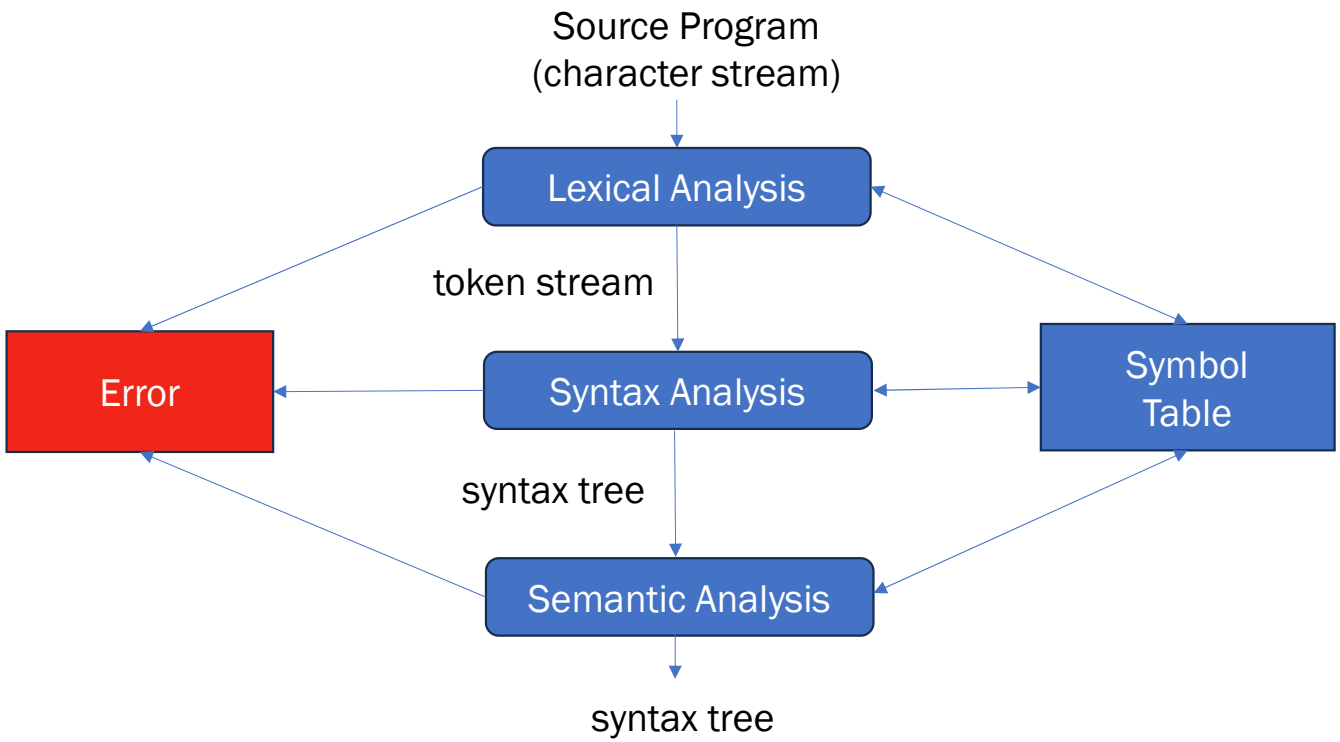
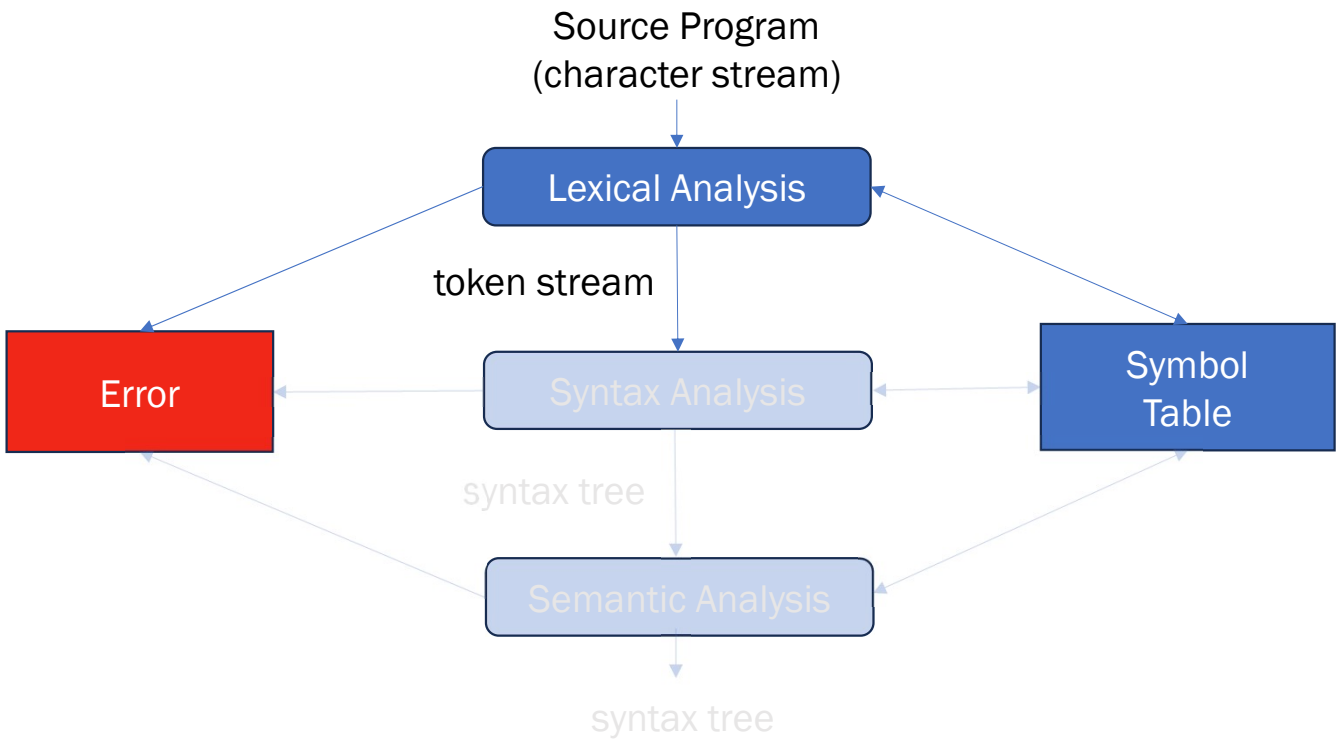
Lexical Analysis

Professor: Suman Saha

Compiler (front-end)



Compiler (front-end)



Lexical Analysis or Scanning

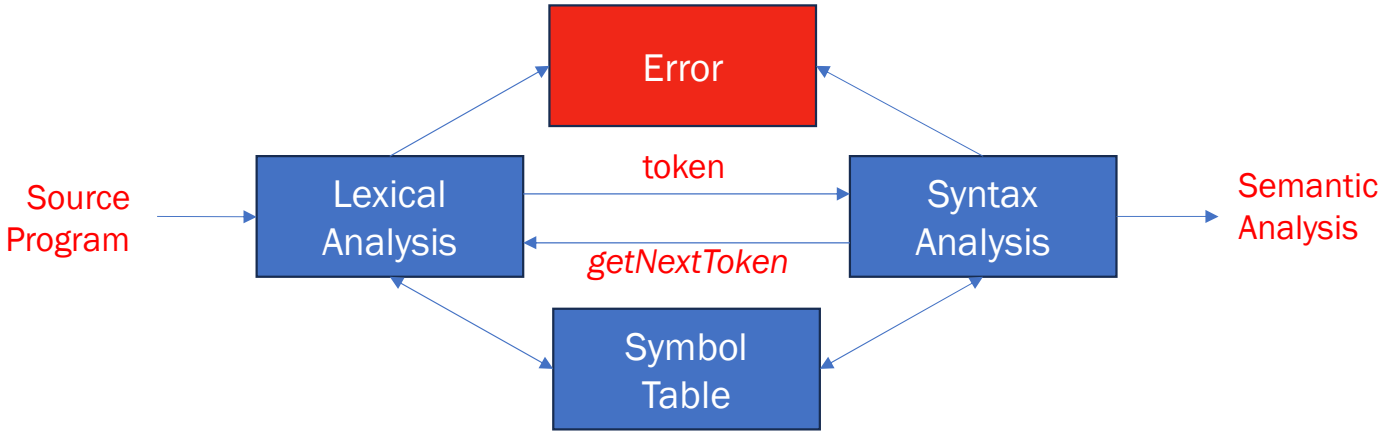
PennState

- Goals of the Lexical Analysis
  - Divide the characters stream into meaningful sequences called lexemes.
  - Label each lexeme with a token that is passed to the parser (syntax analysis)
  - Remove non-significant blanks and comments
  - Optional: update the symbol tables with all identifiers (and numbers)

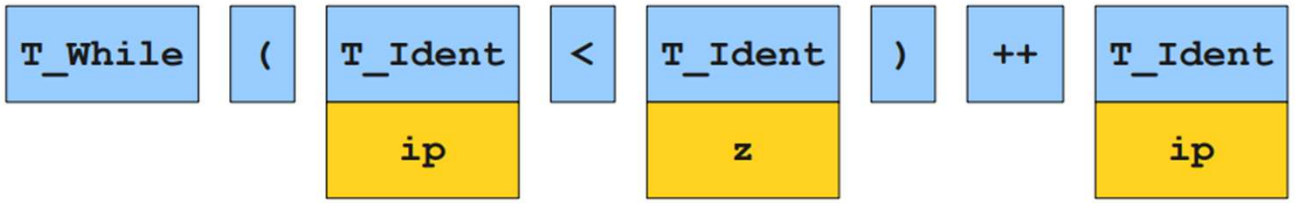
Lexical Analysis or Scanning

PennState

- Goals of the Lexical Analysis
  - Divide the characters stream into meaningful sequences called lexemes.
  - Label each lexeme with a token that is passed to the parser (syntax analysis)
  - Remove non-significant blanks and comments
  - Optional: update the symbol tables with all identifiers (and numbers)
- Provide the interface between the source program and the parser



Example



w	h	i	l	e		(	i	p		<		z	)	\n	\t	+	+	i	p	;

```
while (ip < z)
    ++ip;
```

Tokens, Patterns, and Lexemes

PennState

- A token is a <name, attribute> pair. Attribute might be multi-valued.
  - Example: <*Ident, ip*>, <*Operator, <*>, <*")", NIL*>
- A pattern describes the character strings for the lexemes of the token.
  - Example: a string of letters and digits starting with a letter, {<, >, ≤, ≥, ==}, ")".
- A lexeme for a token is a sequence of characters that matched the pattern for the token
  - Example: **ip**, "<", ")" in the following program

```
while (ip < z)
        ++ip
```

Defining a Lexical Analysis

PennState

- Define the set of tokens
- Define a pattern for each token (ie., the set of lexemes associated with each token)
- Define an algorithm for cutting the source program into lexemes and outputting the token

Choosing the tokens

- Very much dependent on the source language
- Typical token classes for programming languages:
  - One token for each keyword
  - One token for each “punctuation” symbol (left and right parentheses, comma, semicolon…)
  - One token for identifiers
  - Several tokens for the operators
  - One or more tokens for the constants (numbers or literal strings)

Choosing the tokens

- Very much dependent on the source language
- Typical token classes for programming languages:
  - One token for each keyword
  - One token for each “punctuation” symbol (left and right parentheses, comma, semicolon…)
  - One token for identifiers
  - Several tokens for the operators
  - One or more tokens for the constants (numbers or literal strings)
- Attributes
  - Allows to encode the lexeme corresponding to the token when necessary. Example: pointer to the symbol table for identifiers, constant value for constants.
  - Not always necessary. Example: keywords, punctuation…

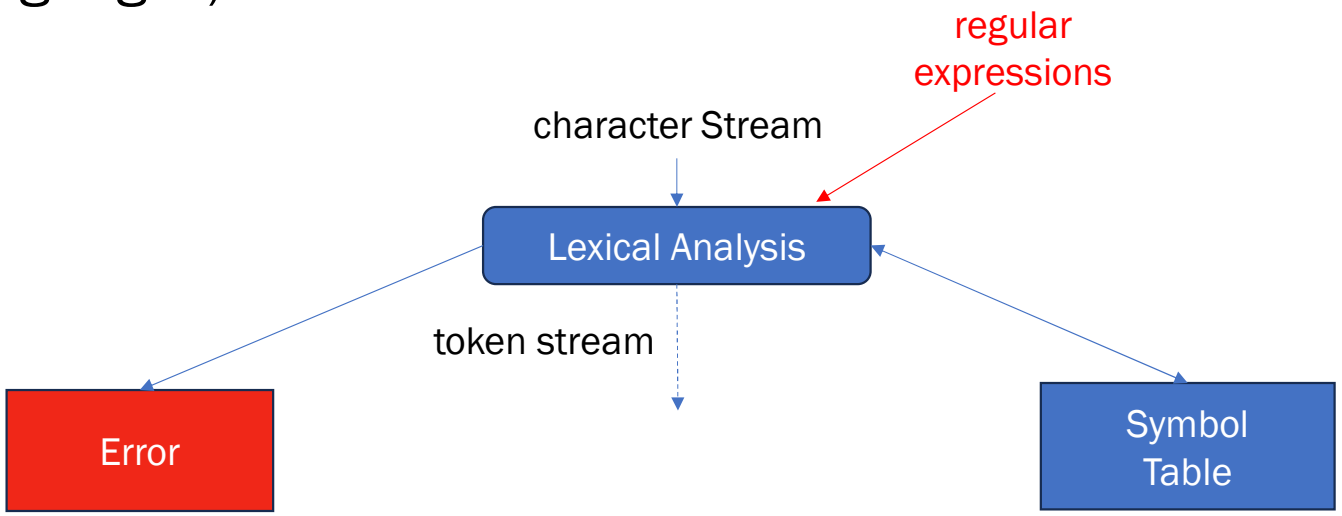
Describing the Patterns

PennState

- A pattern define the set of lexemes corresponding to a token
- A lexeme being a string, a pattern is actually a language.
- Patterns are typically defined through regular expressions (that define regular languages)

Describing the Patterns

- A pattern define the set of lexemes corresponding to a token
- A lexeme being a string, a pattern is actually a language.
- Patterns are typically defined through regular expressions (that define regular languages)



Top Hat

PennState

Is It as Easy as It Sounds?

PennState

- FORTRAN rule: Whitespace is insignificant
- E.g. VAR1 is the same as VA R1

- Consider
  - DO 5 I = 1, 25 ! loop
  - DO 5 I = 1.25 ! Variable declaration

- What is the difference here?
  - Reading left-to-right, the lexical analyzer cannot tell if DO5I is a variable or a DO statement until after "," is reached

Lexical Analysis in FORTRAN

PennState

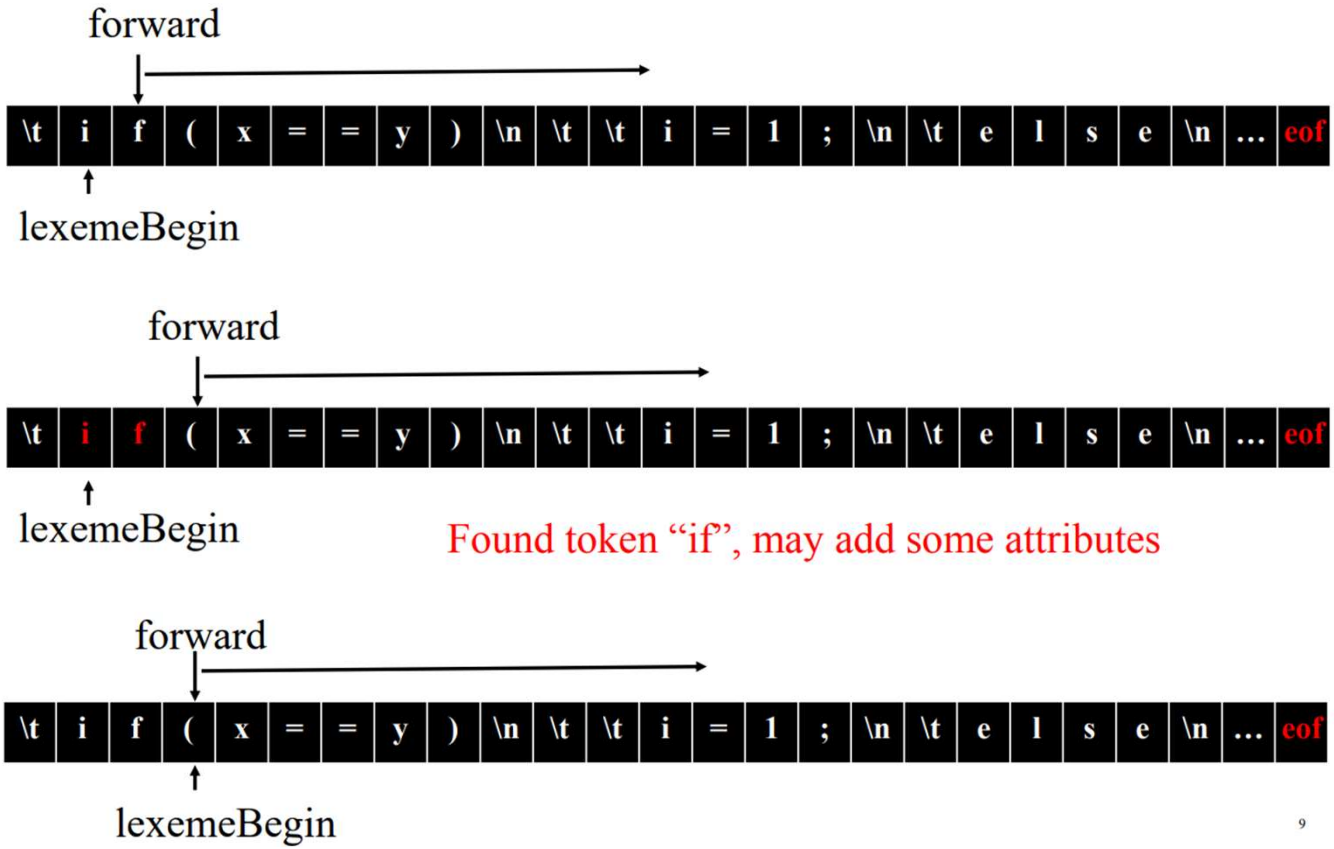
- Two important points:
  - The goal is to partition the string. This is implemented by reading left-to-right, recognizing one token at a time
  - "Lookahead" may be required to decide where one token ends and the next token begins
- Even our simple example has lookahead issues

i **vs.** if

= **vs.** ==

```
if (i == j)
    Z = 0;
else
    Z = 1;
```

Lookahead


forward
\t i f (x = = y) \n \t \t i = 1 ; \n \t e l s e \n … eof
lexemeBegin
forward
\t i f (x = = y) \n \t \t i = 1 ; \n \t e l s e \n … eof
lexemeBegin
Found token "if", may add some attributes
forward
\t i f (x = = y) \n \t \t i = 1 ; \n \t e l s e \n … eof
lexemeBegin

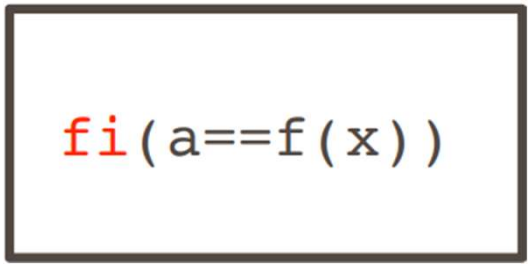
Lexical Errors

PennState

- A lexical error is any input that can be rejected by the lexer
- When a token cannot be recognized by the rules defined token class
  - Example: '@' is rejected as a lexical error for identifiers in Java (it's reserved).

- Recovery
  - Panic Mode: delete successive characters until a valid token is found
  - Delete one character from remaining inputs
  - Insert one character in the remaining input
  - Replace / transpose

Lexical Errors



- Is fi lexical error?
  - It can be a function identifier
  - It is quite difficult for a lexical analyzer to decide whether fi is an error without further information