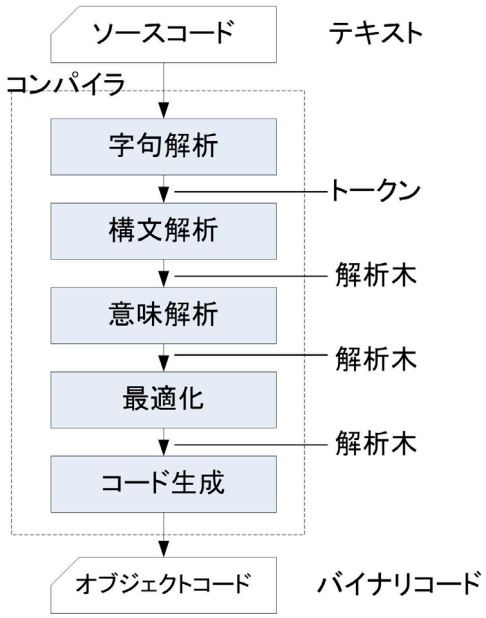
V プログラムの翻訳技術（第1回　字句解析とパージング）

（コンパイラの位置づけとプログラムの構成要素）

前回講義したプログラムの翻訳（コンパイル）処理における字句解析、構文解析、意味解析、最適化について具体的な処理技術について説明する

コンパイラの構成



1. 字句解析

プログラムコードを意味のある語句に分割する。処理は1文字ずつ読み込みながら意味のある区切りに分割し、文字配列に格納する。

[C言語の例]

入力コード

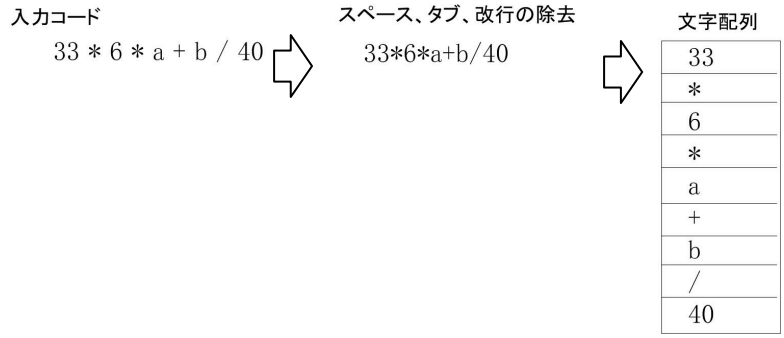
```
while( x == 0 ){
    x = x + x ** x;
    x = x - 1;
}
```

スペース、タブ、改行の除去

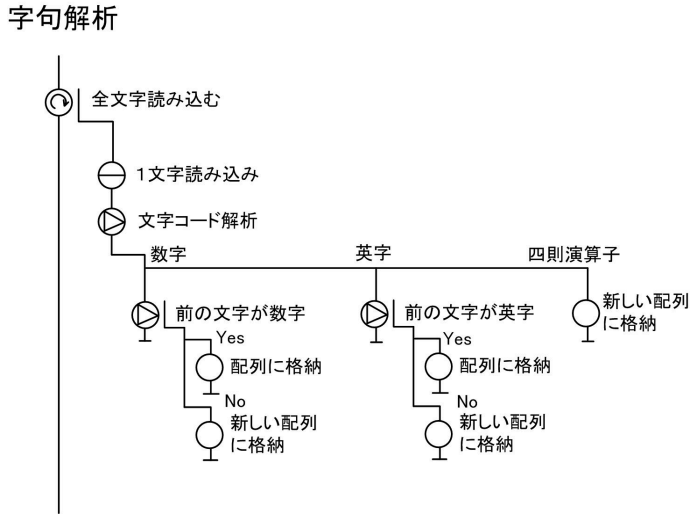
```
while(x==0){x=x+x**x;x=x-1;}
```

While
(
x
==
0
)
{
x
=
x
+
x
*
x
;
x
=
x
-
1
;
}

[数式の例]



[字句解析の処理手順(数式)]･･･数字,変数,四則演算子



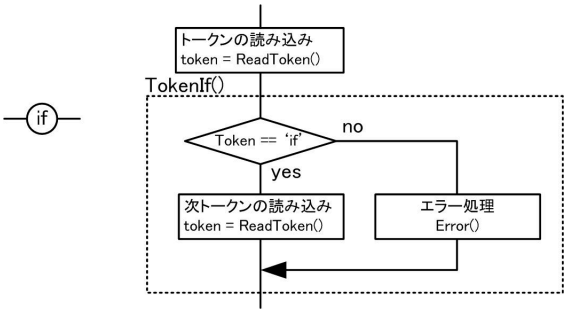
2. 構文解析

構文解析はプログラムが文法(BNF あるいは構文木)通りに記述されているかを確認する作業と、中間コード(解析木)を作成する作業を行う。特にプログラムコードが与えられた文法に合致しているかを判定するためのプログラムをパーザー(Paser)と呼び、パーザを行うことをパージング(Parsing)と呼ぶ。

2. 1パーザの構造(構文図とパーザ)

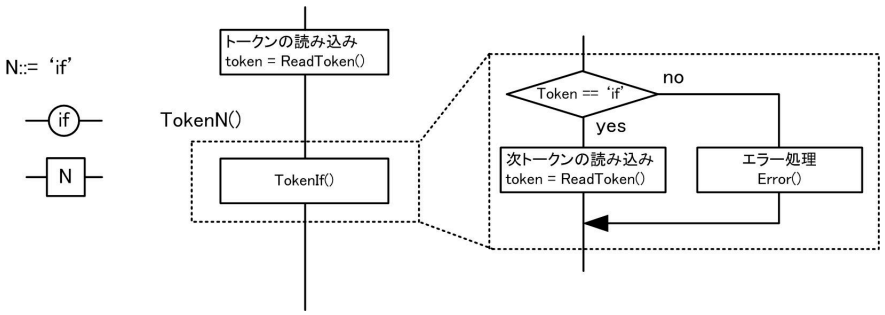
構文図からパーザを行う処理方法を述べる。ここで、ReadToken()は配列からトークンを取り出す関数であり、各処理の実行前に読み出されているとする。また、Error()はエラーを表示する関数である。

(1)トークン




```
if(token==”if”)
      ReadToken();
else
      Error();
```

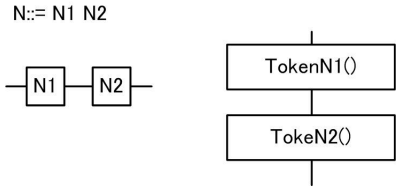
(2)ブロック



TokenN();

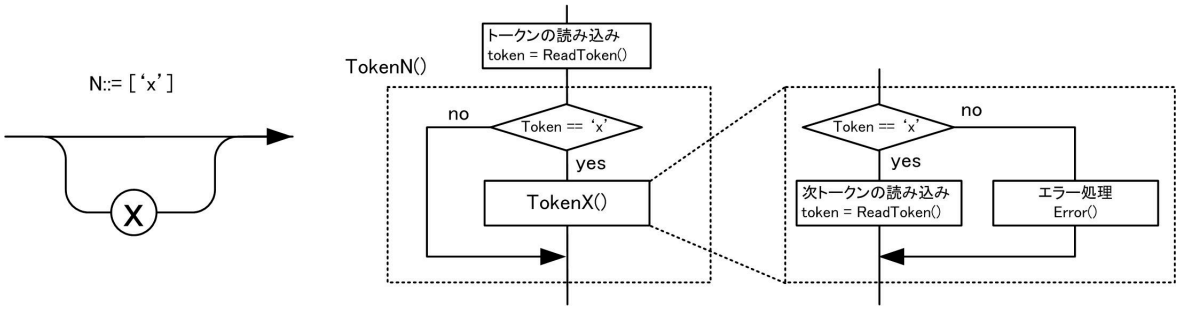
```
if(token==”if”)
      ReadToken();
else
      Error();
```

(3) シーケンス



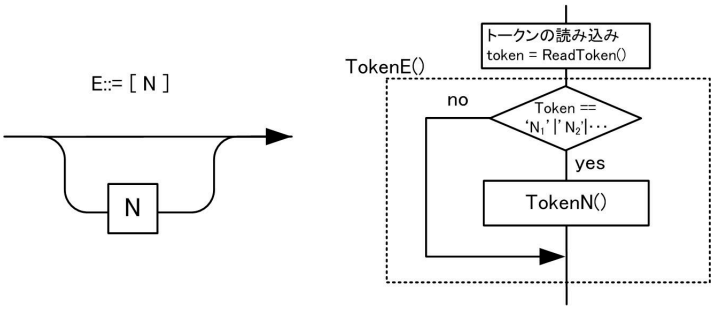

```
TokenN1();
TokenN2();
```

(4) 分岐

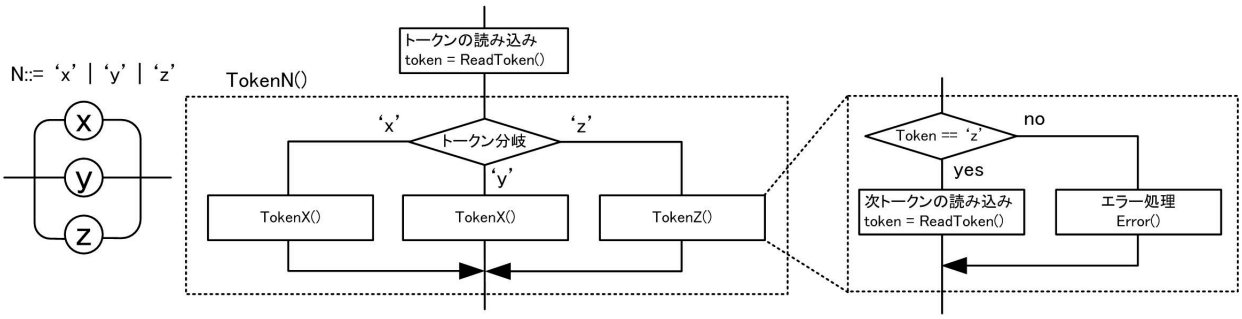



```
if(token=='x'){
    TokenX()
}
```

```
if(token=='x')
    ReadToken();
else
    Error();
```

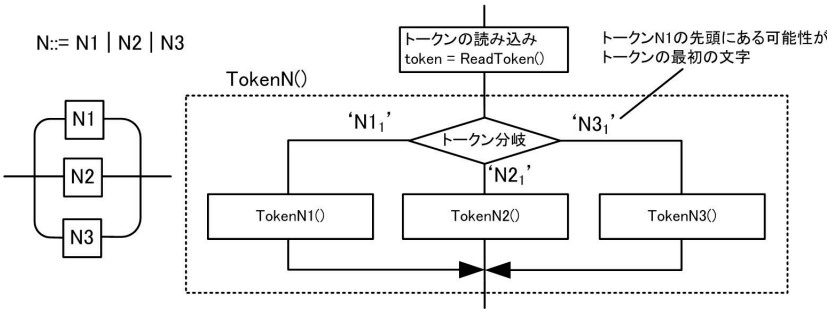



```
if(token== 'N1₁' | | 'N2₁' | | 'N3₁' ••• ){
    TokenN();
}
```



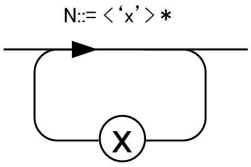

```
switch(token){
  case 'x':
    TokenX();
    break;
  case 'y':
    TokenY():
    break;
  case 'z':
    TokenZ();
    Berak;
  default:
    Error();
}
```

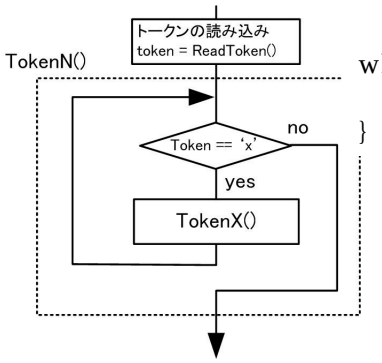
```
if(token=='x')
    ReadToken();
else
    Error();
```




```
switch(token){
  case 'N1₁': case 'N1₂': ・・・
    TokenN1();
    break;
  case 'N2₁': case 'N2₂': ・・・
    TokenN2();
    break;
  case 'N3₁': case 'N3₂': ・・・
    TokenN3();
    Berak;
  default:
    Error();
}
```

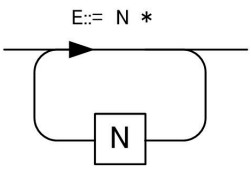
(6) 繰り返し (Zero 回以上)

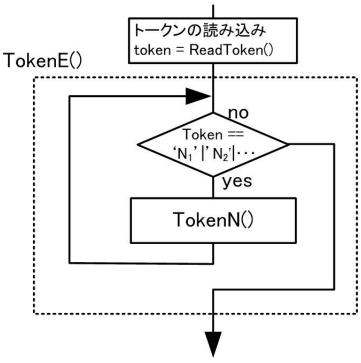





```
while(token=='x'){
    TokenX();
}
```

```
if(token=='x')
      ReadToken();
else
      Error();
```



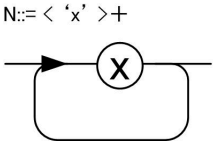


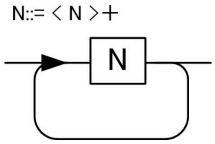

```
while(token== 'N1₁' | | 'N2₁'| | 'N3₁' ・・・ ){
    TokenN();
}
```

この手法は、構文木の根から先にかけて解析していくのでトップダウンの構文解析と呼ぶ。一方、この逆をボトムアップの構文解析と呼ぶ。

練習

1回以上の繰り返しについてパージングプログラムを作成せよ



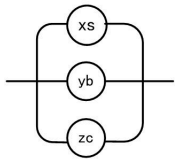


構文図のパージング関数例

構文図は複数のトークンあるいはブロックから構成されている。これを関数として表現する。

[変数]

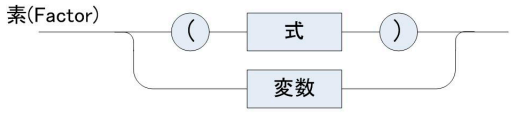
変数 ::= ‘xs’ | ‘yb’ | ‘zc’




```
Hensu(void)                                          Hensu(void)
{
    switch(token){
        case ‘x’:
            if(token==’xs’)          TokenXS()
                ReadToken();
            else
                Error();
            break;
        case ‘y’:
            if(token==’yb’)          TokenYB()
                ReadToken();
            else
                Error();
            break;
        case ‘z’:
            if(token==’zc’)          TokenZC()
                ReadToken();
            else
                Error();
            break;
        defaulut:
            Error();
    }
}
```

[素]

素 ::= 変数 | < ‘(‘ 式 ‘)’ >




```
So(void)
{
    switch(token){
        case ‘x’ : case ‘y’ : case ‘z’ :
            Hensu();                                 Hensu(void)
            break;
        case ‘(’ :
            if(token==’(’)                           TokenLeftCode(void)
                ReadToken();
            else
                Error();
            Siki();                                  Sikivoid)
            if(token==’)’)                           TokenRightCode(void)
                ReadToken();
            else
                Error();
            break;
        defaulut:
            Error();
    }
}
```

(柱)

本講義で作成している関数は、無駄(冗長性)が多いが、あくまでパージングを理解するためであり、実際は最適化されて利用される。また、文字列の比較において token==’xs’と表現しているが、実際の C 言語では strcmp() 関数を用いる必要がある。