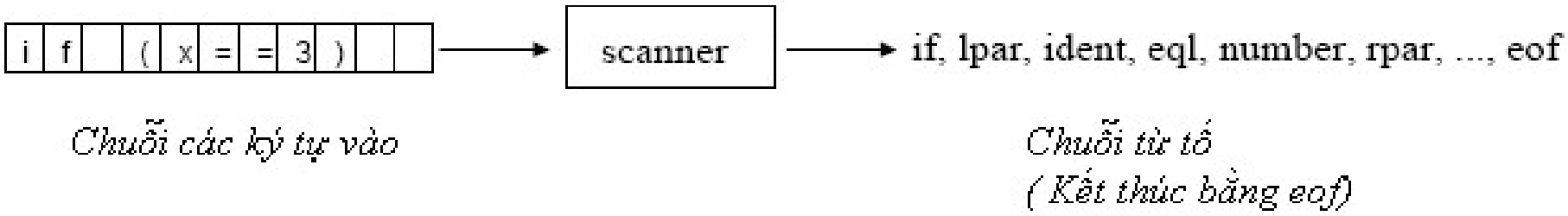
ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài 5
Bộ phân tích từ vựng

Nhiệm vụ của bộ phân tích từ vựng

- Phát hiện các từ tố



- Bỏ qua các ký tự không cần thiết
  - Khoảng trống
  - Dấu tab
  - Ký tự xuống dòng (CR,LF)
  - Chú thích

Từ tố có cấu trúc cú pháp

```
ident =     letter {letter | digit}.
number =    digit {digit}.
if =        "i" "f".
eql =       "=" "=".
...
```

- Tại sao không xử lý các luật này trong giai đoạn phân tích cú pháp ?

Xử lý các luật từ vựng trong bộ phân tích cú pháp ?

- Làm cho bộ phân tích cú pháp trở nên quá phức tạp
  - Phân biệt tên và từ khoá
  - Phải có những luật phức tạp để xử lý chuỗi các ký tự không cần thiết (khoảng trống, tab, chú thích . . . .)

Các từ tố của KPL

- Số nguyên
- Định danh
- Từ khóa: begin,end, if,then, while, do, call, const, var, procedure, program,type, function,of,integer,char,else,for, to,array
- Hằng ký tự
- Dấu phép toán:
  - số học

    + - */
  - so sánh

    = != < > <= >=
- Dấu phân cách

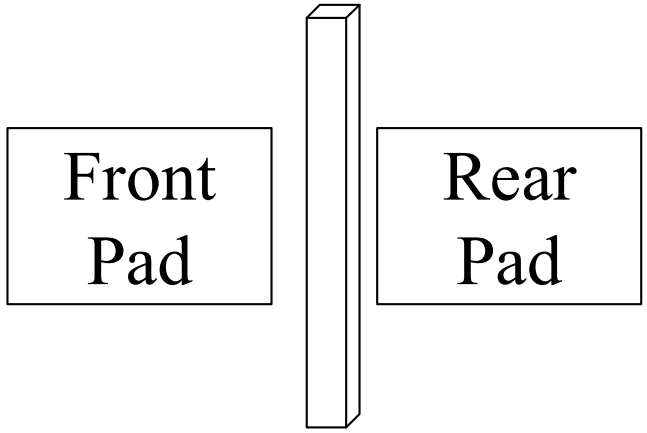
  ( ) . : ; (. .)
- Dấu phép gán :=

Ôtômat hữu hạn

- Ôtômat hữu hạn (máy hữu hạn trạng thái là mô hình được ứng dụng trong nhiều lĩnh vực như kỹ thuật, mạng, xử lý ngôn ngữ
- Nguyên tắc làm việc: Thay đổi trạng thái căn cứ vào trạng thái hiện hành và tín hiệu điều khiển
- Trong kỹ thuật, ôtômat hữu hạn được dùng để điều khiển máy bán hàng tự động, thang máy, biểu diễn mạch, mô tả giao thức truyền thông…
- Trong compiler, ôtômat hữu hạn là mô hình cho bộ phân tích từ vựng
- Bộ sinh phân tích từ vựng là một phần của bộ sinh compiler. Đầu vào là các biểu thức chính quy mô tả các luật từ vựng. Đầu rả là chương trình phân tích từ vựng. Quá trình biến đổi từ biểu thức chính quy sang code đòi hỏi qua nhiều bước trung gian. Tại mỗi bước trung gian, mô hình ôtômat hữu hạn được dung đề biến đổi

Xét một ví dụ: cửa một chiều

- Giả sử cần thiết kế cánh cửa tự động chỉ cho phép đi theo một chiều
- Cửa sẽ có hai vùng được kiểm soát bởi các sensor. Một vùng ở đằng trước và một vùng ở đằng sau cửa. Các vùng đó cho phép nhận biết có người đứng trước hay sau cửa
- Yêu cầu đặt ra là mọi người chỉ được phép đi từ vùng trước cửa sang vùng sau cửa khi cửa mở, không cho phép đi theo hướng ngược lại
- Mô hình ôtômat hữu hạn có thể cho phép giải quyết yêu cầu này



Cửa một chiều

- Trạng thái của cửa Mở/Đóng. Trạng thái thay đổi nhờ các tín hiệu điều khiển sau:

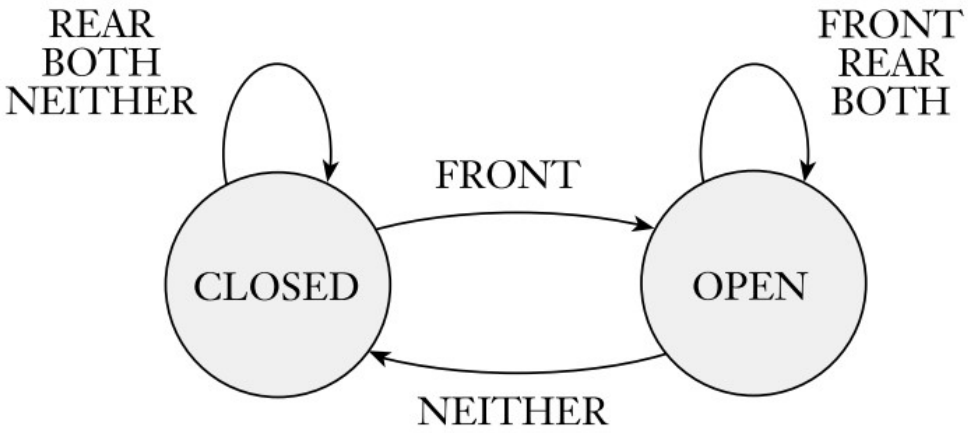
  Không có ai trên vùng trước và vùng sau cửa (NEITHER)

  Chỉ có người ở vùng trước (FRONT)

  Chỉ có người ở vùng sau (REAR)

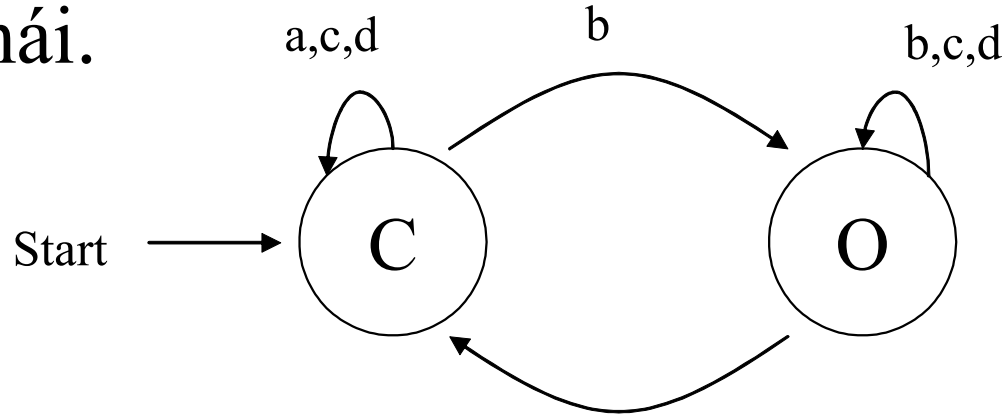
  Có người cả ở vùng trước và vùng sau (BOTH)

- Cửa không được mở khi có người đưng ở vùng sau vì sẽ đập vào họ.
- Ôtômat có thể biểu diễn bằng sơ đồ sau:



Sơ đồ trạng thái của ôtômat hữu hạn

- Đồ thị định hướng với các đỉnh là các trạng thái, các cung có nhãn là các ký hiệu vào
- Trạng thái đầu có mũi tên với nhãn "đầu"
- Các trạng thái kết thúc được ký hiệu bằng vòng tròn kép.
- Ôtômat hữu hạn đơn định (ÔHĐ): Với mọi ký hiệu $a \in \Sigma$ tồn tại nhiều nhất một cung nhãn a xuất phát từ mỗi trạng thái.



Nhận biết các từ tố của KPL

- Ngôn ngữ chứa các từ tố là ngôn ngữ chính quy. Nó có thể được đoán nhận bởi một ôtômat hữu hạn
- Mô hình ôtômat hữu hạn có nhiều loại: đơn định, không đơn định, có dịch chuyển ε…
- Mô hình ôtômat hữu hạn để code bộ phân tích từ vựng là ôtômat hữu hạn đơn định (ÔHĐ)

Định nghĩa hình thức của ôtômat hữu hạn đơn định

- Xuất phát từ 1 trạng thái có tối đa 1 cung ứng với mỗi ký hiệu được xét.
- Khi phải xét hoạt động của một ôtômat hữu hạn trên máy tính, cần phải mô tả nó ở dạng máy đọc được. Mô tả hình thức chính là dạng đó.
- Ôtômat hữu hạn đơn định (ÔHĐ) là bộ 5

  $M = (Q, \Sigma, \delta, qo, F)$, trong đó :

  Σ : Bảng chữ hữu hạn - bảng chữ của xâu vào

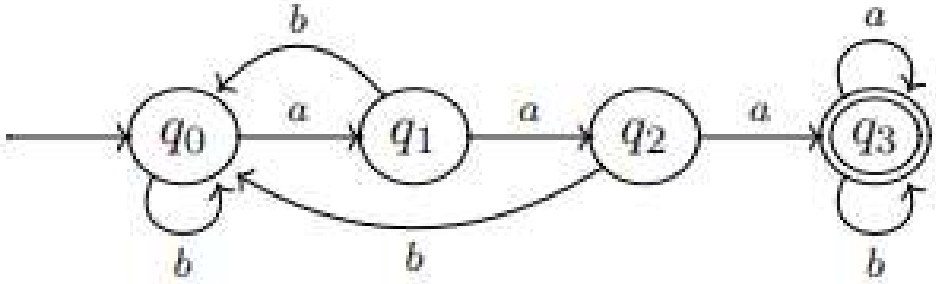
  Q : Tập hữu hạn trạng thái

  $qo \in Q$: Trạng thái đầu

  $F \subseteq Q$: Tập trạng thái kết thúc

  δ : hàm $\Sigma \times Q \rightarrow Q$ gọi là hàm chuyển trạng thái

Ví dụ về ÔHĐ



- Đoán nhận tập các xâu trên {a,b} chứa ba ký hiệu a liên tiếp
- Mô tả hình thức

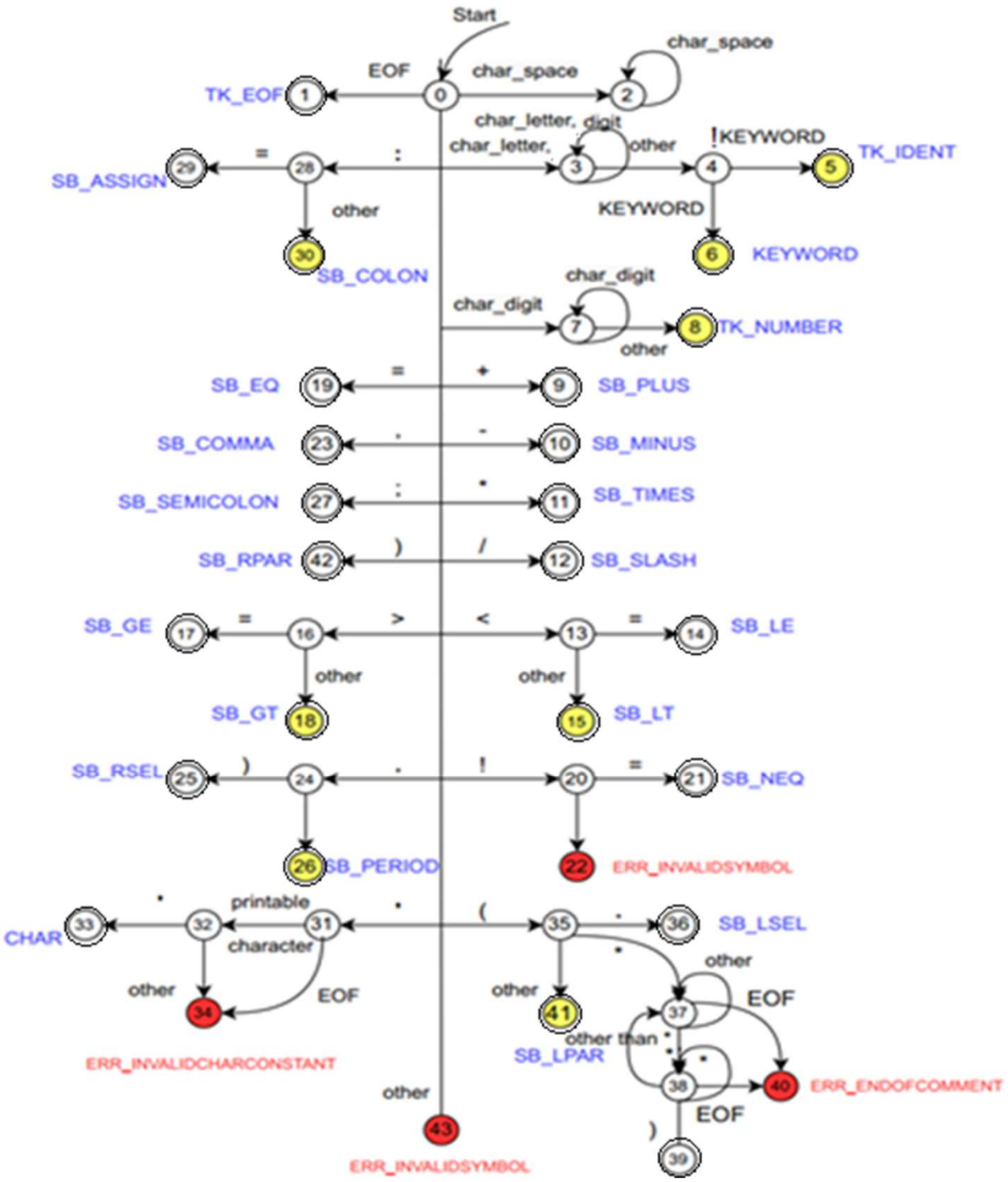
$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a,b\}$, $F = \{q_3\}$

- Hàm chuyển δ được mô tả trong bảng bên

δ	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
q_3	q_3	q_3

Ôtômat hữu hạn của bộ phân tích từ vựng KPL



Mỗi khi đoán nhận được 1 từ tố, ôtômat hữu hạn lại quay về trạng thái 0.

Với những ký tự không đoán nhận được, cần thông báo lỗi.

Nếu ô tô mat đến những trạng thái màu vàng, ký tự hiện hành đã là ký tự đầu của token tiếp theo

Đầu vào, đầu ra của bộ phân tích từ vựng

Vào (CT nguồn)

Program Example1; (* Example 1*)

Begin

End. (* Example1*)

Ra (Dãy từ tố)

1-1:KW_PROGRAM

1-9:TK_IDENT(Example1)

1-17:SB_SEMICOLON

2-1:KW_BEGIN

3-1:KW_END

3-4:SB_PERIOD

Cài đặt bộ phân tích từ vựng

- Phân loại ký tự
- Chọn cấu trúc dữ liệu phù hợp cho token
- Nhận biết các từ tố bằng mô hình ôtômat hữu hạn đơn định

Phân loại các ký tự của CT nguồn

```
typedef enum {
  CHAR_SPACE,              // Khoảng trống
  CHAR_LETTER,             // Chữ cái
  CHAR_DIGIT,              // Chữ số
  CHAR_PLUS,               // '+'
  CHAR_MINUS,              // '-'
  CHAR_TIMES,              // '*'
  CHAR_SLASH,              // '/'
  CHAR_LT,                 // '<'
  CHAR_GT,                 // '<'
  CHAR_EXCLAIMATION,       // '!'
  CHAR_EQ,                 // '='
  CHAR_COMMA,              // ','
  CHAR_PERIOD,             // '.'
  CHAR_COLON,              // ':'
  CHAR_SEMICOLON,          // ';'
  CHAR_SINGLEQUOTE,        // '\''
  CHAR_LPAR,               // '('
  CHAR_RPAR,               // ')'
  CHAR_UNKNOWN             // Ký tự không được phép có mặt trong chương trình
} CharCode;
CharCode charCodes[256] ={.....}
```

Cấu trúc dữ liệu cho từ tố

```
enum {
  TK_NONE, TK_IDENT, TK_NUMBER, TK_CHAR, TK_EOF,

  KW_PROGRAM, KW_CONST, KW_TYPE, KW_VAR,
  KW_INTEGER, KW_CHAR, KW_ARRAY, KW_OF,
  KW_FUNCTION, KW_PROCEDURE,
  KW_BEGIN, KW_END, KW_CALL,
  KW_IF, KW_THEN, KW_ELSE,
  KW_WHILE, KW_DO, KW_FOR, KW_TO,

  SB_SEMICOLON, SB_COLON, SB_PERIOD, SB_COMMA,
  SB_ASSIGN, SB_EQ, SB_NEQ, SB_LT, SB_LE, SB_GT, SB_GE,
  SB_PLUS, SB_MINUS, SB_TIMES, SB_SLASH,
  SB_LPAR, SB_RPAR, SB_LSEL, SB_RSEL
};
```

Các thông tin trong bảng ký hiệu

- Thông tin của định danh
  - Tên: xâu ký tự
  - Thuộc tính: tên kiểu,tên biến, tên thủ tục, tên hằng. . .
  - Kiểu dữ liệu
  - Phạm vi sử dụng
  - Địa chỉ vùng nhớ,kích cỡ vùng nhớ
  - . . .
- Với các số, thông tin về giá trị sẽ được lưu trữ

Thực thi bộ PTTV trên ÔHĐ

```
state = 0;
currentChar = readChar();
token = getToken();
while (token!=EOF)
  {
      state =0;
      token = getToken();
   }
```

Nhận biết từ tố

```
switch (state)
{
case 0 :
  switch (currentChar)
  {
    case space
        state = 2;
    case  lpar
        state = 38;
    case  letter
        state = 3;
    case digit
        state =7;
    case plus
        state = 9;
    case lt
        state = 13
    ……
  }
```

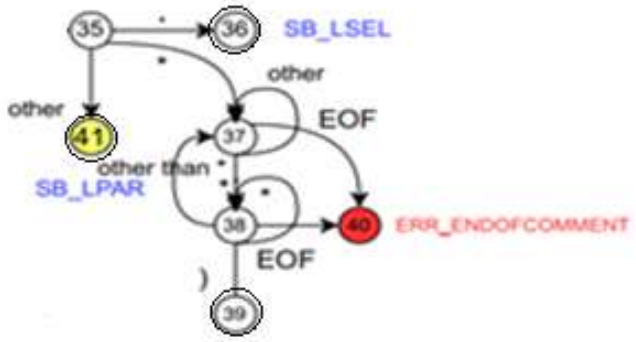
Nhận biết từ tố (tiếp)

```
case 9:
      readChar();
      return SB_PLUS;

 case 13:
      readChar();
      if (currentChar = EQ)state = 14 else state = 15;
 case 14:
       readChar();
       return SB_LE;
  case 15:
       return SB_LT;
```

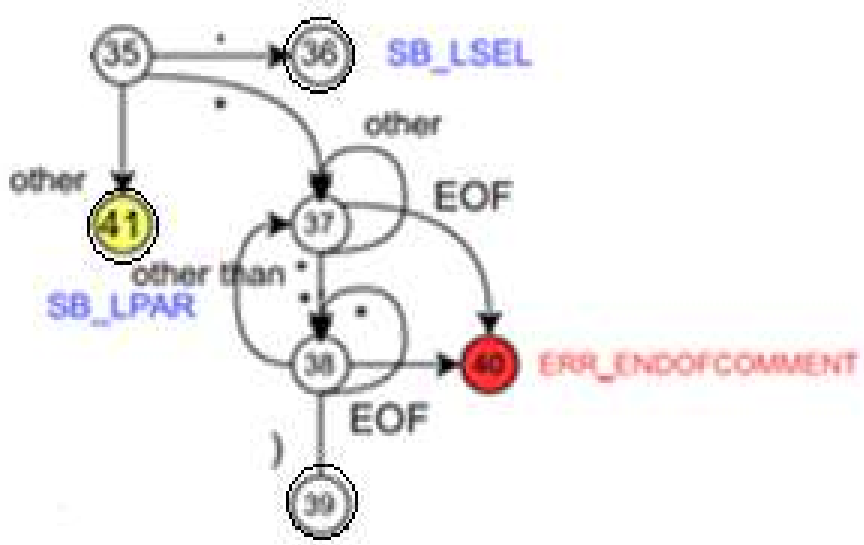
Nhận biết từ tố (tiếp)

```
case 2:
  while (currentChar= space) // skip blanks
        readChar();
  return getToken();
 case 35:
  readChar();
 if (currentChar= EOF)state =41;
 else
        switch  (currentChar)
        {
        case period
                state = 36;//  token lsel
        case times
                state =37; //skip comment
        default
              state =41; // token lpar
        }
        return getToken();
        }
```



Nhận biết từ tố (tiếp)

```
case 37: // skip comment
  readChar();
  while (currentChar != times)
  {
        state = 37;
        readChar();
  }
  state = 38;
case 38:
  readChar();
  while (currentChar == times)
  {
        state = 38;
        currentChar = readChar();
  }
If  (currentChar == lpar)  state = 39; else state =40;
```



Xử lý định danh / từ khoá

- Lập danh mục từ khóa, có thể dùng mảng
- Nếu số lượng từ khóa nhiều có thể phân phối bộ nhớ động
- Lập một hàm trả ra một từ khóa hoặc định danh

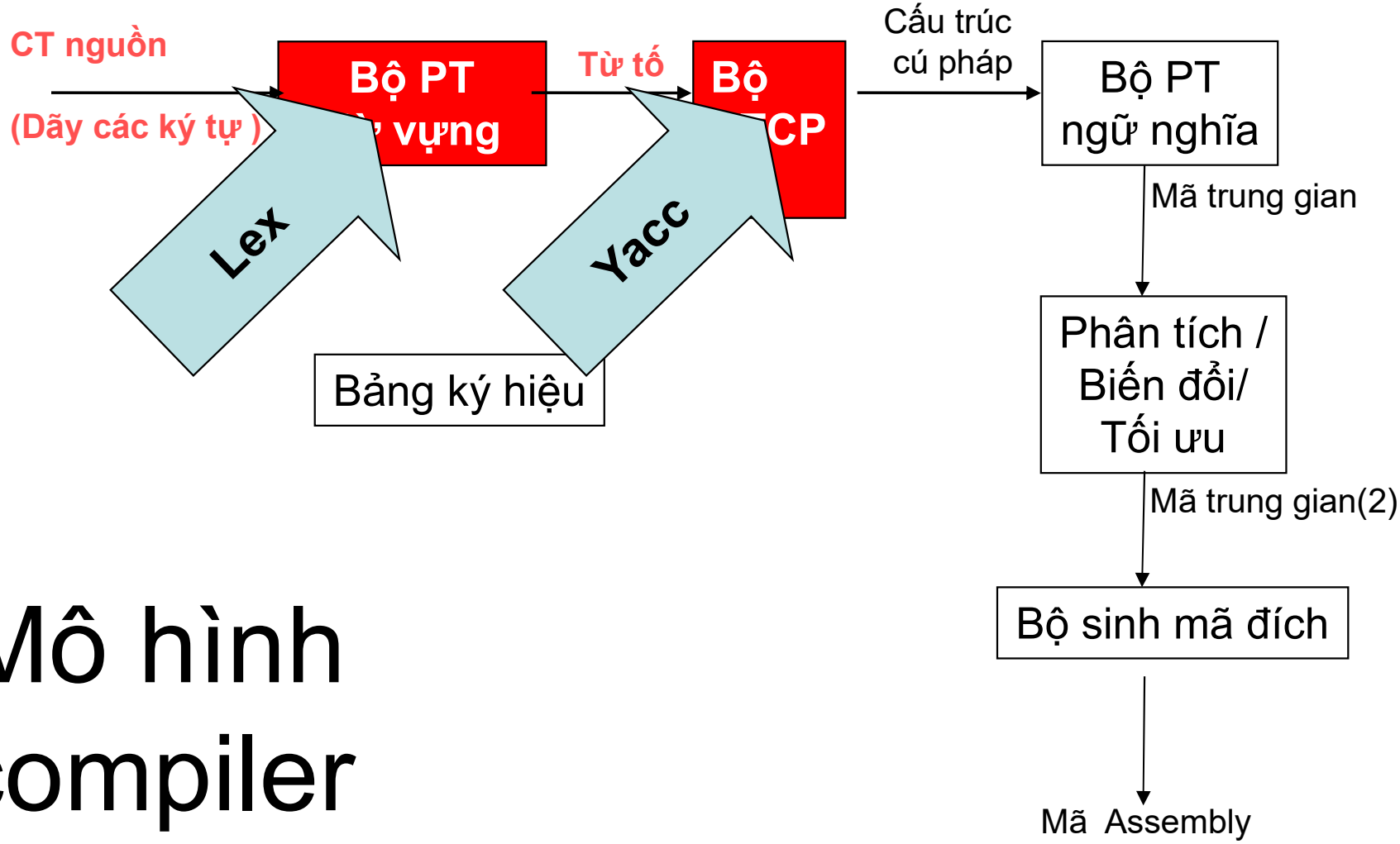
Phân biệt định danh/từ khóa

```
case 4:
    if (checkKeyword (token) ==
 TK_NONE)state = 5; else state =6;
case 5:
 install_ident();// save to symbol
 table
case 6
 return checkKeyword(token);
 ……
```

Bộ sinh phân tích từ vựng

- Công cụ để đặc tả bộ phân tích từ vựng cho nhiều ngôn ngữ
- Một số bộ sinh phân tích từ vựng
  - **Lex và Yacc của AT &T**
  - **Lexer trong ANTLR (**ANother Tool for Language Recognition) ĐH San Francisco
  - **Flex của Berkeley Lab**

Nguyên lý



Mô hình compiler

Bộ sinh phân tích từ vựng

- Vào

  Đặc tả từ vựng của ngôn ngữ lập trình

  (Biểu thức chính quy)

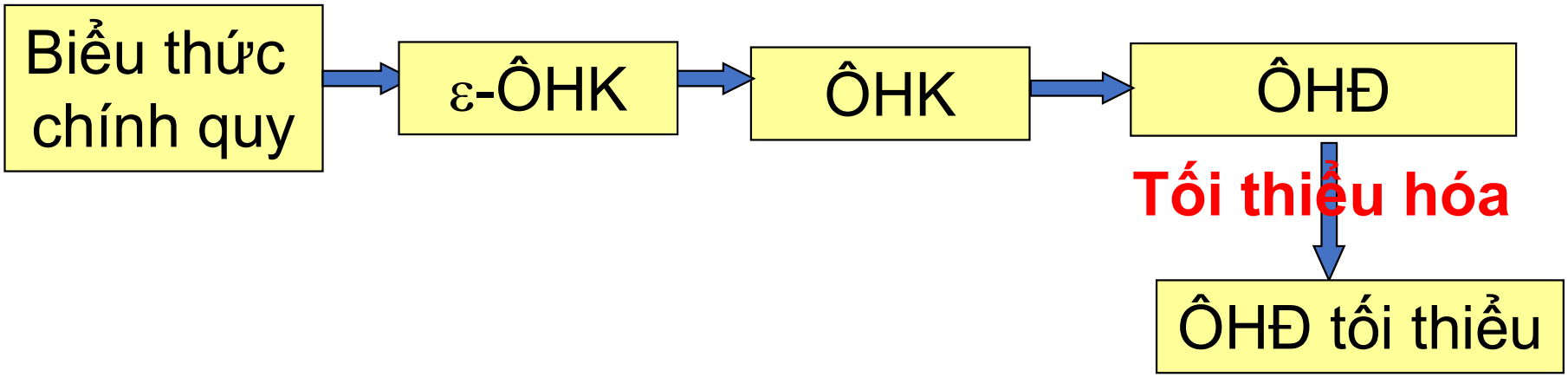
- Ra

  Chương trình phân tích từ vựng

  (chương trình C)

Quá trình biến đổi để xây dựng bộ sinh phân tích từ vựng

- Luật từ vựng được thể hiện bằng các Biểu thức chính quy
- Từ biểu thức chính quy biến đổi tương đương qua các dạng khác nhau của ôtômat hữu hạn.
- Mô tả hình thức của ôtômat hữu hạn đơn định được dung để sinh ra mã nguồn của bộ phân tích từ vựng



Biểu thức chính quy (regular expression)

Định nghĩa hình thức của biểu thức chính quy

Cho Σ là một bảng chữ.

- ∅ là biểu thức chính quy biểu diễn tập ∅
- ε là biểu thức chính quy biểu diễn tập {ε}
- ∀a ∈ Σ , a là biểu thức chính quy biểu diễn tập {a}
- Nếu r và s là các biểu thức chính quy biểu diễn các tập R và S tương ứng thì (r + s), (rs), (r*) là các biểu thức chính quy biểu diễn các tập R ∪ S, RS và R* tương ứng.

Biểu thức chính quy là công cụ tương đương với ô tô mat hữu hạn

Ví dụ

- Tập các xâu trên {0, 1} bắt đầu bằng 1

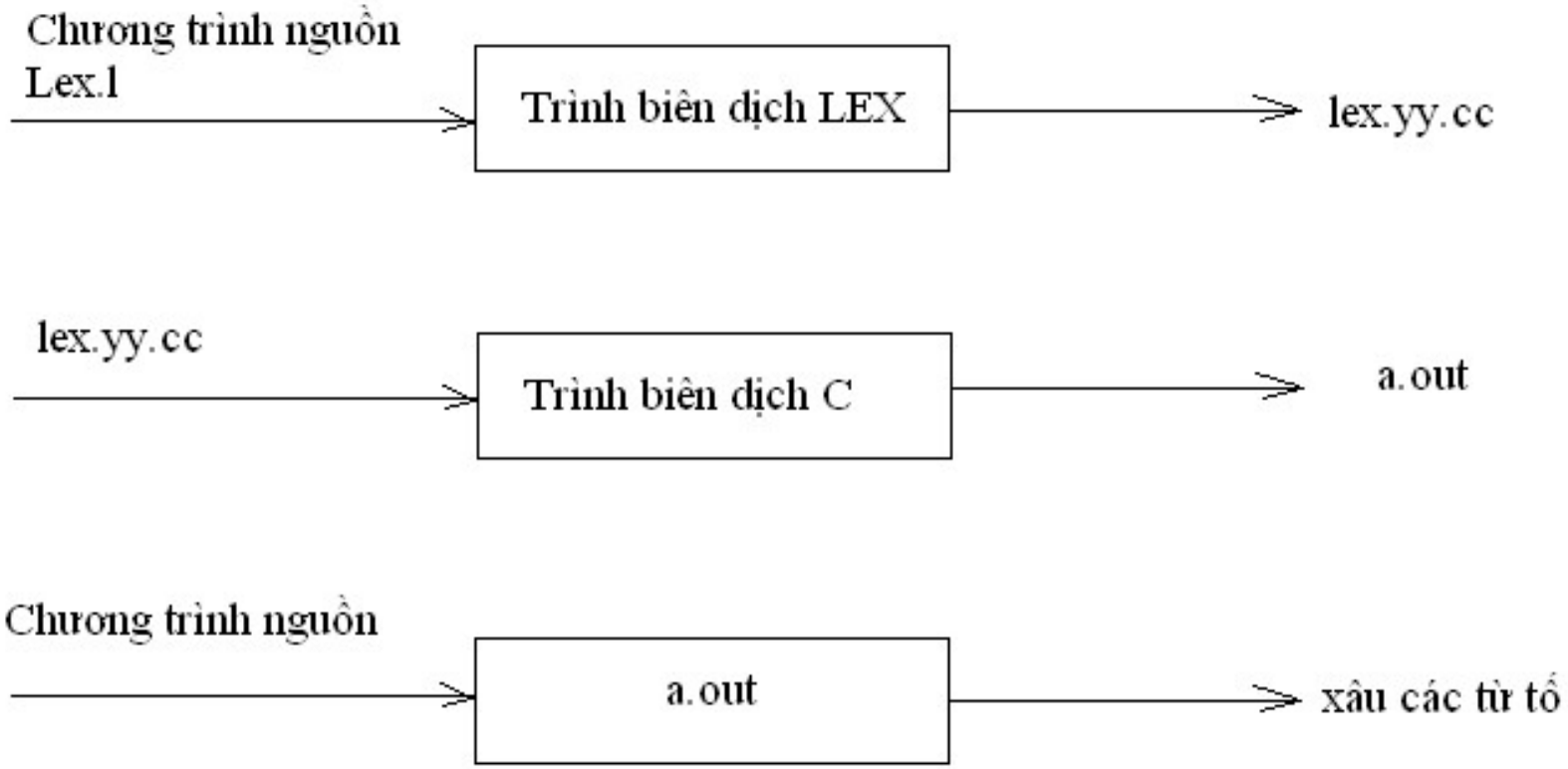
  (1((0+1) *))

- Tập các xâu trên {a,b} có chứa 3 ký hiệu a liên tiếp

((((((a + b)*)a)a)a)((a+b)*))

- Tập các định danh của KPL? Cần sự biến đổi biểu thức chính quy thành định nghĩa chính quy với những định nghĩa cho tập chữ cái, chữ số…

Cấu trúc của bộ sinh phân tích từ vựng



Đặc tả

- Đặc tả bằng một chương trình Lex.l viết bằng ngôn ngữ Lex.
- Chương trình dịch của Lex sẽ sinh ra một chương trình C lex.yy.c.
- Lex.yy.c được cho chạy qua một trình biên dịch C, sinh ra a.out.
- a.out chính là bộ phân tích từ vựng của ngôn ngữ đưa vào

Chương trình Lex.yy.c

- Biểu diễn hình thức của một ôtômat hữu hạn, xây dựng từ các biểu thức chính quy được mô tả trong tệp lex.l ,
- Chứa các thủ tục chuẩn dùng bảng d để nhận dạng các từ tố.

Đặc tả Lex

Phần khai báo

%

Các luật dịch

%

Các thủ tục bổ trợ

Phần khai báo

- Biến
- Hằng hiện
- Định nghĩa chính quy

Các luật dịch

p_1 *{hành động 1}*

p_2 *{hành động 2}*

. . . . .

p_3 *{hành động 3}*

%{Ví dụ

```
enum yytokentype {
NUMBER = 258,
ADD = 259,
SUB = 260,
MUL = 261,
DIV = 262,
ABS = 263,
EOL = 264
};
int yylval;
%}
%%
"+" { return ADD; }
"-" { return SUB; }
"*" { return MUL; }
"/" { return DIV; }
"|" { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n { return EOL; }
[ \t] { /* ignore whitespace */ }
. { printf("Mystery character %c\n", *yytext); }
%%
main(int argc, char **argv)
{
int tok;
while(tok = yylex()) {
printf("%d", tok);
if(tok == NUMBER) printf(" = %d\n", yylval);
else printf("\n");
}
}
```

Kết quả chạy thử nghiệm tệp a.out

flex fb1-4.l

$ cc lex.yy.c -lfl

$ ./a.out

a / 34 + |45

Mystery character a

262

258 = 34

259

263

258 = 45

264

Ví dụ: Dùng Flex sinh scanner cho KPL

- **Dựa trên các đặc điểm từ vựng của KPL**
  - **Số nguyên: không dấu**
  - **Định danh : dùng chữ cái, chữ số, bắt đầu bằng chữ cái**
  - **Hằng ký tự**
  - **Bỏ qua ký tự trắng, chú thích**
- **Xây dựng mô tả từ vựng scanner.l**
- **Dùng Flex sinh ra tệp lex.yy.c**
- **Thực hiện với ví dụ**

Mô tả tệp scanner.l

Phần1: định nghĩa và hàm thư viện :

Tạo ra hàm yyloc() tính số dòng, số cột của từ tố

Tham số truyền vào là text , x, y.Trong đó :

text: từ tố cần phân tích
x số của dòng chứa từ tố

y: số của cột chứa từ tố.

```
%option noyywrap
%{

    void yyloc(char* text, int *x, int *y){
        while (*text != '\0'){
            if (*text == '\n'){
                *y += 1;
                *x = 1;
            } else {
                *x += 1;
            }
            text++;
        }
    }

    int x = 1;
    int y = 1;

%}

NUMBER                  [0-9]+
DELIMITER               [ \n\t\r]
CHAR                    \'[[:print:]]\'
IDENT                   [a-zA-Z][a-zA-Z0-9]*
COMMENT                 \(\*([^*]|(\*+[^*)]))*\*+\)
ERROR                   [^+\-*/,;.:()=a-zA-Z0-9<>]

%%
```

Mô tả tệp scanner.l

Phần 1: định nghĩa và hàm thư viện.

Định nghĩa chính quy:

- NUMBER : các số từ 0→ 9 (có thể mở rộng ra gồm nhiều chữ số) .Biểu thức chính quy : [0-9]+
- DELIMITER : Các dấu trắng , khoảng cách , xuống dòng. Biểu thức chính quy : [\n\t\r]
- CHAR : Nhận diện các kí tự bắt đầu bởi dấu ‘ , kết thúc bởi dấu ‘ va in ra phần bên trong 2 dấu nháy ([:print:] tương đương với hành động in ra) . Biểu thức chính quy : \'[[:print:]]\'
- IDENT: các biến được định nghĩa . Cho phép các kí tự hoặc số. Biểu thức chính quy : [a-zA-Z][a-zA-Z0-9]*
- COMMENT: khớp với các kí tự được bắt đầu bằng “(*” va kết thúc bởi “*)”. Biểu thức chính quy : \(*([^*]|(*+[^*)]))**+\)
- ERROR : khi các kí tự không nằm trong khoảng cho phép (automat kpl) Biểu thức chính quy : [^+\-*/,;.:()=a-zA-Z0-9<>]

```
%option noyywrap
%{

    void yyloc(char* text, int *x, int *y){
        while (*text != '\0'){
            if (*text == '\n'){
                *y += 1;
                *x = 1;
            } else {
                *x += 1;
            }
            text++;
        }
    }

    int x = 1;
    int y = 1;

%}

NUMBER              [0-9]+
DELIMITER           [ \n\t\r]
CHAR                \'[[:print:]]\'
IDENT               [a-zA-Z][a-zA-Z0-9]*
COMMENT             \(\*([^*]|(\*+[^*)]))*\*+\)
ERROR               [^+\-*/,;.:()=a-zA-Z0-9<>]

%%
```

Mô tả tệp scanner.l

- **Phần quy tắc dịch :**
  Do cấu trúc của flex dịch từ trên xuống , do đó ta sẽ ưu tiên thứ tự cho các biểu thức cho phù hợp. Ví dụ như EOF hay COMMENT sẽ duyệt đầu tiên và in ra .
  Hàm phải được thể hiện trên 1 dòng , ta sẽ in ra chữ tương ứng với cú pháp được đưa vào từ input , cộng thêm địa chỉ của từ được phân tích bằng cách sử dụng hàm lấy vị trí đã nêu ở trên.
- Chương trình sẽ in ra bao gồm vị trí của từ “x-y”
  , từ tô.

```
DELIMITER       [ \n\t\r]
CHAR            \'[[:print:]]\'
IDENT           [a-zA-Z][a-zA-Z0-9]*
COMMENT         \(\*([^*]|(\*+[^*)]))*\*+\)
ERROR           [^+\-*/,;.:()=a-zA-Z0-9<>]

%%

<<EOF>>         {return 0;}
{COMMENT}       {yyloc(yytext, &x, &y);}
{DELIMITER}     {yyloc(yytext, &x, &y);}

"+"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_PLUS"); yyloc(yytext, &x, &y);}
"-"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_MINUS"); yyloc(yytext, &x, &y);}
"*"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_TIMES"); yyloc(yytext, &x, &y);}
"/"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_SLASH"); yyloc(yytext, &x, &y);}
";"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_SEMICOLON"); yyloc(yytext, &x, &y);}
","             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_COMMA"); yyloc(yytext, &x, &y);}
"."             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_PERIOD"); yyloc(yytext, &x, &y);}
":="            {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_ASSIGN"); yyloc(yytext, &x, &y);}
"(."            {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_LSEL"); yyloc(yytext, &x, &y);}
".)"            {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_RSEL"); yyloc(yytext, &x, &y);}
"!="            {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_NEQ"); yyloc(yytext, &x, &y);}
">="            {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_GE"); yyloc(yytext, &x, &y);}
"<="            {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_LE"); yyloc(yytext, &x, &y);}
"="             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_EQ"); yyloc(yytext, &x, &y);}
"<"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_LT"); yyloc(yytext, &x, &y);}
">"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_GT"); yyloc(yytext, &x, &y);}
":"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_COLON"); yyloc(yytext, &x, &y);}
"("             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_LPAR"); yyloc(yytext, &x, &y);}
")"             {fprintf(yyout, "%d-%d:%s\n", y, x, "SB_RPAR"); yyloc(yytext, &x, &y);}

"PROGRAM"       {fprintf(yyout, "%d-%d:%s\n", y, x, "KW_PROGRAM"); yyloc(yytext, &x, &y);}
"CONST"         {fprintf(yyout, "%d-%d:%s\n", y, x, "KW_CONST"); yyloc(yytext, &x, &y);}
"TYPE"          {fprintf(yyout, "%d-%d:%s\n", y, x, "KW_TYPE"); yyloc(yytext, &x, &y);}
"VAR"           {fprintf(yyout, "%d-%d:%s\n", y, x, "KW_VAR"); yyloc(yytext, &x, &y);}
"INTEGER"       {fprintf(yyout, "%d-%d:%s\n", y, x, "KW_INTEGER"); yyloc(yytext, &x, &y);}
"CHAR"          {fprintf(yyout, "%d-%d:%s\n", y, x, "KW_CHAR"); yyloc(yytext, &x, &y);}
```

Mô tả tệp scanner.l

- **Phần thủ tục bổ trợ**
- Cho phép file đọc input và in ra file output .
  Khi run file ta sẽ gồm 3 phần là tên chương trình khi biên dịch ra file exe , tên file input , tên output (tùy chọn) .
  Nếu sai cú pháp thoát chương trình , chương trình sẽ tự động in ra cách dùng như hình và thoát .

```
83  int main(argc, argv)
84  int argc;
85  char **argv;
86  {
87      extern FILE* yyin;
88      extern FILE* yyout;
89
90      if (argc == 3) {
91          yyin = fopen(argv[1],"r");
92          yyout = fopen(argv[2], "w");
93      } else {
94          printf("Usage: scanner.exe [input_file] [output_file]");
95          exit(1);
96      }
97      yylex();
98      return 0;
99  }
```