Government College University Lahore

Compiler Construction

Assignment_01

Name: HUSNAIN

Roll No: 0112-BSCS-20

Section: A1

Submitted To: Sir Atif Ishaq

Introduction:

In this document I have explained every step of **Lexical Analyzer**. Lexical Analyzer is first phase of compiler in which it takes stream of string as input, convert it into tokens and pass to syntax analyzer. Main task of Lexical Analyzer is read the input and identify each lexeme and convert them into relevant tokens. How it works? How it identifies each lexeme? How it separates each lexeme token by token? All these questions are

solved in this document. To solve these problems, we must understand how patterns match.

Step_1:

Language: I have Selected c++ high level language to solve this problem.

Description of Code:

Class T0KEN: Token contains two values first is token name and second is attribute value (name, attribute value). So, I have used OOP concepts and make a class of name TOKEN and its attributes named string TName , string AtValue.

Display (): This function is using for display attribute name and value along with specific token and write output on the text file (Output.txt) in Class TOKEN.

Class BUFFER: This class is for reading input from the input file and store it on the string variable (Str). This class contains an array of type string which contains 21 keywords ("int" , "float" , "double" , "short" , "char" ,

"bool" , "if" , "else" , "while" , "for" , "break" , "continue" , "class" ,

"long" , "string" , "const" , "static" , "goto" , "do" , "return" , "default"). It also contains lexeme begin and lexeme forward pointers.

Begin: This is working as pointer which point beginning position of each lexeme.

Forward: This is working as pointer which point end of the lexeme when whole lexeme is read.

readFile (): This function is used for reading input from text file line by line.

getChar (): This function is used for getting input character by character from Buffer.

fail (): This function is used when character is not matched to one automata and it makes forward pointer equal to begin pointer and return state number of next automata.

retract (): Used to decrement value of forward pointer when forward read the extra character which is not the part of pattern.

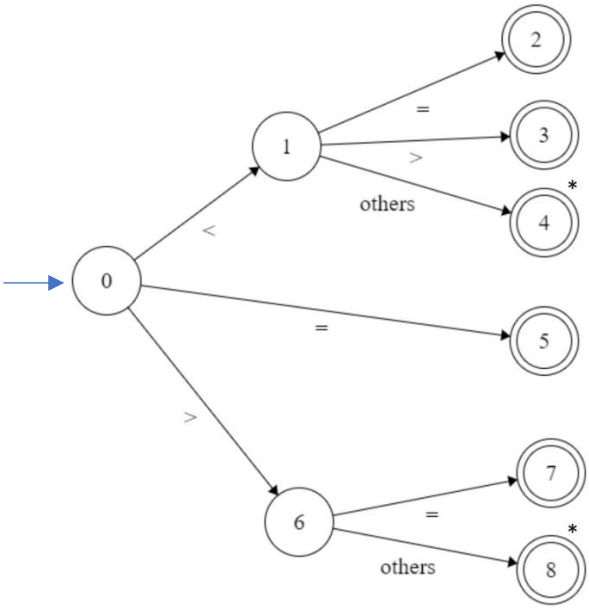
tokSuccess (): When the token is successfully created then there is need to read the next lexeme and begin pointer is shifted to place where forward is pointing.

chkKeyword(string Str): Used to check whether lexeme is identifier or keyword.

stop (): Used to finish the process when whole String is tokenized.

Automata

Relational Operator Machine



Relevant c++ Code

```
case 0:
B.setTGc(0);
C = B.getChar();
if(C == '<')State=1;
else if(C =='=')State=5;
else if(C == '>')State=6;
else State = B.fail();
break;
case 1:
C = B.getChar();
```

```
if(C == '=')State=2;
else if(C == '>')State=3;
else State=4;
break;
case 2:
RetToken.setTName("RelOp");
RetToken.setAtValue("LE");
RetToken.Display();
B.tokSuccess();
cout<<B.stop();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;

case 3:
RetToken.setTName("RelOp");
RetToken.setAtValue("NE");
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
case 4:
B.retract();
RetToken.setTName("RelOp");
RetToken.setAtValue("LT");
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;

case 5:
RetToken.setTName("RelOp");
RetToken.setAtValue("EQ");
RetToken.Display();
```

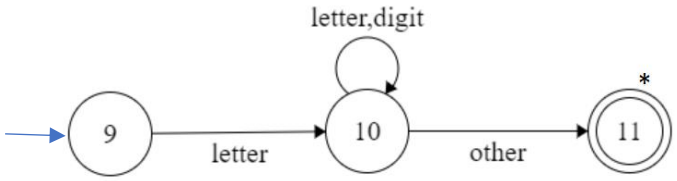
```
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
case 6:
C = B.getChar();
if(C == '=')State=7;
else State=8;
break;
case 7:
RetToken.setTName("RelOp");
RetToken.setAtValue("GE");
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
case 8:
B.retract();
RetToken.setTName("RelOp");
RetToken.setAtValue("GT");
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
```

Explanation of Relational Operator Machine:

Each case in switch is the state of machine. In this machine state 0 is starting state of machine. If control is at 0 state and read '<' operator it will move on state 1 and at state 1 if it read '=' then it will move on state 2 and return the name and value of token (RelOP,LE). If it reads '>' at state 1 then it will move on state 3 and return the token name and value (RelOP,NE). If it reads any other character, then it will

move on state 4 and retract and return the name and value (RelOP,LT). If it reads '=' at state 0 then it will return name and value (RelOP,EQ). If it reads '>' at state 0 it will move on state 6 and at state 6 if it reads '=' then it will move on state 7 and it will return name and value of token (RelOP,GE). If it reads any other character, then it will move on state 8 and retract and return the name and value of token (RelOP,GT). If no operator in lexeme it will move on other automata.

Identifiers Machine



Relevant c++ Code

```
case 9:

Str = "";

C = B.getChar();

Temp = int(C);

if((Temp>=65 && Temp<=90) ||
(Temp>=97 && Temp<=122))

{

        Str = C;

        State=10;

}

else State = B.fail();

break;


case 10:

C = B.getChar();

Temp = int(C);

if((Temp>=65 && Temp<=90) ||
(Temp>=97 && Temp<=122) || (Temp>=48
&& Temp<=57))

{

        Str += C;

        State=10;

}

else State = 11;

break;


case 11:

if(B.chkKeyword(Str) == true)

{

        State = 12;
```

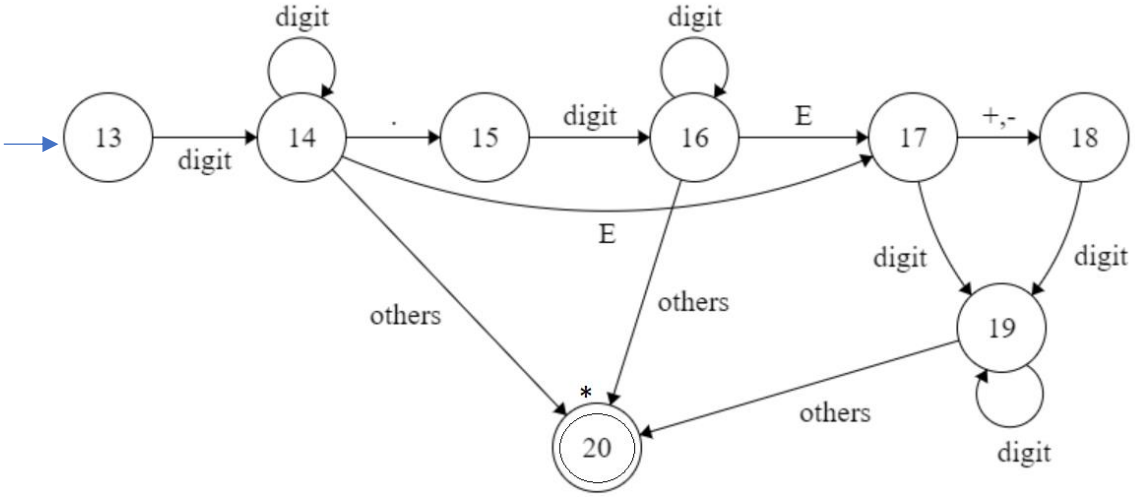
```
}
else
{
        B.retract();
        RetToken.setTName("ID");
        RetToken.setAtValue(Str);
        RetToken.Display();
        B.tokSuccess();
        if(B.stop())
        {
                State=0;
        }
        else
        {
                Test = false;
        }
}
break;

case 12:
B.retract();
RetToken.setTName("Keyword");
RetToken.setAtValue(Str);
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
```

Explanation of Identifiers Machine:

Starting state of Identifier Machine is state 9.

- ➔ If it reads any letter ((a-z) or (A-Z)) it will move on state 10.
- ➔ At state 10 there is loop of letter and digit ((letter, digit)*). If it reads any letter or digit it will remain on state 10 until other character is read.
- ➔ If it reads any other character at state 10 then it will move on state 11 and then it will check whether it is keyword or not if yes then it will move on state 12 and return the name and value of token (KeyWord, val) if not then it will return the name and value (Id,val). If at starting state there is no letter then it will move on next automata.

Numbers Machine



Relevant c++ Code

```
case 13:
Str = "";
C = B.getChar();
Str = C;
Temp = int(C);
if(Temp>=48 && Temp<=57)State = 14;
else if(C == '.')State = 15;
else State  = B.fail();
break;


case 14:
C = B.getChar();
Temp = int(C);
if(Temp>=48 && Temp<=57){Str += C;State
= 14;}
else if(C == '.'){Str += C;State = 15;}
else if(C == 'E'){Str += C;State = 17;}
else State  = 20;
break;


case 15:
C = B.getChar();
Temp = int(C);
if(Temp>=48 && Temp<=57){Str += C;State
= 16;}
break;


case 16:
C = B.getChar();
Temp = int(C);
```

```
if(Temp>=48 && Temp<=57){Str += C;State = 16;}

else if(C == 'E'){Str += C;State = 17;}

else State = 20;

break;


case 17:

C = B.getChar();

Temp = int(C);

if(C=='-' || C=='+'){Str += C;State = 18;}

else if(Temp>=48 && Temp<=57){Str += C;State = 19;}

break;


case 18:

C = B.getChar();

Temp = int(C);

if(Temp>=48 && Temp<=57){Str += C;State = 19;}

break;


case 19:

C = B.getChar();

Temp = int(C);

if(Temp>=48 && Temp<=57){Str += C;State = 19;}

else State = 20;

break;


case 20:

B.retract();

RetToken.setTName("Number");

RetToken.setAtValue(Str);

RetToken.Display();

B.tokSuccess();

if(B.stop())

{

        State=0;

}

else

{

        Test = false;

}

break;
```

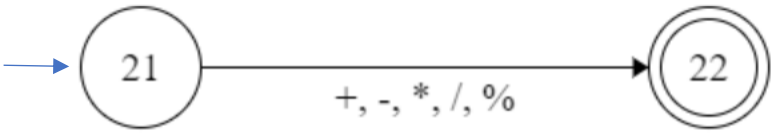
Explanation of Digits Machine:

Starting State of Digits machine is state 13.

- ➔ If it reads digit (0-9) at state 13 it will move on state 14.
- ➔ At state 14 when it reads digit it will remain on 14 if reads '.' (dot) it will move on state 15 if it reads E it will move on state 17 other then digit or '.' or E It will move on state 20 and retract and return the name and value of token (Number, val).
- ➔ At state 15 if it reads digit it will move on state 16.

- ➔ At state 16 if it reads digit it will remain at state 16. If it reads E it will move in state 17. If any other character, it will move on state 20. It retracts and return name and value of token (Number,val).
- ➔ At state 17 if it reads '+' or '–' it will move on state 18 and if digit it will move on state 19.
- ➔ At state 18 if it reads digit it will move on 19.
- ➔ At state 19 when it reads digit it will remain at 19 otherwise it will move on state 20 and then retract and return name and value of token (Number, val).

Arithmetic Operator Machine



Relevant c++ Code

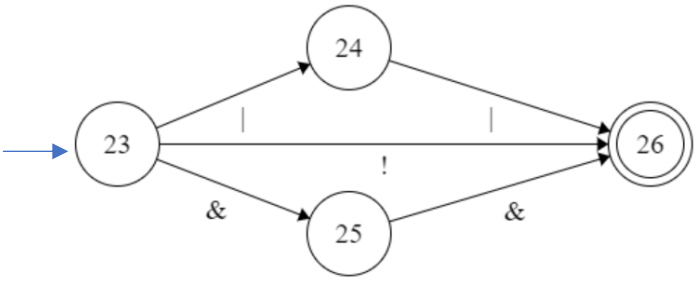
```
case 21:
Str = "";
C = B.getChar();
if(C == '+' || C == '-' || C == '*' || C == '/' || C == '%'){Str = C;State = 22;}
else State = B.fail();
break;

case 22:
RetToken.setTName("ArithOp");
RetToken.setAtValue(Str);
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
```

Explanation of Arithmetic Operator Machine:

Starting state of this machine is state 21.

- ➔ If it reads any arithmetic operator (+ , - , * , / , %) it will move on state 22 and return the name and value of token (ArthOp,val). If it reads any other character it will move on next automata.

Logical Operator Machine



Relevant c++ Code

```
case 23:
Str = "";
C = B.getChar();
if( C == '|'){Str = C;State = 24;}
else if(C == '&'){Str = C;State = 25;}
else if(C == '!'){Str = C;State = 26;}
else State = B.fail();
break;
```

```
case 24:
C = B.getChar();
if(C == '|' && !(Str.compare("|"))){Str += C;State = 26;}
else {B.retract2();State = 34;}
break;
```

```
case 25:
C = B.getChar();
if(C == '&' && !(Str.compare("&"))){Str += C;State = 26;}
else {B.retract2();State = 34;}
break;
```

```
case 26:
RetToken.setTName("LogicOp");
RetToken.setAtValue(Str);
RetToken.Display();
B.tokSuccess();
if(B.stop())
```

```
{
        State=0;
}
else
```

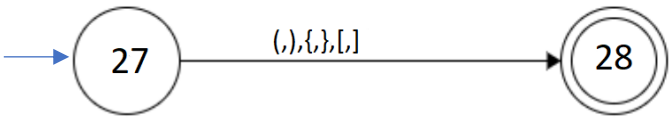
```
{
        Test = false;
}
```

Explanation of Logical Operator Machine:

Starting state of Logical Operator is state 23.

- ➔ If it reads OR operator at state 23 it will move on state 24, if it reads AND operator it will move on state 25, and if it reads NOT operator it will move on final state 26 and return the name and value of Token.
- ➔ At state 24 when it reads OR operator it will move on final state 26 and return the name and value of Token. If any other, then it will show that previous OR is not part of language.
- ➔ At state 25 when it reads AND operator it will move on final state 26 and return the name and value of Token. If any other, then it will show that previous AND is not part of language.

Brackets Machine



Relevant c++ Code

```
case 27:
Str = "";
C = B.getChar();
if(C == '(' || C == ')' || C == '{' || C == '}' || C == '[' || C == ']') {Str = C;State = 28;}
else State = B.fail();
break;
```

```
case 28:
RetToken.setTName("BracKet");
RetToken.setAtValue(Str);
RetToken.Display();
B.tokSuccess();
```

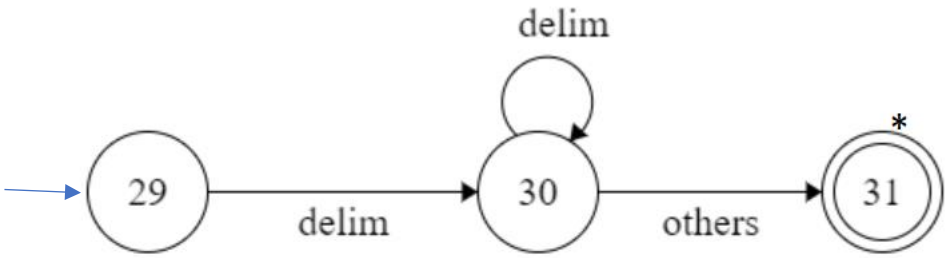
```
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
```

Explanation of Brackets Machine:

Starting state of this machine is state 27.

- ➔ If it reads any bracket ((,),{,},[,]) it will move on state 28 and return the name and value of token (bracket,val). If it reads any other character it will move on next automata.

Delimiters Machine



Relevant c++ Code

```
case 29:
Str = "";
C = B.getChar();
Temp = int(C);
if(Temp == 9){Str = "VERTICAL TAB";State = 30;}
else if(Temp == 10){Str += "LINE FEED";State = 30;}
else if(Temp == 11){Str += "HORIZONTAL TAB";State = 30;}
else if(Temp == 32){Str += "SPACE";State = 30;}
```

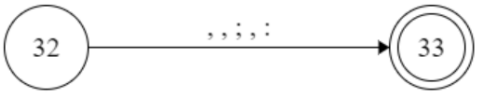
```
else State = B.fail();
break;

case 30:
C = B.getChar();
Temp = int(C);
if(Temp == 9){Str += "VERTICAL TAB";State = 30;}
else if(Temp == 10){Str += "LINE FEED";State = 30;}
else if(Temp == 11){Str += "HORIZONTAL TAB";State = 30;}
else if(Temp == 32){Str += "SPACE";State = 30;}
else State = 31;
break;

case 31:
B.retract();
RetToken.setTName("Delim");
RetToken.setAtValue(Str);
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
```

Explanation of Delimiters Machine:

Starting state of Delimiters Machine is state 29.

- ➔ If it reads any delimiter (space, tab, line feed …) it will move on state 30.
- ➔ At state 30 there is loop of delimiter ((delimiter)*). If it reads any delimiter it will remain on state 30 until other character is read.
- ➔ If it reads any other character at state 30 then it will move on state 31 and return the name and value of token (Delim, val). If at starting state there is no delimiter then it will move on next automata

Punctuators Machine



Relevant c++ Code

```
case 32:
Str = "";
C = B.getChar();
if(C == ';' || C == ',' || C == ':'){Str = C;State = 33;}
else State = B.fail();
break;

case 33:
RetToken.setTName("Puntuator");
RetToken.setAtValue(Str);
RetToken.Display();
B.tokSuccess();
if(B.stop())
{
        State=0;
}
else
{
        Test = false;
}
break;
```

Explanation of Punctuators Machine:

Starting state of this machine is state 32.

➔ If it reads any punctuator (, , ; , :) it will move on state 33 and return the name and value of token (punctuator, val). If it reads any other character it will move on next automata.

NOT in Language:

```
                                        if(B.stop())
case 34:                                {
Str = "";                               State=0;
C = B.getChar();                        }
Str = C;                                else
RetToken.setTName("NotInLang");         {
RetToken.setAtValue(Str);               Test = false;
RetToken.Display();                     }
B.tokSuccess();                         break;
```

Case 34: This is the case in which all those characters that are not the part of language are identified and returned.