3. Transition Diagrams to Code

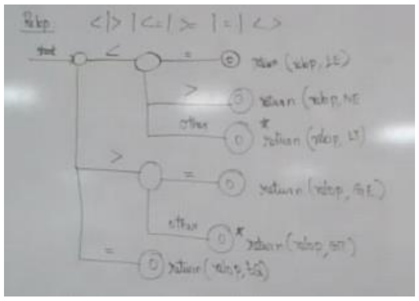
Wednesday, February 17, 2021 10:16 AM

1. Specify the tokens
2. Recognize the tokes (done by a lexical analyzer)

Regular expression->transition diagram -> code

Transition diagram consists of:
1. Nodes (represent states)
2. Edges (represent transitions)

Start state
Final state - return token when you reach accepting state
- * --> retract forward pointer by 1 position



To convert the transition diagram to code

1. Number the states (starting 0)
2. Work with switch case

OR Create a huge transition table for one big code

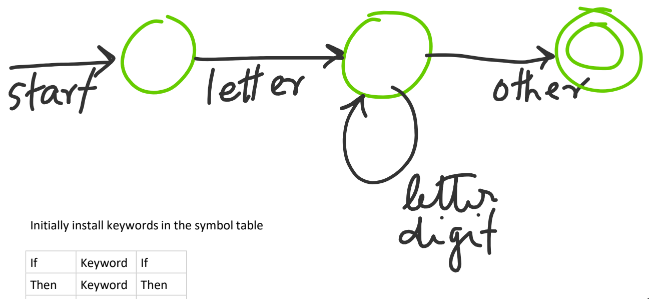
Lex takes input specifications
Lex.yy.c: regular expression to transition diagram to code

Conflict resolution in Lex
If a pattern matches more than 1 lexeme

Matches if and identifier
<= matches < or <=

Always match longest prefix
If longest matching prefix, use rule that appears first in the set

identifier & keyword



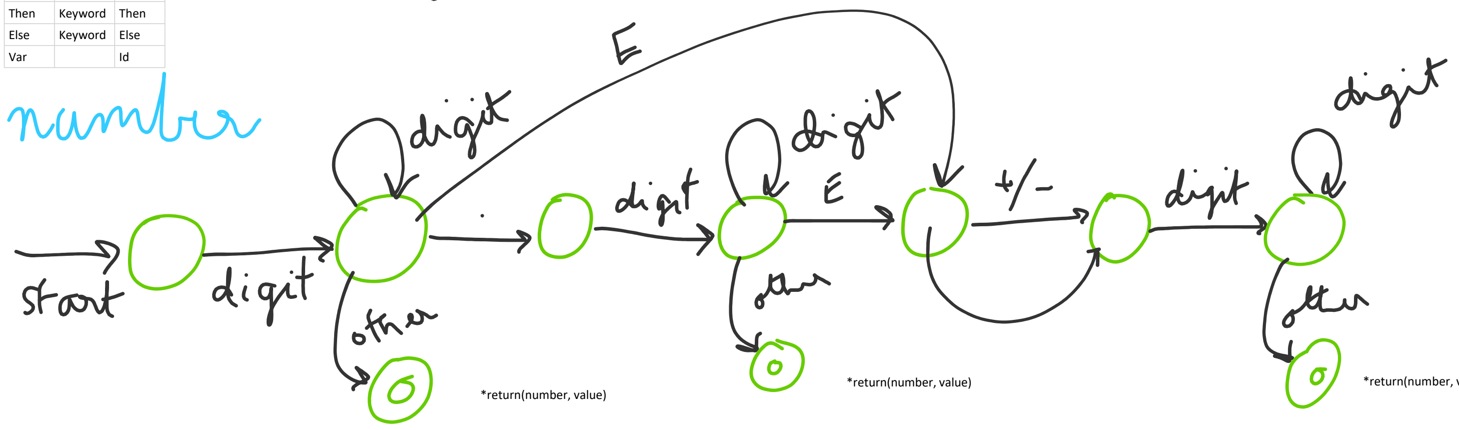
*return (ID, pointer to ST)

*return (getToken(), installID())
GetToken():
Take Lexeme and find corresponding token name
InstallID():
Lexeme installed if not present in the symbol table

Initially install keywords in the symbol table

If	Keyword	If
Then	Keyword	Then
Else	Keyword	Else
Var		Id

number



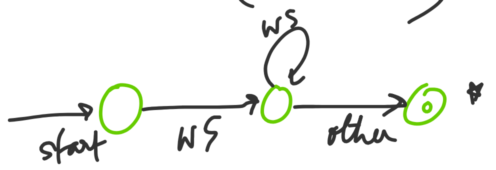
*return(number, value)

*return(number, value)

*return(number, value)

delimiter → space / tab / newline

ws → (delimiter)$^+$



start
ws
ws
other
*