Compiler Design

Lexical Analyzer

Samit Biswas[1]

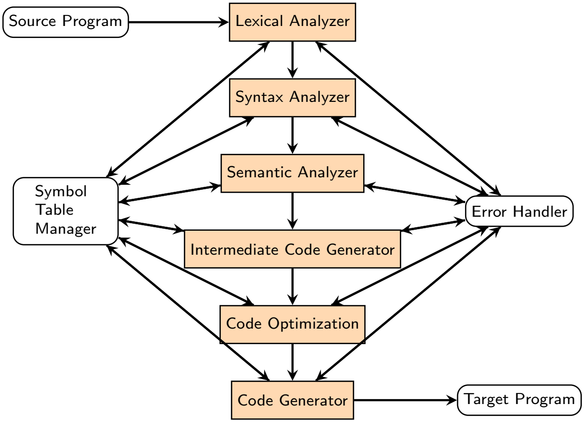
[1]Department of Computer Science and Technology,
Indian Institute of Engineering Science and Technology, Shibpur
Email: samit@cs.iiests.ac.in

Table of Contents

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

The Phases of a Compiler

- Conceptually, a compiler operates in phases, each of which translates the source program from one representation to another.



IIEST, Shibpur
आई आई ई एस टि, शिवपुर

Lexical Analyzer

- The Main task is to read the input characters and produce as output a **sequence of tokens**.

Lexical Analyzer

- The Main task is to read the input characters and produce as output a **sequence of tokens**.
- Stripping from the source program comments and white space in the form of blank, tab, and newline characters.

Lexical Analyzer

- The Main task is to read the input characters and produce as output a **sequence of tokens**.
- Stripping from the source program comments and white space in the form of blank, tab, and newline characters.
- Correlating error messages from the compiler with the same source program

Lexical Analyzer

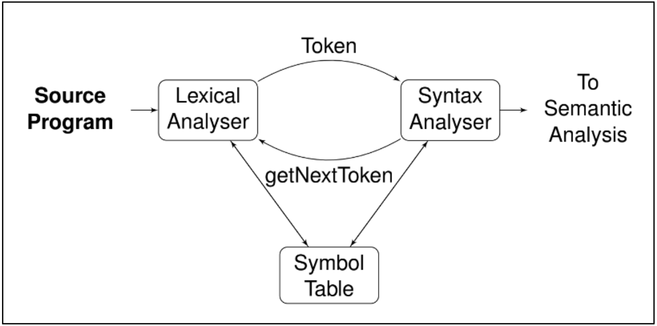


Table of Contents

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Tokens, Patterns and Lexemes

- A **token** is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or a sequence of input characters denoting an identifier.

Tokens, Patterns and Lexemes

- A **token** is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or a sequence of input characters denoting an identifier.
- This set of strings is described by a rule called **pattern** associated with that token. The pattern is said to match each string in the set.

Lexical Analyzer

Tokens, Patterns and Lexemes

- A **token** is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or a sequence of input characters denoting an identifier.
- This set of strings is described by a rule called **pattern** associated with that token. The pattern is said to match each string in the set.
- A **lexeme** is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyser as an instance of that token. These are smallest logical unit (words) of a program such as A, B, 1.0, true, +, ¡= ....

Examples - Tokens, Patterns and Lexemes

Consider The Following C Statement:

printf ("Total = %d", score) ;

- **printf** and **score** are lexemes that match the pattern for the token **id**
- "Total = %d" is a lexeme matching literal.

Token	Sample lexemes	Pattern
if	if	Characters i, f
else	else	Characters e, l, s, e
comparison		
id	pi, score, d2	letters followed by letters and digit
number		any numeric constant
literal	"Total = %d"	Total = %d

Table of Contents

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Specification of Tokens

- Regular Expression.
- Deterministic Finite Automata.
- Non-Deterministic Finite Automata.
- Non-Deterministic Finite Automata with empty transitions.

- The Language accepted by **Finite Automata** are easily described by simple expressions called **Regular Expressions**.

Regular Expressions

- The Language accepted by **Finite Automata** are easily described by simple expressions called **Regular Expressions**.
- **Regular Expression** is a way to represent a language.

IIEST, Shipbur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Regular Expressions

- The Language accepted by **Finite Automata** are easily described by simple expressions called **Regular Expressions**.
- **Regular Expression** is a way to represent a language.
- **Regular expressions** describe **Regular language**.

Regular Expressions

- The Language accepted by **Finite Automata** are easily described by simple expressions called **Regular Expressions**.
- **Regular Expression** is a way to represent a language.
- **Regular expressions** describe **Regular language**.
- **Example:** $(a + bc)^*$
  describes the language
  $\{a, bc\}^* = \{\lambda, a, bc, aa, abc, bca, \ldots\}$

IIEST, Shipbpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Regular Expression

- **Recursive Definition:**
  - Primitive regular expressions: Φ, λ, α

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Regular Expression

- **Recursive Definition:**
  - Primitive regular expressions: Φ, λ, α
  - Given Regular Expressions: r_1 & r_2

Regular Expression

- **Recursive Definition:**
  - Primitive regular expressions: Φ, λ, α
  - Given Regular Expressions: r_1 & r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 . r_2 \\ r_1^* \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Regular Expression

- **Recursive Definition:**
  - Primitive regular expressions: Φ, λ, α
  - Given Regular Expressions: r_1 & r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 . r_2 \\ r_1^* \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

  - A regular expression: $(a + bc)^*(c + \Phi)$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

- **Recursive Definition:**
  - Primitive regular expressions: Φ, λ, α
  - Given Regular Expressions: r_1 & r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 . r_2 \\ r_1^* \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

- A regular expression: $(a + bc)^*(c + \Phi)$
- Not a regular expression: (a+b+)

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Languages of Regular Expressions

- L(R): language of regular expression, r
  Example: $L((a+b+c)^*) = \{\lambda, a, bc, aa, abc, bca, \ldots\}$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Languages of Regular Expressions

- L(R): language of regular expression, r
  Example: $L((a+b+c)^*) = \{\lambda, a, bc, aa, abc, bca, \dots\}$
- For primitive regular expressions:
  - $L(\Phi) = \Phi$
  - $L(\lambda) = \{\lambda\}$
  - $L(a) = \{a\}$
  - $L(r_1 + r_2) = L(r_1) \bigcup L(r_2)$
  - $L(r_1 . r_2) = L(r_1) L(r_2)$
  - $L(r_1^*) = (L(r_1))^*$
  - $(L(r_1)) = L(r_1)$

Languages of Regular Expressions

- **Example:**
  Regular Expression: $(a+b).a^*$
  $L((a+b).a^*) = L(a+b)L(a^*) = (L(a) \bigcup L(b))L(a^*) =$
  $(\{a\} \bigcup \{b\}\{(a)\}^* = \{a, b\}\{\lambda, a, aa, aaa, \ldots\} =$
  $\{a, aa, aaa, \ldots, b, bb, bbb, \ldots\}$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Languages of Regular Expressions

- **Example:**
  Regular Expression: $(a+b).a^*$
  $L((a+b).a^*) = L(a+b)L(a^*) = (L(a)\bigcup L(b))L(a^*) =$
  $(\{a\}\bigcup\{b\}\{(a)\}^* = \{a,b\}\{\lambda, a, aa, aaa, \ldots\} =$
  $\{a, aa, aaa, \ldots, b, bb, bbb, \ldots\}$
- **Regular Expression:** $r = (a+b)^*(a+bb)$
  $L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Languages of Regular Expressions

- **Example:**
  Regular Expression: $(a+b).a^*$
  $L((a+b).a^*) = L(a+b)L(a^*) = (L(a) \bigcup L(b))L(a^*) = (\{a\} \bigcup \{b\}\{(a)\}^* = \{a,b\}\{\lambda, a, aa, aaa, \ldots\} = \{a, aa, aaa, \ldots, b, bb, bbb, \ldots\}$
- **Regular Expression:** $r = (a+b)^*(a+bb)$
  $L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}$
- **Regular Expression:** $r = (aa)^*(bb)^*b$
  $L(r) = \{a^{2n}b^{2m}b : n, m \geq 0\}$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

- **Example:**
  Regular Expression: $(a+b).a^*$
  $L((a+b).a^*) = L(a+b)L(a^*) = (L(a)\bigcup L(b))L(a^*) = (\{a\}\bigcup\{b\}\{(a)\}^* = \{a,b\}\{\lambda, a, aa, aaa, \ldots\} = \{a, aa, aaa, \ldots, b, bb, bbb, \ldots\}$
- **Regular Expression:** $r = (a+b)^*(a+bb)$
  $L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}$
- **Regular Expression:** $r = (aa)^*(bb)^*b$
  $L(r) = \{a^{2n}b^{2m}b : n, m \geq 0\}$
- **Regular Expression:** $r = (0+1)^*00(0+1)^*$
  $L(r) = \{$all strings with at least two consecutive 0$\}$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

- **Example:**
  Regular Expression: $(a+b).a^*$
  $L((a+b).a^*) = L(a+b)L(a^*) = (L(a)\bigcup L(b))L(a^*) = (\{a\}\bigcup\{b\}\{(a)\}^* = \{a,b\}\{\lambda, a, aa, aaa, \ldots\} = \{a, aa, aaa, \ldots, b, bb, bbb, \ldots\}$
- **Regular Expression:** $r = (a+b)^*(a+bb)$
  $L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}$
- **Regular Expression:** $r = (aa)^*(bb)^*b$
  $L(r) = \{a^{2n}b^{2m}b : n, m \geq 0\}$
- **Regular Expression:** $r = (0+1)^*00(0+1)^*$
  $L(r) = \{\text{all strings with at least two consecutive } 0\}$
- **Regular Expression:** $r = (1+01)^*(0+\lambda)$
  $L(r) = \{\text{all strings without two consecutive } 0\}$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Equivalent Regular Expressions

- Regular expressions r_1 and r_2 are equivalent if $L(r_1) = L(r_2)$
- **Example:** L = { all strings without two consecutive 0 }
  $r_1 = (0 + 01)^*(0 + \lambda)$
  $r_2 = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda)$
  $L(r_1) = L(r_2) = L \implies r_1$ and r_2 are equivalent regular expression.

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Regular Expressions and Regular Languages

$$\left.\begin{array}{l}\text{Languages Generated by}\\ \text{by}\\ \text{Regular Expressions}\end{array}\right\} = \{\textbf{Regular Languages}\}$$

$$\left.\begin{array}{l}\text{Languages Generated by}\\ \text{by}\\ \text{Regular Expressions}\end{array}\right\} = \{\textbf{Regular Languages}\}$$

- **Theorem 1:** For any regular expression r the language $L(r)$ is regular.
- **Theorem 2:** For any regular language, L (r), there is a regular expression r with L(r) = L.

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Deterministic Finite Automata (DFA)

- **Deterministic Finite Automata (DFA):**

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

Q: Finite set of states

Σ: Input alphabet

δ: Transition Function: $\delta : Q \times \Sigma \rightarrow Q$

q_0: Initial State

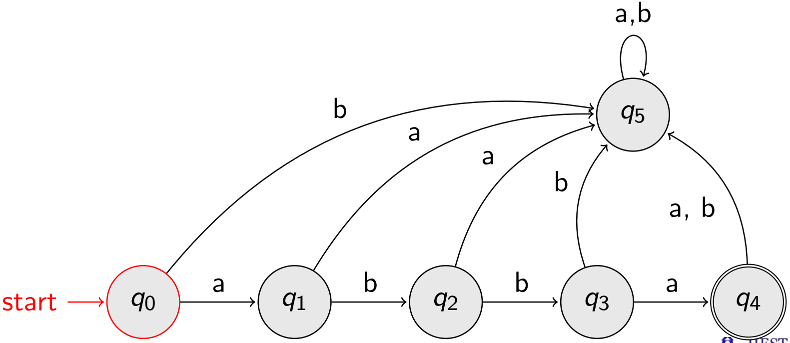
F: Finite set of final states: $F \subseteq Q$

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Formalities

- **Transition Function** , δ:

$$\boxed{\delta : Q \times \Sigma \rightarrow Q}$$

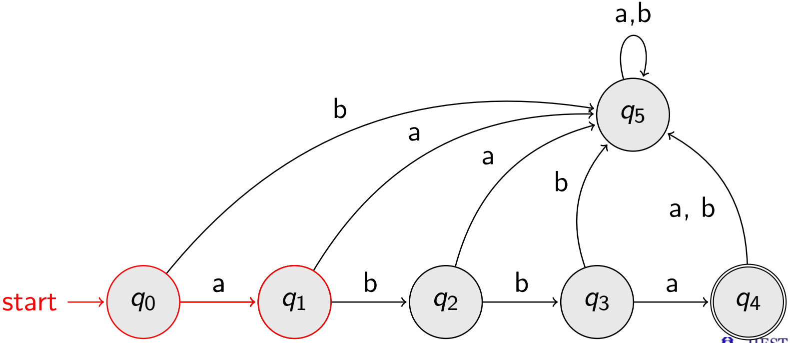


Formalities

- **Transition Function , δ:**

$$\boxed{\delta : Q \times \Sigma \rightarrow Q}$$

$$\delta(q_0, a) = q_1$$



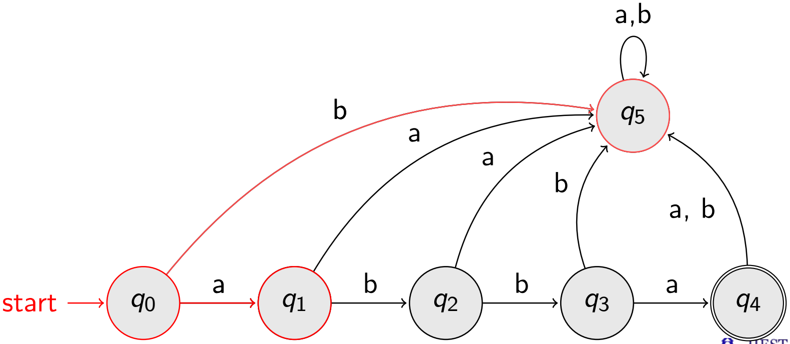
Formalities

- **Transition Function** , δ:

$$\boxed{\delta : Q \times \Sigma \rightarrow Q}$$

$$\delta(q_0, a) = q_1$$

$$\delta(q_0, b) = q_5$$



Formalities

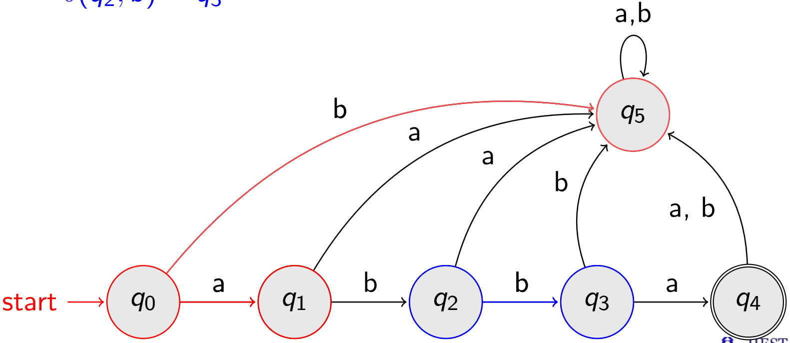
- **Transition Function** , δ:

$$\boxed{\delta : Q \times \Sigma \to Q}$$

$$\delta(q_0, a) = q_1$$

$$\delta(q_0, b) = q_5$$

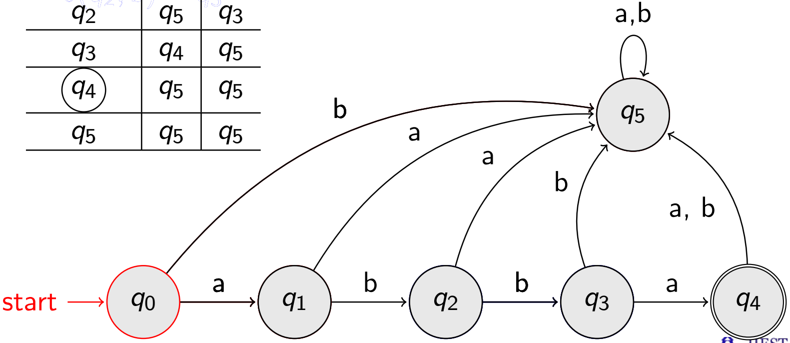
$$\delta(q_2, b) = q_3$$



- **Transition Function**, δ:

$$\boxed{\delta : Q \times \Sigma \rightarrow Q}$$

States	a	b
$\rightarrow q_0$	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5



Languages Accepted by DFAs

- The language accepted by a DFA, $M = (Q, \Sigma, \delta, q_0, F)$ can be defined as follows:
  - $L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}$
    Here,
    Q: Finite set of states
    Σ: Input alphabet
    δ: Transition function: $\delta^* : Q \times \Sigma^* \to Q$
    q_0: Initial State
    F: Finite set of final states

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

- **Nondeterministic Finite Accepter (NFA):**

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

Q: Finite set of states

Σ: Input alphabet

δ: Transition Function: $\delta : Q \times \Sigma \to 2^Q$

q_0: Initial State

F: Finite set of final/accepting states: $F \subseteq Q$

Nondeterministic Finite Accepter with ϵ-transitions (ϵ-NFA)

- **Nondeterministic Finite Accepter with ϵ-transitions (ϵ-NFA):**

$$M = (Q, \Sigma \bigcup \{\epsilon\}, \delta, q_0, F)$$

Q: Finite set of states

$\Sigma \bigcup \{\epsilon\}$: Input alphabet, $\{\epsilon\}$ is a member of this alphabet.

δ: Transition Function: $\delta : Q \times \Sigma \bigcup \{\epsilon\} \rightarrow 2^Q$

q_0: Initial State

F: Finite set of final/accepting states: $F \subseteq Q$

Table of Contents

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Recognitions of Tokens

Regular expression pattern

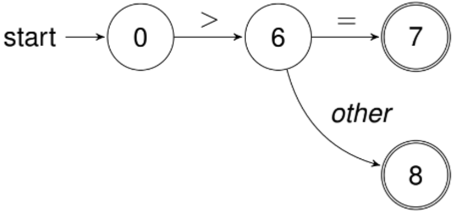
Regular Expression	Token	Attribute Value
WS	–	–
if	if	–
then	then	–
else	else	–
id	id	pointer to table entry
num	num	pointer to table entry
$<$	relop	LT
$<=$	relop	LE
$>$	relop	GT
$<=$	relop	LE
$=$	relop	EQ
$<>$	relop	NE

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

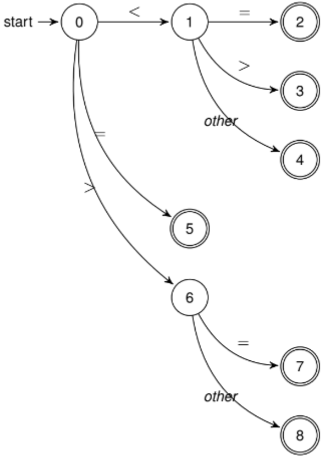
Isolate Lexeme

Construct a lexical analyser that will isolate the lexeme for the next token in the input buffer and produce as output a pair consisting of the appropriate token and attribute-value, using the given translation table.

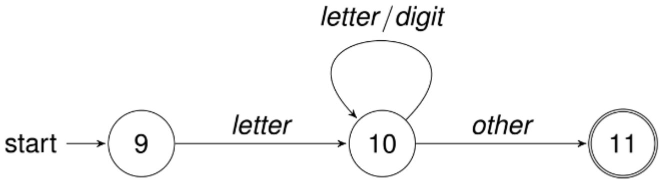
Transition Diagram for $>=$



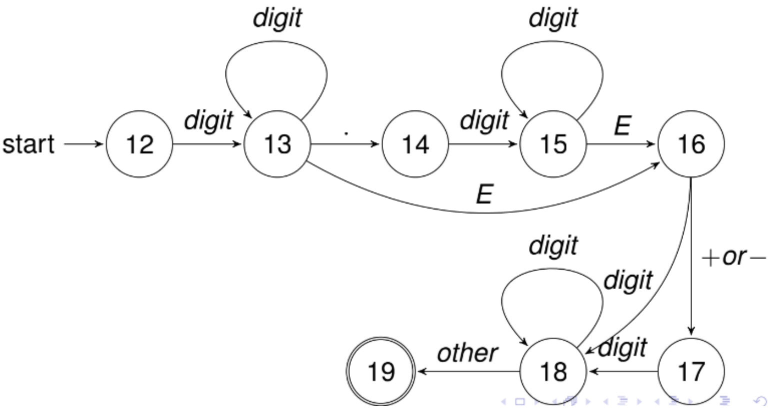
Transition Diagrams for Relational Operators



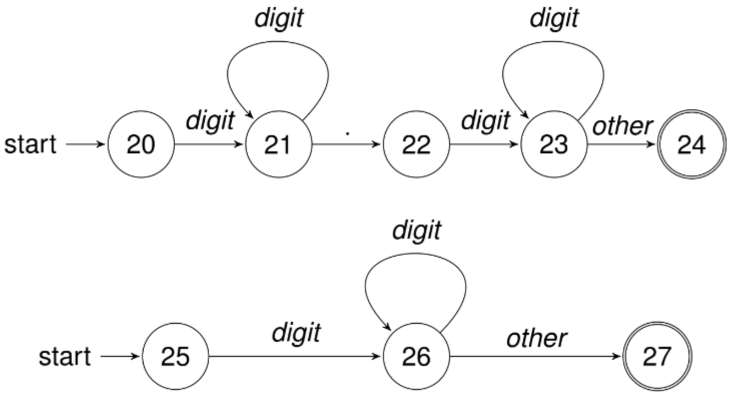
Transition Diagrams for Identifiers or Keywords



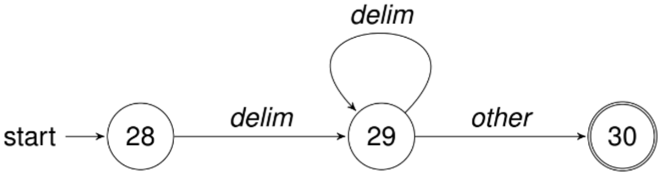
Transition Diagram for Numbers



Transition Diagram for Numbers



Transition Diagrams for White spaces

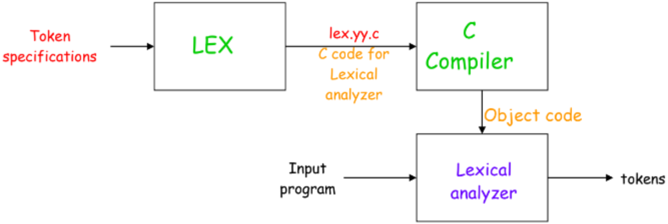


Isolate Lexeme

Implementing a Transition Diagram

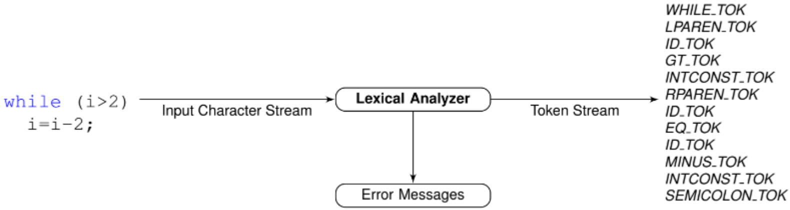
```
token nexttoken()
{ while (1) {
    switch (state) {
    case 0: c = nextchar();
       if (c==blank || c==tab || c==newline) {
         state = 0;
         lexeme_beginning++;
       }
       else if (c=='<') state = 1;
       else if (c=='=') state = 5;
       else if (c=='>') state = 6;
       else state = fail();
       break;
     case 1:
       ...
     case 9: c = nextchar();
       if (isletter(c)) state = 10;
       else state = fail();
       break;
     case 10: c = nextchar();
       if (isletter(c)) state = 10;
       else if (isdigit(c)) state = 10;
       else state = 11;
       break;
     ...
```

Lexical Analyzer Generators - Lex



Lexical Analyzer Generators - Lex

- converts the input program into a sequence of Tokens.
- can be implemented with the help of Finite Automata.



IIEST, Shibpur
आई आई ई एस टि, शिवपुर

Lexical Analyzer Generators - Lex

```
FILE *yyin;
char *yytext;
main(int argc, char *argv[]){
int token;
if (argc != 2){

}else{
        yyin = fopen(argv[1], "r");
        while(!feof(yyin)){
                token = yylex();
                printf("%d", token);
        }
        fclose(yyin);
      }
}
```

```
int yylex(){
    ...
    ...
}
```

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Lexical Analyzer Generators - Lex

Loop and switch Approach

```
/* Single caharacter lexemes */
#define LPAREN_TOK '('
#define GT_TOK '>'
#define RPAREN_TOK ')'
#define EQ_TOK '='
#define MINUS_TOK '-'
#define SEMICOLON_TOK ';'
/*.....................
.......................*/
/* Reserved words */
#define WHILE_TOK 256
/*.....................
.......................*/
/* Identifier, constants..*/
#define ID_TOK 350
#define INTCONST 351
/*.....................
.......................*/
```

References

- Alfred V. Aho, Ravi Sethi, Jeffrey D Ullman, "Compilers Principles Techniques and Tools", Pearson Education.