパイラ演習資料(LL(1)文法)　担当：佐々木晃

□演習問題 A7

問題番号: A7
課題名：LL(1)文法
問題：スライド p41 問題 A(1)から(4)。（(4)の e は ε ではなく e という終端記号である。）

□字句解析(Lexical Analysis)（復習）

□文（プログラム）を単語に分ける。

→文の中の単語を一つ一つ「辞書」を引きながら（パターンマッチング）

→"Ihaveapen."

□"abc=123+ cde"

□「辞書」の内容

○0,1,2,3, ...→数字

○a, aa, aaa, ... ab, ... a1..　→　識別子（名前）

○= →　演算子=

○+ →　演算子+

○タブ文字、空白文字、改行文字（列）など　→　空白空(White Space)

□単語（これを字句、（あるいは符、トークンとなどとも）という）の列に変換

[("abc", 識別子), ("=", 演算子=), ("123", 数字), ("+", 演算子+), ("efg", 識別子)]

→White Space は通常省略される。

□難しい点

□処理を行うときは、先頭から一文字ずつ取り出しながら行う。コンピュータは全体を同時に見渡せない。

□最長一致、a も、ab も識別子の辞書に含まれているけれど、abc で一つの識別子として認識する。

→数字や識別子のようなものは、どこまでが単語かは少なくとも一つ先まで調べて見ないとわからない。

□構文解析(Parsing, Syntax Analysis)

□単語の並びを構造化する（→単語の並びを大雑把な塊としてとらえる。）

→単語の並びが、どういう文型にマッチするかを調べる。（パターンマッチング）

□[("abc", 識別子), ("=", 演算子=), ("123", 数), ("+", 演算子+),("efg", 識別子)]

□文型カタログ（代入式の文法）（→文脈自由文法、BNF）

文法 G = (P, <代入式>)

P = {

<代入式> → **識別子　演算子=** <式>

<式>　　→ <式>　**演算子+**　<式>

　　| **識別子**

　　| **数**　　}

□L(G)が、代入式として許される単語の並びの全てパターン。

L(G)＝{[**識別子　演算子=　識別子**], [**識別子　演算子=　数**],

[**識別子　演算子=　識別子 演算子+ 数**].....}

□解析木あるいは抽象構文木(AST- Abstract Syntax Tree)として構造化する。

→構造化されていると、意味をとりだしやすくなる。

<式>　→ <式>　**演算子+**　<式>

（左の部分式の結果、右の部分式の結果に演算＋を適用すると、式全体の結果が得られる）

→意味をとりだせなければ翻訳（コンパイル）できない。

□難しい点

□処理を行うときは、先頭から一単語ずつ取り出しながら行う。コンピュータは全体を同時に見渡せない。

→いかに非決定性プロセスをなくすか？（パターンマッチの効率化）

□練習問題 1

□スライド p2 の文法 G1 で (a + b) * c の解析木と抽象構文木(AST)を作成せよ。

□上記の解析木の最右導出、最左導出を示せ。

□練習問題 2：methodS()から作られる呼び出し木(call tree)の全てのパターンを列挙せよ。

```
static void main(...){
   methodS();
}
static void methodS(){
    methodA();
    methodB();
}
static void methodA(){
    if(ある条件){
        read_a();
    } else {
        read_e();
    }
}
static void methodB(){
    if(ある条件){
        read_b();
    } else {
        read_e();
    }
}
```

```
static void read_a(){
   //a を読んだ時の処理
}
static void read_b(){
   //b を読んだ時の処理
}

static void read_e(){
   //何もしない
}
```

□例題 次の文法 G は LL(1)文法である。

```
G=(P, S)
P={
 S→AB,
 A→a,
 A→ε,
 B→b,
 B→ε}
```

ちなみに、L(G) = {ε, a, b, ab}

LL(1)文法かどうかを判定するために Director 集合を求める。Director 集合を求めるには、First 集合と Follow 集合を計算する必要がある。

○First(α1α2...) : 文法記号列α1α2...から生成される終端記号列を考えたとき、その先頭にくる可能性のあるものは何か？を表したもの（集合)。記号列からεが生成される可能性がある場合は、εも含まれる。葉に近いほうから求めるのが良い。

```
First(a) = {a}
First(b) = {b}
First(ε) = {ε}

First(A) = {a, ε}
First(B) = {b, ε}
First(AB) = {b, a, ε}
First(S) = First(AB)
```

注意：First にはεが含まれる。$は含まれない。

○Follow(X)（X は非終端記号）: X の後にくる可能性のある終端記号列のうち先頭のものは何か？を表し

たもの（集合)。X の後に何もこない可能性がある場合は、$（単語の終わりを表す）という特別なトークンが含まれるものとする。

Follow(S)={$}　（S'→S$という規則があるものと考える。）
Follow(B) = {$}　B の後に来るもの＝S の後ろに来るもの（すなわち$）
Follow(A)= {b, $} A の後に来るもの＝B。ただし、B からはεも生成されるので、その場合は A の後ろに来るもの＝B の後に来るもの＝$。B からε以外が作られる場合は、B の始まりの文字＝b

仮に S→ABa の場合
Follow(B)は、B の後に来るもの＝a となる。（Follow(A)も上とは違うものになる。）

注意：Follow にはεは含まれない、$は含まれる。

○Director(X, α1α2...)：X→α1α2...を適用する際に、次に読むべきトークンの集合。

Director(S, AB) = {a, b, $}
Director(A, a) = {a}
Director(A, ε)= {b, $}
Director(B, b)= {b}
Director(B, ε) = {$}

注意：Director にはεは含まれない、$は含まれる。

○LL(1)文法であるかどうかの判定

複数個を持つ Director は、A,B であるが、それぞれ、
Director(A,a) ∩Director(A, ε)= {a}∩{b, $} = {}
Director(B,b) ∩ Director(B, ε) = {b}∩{$} = {}
であり、Director 間には共通部分がないから、G は LL(1)文法である。

→非終端記号から複数の導出可能性がある場合、その生成規則の Director 集合を比べて、同じものが含まれていなければ（共通部分が空）次に読むトークンによってどの生成規則を選べばよいか自動的に決められるということである。
→例えば、非終端記号 A に対しては、次のトークンが a の場合は A→a、次のトークンが b か$(読む単語がなし)の場合は、A→ε を選択すればよい。（ちなみに、それ以外のトークンがきたときは構文エラーである（Syntax Error, 文法違反＝その文法で想定されない単語の並びになっている）ことを示す。）