構文解析とコンパイラ 期末試験 2016 年度

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、それぞれひとつ、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。

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

コンパイラは原始プログラムを読み込むと、まず、字句解析を行い、[4] に分割する。そしてその後構文解析を行い、[1] を出力する。このとき、実行速度を上げるために [5] を行うこともある。

a. オートマトン　b. トークン　c. lex　d. 最適化　e. 目的コード
f. 原始プログラム　g. C 言語　h. CPU　i. 状態遷移図

2　正規表現とオートマトン

1. 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。終了状態は二重丸で書くこと。

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

2. 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 (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

表の穴①をうめよ。（英字には_（下線）を含む）

3　文法 1

四則演算を表す文法 G1 を次に示す。これについて各問いに答えよ。（定義中の **num** は数字（[0-9][0-9]*）を示す）

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

1. 上の文法に従って次の式を最左導出せよ。解析木も書け。

$$4 + 5 * 6$$

4　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）

$$
\begin{aligned}
G1 = (\quad & N \quad , \ T \ , \ P \ , \ S \ ) \\
N = \{\quad & S \quad , \ L\} \\
T = \{\quad & ''\,,'' \ , \ (\ , \ ) \ , \ a \ \} \\
P = \{\quad & S \quad \ \to \ (\ L \ ) \ \mid \ a \\
& L \quad \ \to \ L \ , \ S \ \mid \ S \\
S = \{\quad & S \quad \}
\end{aligned}
$$

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)))

5 lex/yacc プログラム

C 言語風の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 yacc プログラムを作成した。これについて次の問いに答えよ。

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 DO
 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  stmt : assignstmt | ifstmt
19  ;
20
21
22  ifstmt : IF LP condE RP {printf("JNZ L%d\n",label);} stmt {printf("L%d\n",label++);}
23  ;
24
25  condE : E EQ E { printf("EQ?\n");}
26  ;
27
28  assignstmt :ID ASSIGN E SC { printf("POP%c\n",$1); }
29
30  E : E ADD E { printf("ADD\n"); }
31  | E SUB E { printf("SUB\n"); }
32  | E MUL E { printf("MUL\n"); }
33  | E DIV E { printf("DIV\n"); }
34  | LP E RP { $$ = $2; }
35  | ID { printf("PUSH %c\n",$1); }
36  | NUMBER { printf("PUSH %d\n",$1); }
37  ;
38  %%
39
```

1. yacc1.y で定義されている非終端記号を全て示せ
2. このプログラムの開始状態となる非終端記号は何か？
3. C 言語では、次に示すような do-while 文が定義されている。do と while の間には実行したい文が入り、while の後のカッコ内にはループの条件式が入るものである。最後にセミコロンが付く。

do-while 文 （例１）

```
do
   x=x+1;
while ( x == 100);
```

(a) この文の構文図を示せ。なお、終端記号は○で囲み、非終端記号は四角で囲み表記せよ。非終端記号や、終端記号は、yacc プログラム内で定義してあるものを使用するといい。

(b) この文を解析できるように yacc プログラムを改良せよ。（追加プログラムを示して、それが何行目に入るか示す）**アセンブリは作成しなくてよい**

4. このプログラムでは、一文（一行）から構成されるプログラムしか入力として受け付けない。改造して複数の文を入力として受け付けることができるようにせよ。

5. 次に示す（例２）のように、if 文内で複数の文を扱うために、**{ }** を導入して、複合文を扱えるようにプログラムを改変せよ。

（例２）

```
if(x == 0){
  x = x/3;
  v = 100;
  }
```

6. `x=v+w-z;` が入力されたときに出力されるアセンブリプログラムを書け

7. 次のプログラム 1 が入力されたときに出力されるアセンブリプログラムを書け

8. このプログラムに対して、解析木も書け

プログラム 1

```
if(x == 0)
  z = 5;
v = y;
```

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  "do"  {return DO;}
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  }
```

授業では、+(プラス記号）など記号を扱うときは、\ を用いて \+ としていたが、"" で該当する記号を囲んでも、同じことになる。