# 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

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

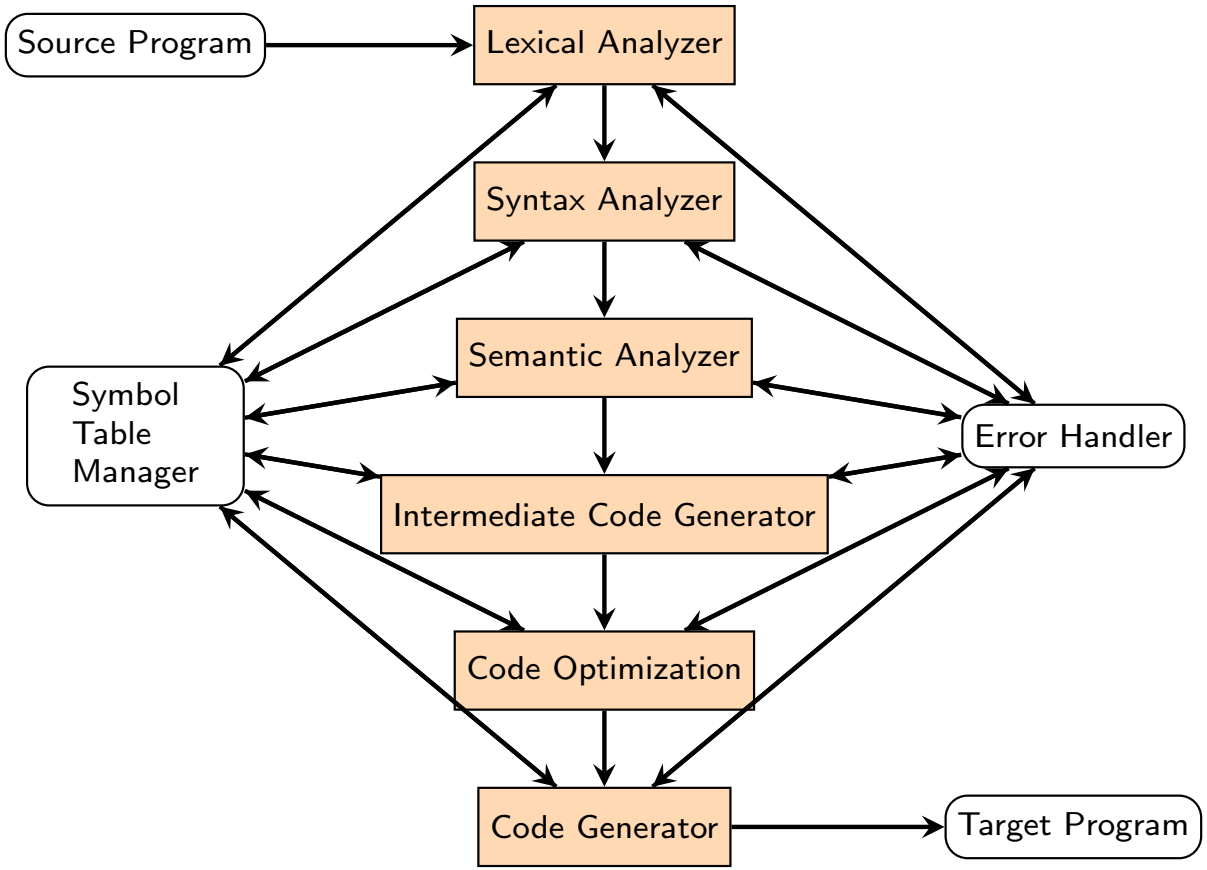
# Table of Contents



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
Erstwhile B E College (Estd 1856)

# Lexical Analyzer

## Lexical Analyzer

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

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

# Lexical Analyzer

**Lexical Analyzer**

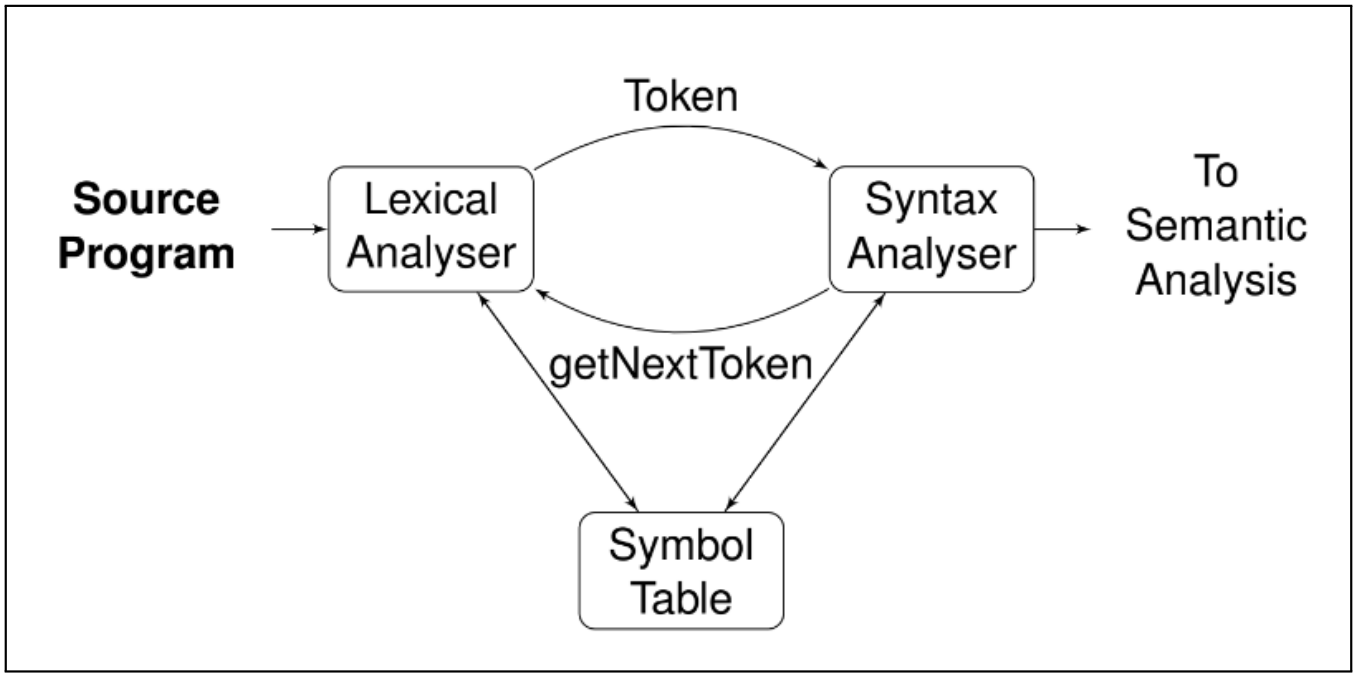
- The Main task is to read the input characters and produce as output a **sequence of tokens**.
- Stripping from the source program comments and white space in the form of blank, tab, and newline characters.

# Lexical Analyzer

**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)

# 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.

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

# 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.

# 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

**Specification of Tokens**

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

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

# Recognitions of Tokens

**Regular expression pattern**

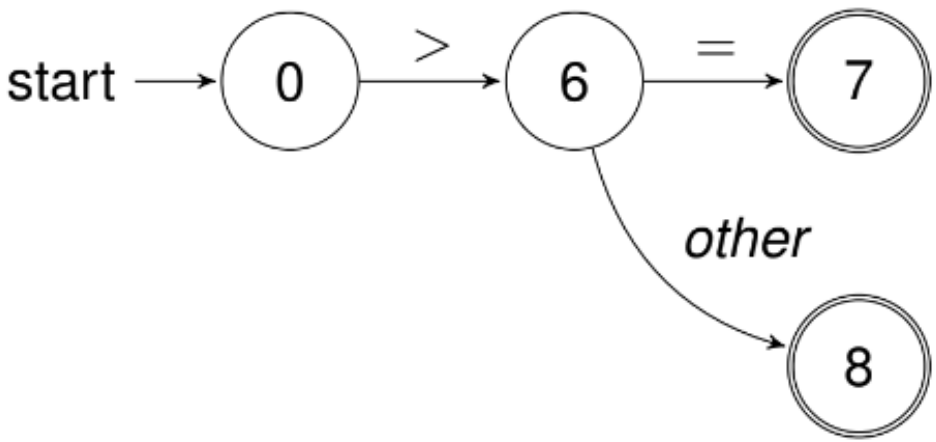
| Regular Expression | Token | Attribute Value |
|---|---|---|
| WS | – | – |
| if | if | – |
| then | then | – |
| else | else | – |
| id | id | pointer to table entry |
| num | num | pointer to table entry |
| < | relop | LT |
| <= | relop | LE |
| > | relop | GT |
| <= | relop | LE |
| = | relop |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

# Isolate Lexeme

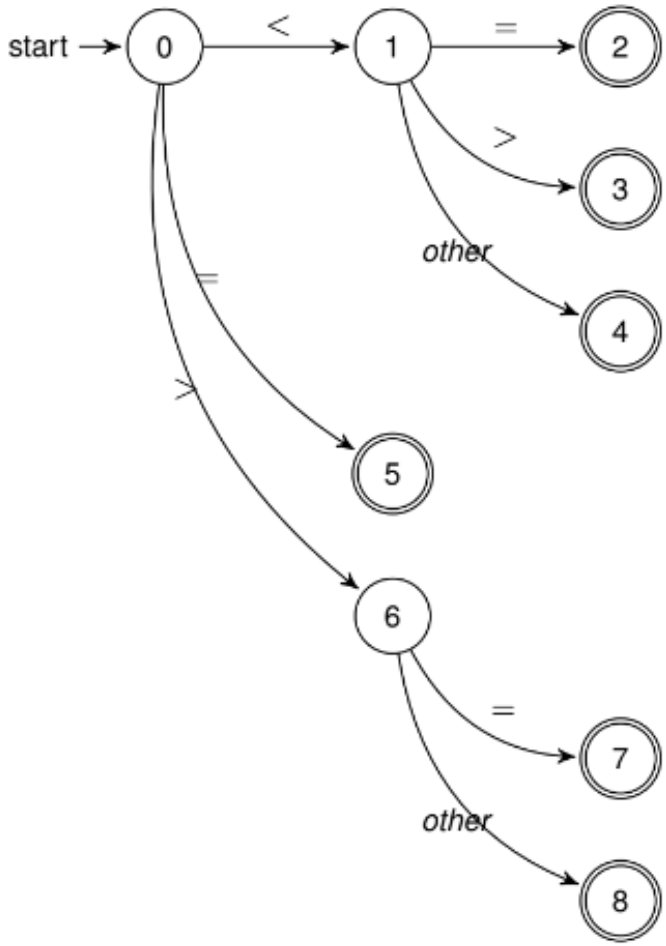
Construct a lexical analyser that will isolate the lexeme for the next token in the input buffer and produce as output a pair consisting of the appropriate token and attribute-value, using the given translation table.

# Isolate Lexeme

**Transition Diagram for $>=$**

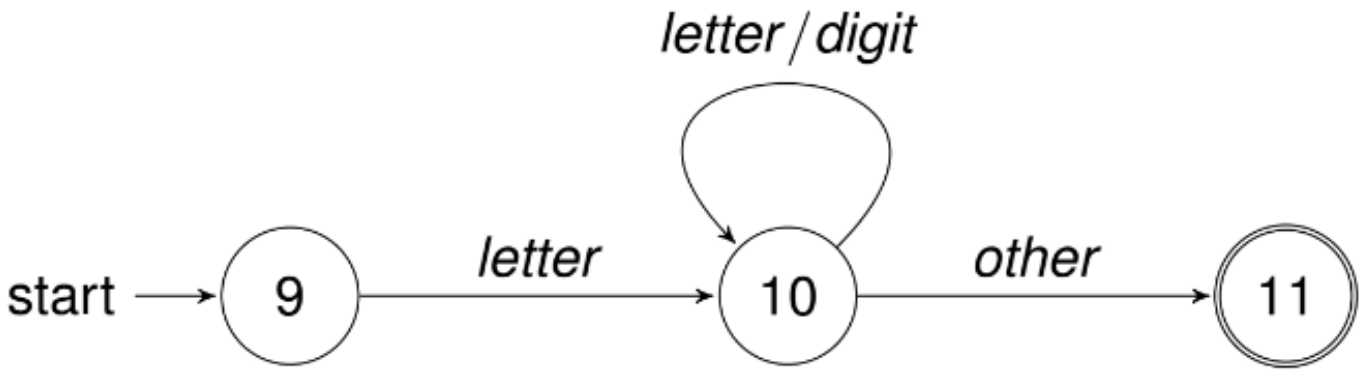


## Transition Diagrams for Relational Operators



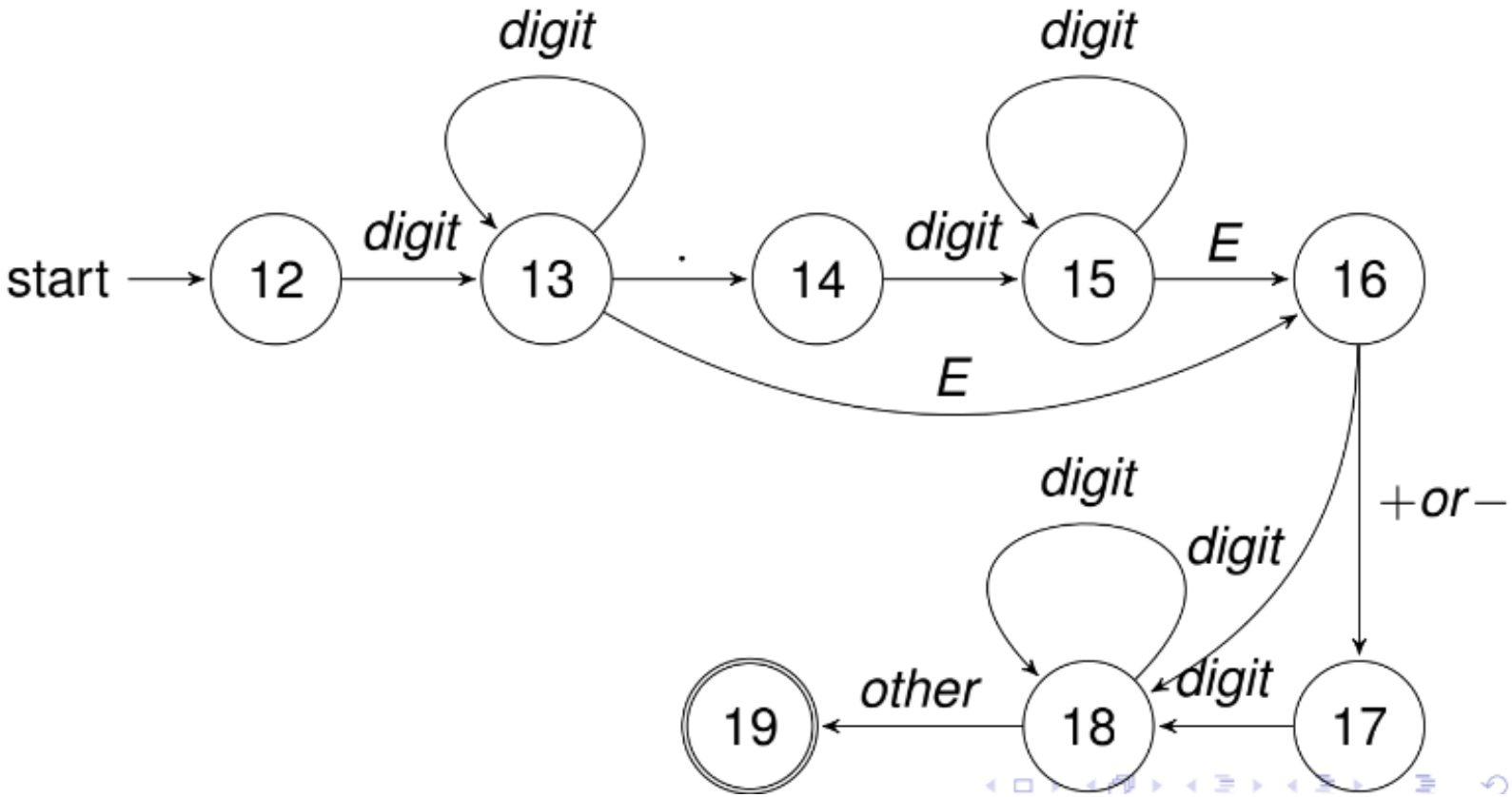
# Isolate Lexeme

**Transition Diagrams for Identifiers or Keywords**

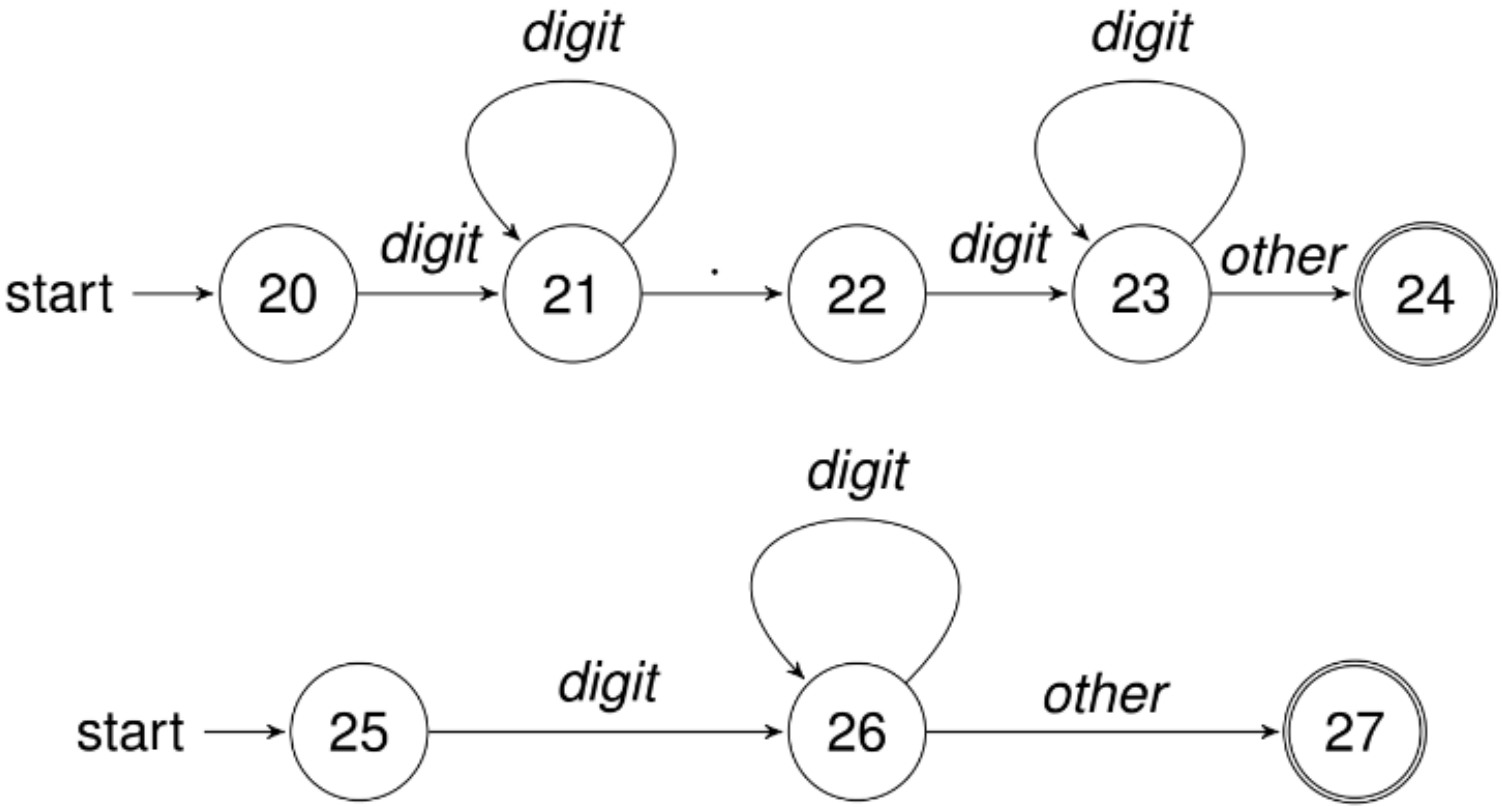



IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

## Transition Diagram for Numbers

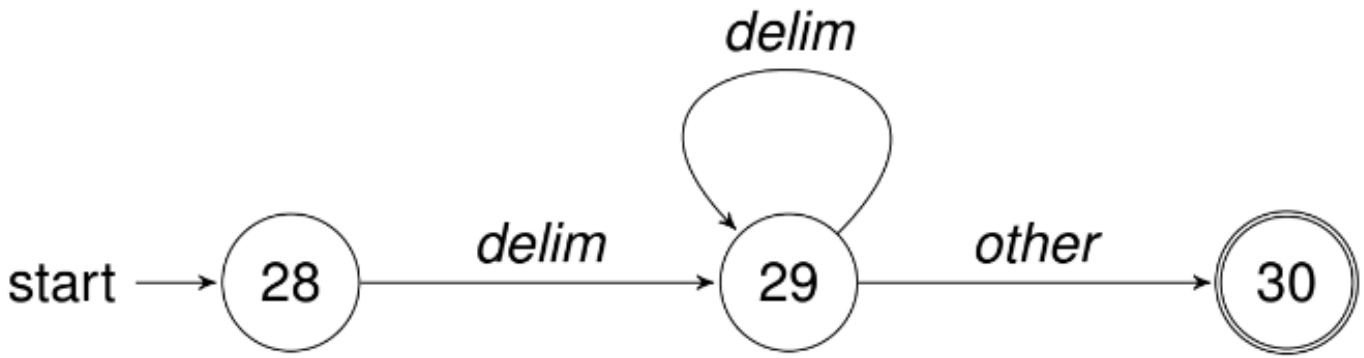


## Transition Diagram for Numbers



# Isolate Lexeme

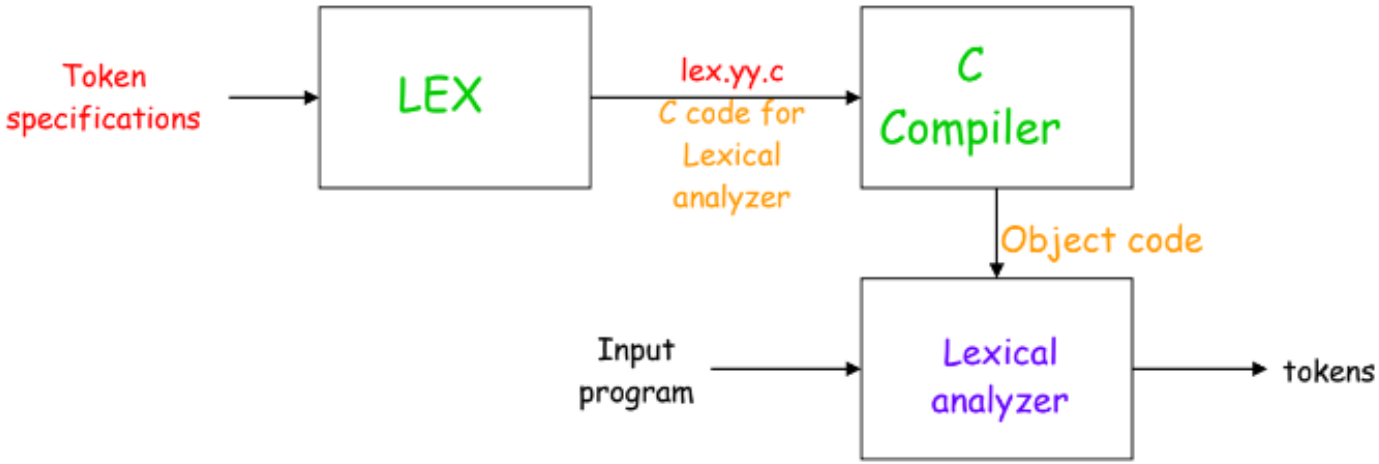
**Transition Diagrams for White spaces**



## Implementing a Transition Diagram

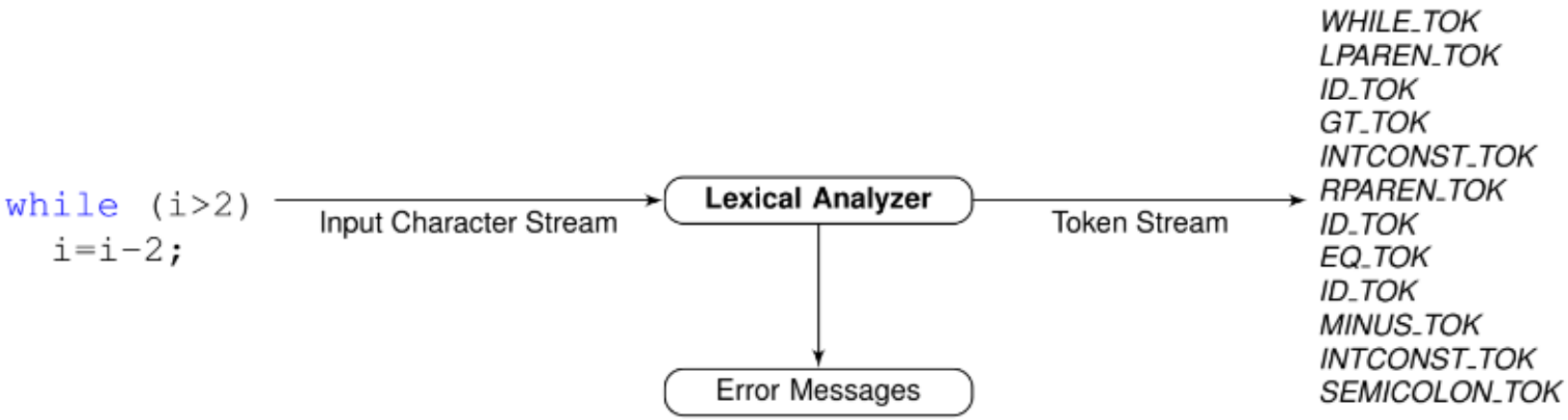
```
token nexttoken()
{ while (1) {
    switch (state) {
    case 0: c = nextchar();
       if (c==blank || c==tab || c==newline) {
         state = 0;
         lexeme_beginning++;
       }
       else if (c=='<') state = 1;
       else if (c=='=') state = 5;
       else if (c=='>') state = 6;
       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
Erstwhile B E College (Estd 1856)

## 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(){
    ...
    ...
}
```

## 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
/*.....................
.......................*/
```

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

# References


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


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