プログラミング応用
http://bit.ly/kosen02

Week13@後期(week28 in 2016)
2016/12/22

お知らせ

- 補講は1/26(木; 水曜日課)の7-8限に行います
  (ホームルーム終了後、演習室に来てください)

本日の内容

- 講義(計算機による言語解析、コンパイラ)
  - 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
- 演習
  - 簡単な字句解析、構文解析システムを実装する

言語の種類

1. 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
     I want to eat somewhere near the sea.
2. 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言語、シェルスクリプト(Bash)
     楽譜なども人工言語
     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)」の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
   - 人工言語ではほぼ不要、自然言語では必要な処理

例) C言語の場合

- 字句解析

int i = 0;

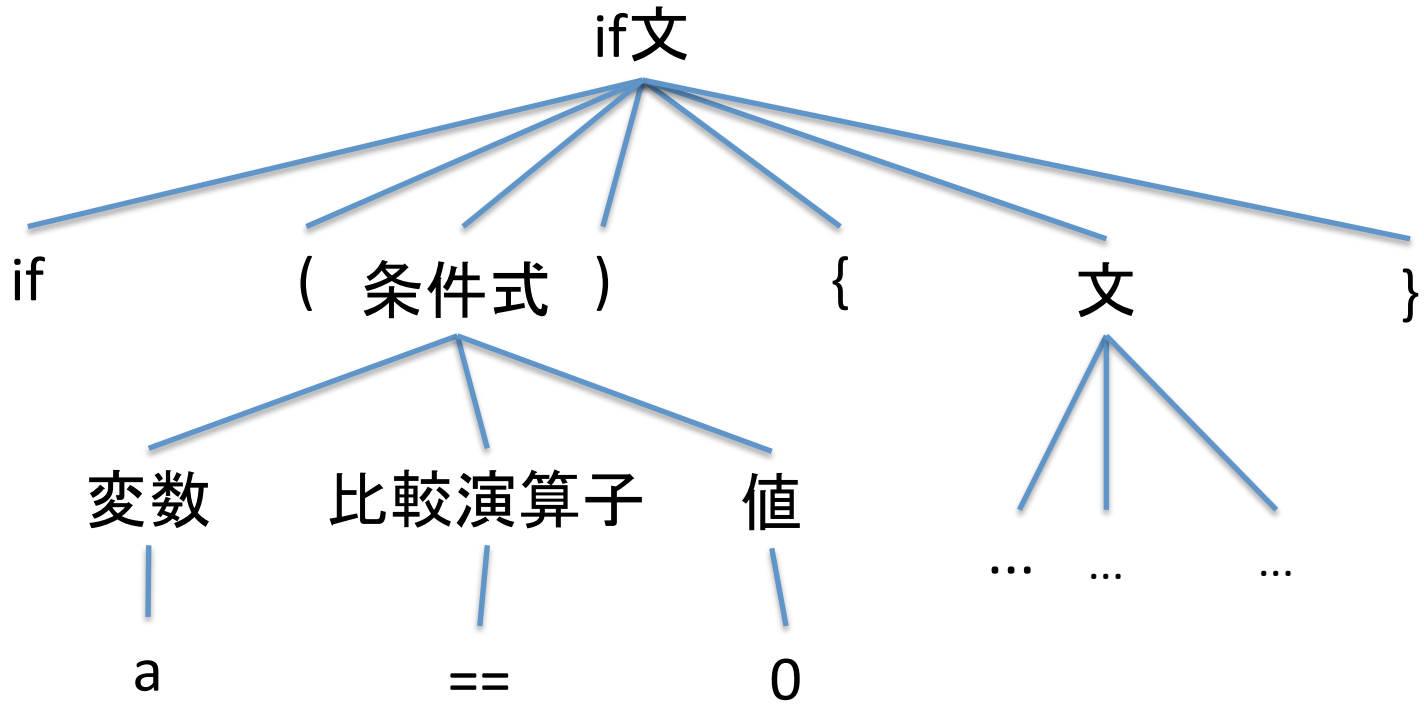
int	i	=	0	;

if (i == 0) { printf(“OK\n”); }

if	(	i	==	0	{	…

例) C言語の場合

- 構文解析



コンパイラ

- (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)より低レベルの言語に直す
- 構文木が出来なければエラーを吐く

lex/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

今回はlex/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作成

lex/yaccによる電卓作成手順

1. 完成形の確認
2. 電卓の文脈自由文法の確認
3. 字句解析プログラム(calc.lex)の作成
4. 構文解析プログラム(calc.yacc)の作成
5. コンパイル/実行して動作確認

1. 完成形の確認

- 完成形のプログラムは以下のコマンドで確認
  $ ~ishigaki/3d/calc

  出力例)
  $ ~ishigaki/3d/calc
  1 + 2 * 3
  7
  1 * (2 + 3)
  5

2. 電卓の文脈自由文法

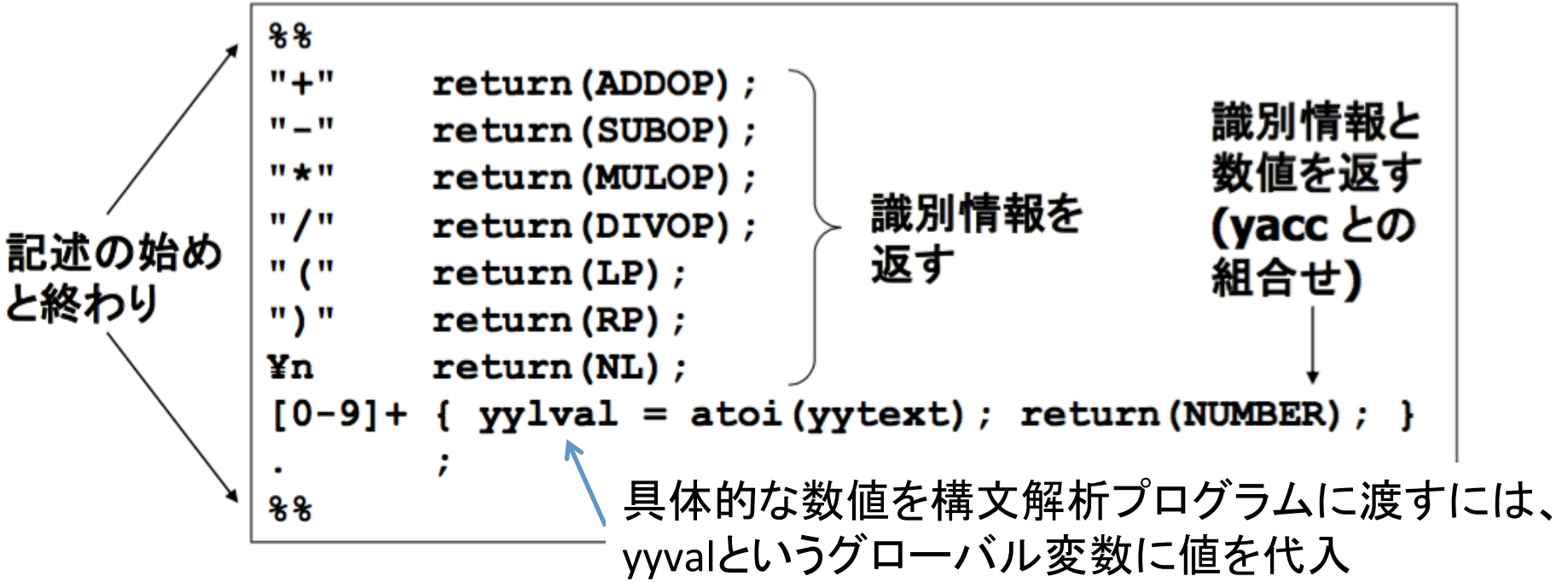
```
リスト  ::=   (空)
            |  リスト  式  改行

式      ::=   式  ＋  式
            |  式  －  式
            |  式  ＊  式
            |  式  ／  式
            |  (   式   )
            |  数値
```

加減乗除とカッコの計算を受理する

3. 字句解析プログラムの作成(2/2)

- 実装例(calc.lex)
  - 『パターン』と『アクション』を組み合わせる



4. 構文解析プログラムの書き方

- 実装例(calc.yacc)

```
%token  NL
%token  NUMBER
%token  LP RP
%token  ADDOP SUBOP MULOP DIVOP
%%


list    : /* 空でも良い */
        | list expr NL    { printf("%d¥n", $2); }
        ;
expr    : expr ADDOP expr { $$ = $1 + $3; }
        | expr SUBOP expr { $$ = $1 - $3; }
        | expr MULOP expr { $$ = $1 * $3; }
        | expr DIVOP expr { $$ = $1 / $3; }
        | LP expr RP      { $$ = $2; }
        | NUMBER          { $$ = $1; }
        ;
%%
#include "lex.yy.c"
```

4. 構文解析プログラムの書き方

- 実装例(calc.yacc)

```
%token  NL
%token  NUMBER
%token  LP RP
%token  ADDOP SUBOP MULOP DIVOP
%%
```

1. lexで定義したトークンを列挙

```
list    : /* 空でも良い */
        | list expr NL      { printf("%d¥n", $2); }
        ;
expr    : expr ADDOP expr   { $$ = $1 + $3; }
        | expr SUBOP expr   { $$ = $1 - $3; }
        | expr MULOP expr   { $$ = $1 * $3; }
        | expr DIVOP expr   { $$ = $1 / $3; }
        | LP expr RP        { $$ = $2; }
        | NUMBER            { $$ = $1; }
        ;
%%
#include "lex.yy.c"
```

2. CFGの規則をBNFで記述

3. どのような計算を行うか記述

lexで作った字句定義を読み込み

5. コンパイル/実行

- 以下のコマンドを順に入力

  1. yaccの実行(y.tab.c)
  $ yacc calc.yacc

  2. lexの実行(lex.yy.cが生成される)
  $ lex 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

演習0

- 講義資料の手順通りに電卓の字句解析、構文解析を行うプログラムを記述しなさい。
  - 以下のファイルを提出
    - calc.lex
    - calc.yacc

演習1

- 前置記法と呼ばれる数式の書き方がある。
- 前置記法では以下のように演算子を数字よりも前に置く。

  + 2 3　　// これは2＋3と同義
  * (+ 2 3) (- 3 2) // これは(2+3) * (3-2)と同義

  前置記法の電卓を正しく構文解析する
  calc_zenchi.yaccを作成しなさい。

次回以降

- 次回1/19は自然言語を計算機で扱う方法を学びます。

- 1/12は3D研修旅行のため休講
- 1/26(木)は水曜日課ですが7-8限に授業をします。

試験

- 日時：未定(試験期間中の木曜日)
- 内容：
  - 正規表現(grepコマンド)、状態遷移図(+C言語実装)
  - 文脈自由文法、BNF
  - lex/yaccによる字句解析・構文解析
  - 自然言語処理

- マシン内の資料、プログラムのみ参照可