19CSE401 - Compiler Design

Lab Sheet 4

S Abhishek

AM.EN.U4CSE19147

A. Try the following JLex Program to recognize a 5 letter word which starts with P/p and ends with T/t.

1. Create a file **Ylex** and type in the following code. **Don't COPY and PASTE, it will result in error.**

```
import java.io.*;
class Main{
      public static void main(String args[]) throws IOException{
      Ylex lex=new Ylex(System.in);
     Token token=lex.yylex();
      while(token.text != null ) {
        System.out.println("\t" + token.text);
        token= lex.yylex();
       }
   }
}
class Token{
  String text;
  Token(String t){text = t;}
}

%%
digit = [0-9]
letter = [a-zA-Z]
special = [!@#$%^&*()_+]
whitespace = [ \t\n]

%type Token

%eofval{
   return new Token(null);
%eofval}
%%
[Pp]{letter}{letter}{letter}[Tt] { return new Token(yytext() ) ;  }
{whitespace}+ {/*Skip white spaces*/}
.                  { }
```

```
Paint
perst
Palet
Payout
petit
picot
pilot
Packet
Pipit
peart
pivot
adopt
pratt
Point
poult
pinot
adust
Pinapple
```

```
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ jflex 1.jlex
Reading "1.jlex"

Warning : Macro "special" has been declared but never used.
Constructing NFA : 18 states in NFA
Converting NFA to DFA :
........
10 states before minimization, 8 states in minimized DFA
Writing code to "Yylex.java"
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ javac Yylex.java
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ java Main
Match: Paint
Match: perst
Match: Palet
Match: petit
Match: picot
Match: pilot
Match: Pipit
Match: peart
Match: pivot
Match: pratt
Match: Point
Match: poult
Match: pinot
```

B. Try the following JLex Program to recognize an identifier which starts with a letter.

```
import java.io.*;
class Main {
public static void main(String args[]) throws IOException {
Yylex lex = new Yylex(System.in);
Token token = lex.yylex();
while(token.text != null ) {
token = lex.yylex();
}
}
}
class Token{
String text;
Token(String t) { text = t; }
}
%%
%public
%class Yylex
%type void
digit = [0-9]
letter = [a-zA-Z]
special = [!@#$%^&*()_+]
whitespace = [ \t\n]
%type Token
%eofval{
return new Token(null);
%eofval}
%%
{letter}({letter}|{digit})*        { System.out.print("<A valid Identifier,"+yytext()+">");}
{whitespace}+                      { /*Skip white spaces*/}
```

```
abhi123
_abhi123
hello_123
12345
a123bhi
a3x3k
n1sh4
100034
compiler_lab
system_design
computer
0123
_1235
```

```
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ jflex 2.jlex
Reading "2.jlex"

Warning : Macro "special" has been declared but never used.
Constructing NFA : 16 states in NFA
Converting NFA to DFA :
.....
7 states before minimization, 5 states in minimized DFA
Old file "Yylex.java" saved as "Yylex.java~"
Writing code to "Yylex.java"
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ javac Yylex.java
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ java Main
<A valid Identifier,abhi123>
<An invalid Identifier,_abhi123>
<An invalid Identifier,hello_123>
<An invalid Identifier,12345>
<A valid Identifier,a123bhi>
<A valid Identifier,a3x3k>
<A valid Identifier,n1sh4>
<An invalid Identifier,100034>
<An invalid Identifier,compiler_lab>
<An invalid Identifier,system_design>
<A valid Identifier,computer>
<An invalid Identifier,0123>
<An invalid Identifier,_1235>
```

1. Write JLex code for the following and output the token of the form **<token_name, lexem>**
   i. To recognize any Java identifier (a sequence of one or more letters and/or digits and/or underscores, starting with a letter or underscore. Token Name is **ID**
   ii. To recognize any Java identifier that does not end with an underscore. Token Name is ID
   iii. To recognize the keyword "if" in addition to identifiers. (Place the rule of "if" above the rule of identifier.) Token Name is **IF**
   iv. Move the "if" rule below that of identifier rule and check the effect on your input. Do you see any difference in the output?
   v. Add the rule for other keywords, **for**, **while**, **do** and all types of parentheses in a similar fashion and try with several inputs to convince yourself of its working.
   vi. To recognize the integer constant. Token Name is **INT_CONST**
   vii. To recognize the floating-point constant. Token Name is **FLOAT_CONST**
   viii. To recognize comments of the type "// xxxx". Token Name **SINGLE_COMMENT**
   ix. Add rule(s) to recognize comments of type /* xxxx */. Token name **MULTI_COMMENT.**

```
int main()
{
    int n = 0, _abhi = -1;
    float a3x3k = 10.0;
    //True if number is less than 0
    if (n < 0)
    {
        _abhi = 1;
    }
    /*False if the number is greater than 0*/
    else
    {
        _abhi = 0;
    }
    return 0;
}
```

```
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ jflex 3.jlex
Reading "3.jlex"
Constructing NFA : 95 states in NFA
Converting NFA to DFA :
.................................................
53 states before minimization, 46 states in minimized DFA
Old file "Yylex.java" saved as "Yylex.java~"
Writing code to "Yylex.java"
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ javac Yylex.java
```

```
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ java Main
<INT> <ID,main><OPEN_PARENTHESIS,(><CLOSE_PARENTHESIS,)>

<OPEN_CURLYBRACKET,{>

    <INT> <ID,n> <SYMBOL> <INT_CONST,0><COMMA,,> <ID,_abhi> <SYMBOL> <SYMBOL><INT_CONST,1><SEMICOLON,;>

    <FLOAT> <ID,a3x3k> <SYMBOL> <FLOAT_CONST,10.0><SEMICOLON,;>

    <SINGLE_COMMENT>

    <IF> <OPEN_PARENTHESIS,(><ID,n> <SYMBOL> <INT_CONST,0><CLOSE_PARENTHESIS,)>

    <OPEN_CURLYBRACKET,{>

        <ID,_abhi> <SYMBOL> <INT_CONST,1><SEMICOLON,;>

    <CLOSE_P_CURLYBRACKET,}>

    <MULTI_COMMENT>

    <ELSE>

    <OPEN_CURLYBRACKET,{>

        <ID,_abhi> <SYMBOL> <INT_CONST,0><SEMICOLON,;>

    <CLOSE_P_CURLYBRACKET,}>

    <RETURN> <INT_CONST,0><SEMICOLON,;>

<CLOSE_P_CURLYBRACKET,}>
```

- If we move the **“If”** rule below the identifier rule then this if rule won’t be considered since the compiler encounters the **“ID”** rule first which has the similar regex expression and constrain declaration and directly proceeds without checking the following rules.
- In the image above **“If”** keyword is considered as an **“IF”** Token itself since the rule for the **“If”** is given above the identifier rule to make the compiler to encounter this **“IF”** rule first and **“ID”** rule next.
- In the image below **“If”** keyword is considered as an **“ID”** since the rule for the **“If”** is given below the identifier rule, thus the compiler encounters the **“ID”** rule first and the **“If”** rule next.

```
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ javac Yylex.java
root at Abhishek in /mnt/h/Compiler Design/Lab/Lab 4
 ○ java Main
<INT> <ID,main><OPEN_PARENTHESIS,(><CLOSE_PARENTHESIS,)>

<OPEN_CURLYBRACKET,{>

    <INT> <ID,n> <SYMBOL> <INT_CONST,0><COMMA,,> <ID,_abhi> <SYMBOL> <SYMBOL><INT_CONST,1><SEMICOLON,;>

    <FLOAT> <ID,a3x3k> <SYMBOL> <FLOAT_CONST,10.0><SEMICOLON,;>

    <SINGLE_COMMENT>

    <ID,if> <OPEN_PARENTHESIS,(><ID,n> <SYMBOL> <INT_CONST,0><CLOSE_PARENTHESIS,)>

    <OPEN_CURLYBRACKET,{>

        <ID,_abhi> <SYMBOL> <INT_CONST,1><SEMICOLON,;>

    <CLOSE_P_CURLYBRACKET,}>

    <MULTI_COMMENT>

    <ELSE>

    <OPEN_CURLYBRACKET,{>

        <ID,_abhi> <SYMBOL> <INT_CONST,0><SEMICOLON,;>

    <CLOSE_P_CURLYBRACKET,}>

    <RETURN> <INT_CONST,0><SEMICOLON,;>

<CLOSE_P_CURLYBRACKET,}>
```

Thankyou!!