Lexical Analysis

Introduction

- function of the lexical analyzer
  - read the source program, one character at a time
  - group them into tokens
- We need a method to describe the possible tokens that can appear: (RE)
- We need some mechanism to recognize these tokens in the input stream (transition diagram and finite automata)
- we need to perform actions as the tokens are recognized

Role of the lexical analyzer

- Two approach for implementing Lexical Analyzer

1. Lexical analyzer as a single pass
2. Combining Lexical analyzer and parser in s a single pass

- **Lexical analyzer as a single pass**
  - lexical analyzer could be a separate pass by placing its output in an intermediate file
  - the parser would then take its input

Role of the lexical analyzer

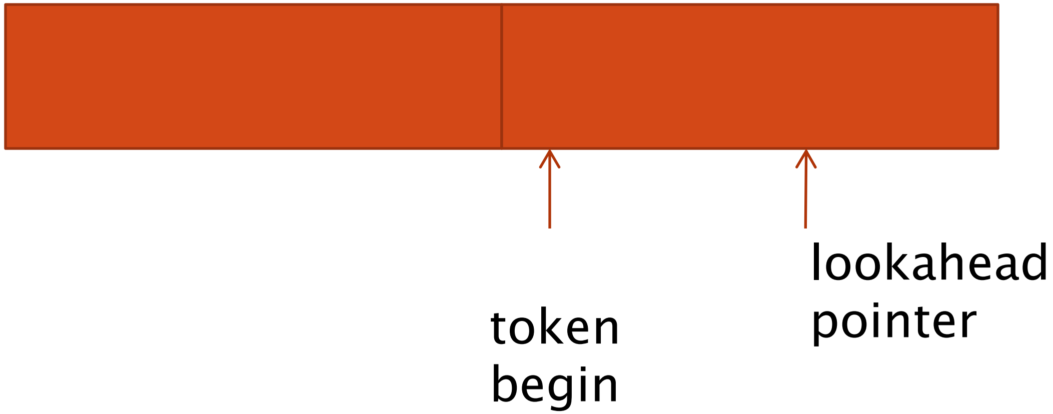
- Combining Lexical analyzer and parser in s a single pass
  - lexical analyzer and the parser are together in the same pass
  - the lexical analyzer acts as a subroutine or co-routine which is called by the parser whenever it needs a new token
  - the lexical analyzer returns a representation of the token it has found
  - this representation is
    - an **integer code** if the token is a simple construct
    - or a **pair** consisting of an **integer code** and a **pointer** to a table if it is an identifier or a constant

Role of the lexical analyzer

- Other functions
  - keeping track of line numbers
  - produce o/p listing
  - stripping out white space
  - deleting comments

Input Buffering

- During scanning, many characters beyond the next token may be examined before the next token can be determined
- It is desirable for the lexical analyzer to read its input from an input buffer



Input Buffering

- example from Pl/I
- DECLARE(arg1, arg2,…argn)
- If the lookahead pointer travels beyond the buffer half in which it began , the other half must be loaded with the next characters from the source file
- since the buffer is limited in size, there is a constraint of how much to lookahead before the next token is discovered.

↑

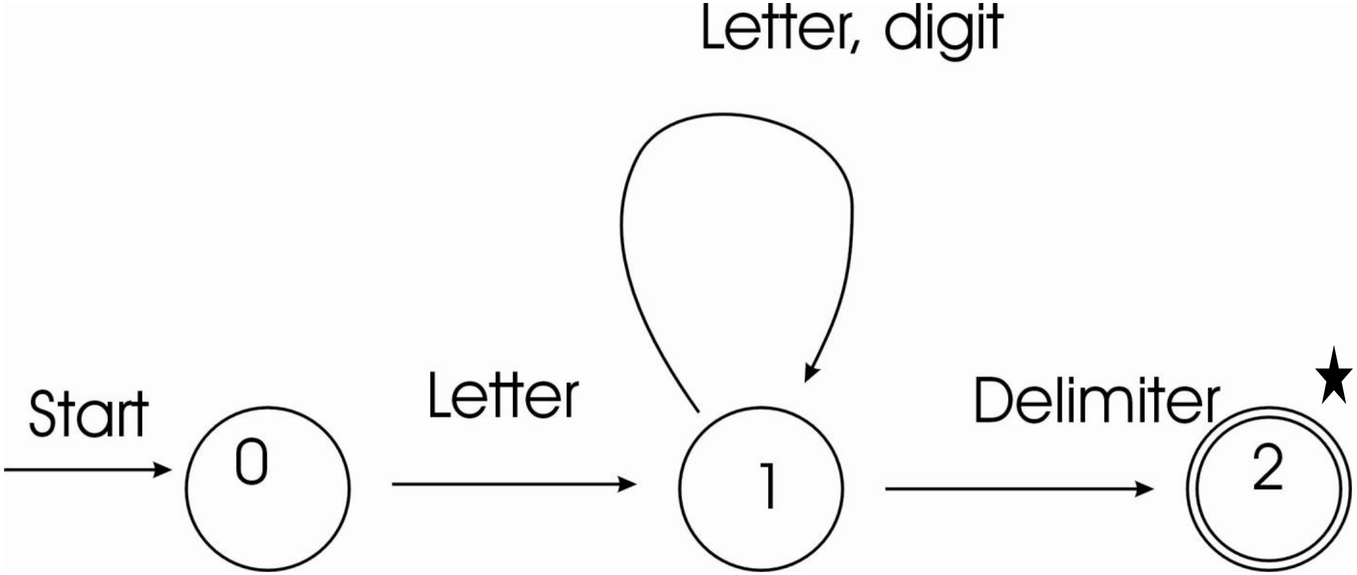
Preliminary scanning

- Performed with the help of an extra buffer
- delete comments
- collapse several strings of blanks to one blank
- a count of lines

A simple approach to the design of a Lexical Analyzer

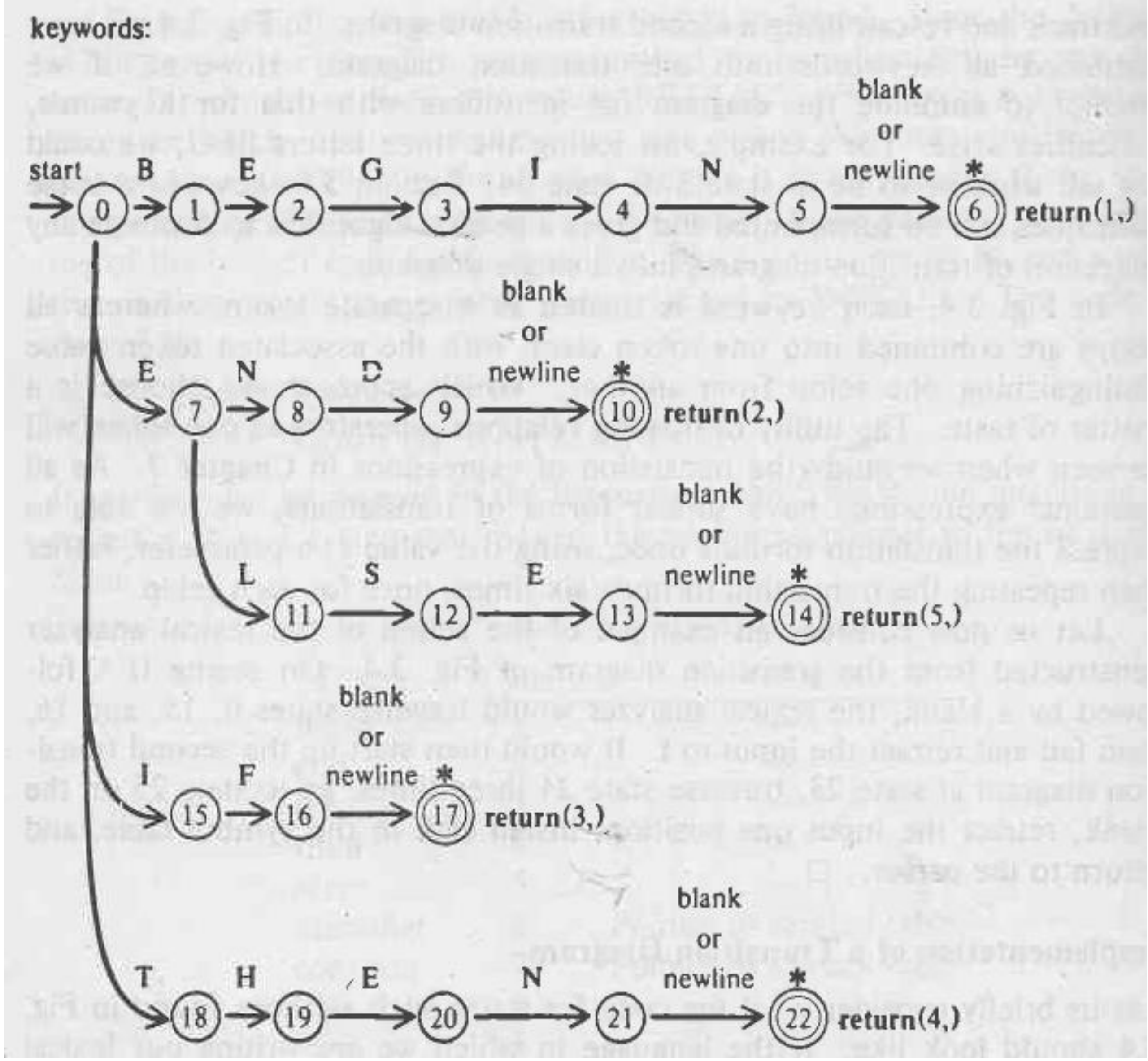
- Transition diagram are used to describe the behavior of a program
- the *circles* are *states* and the states are connected by *edges*
- Example of one transition diagram : identifier
- Turning a collection of transition diagrams into a program
- construct a segment of code for each state

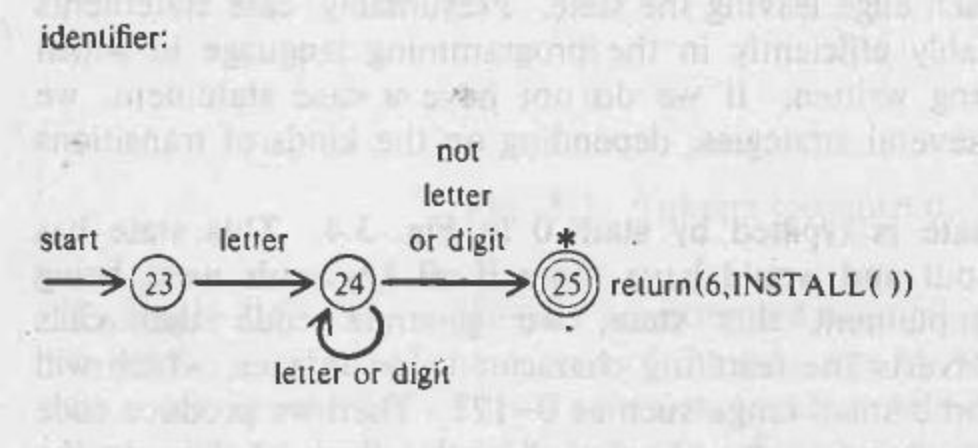
Transition diagram for identifier

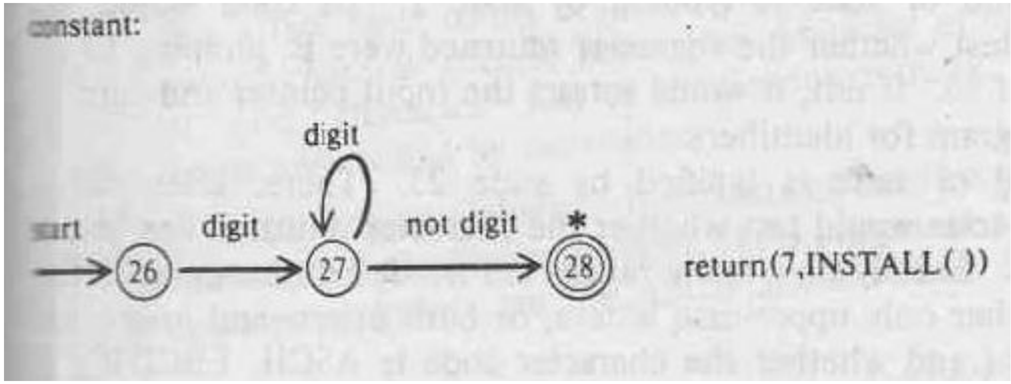


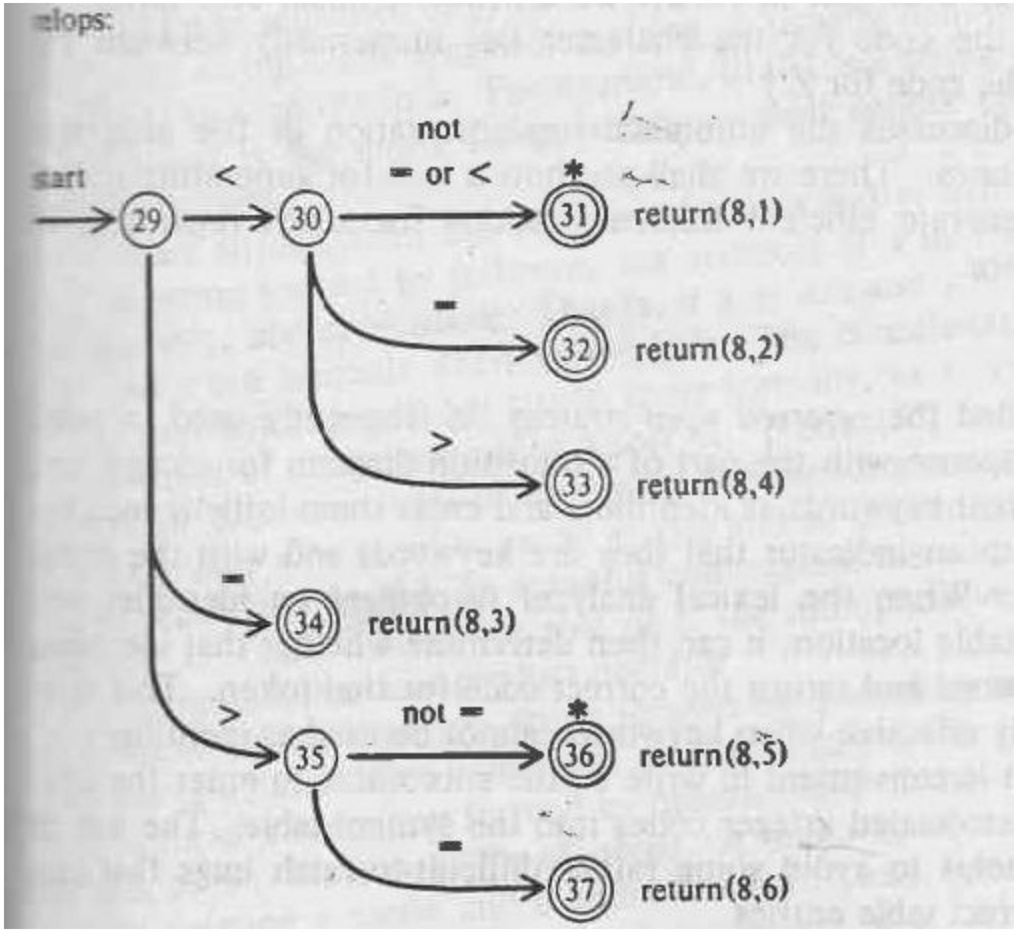
Lexical Analyzer: Algorithm

- use GETCHAR (which returns the next character and advancing the lookahead pointer)
- determine which edge, if any, out of the state
  - If such an edge is found control transfers to that state
  - If no such edge is found , we have failed to find the token. The *lookahead* pointer must be retracted to where the *begin pointer* is and another token must be searched


keywords:
start
B
E
G
I
N
blank or newline
*
0
1
2
3
4
5
6
return(1,)
E
N
D
blank or newline
*
7
8
9
10
return(2,)
L
S
E
blank or newline
*
11
12
13
14
return(5,)
I
F
blank or newline
*
15
16
17
return(3,)
T
H
E
N
blank or newline
*
18
19
20
21
22
return(4,)


identifier:
start
23
letter
24
not letter or digit
*
25
return(6,INSTALL( ))
letter or digit


constant:
start
26
digit
27
digit
not digit
*
28
return(7,INSTALL( ))


elops:
sart
<
not
= or <
*
29
30
31
return(8,1)
=
32
return(8,2)
>
33
return(8,4)
=
34
return(8,3)
>
not =
*
35
36
return(8,5)
=
37
return(8,6)

Lexical Analyzer: Algorithm

- If all transition diagrams have been tried without success , a lexical error has been detected
- Error correction routine must be called

Lexical Analyzer: Algorithm

```
state 0: c=GETCHAR()
if LETTER(c) then goto state 1
else FAIL()

state 1: c=GETCHAR()
 if LETTER(c) or DIGIT(c) then goto state 1
else if DELIMETER (c) then goto state 2
else FAIL()

state 2:RETRACT();
return (id, INSTALL())
```

Implementation of a Transition Diagram

- If the language in which the lexical analyzer is being written has a case statement
  - A case can be use for each state

Reserved words

- If we do away with transition diagram for recognizing keywords then
- Keywords are treated as identifiers and are initially entered in the symbol table
  - Each keyword also have an indicator and a code
- When the lexical analyzer recognizes an identifier and finds its symbol table entry location
  - it can be determined that it is a keyword
- This technique is very useful for those languages which does not allow a keyword to be use as an identifer