UCS1602:
COMPILER DESIGN

Role of

Lexical analyser & Input buffering

Session Objectives

- To learn concepts tokens, input buffering
- To study about specification of tokens

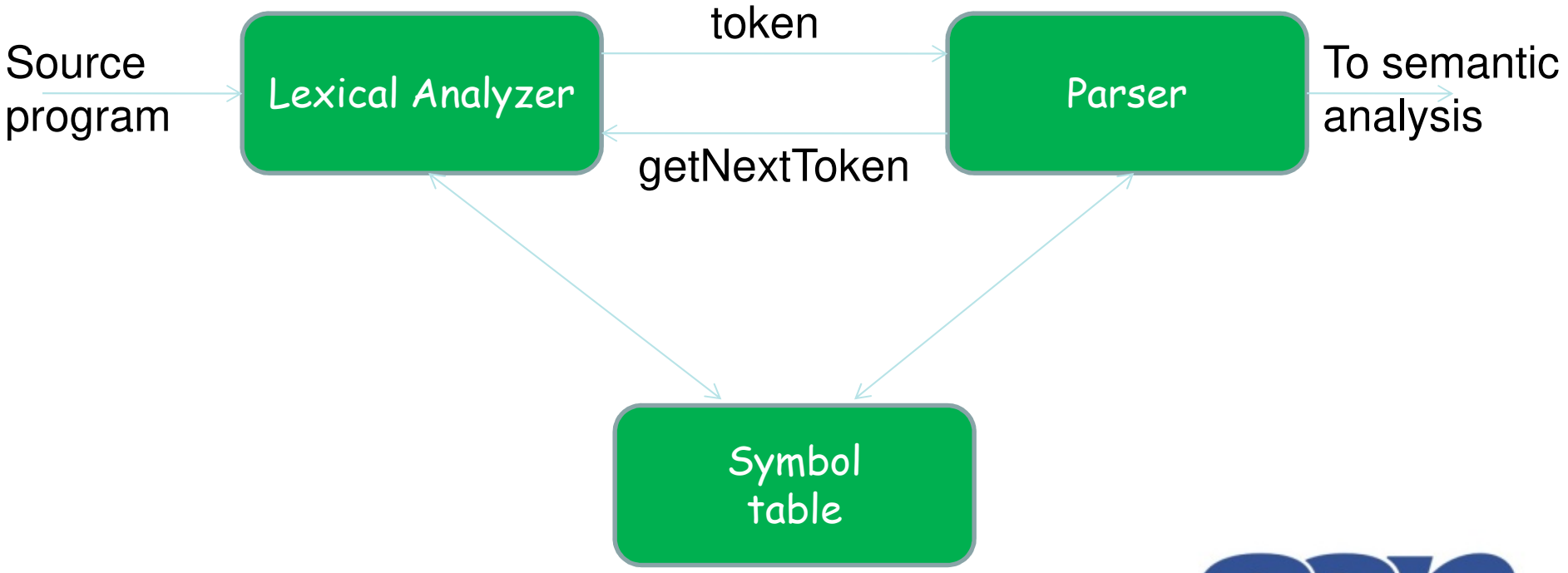
Session Outcomes

- At the end of this session, participants will be able to
  - Understand the concepts of tokens, input buffering

Outline

- Role of lexical analyzer
- Input buffering

The Role of Lexical Analyzer



Why to separate Lexical analysis and Parsing

1. Simplicity of design

   Removing white space by Lexical Analyzer pays way for easy implementation of parsing

2. Improving compiler efficiency

   Large amount of time is spend in reading the source program. Buffering techniques for reading input characters.

3. Enhancing compiler portability

   The representation of special symbols can be isolated in Lexical Analyzer

Tokens, Patterns and Lexemes

- A **token** is a pair of a token name and an optional token value
- A **pattern** is a description of the form that the lexemes of a token may take
- A **lexeme** is a sequence of characters in the source program that matches the pattern for a token

Example

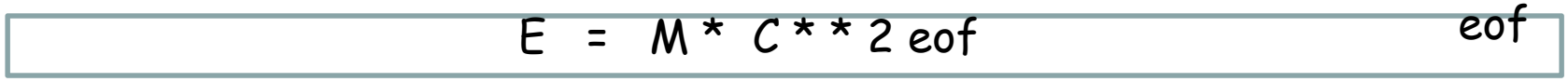
Token	Informal description	Sample lexemes
if	Characters i, f	if
else	Characters e, l, s, e	else
comparison	< or > or <= or >= or == or !=	<=, !=
id	Letter followed by letter and digits	pi, score, D2
number	Any numeric constant	3.14159, 0, 6.02e23
literal	Anything but " sorrounded by "	"core dumped"

Attributes for tokens

- E = M * C ** 2
  - <id, pointer to symbol table entry for E>
  - <assign-op>
  - <id, pointer to symbol table entry for M>
  - <mult-op>
  - <id, pointer to symbol table entry for C>
  - <exp-op>
  - <number, integer value 2>

Input Buffering

- Sometimes lexical analyzer needs to look ahead some symbols to decide about the token to return
  - In C language: we need to look after -, = or < to decide what token to return
- We need to introduce a two buffer scheme to handle large look-ahead safely



Buffer pairs

- Buffer is divided into two N-characters halves.
- N is number of characters eg: 1024 or 4096.

Figure: An input buffer in two halves.

Buffer pairs

```
If forward at end of first half then begin
      reload second half
      forward := forward + 1
  end
  else if forward at end of second half then begin
          reload first half
          move forward to beginning of first half
  end
  else forward:=forward +1
```

Buffer pairs

For Eg: consider

abc = pqr * xyz;

Buffer pairs

For Eg: consider

abc = pqr * xyz;

Buffer pairs

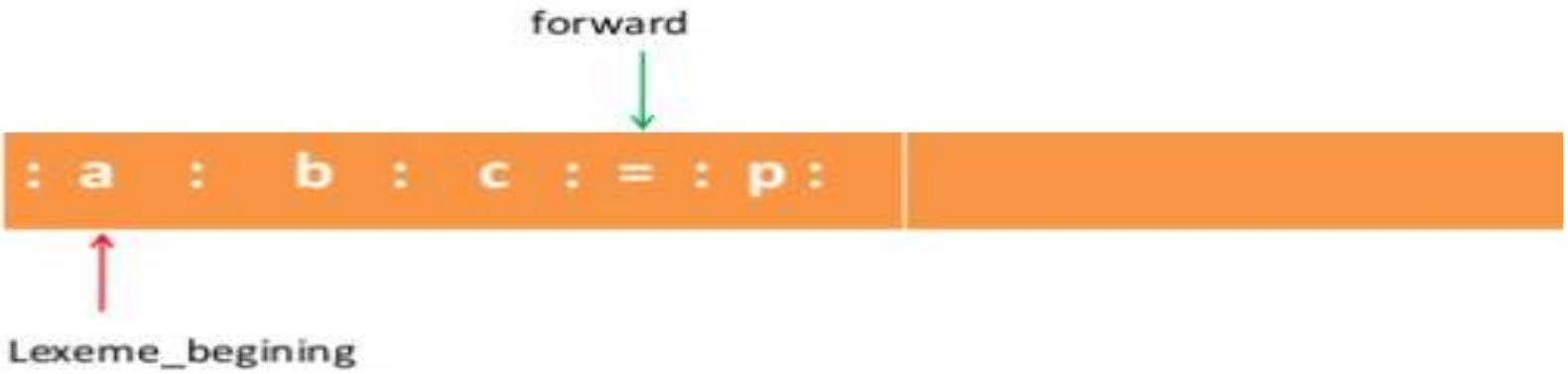
For Eg: consider

abc = pqr * xyz;

Buffer pairs

For Eg: consider

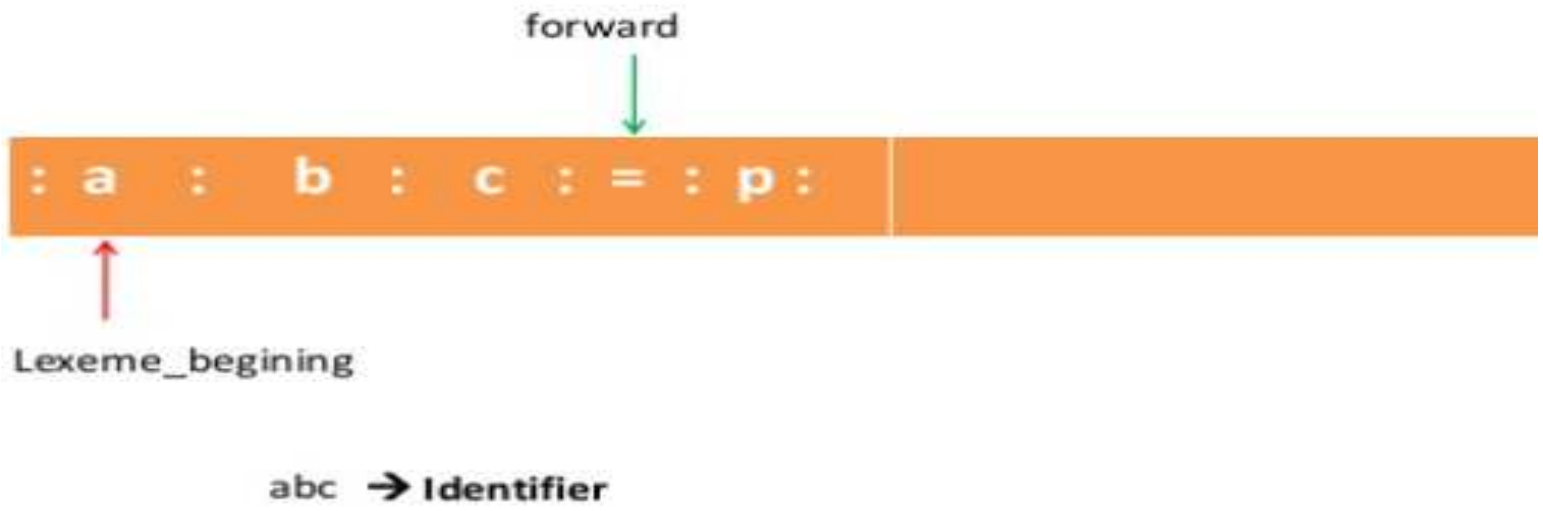
abc = pqr * xyz;



Buffer pairs

For Eg: consider

abc = pqr * xyz;

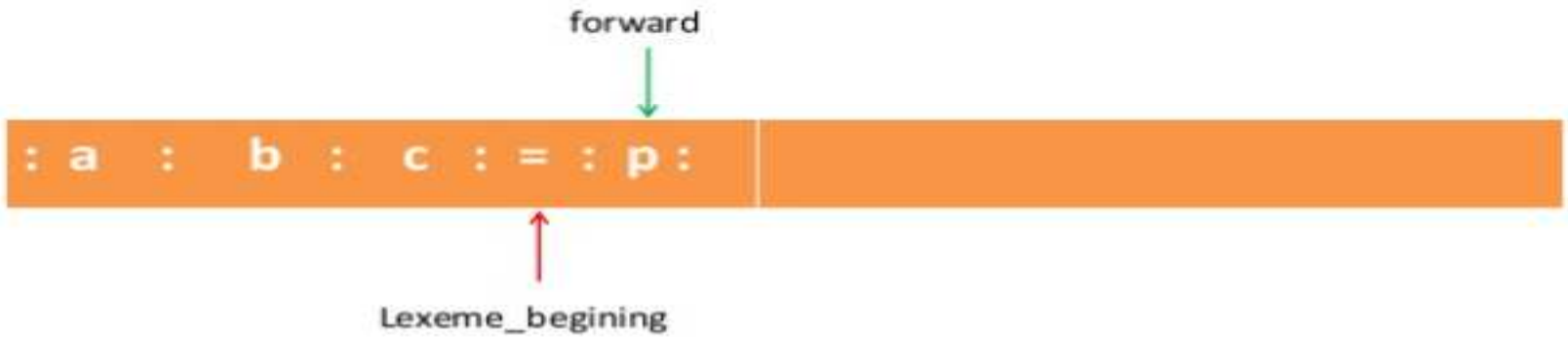


abc → **Identifier**

Buffer pairs

For Eg: consider

abc = pqr * xyz;

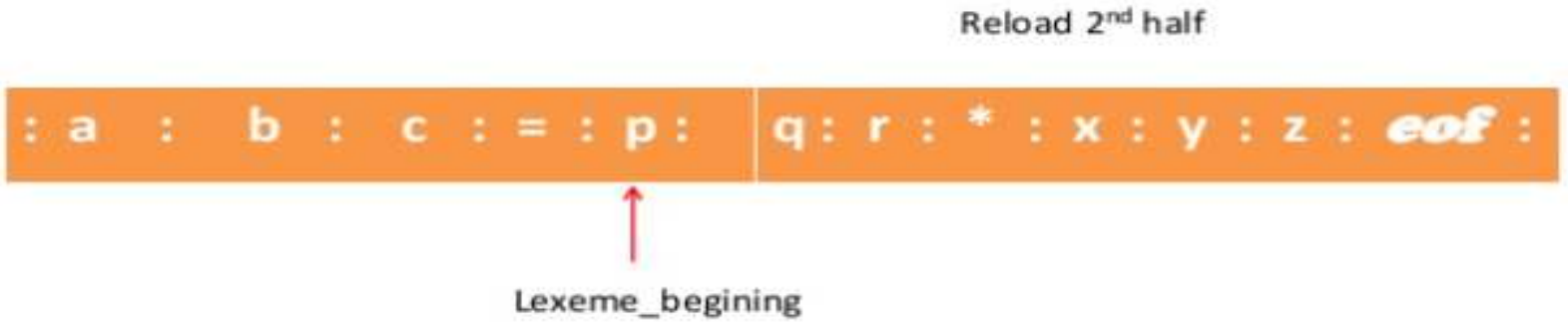


=

Buffer pairs

For Eg: consider

abc = pqr * xyz;



Sentinels


```
Switch (*forward++) {
   case eof:
        if (forward is at end of first buffer) {
              reload second buffer;
              forward = beginning of second
   buffer;
        }
```

```
else if {forward is at end of second buffer) {
            reload first buffer;
            forward = beginning of first buffer;
      }
      else /* eof within a buffer marks the end of
   input */
            terminate lexical analysis;
      break;
   cases for the other characters;
```

Lexical errors

- Some errors are out of power of lexical analyzer to recognize:
  - fi (a == f(x)) ...
- However it may be able to recognize errors like:
  - d = 2r
- Such errors are recognized when no pattern for tokens matches a character sequence

Error recovery

- Panic mode: successive characters are ignored until we reach to a well formed token
- Delete one character from the remaining input
- Insert a missing character into the remaining input
- Replace a character by another character
- Transpose two adjacent characters

Summary

- Lexical anlyser
- Input buffering
- Token
- Lexeme
- Pattern

Check your understanding?

1. Tokenize the following expression

   a + b / 2.6

2. What is lexeme?
3. What is the advantage of using input buffering scheme?