平成 26 年度後期期末試験
3 学年 9 班　構文解析とコンパイラ (岩井)
2015 年 2 月 23 日 (月)

1　コンパイラとは

次の文章の書く四角の中に入る語句を、下のリストから選んで、答えとしてその番号を書きなさい。

コンパイラとは [1] を [2] に変換するプログラムである。[1] は、高級言語で表現され、これには [3] や、FORTRAN などがある。高級言語は、人間には理解しやすいが、[4] はこれをそのまま実行することはできない。

コンパイラは [1] を読み込むと、まず、字句解析を行い、[5] に分割する。そしてその後構文解析を行い、[2] を出力する。このとき、実行速度を上げるために [6] を行うこともある。

1. C 言語　2. アセンブラ　3. lex　4. 最適化　5. 目的コード
6. 原始プログラム　7. オートマトン　8. CPU　9. トークン　10. 状態遷移図

2　正規表現とオートマトン

次の正規表現からオートマトンを作成せよ。非決定性オートマトン (NFA)、決定性有限オートマトンのどちらでもよい。

$$ba(a|b)*a$$

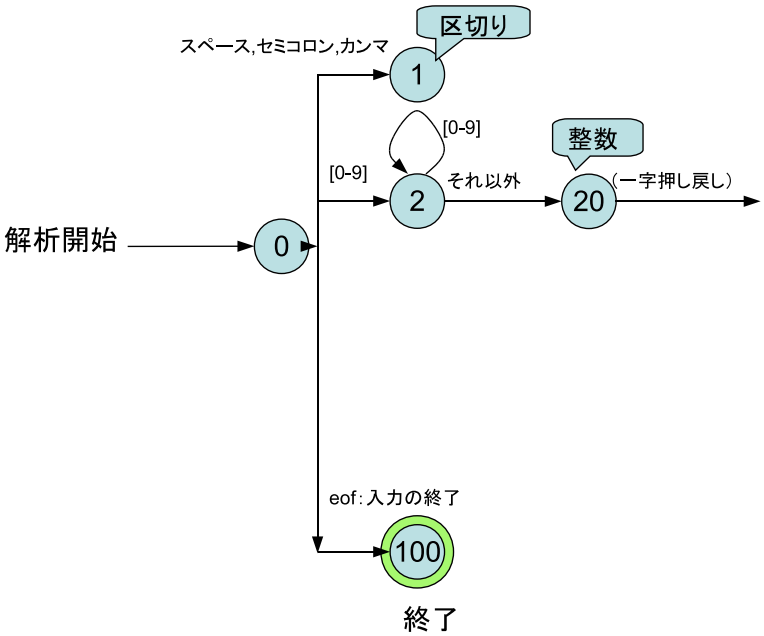
3　C 言語のトークンと lex

C 言語（に似た言語）を字句解析するためのプログラムを考える。この言語は次の表に示す４つのトークンをもつ。

トークンの種類	正規表現	備考
整数	[0-9](0-9)*	10 進の整数
キーワード (for)	for	for
キーワード (int)	int	int
識別子	①	英文字で始まり、英文字と数字で表現される任意の長さの文字列
区切り	スペース, セミコロン , カンマ	区切りをあらわす
括弧	(,)	括弧
OPERATOR（演算子）	+,=	四則演算

1. 表の穴①をうめよ
2. この字句解析プログラムは次のオートマトンで示される。ただしこの図は未完成である。完成したオートマトンを示せ。このとき、解析した一文字の押し戻しが必要な場合は、図に示してあるとおり、エッジにそえて（一文字押し戻し）表記せよ。また、非決定性オートマトンであってもよい。

   ヒント プログラムの字句解析を行うためには、トークンの解析を繰り返す必要があることに注意



4　文法 1

四則演算を表す文法 G1 を次に示す。これについて次の問いに答えよ。

$$
\begin{aligned}
G1 = (\quad & N \quad && ,T,P,S) \\
N = \{\quad & E \quad && \} \\
T = \{\quad & + \quad && ,-,*,/,\mathbf{num}\} \\
P = \{\quad & E \quad && \to E+E \\
& E \quad && \to E-E \\
& E \quad && \to E*E \\
& E \quad && \to E/E \\
& E \quad && \to (E) \\
& E \quad && \to \mathbf{num}\} \\
S = \{\quad & E \quad && \}
\end{aligned}
$$

5　文法 2

次に示す文法について、問いに答えよ。(終端記号、非終端記号の区切りにカンマ（,）を用いているが、T(終端記号）には ”, ”があることに注意）

$$
\begin{aligned}
G1 = (\quad & N \quad && , \ T \ , \ P \ , \ S \ ) \\
N = \{\quad & S \quad && , \ L\} \\
T = \{\quad & ″,″ \quad && , \ (\ , \ ) \ , \ a \ \} \\
P = \{\quad & S \quad && \to \ (\ L \ ) \ \mid \ a \\
& L \quad && \to \ L \ , \ S \ \mid \ S \\
S = \{\quad & S \quad && \}
\end{aligned}
$$

この文法に基づいて、次の文の最左導出を示せ。あわせて解析木を示せ

1. (a,a)
2. (a, ((a, a),(a, a)))

1. 上の文法を構文図で描け (終端記号は○で囲み、非終端記号は□で囲んで示すこと）
2. 開始状態を E として、3*(4+5) のを最右導出せよ（途中の手順を省略せずに書く)。あわせて、それに対応する解析木をかけ。

6 lex/yacc プログラム

下に示す簡単なプログラムに対して、構文解析を行い、スタックマシン用のアセンブリ言語を出力する lex/yacc を後に示す。これについて次の問いに答えよ。

プログラム例

```
x = y;
if(x == 0){
  z = 0;
}
```

1. この lex/yacc プログラムで定義している文法の終端記号を全て示せ。
2. yacc プログラム 25-28 行目で定義している文法について、構文図で示せ。
3. yacc プログラム 30 行目で定義している文法について、構文図で示せ。
4. 21-23 行目を次のように書き換えると、この文法が生成する解析木が変わる。どのように変化するか、簡単に説明せよ

```
21  statementlist :
22          statement statementlist |
23  ;
```

5. この yacc プログラムでは、if 文などで、{} を用いた複数の文（複合文) を扱うことができない。**{ }** を導入して、複合文を扱えるようにプログラムを加筆または改変せよ。解答は加筆・修正部分の記述だけでよい。
6. 次のような while 文をコンパイルできるように、yacc プログラムを加筆せよ。加筆部分の記述だけでよい。その場合、yacc プログラムの何行目に付け足すかも記述せよ
   while の例

```
while (x == 10)
    x = x + 1;
```

lex プログラム (minic.l)

```
 1  %{
 2  #include "y.tab.h"
 3  extern int yylval;
 4  int line = 1;
 5  %}
 6
 7  %%
 8  "+"   {return ADD;}
 9  "-"   {return SUB;}
10  "*"   {return MUL;}
11  "/"   {return DIV;}
12  "="   {return ASSIGN;}
13  "=="  {return EQ;}
14  "{"   {return LC;}
15  "}"   {return RC;}
16  "("   {return LP;}
17  ")"   {return RP;}
18  ";"   {return SC;}
19  ","   {return CM;}
20  "if"    {return IF;}
21  "while"   {return WHILE;}
22
23  [0-9]+ {  yylval = atoi(yytext); return NUMBER;}
24  [a-zA-Z] {  yylval = yytext[0];  return ID;}
25
26  "\n" {printf("line %d:\n",line++);}
27  [ \t] ;
28
29  %%
30
31  main(){
32      return yyparse();
33  }
34
35  yyerror(char *s){
36    printf("%s\n",s);
37  }
```

yacc プログラム (minic.y)

```
 1  %{
 2          int label=0;
 3  %}
 4
 5  %token ID
 6  %token NUMBER
 7  %token IF WHILE
 8  %token LP RP
 9  %token LC RC
10  %token SC CM
11  %token ASSIGN
12  %token EQ
13  %left  MUL DIV
14  %left  ADD SUB
15
16  %%
17
18  program        : statementlist {puts("program end");}
19  ;
20
21  statementlist :
22         statementlist statement |
23  ;
24
25  statement : ID ASSIGN expression SC { printf("\tPOPV\t%c\n",$1); }
26             | ifstatement {printf("L%d\n",label++);}
27  ;
28
29
30  ifstatement : IF LP condexpression RP {
printf("\tJNZ\tL%d\n",label);} statement
31  ;
32  condexpression : expression EQ expression { printf("\tEQ?\n");}
33  ;
34
35  expression : expression ADD expression { printf("\tADD\n"); }
36  | expression SUB expression { printf("\tSUB\n"); }
37  | expression MUL expression { printf("\tMUL\n"); }
38  | expression DIV expression { printf("\tDIV\n"); }
39  | LP expression RP { $$ = $2; }
40  | ID { printf("\tPUSH\t%c\n",$1); }
41  | NUMBER { printf("\tPUSH\t%d\n",$1); }
42  ;
43  %%
```