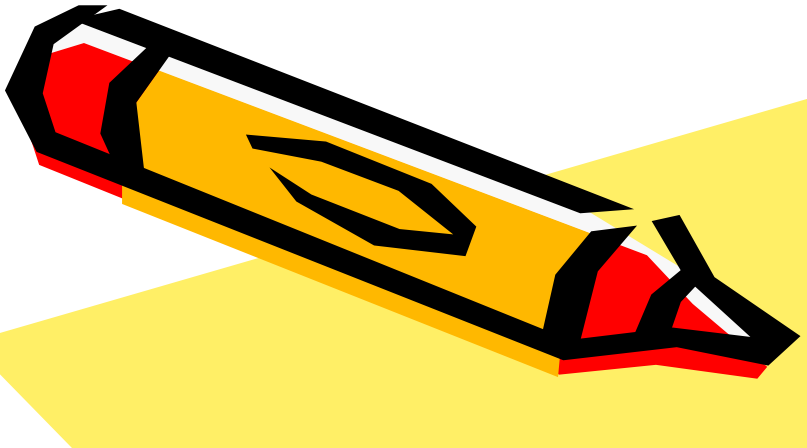
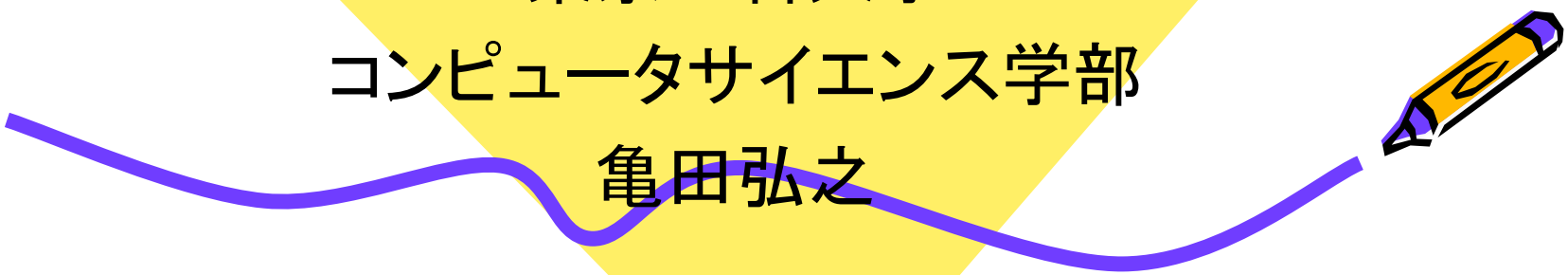


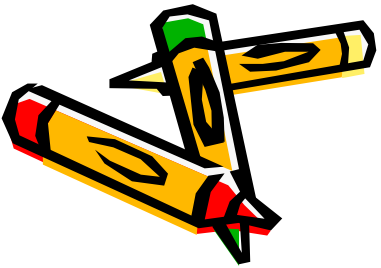
言語プロセッサ2014
-No.3-

東京工科大学
コンピュータサイエンス学部
亀田弘之



これからの内容

1. 字句解析プログラムの作成方法
   - 手書きの方法
   - Flexを利用する方法
2. 構文解析
   - 解析手法の種類
   - 左再帰とその除去
   - 括りだし
   - FirstとFollow

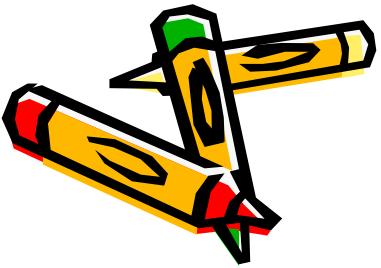


参考資料(発展)

Intermediate Representations in Imperative Compilers: A Survey,

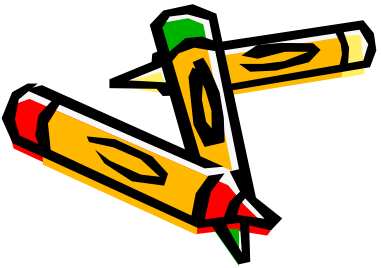
James Stanier and Des Watson,

ACM Computing Surveys, Vol.45, No.3, Article 26(27 pages), 2013.



正規表現の練習

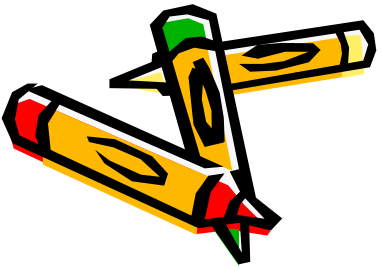
- PCを立ち上げてください。
- HTMLのソースコードから情報を抽出してみよう！



演習

- 準備
  - 正規表現の使えるエディタを準備
  - 大学のホームページのソースコードをコピーする。

1. 正規表現の利用練習をする。

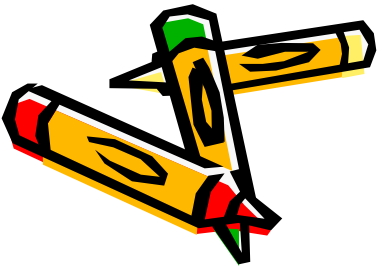


練習課題

1. HTMLのタグをすべて除去する。
   - Remove all tags in the HTML page.
2. メールアドレスをすべて抜き取る。
   - Extract all e-mail addresses in it.
3. 電話番号をすべて抜き取る。
   - Extract all telephone numbers in it.
4. 数字だけをすべて抜き取る。
   - Extract all figures in it.

練習課題のヒント

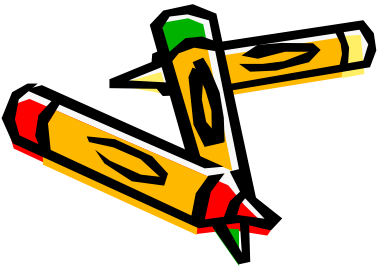
- サイト
  http://hodade.adam.ne.jp/seiki/page.php?chapter_1



試してみよう (Let's try it!)

<(" [^"]*" | ' [^']*' | [^'">])*>

参考ページ
http://hodade.adam.ne.jp/seiki/page.php?chapter_1



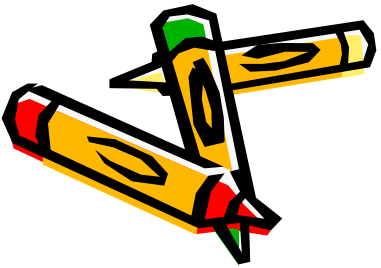
字句解析の実際

- 「符号なし数」の字句解析

  num → digit+ (. digit+)? (E(+|-)?digit+)?

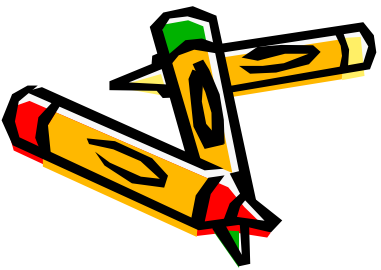
  => n → d+(. d+)? (E(+|-)?d+)?

  => num → d+(.d+|ε)((E((+|-)|ε)) d+|ε) |ε)



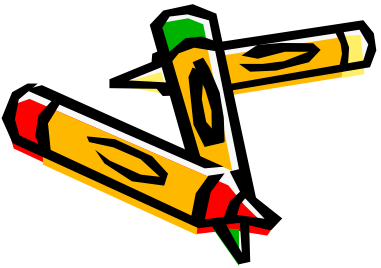
Flexのソースコード例


```
D [0-9]

%%
[¥t ]  { printf(″Tab or Space¥n″); }
{D}+(¥.{D}+)?(E(¥+|¥-)?{D}+)?    {printf(″%s¥n″, yytext); }
.  { printf(″NG %s¥t″, yytext); }
%%
```



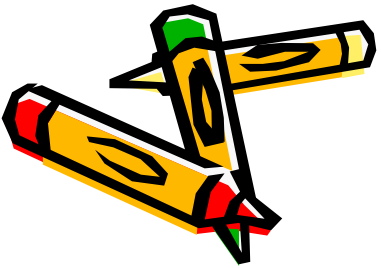
識別子のシンタックス

- letter → a|b|c|...|z|A|B|C|...|Z
- digit → 0|1|2|...|9
- id → letter (letter|digit)*

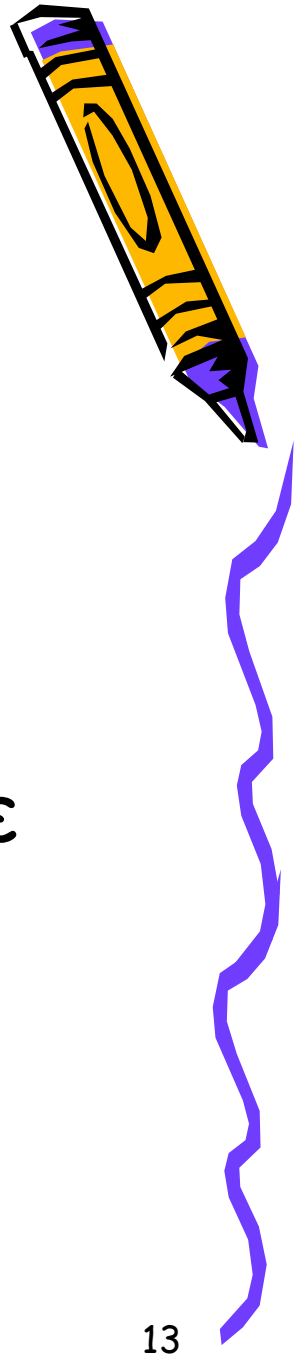


Flexのコード例

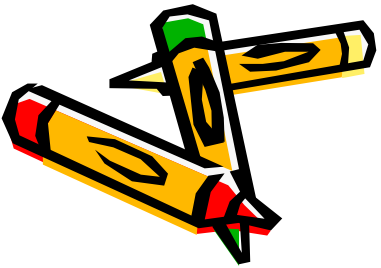
```
LETTER      [a-zA-Z]
DIGIT       [0-9]

%%
[¥t ]   { printf("Tab or Space¥n"); }
{LETTER}({LETTER}|{DIGIT})* {printf("id=%s¥n",
    yytext); }
.   { printf("NG %s¥t", yytext); }
%%
```



符号なし数のシンタックス

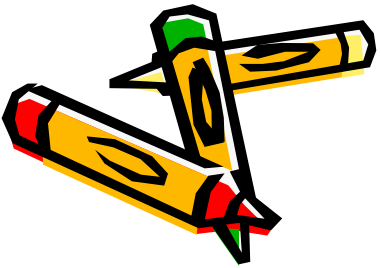


- digit → 0|1|2|3|4|5|6|7|8|9
- digits → digit digit*
- optional_fraction → . digits |ε
- optional_exponent → (E(+|-|ε)digits)|ε
- num → digits optional_fraction
  optional_exponent

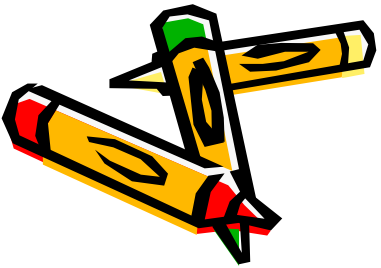


トークン認識プログラム作成

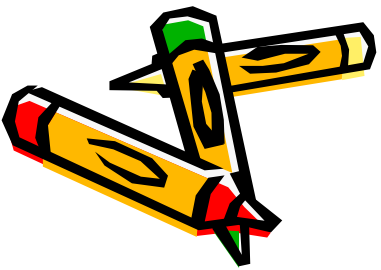
- （教科書を参考に作成してみよう！）



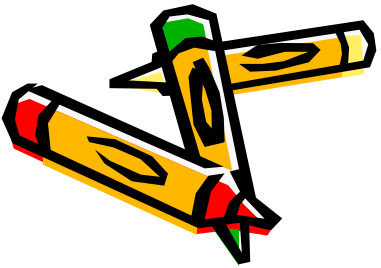
プログラムの文法(例)



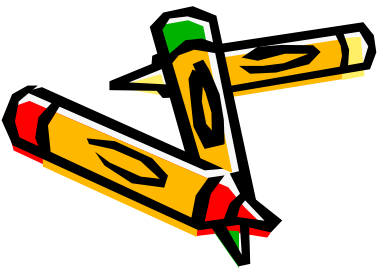
- stmt → if expr then stmt
  | if expr then stmt else stmt
  | ε
- expr → term relop term
  | term
- term → id
  | num



- if → if
- then → then
- else → else
- relop → <|<=|=|<>|>|>=
- id → letter(letter|digit)*
- num → digit$^+$(.digit$^+$)?(E(+|–)?



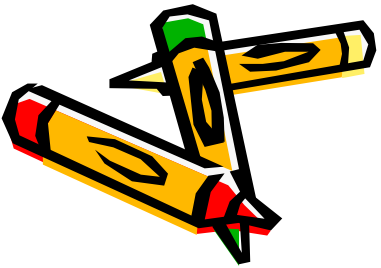
- delim $\rightarrow$ blank|tab|newline
- ws $\rightarrow$ delim^{+}

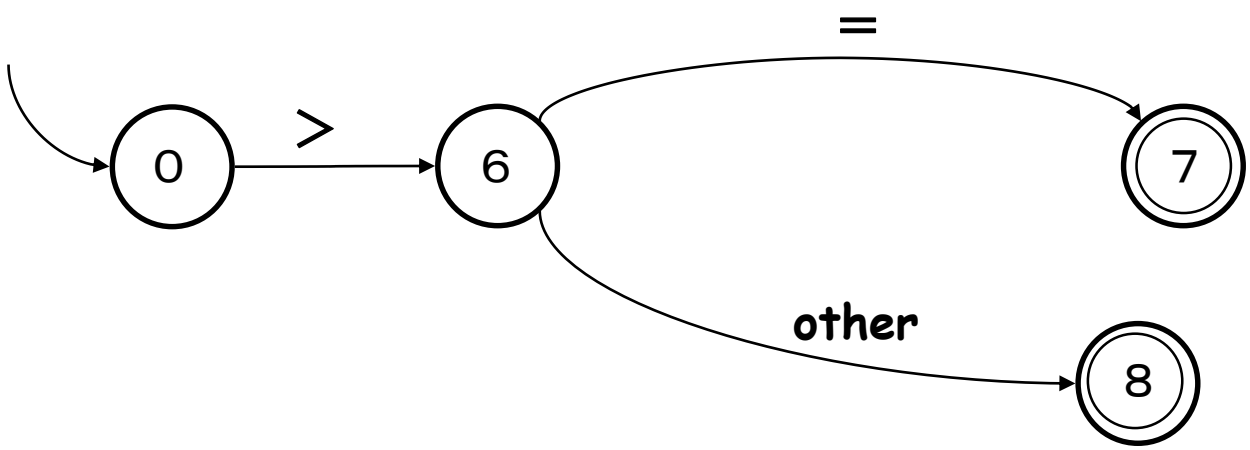


トークン-正規表現パタン

正規表現	トークン	属性値
ws	（なし）	（なし）
if	if	（なし）
then	then	（なし）
else	else	（なし）
id	id	記号表のエントリへのポインタ
num	num	記号表のエントリへのポインタ
<	relop	LT
<=	relop	LE
=	relop	EQ
<>	relop	NE
>	relop	GT
>=	relop	GE

オートマトンの作成




0
>
6
=
7
other
8

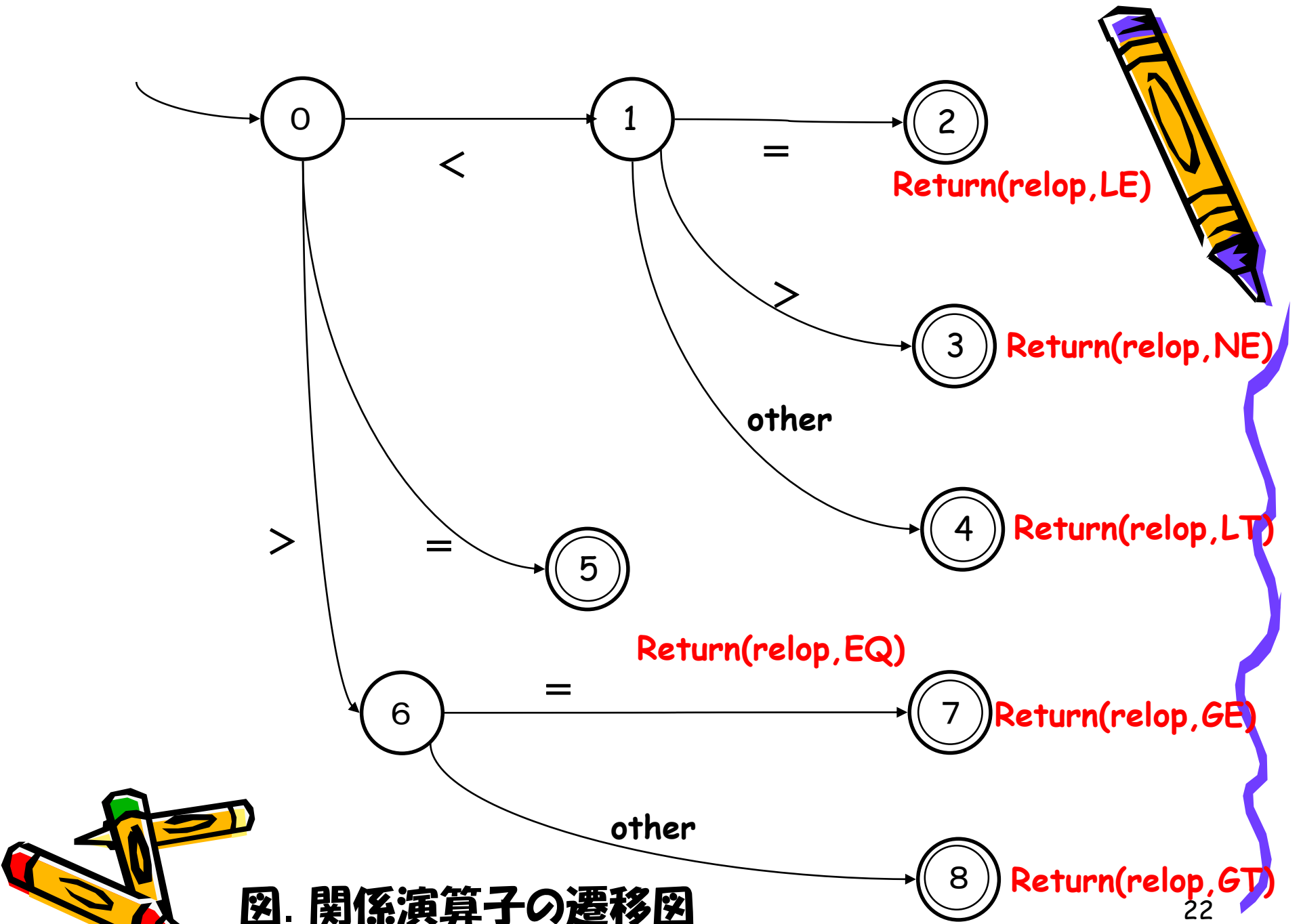

0
1
2
<
=
Return(relop,LE)
>
3
Return(relop,NE)
other
4
Return(relop,LT)
>
=
5
Return(relop,EQ)
=
6
7
Return(relop,GE)
other
8
Return(relop,GT)

図. 関係演算子の遷移図

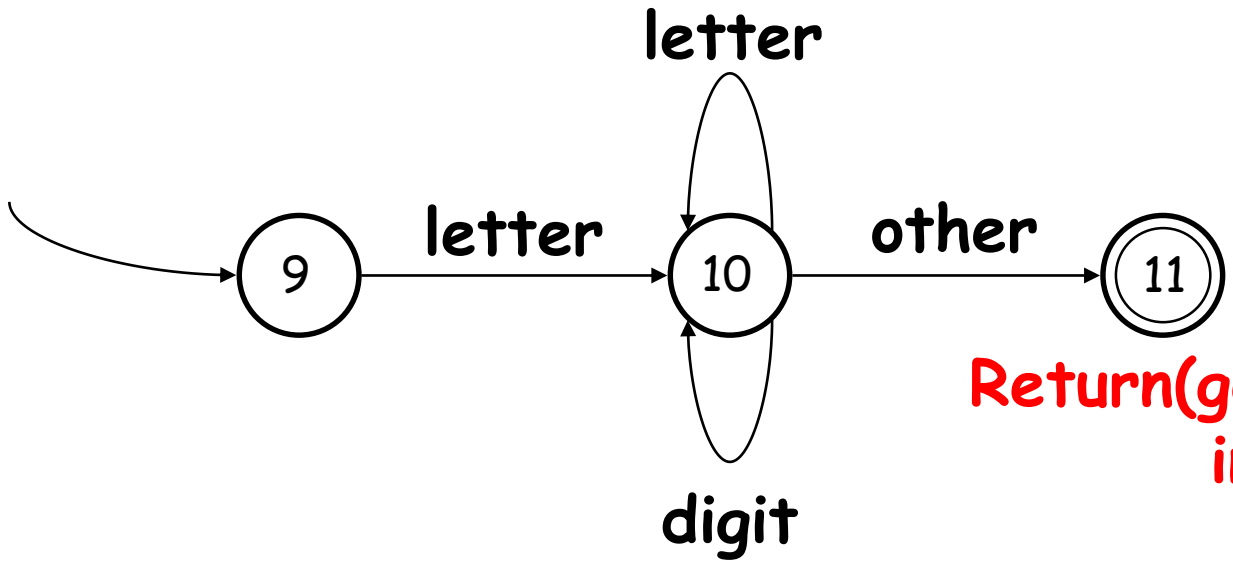


図. 識別子とキーワードに対する遷移図

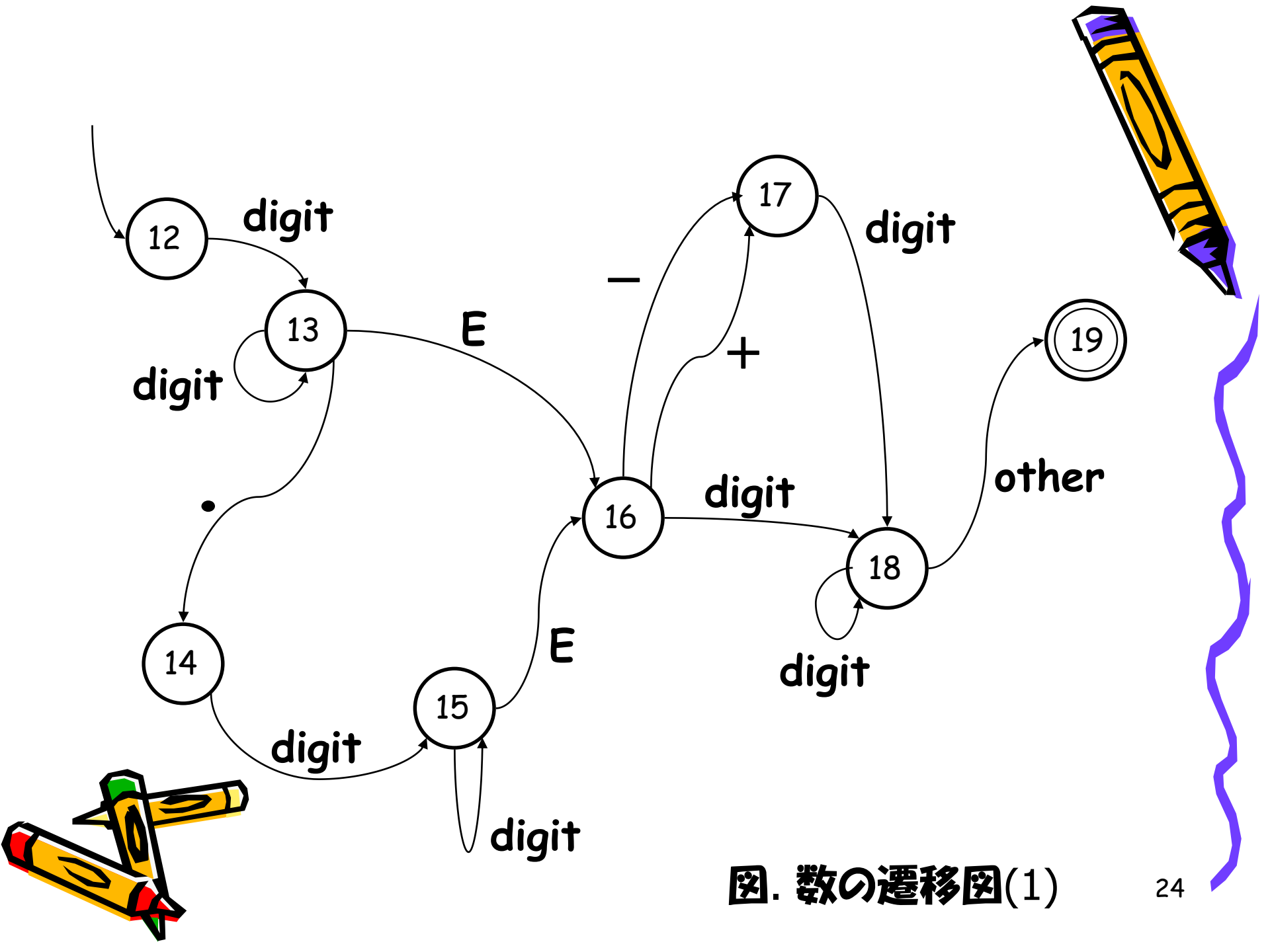

12
digit
13
digit
E
17
digit
−
+
19
16
digit
18
other
14
•
15
E
digit
digit
digit

図. 数の遷移図(1)

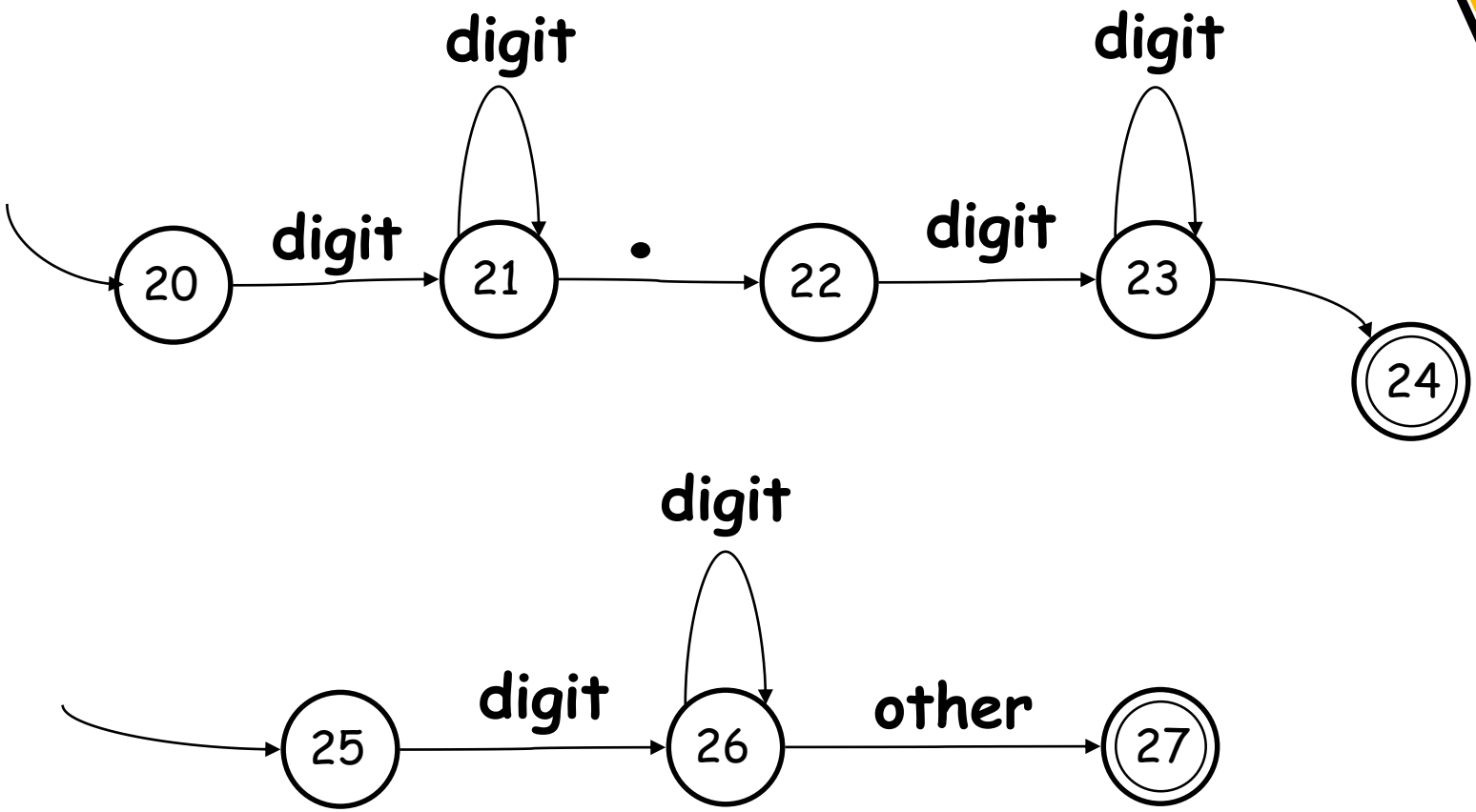
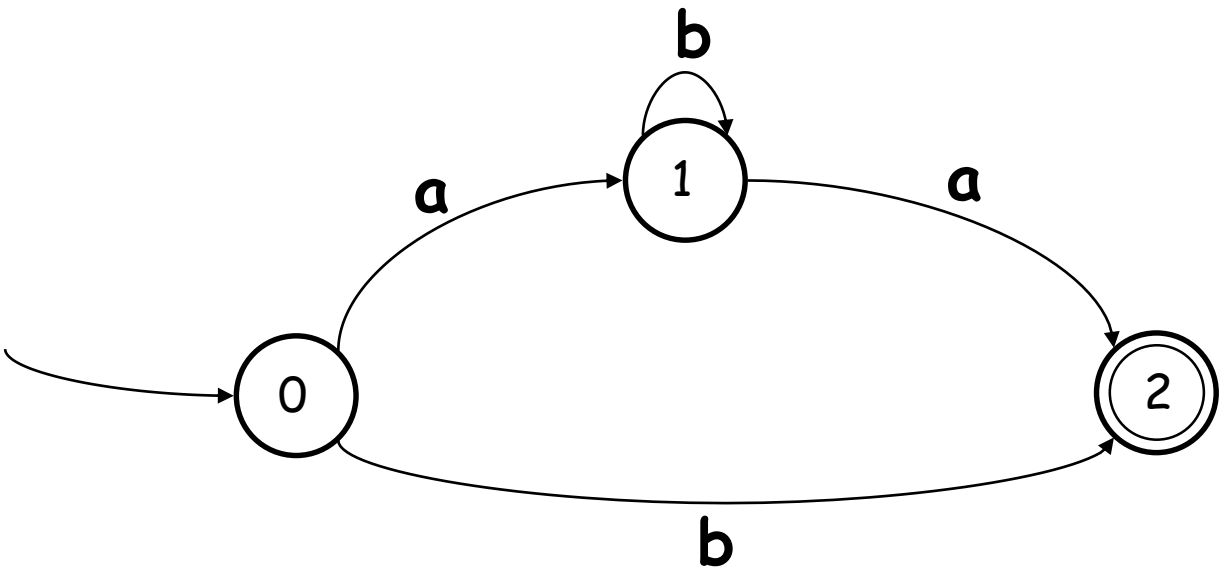
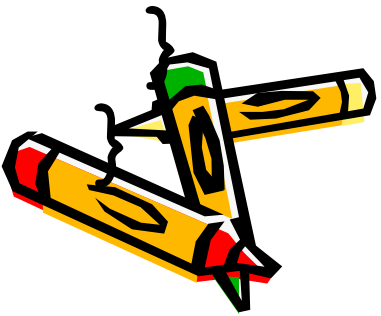


図. 数の遷移図(2)

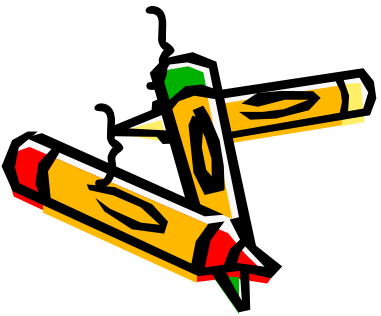
単純な遷移図



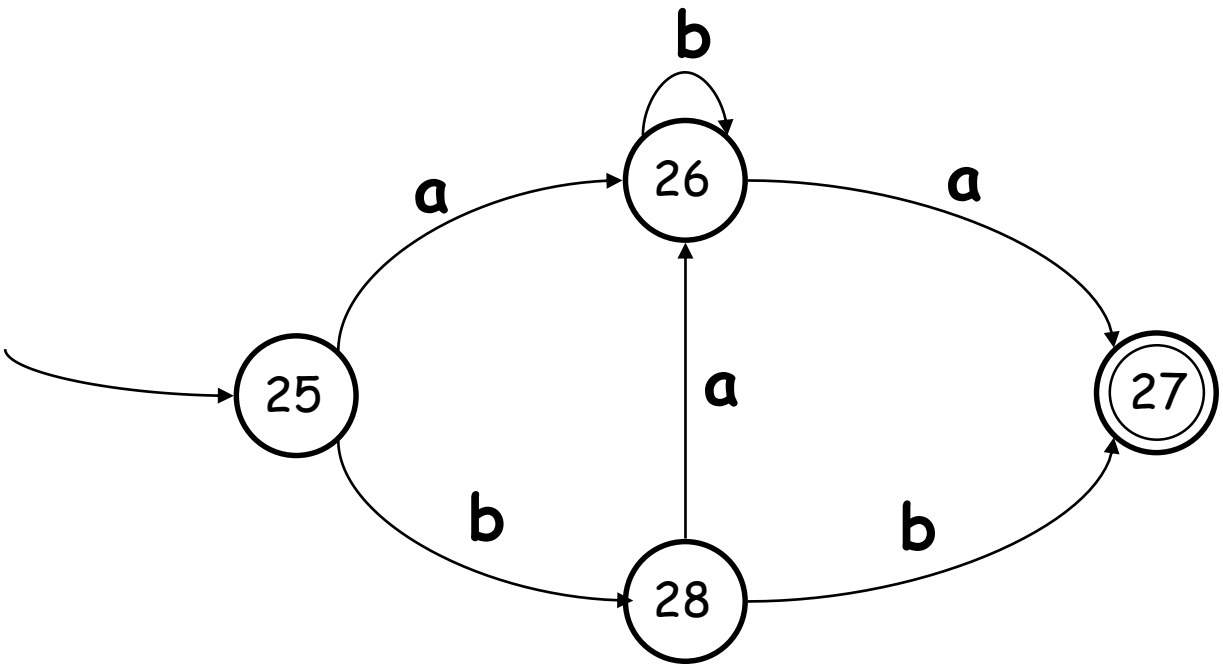
```
state = 0;
while(TRUE) {
 switch(state) {
   case 0: c = nextChar( ); /* 先読み */
     switch(c){
       case ‘a’: state = 1; break;
       case ‘b’: state = 2; break; }
   case 1: c = nextChar( );
     switch(c){
       case ‘a’: state = 2; break;
       case ‘b’: state = 1; break; }
```



```
state = 0;
while(TRUE) {
 switch(state) {
   case 0: c = nextChar( ); /* 先読み */
     switch(c){
       case 'a': state = 1; break;
       case 'b': state = 2; break; }
   case 1: c = nextChar( );
     switch(c){
       case 'a': state = 2; break;
       case 'b': state = 1; break; }
```

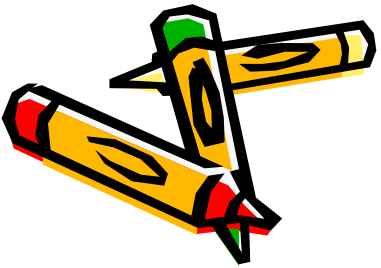


簡単な遷移図

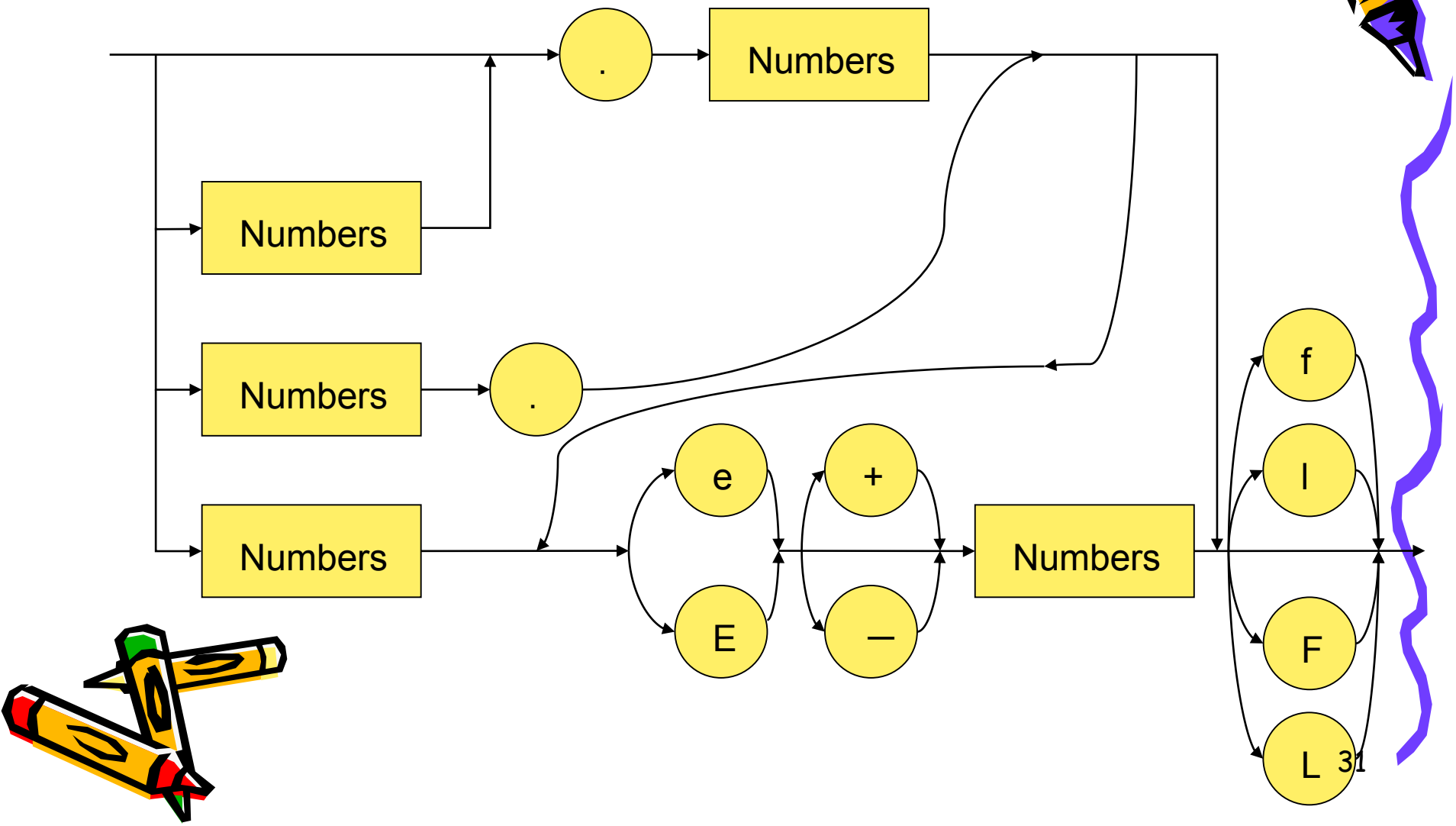


練習問題

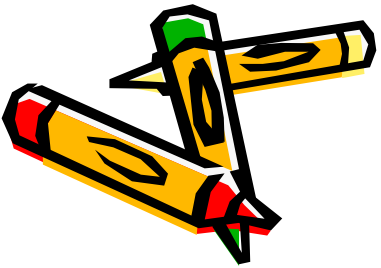
1. 直前のページのオートマトンをシミュレートするプログラムを作成せよ。
2. 符号なし数を処理する字句解析プログラムを作成せよ。
3. 次ページに示した（C言語の）浮動小数点定数を処理する字句解析プログラムを作成せよ。



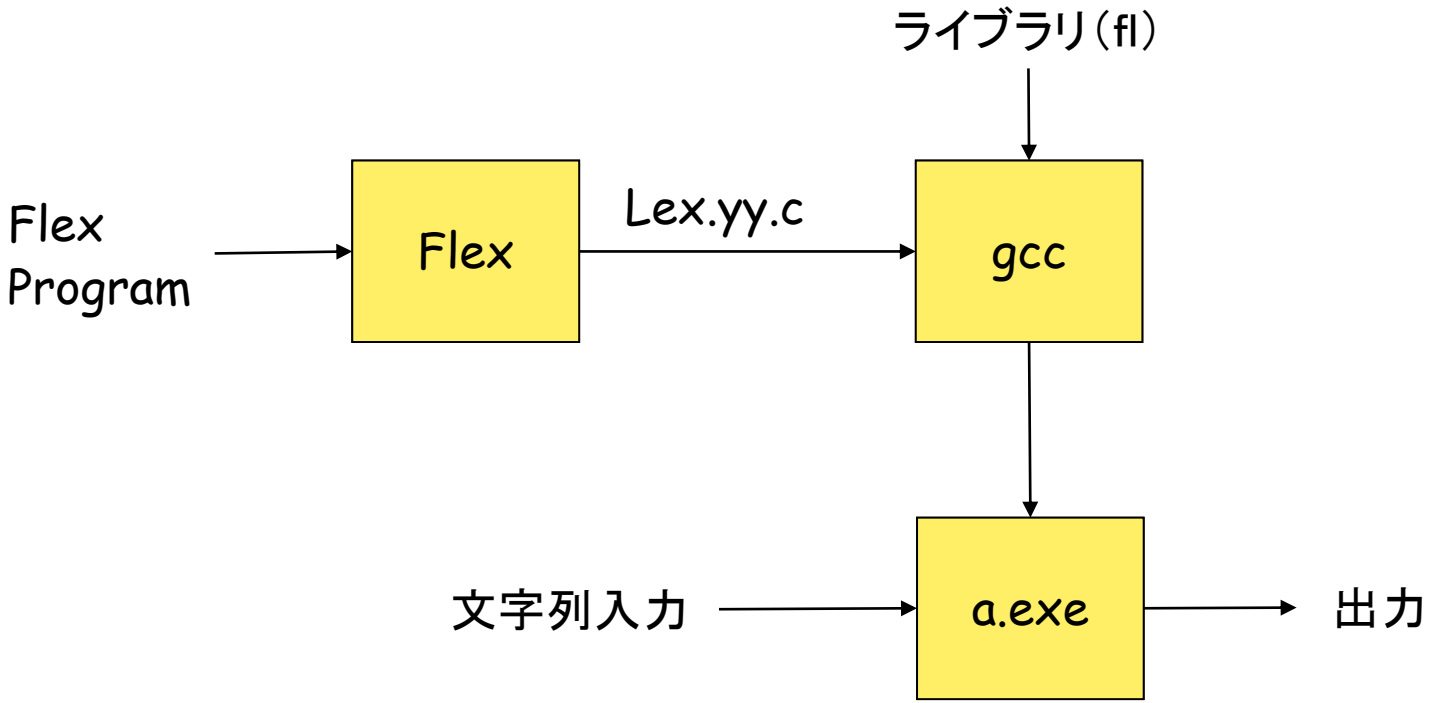
参考：（C言語の）浮動小数点定数



今度はflexを使ってみよう！

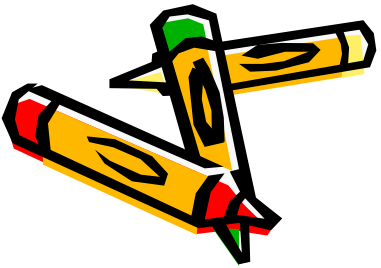


手順

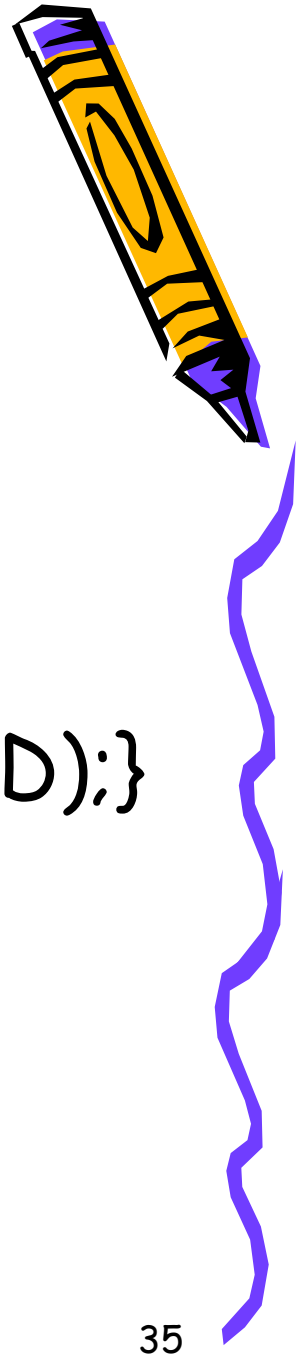


Flexプログラムの記述(1)

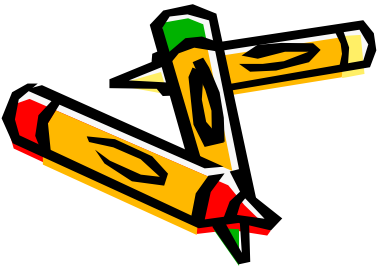
```
delim     [ ¥t¥n]
ws        {delim}+
letter    [a-zA-Z]
digit     [0-9]
id        {letter}({letter}|{digit})*
number    {digit}+(¥.{digit}+)?(E[+¥-]?{digit}+)?
%%
```



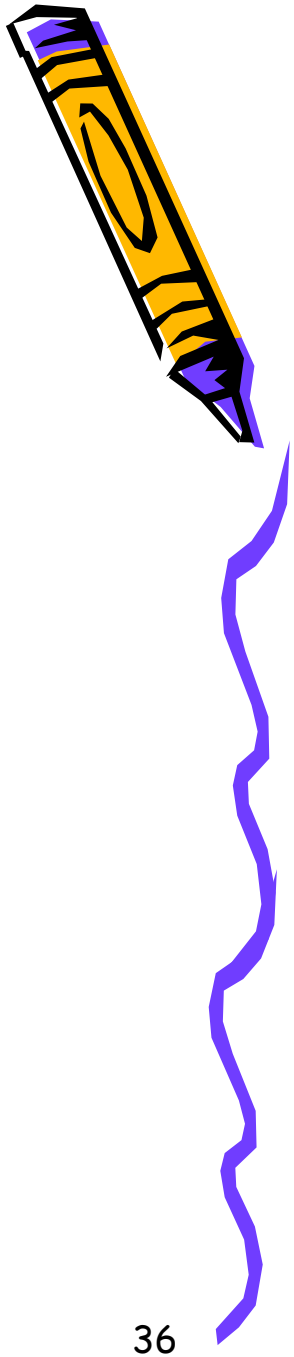
Flexプログラムの記述(2)



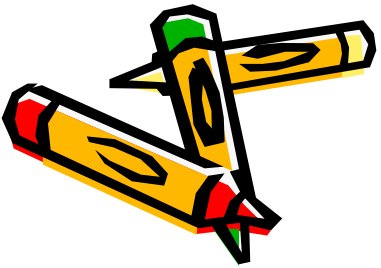
```
{ws}      { /* do nothing */ }
If        {return(IF);}
Then      {return(THEN);}
else      {return(ELSE);}
{id}      {yylval = install_id( ); return(ID);}
{number}  {yylval = install_num();
            return(NUMBER);}
```



Flexプログラムの記述(3)

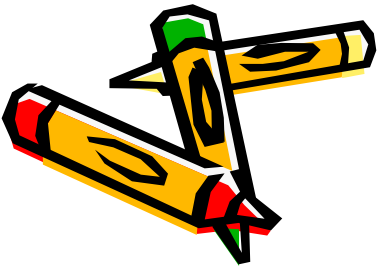


```
"<"     {yylval = LT; return(RELOP);}
"<="    {yylval = LE; return(RELOP);}
"="     {yylval = EQ; return(RELOP);}
"<>"    {yylval = NE; return(RELOP);}
">"     {yylval = GT; return(RELOP);}
">="    {yylval = GE; return(RELOP);}
%%
```

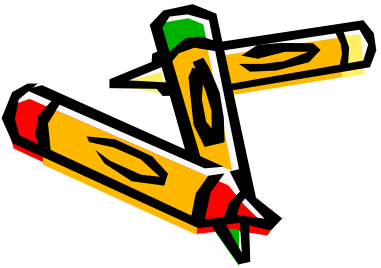


Flexプログラムの記述(4)

```
install_id( ){
   static int id_ptr=0;
   return(id_ptr); }

install_num( ){
   static int num_ptr=0;
   return(num_ptr); }
```



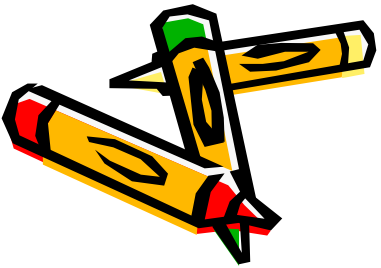
Flexのデモ



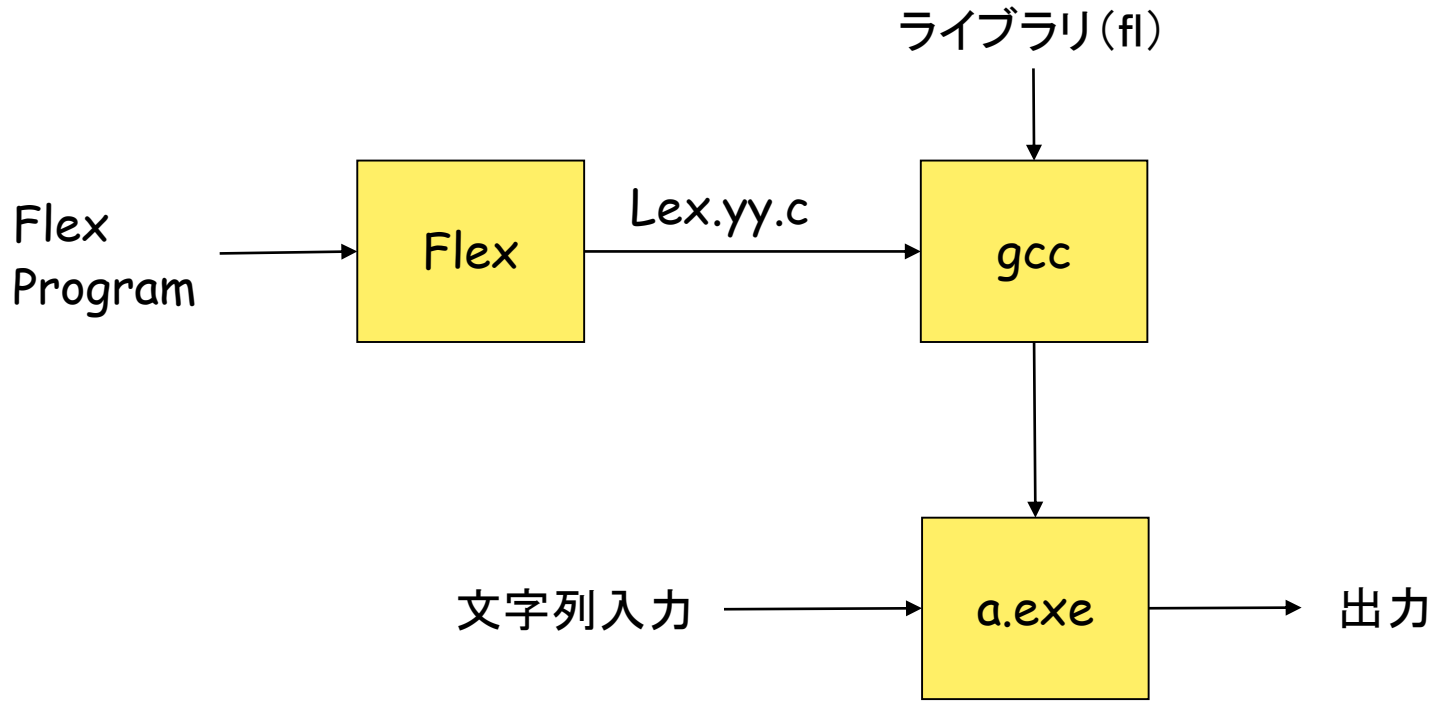
手順

1. Flexのプログラムを書く。
2. Flexのプログラムをflexにかける。
3. 出力ファイルlex.yy.cをgccでコンパイルする。

4. 出力a.exeを実行する。
5. さまざまな文字列を入力する。



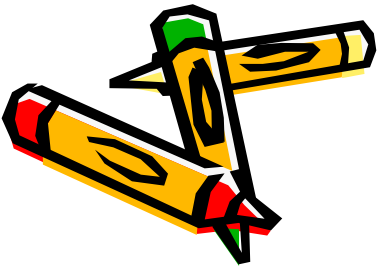
手順



実際の手順

```
C:¥> flex  sample01.l
C:¥> gcc lex.yy.c –lfl
C:¥> a.exe
```

それでは、実際にやってみよう。



字句解析から構文解析へ

以上で、字句解析は終わり。

字句解析の次の処理は、構文解析でしたね。