プログラミング応用

http://bit.ly/ouyou3d

言語処理系（1）

後期 第8週

2018/11/15

本日は・・・

- 計算機による言語解析
- lex, yaccによる言語解析プログラムの実装

言語の種類

自然言語

人間が普段情報伝達に用いる言語

特徴：曖昧性を含む

人工言語

人間が何かしらの目的のために作り出した言語（例：C, シェルスクリプト）

特徴：曖昧性がない

計算機による言語解析の手順

1. **字句解析**
   - 入力言語を「字句（トークン）」並びに分割する処理
   - 字句：受理する文字列を正規表現で定義
2. **構文解析**
   - 字句解析した言語から構文木を構築
   - 構文木構築：文脈自由文法（CFG）で規則を定義
3. **意味解析**
   - 構文解析した結果に曖昧性がある場合に
     もっともらしい意味を推定
   - 人工言語でほぼ不要、自然言語で必要な処理

字句解析の例（Cの場合）

```
int x = 0;
```

字句解析

int	x	=	0	;

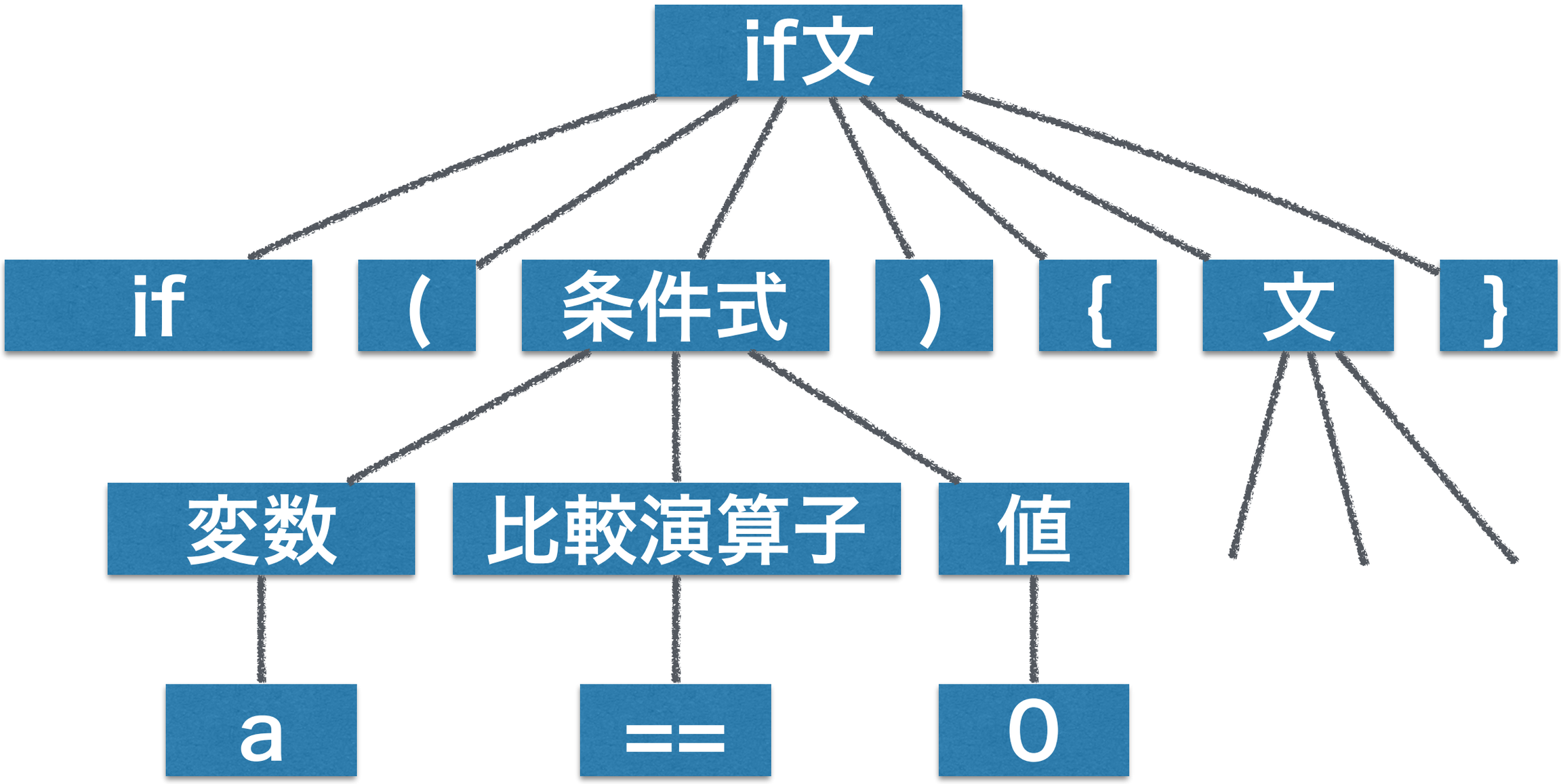
```
if (i == 0) { printf("OK\n"); }
```

字句解析

if	(	i	==	0	)	{	…

構文解析の例（Cの場合）

構文は木構造で解析される



コンパイラの役割

- （多くの場合）文脈自由文法で定義された**規則**をもとに字句解析、構文解析を行う
- 構文木が出来れば（=受理されれば）、より低水準の言語に**翻訳する**
- 構文木が出来なければ、**エラー**を出す

lexとyacc

- lex

  字句規則を正規表現で与えると、字句解析を行うプログラムを自動生成するツール

- yacc

  文脈自由文法による規則を与えると 構文解析を行うプログラムを自動生成するツール

本日は・・・

lexとyaccを用いて、
簡単な電卓を字句解析・構文解析する
プログラムを生成して動かしてみます。

字句解析プログラムの作成

calc.lex

```
%%
"+" return ADD;
"-" return SUB;
"*" return MUL;
"/" return DIV;
"(" return LP;
")" return RP;
"\n" return NL;

[0-9]+ {
  yylval = atoi(yytext);
  return NUMBER;
}
%%
```

詳しくは次回説明します

構文解析プログラムの作成

calc.yacc

```
%token NL LP RP NUMBER
%token ADD SUB MUL DIV
%%
list :
     | list expr NL { printf("%d\n", $2);}
     ;
expr : expr ADD expr { $$ = $1 + $3;}
     | expr SUB expr { $$ = $1 - $3;}
     | LP expr RP    { $$ = $2;}
     | NUMBER        { $$ = $1;}
     ;
%%
#include "lex.yy.c"
```

詳しくは次回説明します

プログラム生成から実行までの流れ

1. yaccの実行（y.tab.cが生成される）

```
$ yacc calc.yacc
```

2. lexの実行（lex.yy.cが生成される）

```
$ flex calc.lex
```

3. Cコンパイラ実行（警告が出るが今回は無視）

```
$ cc y.tab.c —ly -lfl
```

4. 実行して動作を確かめる

```
$ ./a.out
```

【練習8-1】

先に示した手順で、「加算と減算のみをする」簡単な電卓のプログラムを実行してみましょう。

[実行結果]

```
$ ./a.out
1+2             (←数式を入力すると)
3               (←計算結果が表示される)
5-1
4
1+(7-3)
5
```