

Project Report

On TALcompiler



University Institute of Engineering and Technology
Chhatrapati Shahu Ji Maharaj University, Kanpur

Submitted To:

Dr. Alok Kumar
CSE Department
UIET CSJMU Kanpur

Submitted By:

Abhishek Singh
Roll No.-CSJMA16001390001

Anish Srivastava
Roll No.-CSJMA16001390008

Ayush Chaurasia
Roll No.-CSJMA16001390016

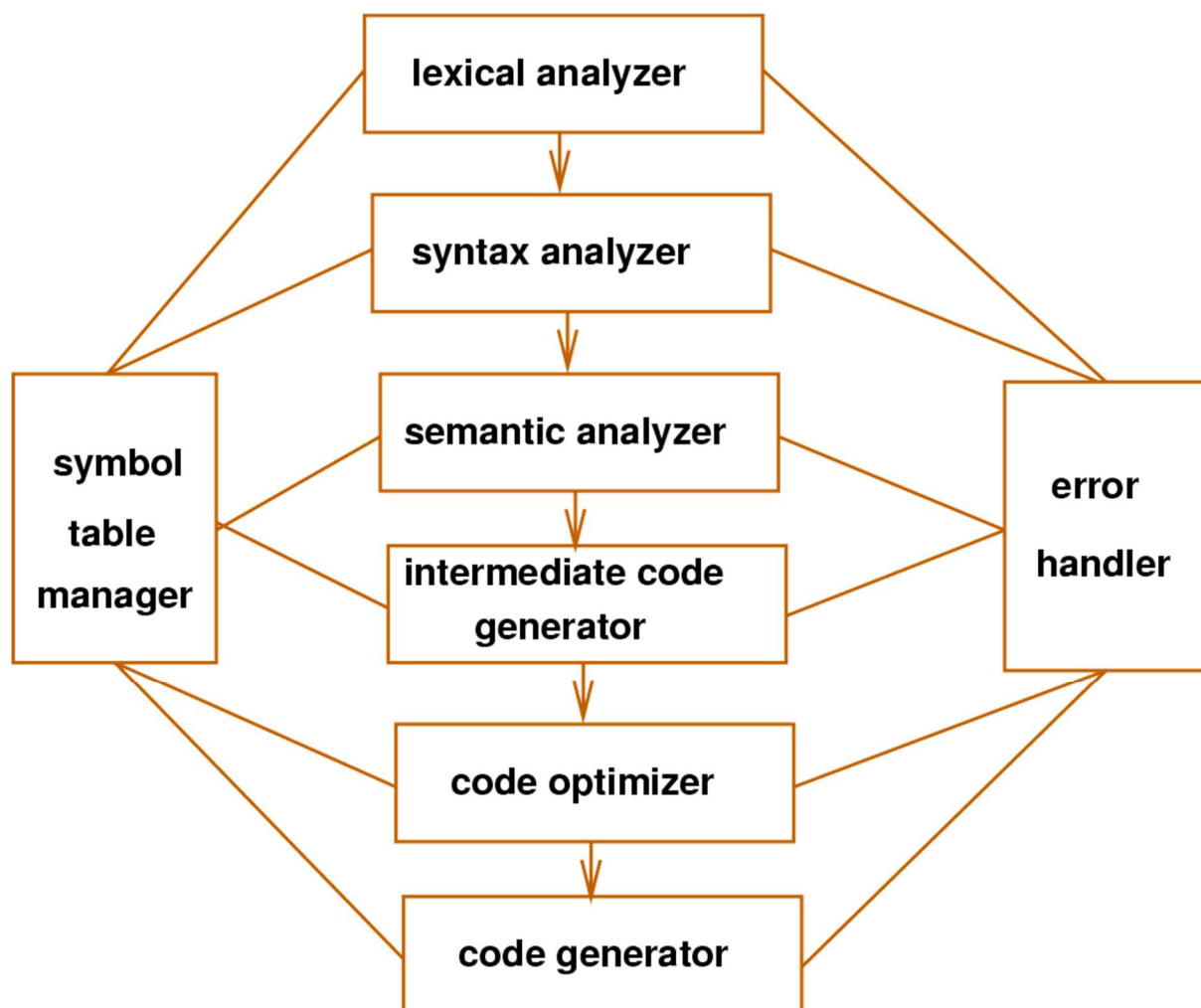
Table of Contents:

1. Introduction
2. Objective.
3. Features of TALcompiler.
4. Some Rules of TA language.
5. Implementation.
6. Execution of Sample Programs

Introduction

“A compiler is a computer program that translates computer code written in one programming language (the source language) into another programming language (the target language).”

Architecture of Compiler:



Objective

- To build a compiler software for a Hypothetical Language which is **Triple A Language(TAL)** in our case.

Features Of TALcompiler

- A program file with “.tal” extension should be given as input.
- Can detect **Lexical Errors** i.e. invalid identifiers, invalid characters, invalid symbols etc.
- Can detect syntax error in programs.
- Generates **tagtrace.log** file containing **Token values and tags** during Lexical Analysis.
- Generate **parselog.log** file containing all information about Syntax Analysis of entered program.
- Generates corresponding **Error Messages** if there is an error in input program.

Some Rules of Our Triple A Language

- Program should start with a pre-defined function named exe().
- Operations supported by the language are
 - Declaration .
 - Mathematical Expressions.
 - show() function.
 - showLine() function.

Implementation

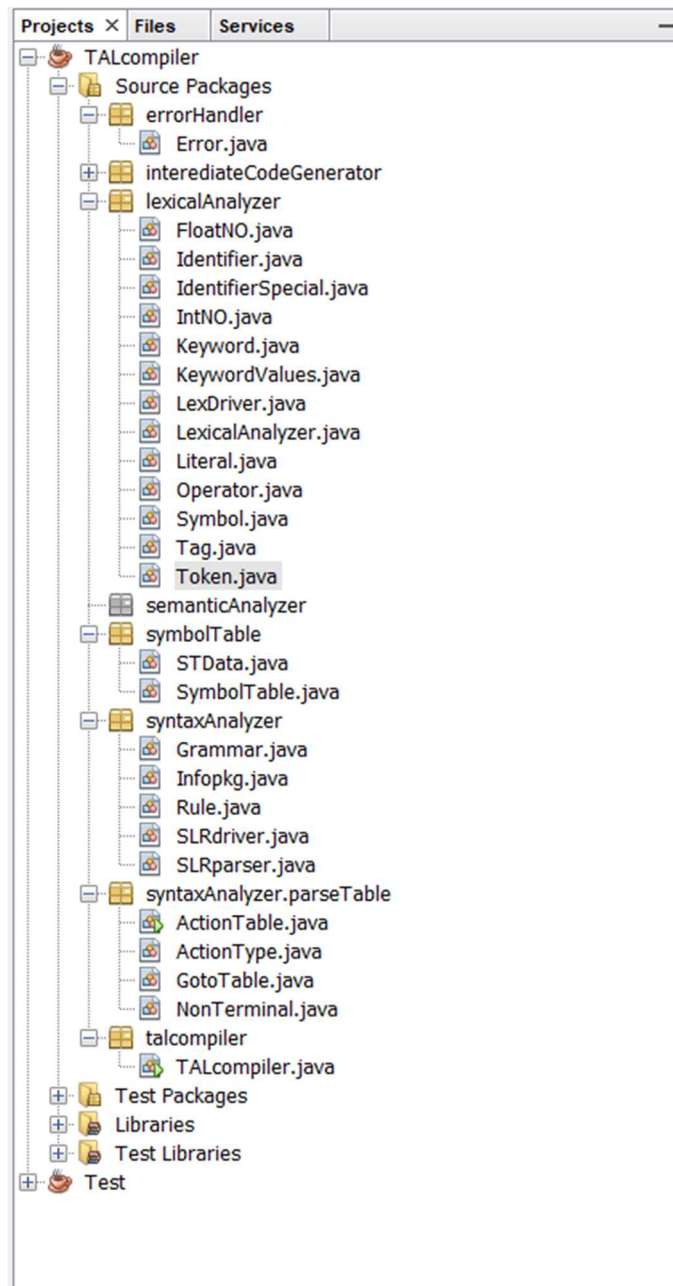


Image : file hierarchy of TALcompiler

Lexical Analyzer

Token class: Parent class of the following classes-

- FloatNO.java
- IntNo.java
- Identifier.java
- IdentifierSpecial.java
- Keyword.java
- Literal.java
- Operator.java
- Sysmbol.java

Code:

```

1  /*
2
3
4  TALcompiler project
5
6  Developed by-
7  * Abhishek Kumar Singh
8  * Anish Shrivastav
9  * Ayush Chaurasia
10
11
12 */
13 package lexicalAnalyzer;
14
15 /**
16  *
17  * @author ayush
18  */
19 public class Token {
20
21     public String value;
22     public String tag;
23
24     public Token() {}
25     public Token(String tag, String value) {
26         this.value = value;
27         this.tag = tag;
28     }
29
30
31 }
32

```

Tag class: Contains Tag values for all types of tokens.

```

1  ...3 lines
4  package lexicalAnalyzer;
5
6  public class Tag {
7      public static final String SYMBL_OPBR = "(";
8      public static final String SYMBL_CLBR = ")";
9      public static final String SYMBL_OPCR = "{";
10     public static final String SYMBL_CLCR = "}";
11     public static final String SYMBL_SEMICOLON = ";";
12     public static final String SYMBL_QUOTE = "\"";
13
14     public static final String OP_EQ = "==";
15     public static final String OP_PLUS = "+";
16     public static final String OP_MINUS = "-";
17     public static final String OP_DEVIDE = "/";
18     public static final String OP_MULTIPLY = "*";
19     public static final String OP_GE = ">=";
20     public static final String OP_ASGN = "=";
21
22     public static final String LTRL = "Literal";
23     public static final String ID = "Identifier";
24     public static final String END = "END";
25
26     public static final String IDSP_EXE = "exe";
27     public static final String IDSP_SHOW = "show";
28     public static final String IDSP_SHOWINE = "showLine";
29     public static final String KW_DATATYPE = "KW_datatype";
30     public static final String FLOAT_NO = "FloatNO";
31     public static final String INT_NO = "IntNO";
32     public static final String LOOP = "KW_loop";
33     public static final String PCOND = "KW_pcond";
34     public static final String SCOND = "KW_scond";
35     public static final String FCOND = "KW_fcond";
36     public static final String CTRUE = "KW_tf";
37     public static final String CFALSE = "KW_tf";
38     public static final String ENDURE = "KW_endure";
39     public static final String INTERRUPT = "KW_interrupt";
40     public static final String REPLY = "KW_reply";
41
42     public static final String BADCHAR = "BADCHAR";
43 }
44

```

LexicalAnalyzer class: This class contains all the functionality provided by the Lexical Analyzer.

Syntax Analyzer

We have used SLR(1) parser in our project. Some classes that are used are described below.

ActionType class: Contains information about different types of action performed in SLR(1) parser.

```

1  /*
2
3
4  -----
5  TALcompiler project
6  -----
7  Developed by-
8  * Abhishek Kumar Singh
9  * Anish Shrivastav
10 * Ayush Chaurasia
11 -----
12 */
13 package syntaxAnalyzer.parseTable;
14
15 /**
16  *
17  * @author ayush
18  */
19 public class ActionType {
20
21     public static final String SHIFT = "SHIFT";
22     public static final String REDUCE = "REDUCE";
23     public static final String ACCEPT = "ACCEPT";
24     public static final String REJECT = "REJECT";
25     public static final String SUGGEST = "SUGGEST";
26
27
28
29
30 }
31

```

Grammar class: Class for Grammar of our Triple A Language.

```

1  /*
2
3
4  -----
5  TALcompiler project
6  -----
7  Developed by-
8  * Abhishek Kumar Singh
9  * Anish Shrivastav
10 * Ayush Chaurasia
11 -----
12 */
13 package syntaxAnalyzer;
14
15 import java.util.HashMap;
16 import java.util.HashSet;
17 import syntaxAnalyzer.parseTable.NonTerminal;
18
19 /**
20  *
21  * @author ayush
22  */
23 public class Grammar {
24
25     public HashMap<Integer, Rule> grammar = new HashMap();
26     public HashSet<String> nonTerminal = new HashSet();
27     public static Grammar G1;
28
29     {...50 lines }
30
31

```

Image: code segment of Grammar Class

InfoPackage class: Data structure for sending and receiving information from Action and Goto Table.

```

19 public class Infopkg {
20
21     public int state;
22     public int ruleNO;
23     public String actionType;
24
25     public String terminal;
26     public String nterminal;
27
28     public Infopkg() {
29
30         this.actionType = "NO_ACTION";
31         this.nterminal = "-1";
32         this.ruleNO = -1;
33         this.state = -1;
34         this.terminal = "-1";
35     }
36
37
38     public Infopkg(int state, String actionType){
39
40         this.state = state;
41         this.actionType = actionType;
42     }
43
44
45     public Infopkg(int ruleNO, String actionType, int a){
46
47         this.ruleNO = ruleNO;
48         this.actionType = actionType;
49     }
50
51 }

```

ActionTable class: contains information and functionality of Action table.

```

83
84 public Infopkg getAction(String k){
85
86     Block b = null;
87
88     try{
89         b = atable.get(k);
90     }catch(Exception e){
91         System.out.println("REexcept");
92     }
93
94     Infopkg retInfo = null;
95
96
97     if(b != null){
98
99         if(b.actiontype == ActionType.SHIFT)
100             retInfo = new Infopkg(b.intInfo,b.actiontype);
101
102         else if(b.actiontype == ActionType.REDUCE)
103             retInfo = new Infopkg(b.intInfo,b.actiontype,0);
104
105         else if(b.actiontype == ActionType.ACCEPT){
106             retInfo = new Infopkg(0, ActionType.ACCEPT);
107         }
108     }
109     else{
110
111         retInfo = new Infopkg();
112         retInfo.nterminal = this.getAllKey(k);
113         retInfo.actionType = ActionType.SUGGEST;
114
115     }
116
117     return retInfo;
118
119 }
120

```

Image: getAction method from ActionTable

GotoTable class: contains information and functionality of Goto Table.

```

1  /*
2
3
4  -----
5  TALcompiler project
6  -----
7  Developed by-
8  * Abhishek Kumar Singh
9  * Anish Shrivastav
10 * Ayush Chaurasia
11 -----
12 */
13 package syntaxAnalyzer.parseTable;
14
15 import java.util.HashMap;
16 import syntaxAnalyzer.Infopkg;
17
18 /**
19 *
20 * @author ayush
21 */
22 public class GotoTable {
23
24     HashMap<String,Integer> gototable = new HashMap();
25
26     public static final GotoTable GT1 = new GotoTable();
27
28     public int getGoto(String key){
29
30         int gotostate = gototable.get(key);
31         return gotostate;
32     }
33     {...50 lines }
34
35 }

```

Image: code of GotoTable

SLRparser class: Contains all the functionality of SLR parser.

```

30 public class SLRparser {
31
32     ActionTable aTble;
33     GotoTable gotoTble;
34     HashMap<Integer, Rule> grammar;
35     HashSet<String> nonTerminal;
36     Token currentToken;
37     int errorCount = 0;
38
39     LexicalAnalyzer lex = new LexicalAnalyzer();
40     Stack<String> stack = new Stack();
41
42
43     public SLRparser(ActionTable a, GotoTable gtable, Grammar gm){
44
45         this.aTble = a;
46         this.gotoTble = gtable;
47         this.grammar = gm.grammar;
48         this.nonTerminal = gm.nonTerminal;
49         stack.push("0");
50         this.incrReadPtr();
51     }
52
53
54     private void incrReadPtr(){
55
56         currentToken = lex.getToken();
57         if(currentToken.tag == null)
58             incrReadPtr();
59         else if(currentToken.tag == Tag.BADCHAR)
60             Error.exitPrg();
61     }
62 }

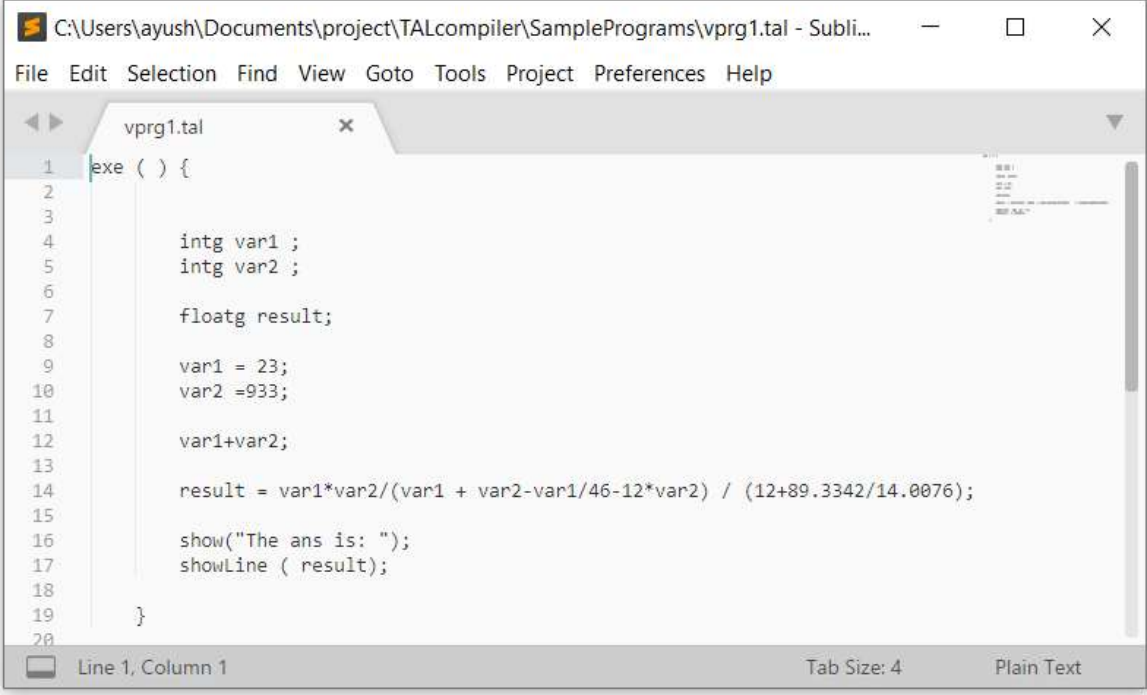
```

Execution

Execution of some sample program is shown below:

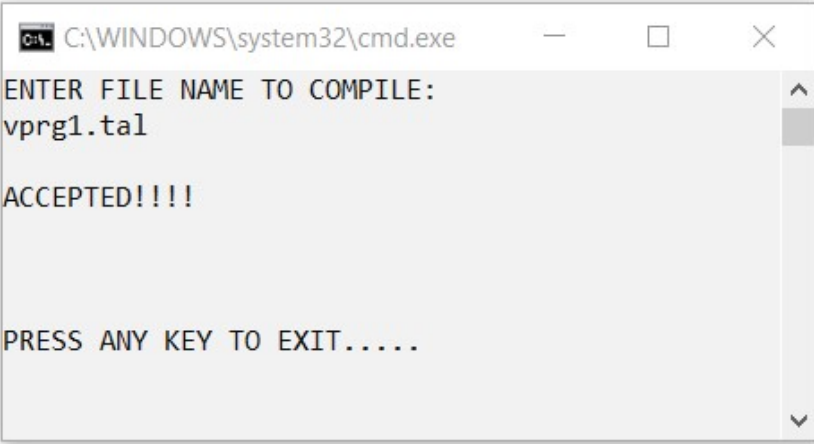
Vprg1.tal

Input:



```
1 exe ( ) {  
2  
3  
4     intg var1 ;  
5     intg var2 ;  
6  
7     floatg result;  
8  
9     var1 = 23;  
10    var2 =933;  
11  
12    var1+var2;  
13  
14    result = var1*var2/(var1 + var2-var1/46-12*var2) / (12+89.3342/14.0076);  
15  
16    show("The ans is: ");  
17    showLine ( result);  
18  
19 }  
20
```

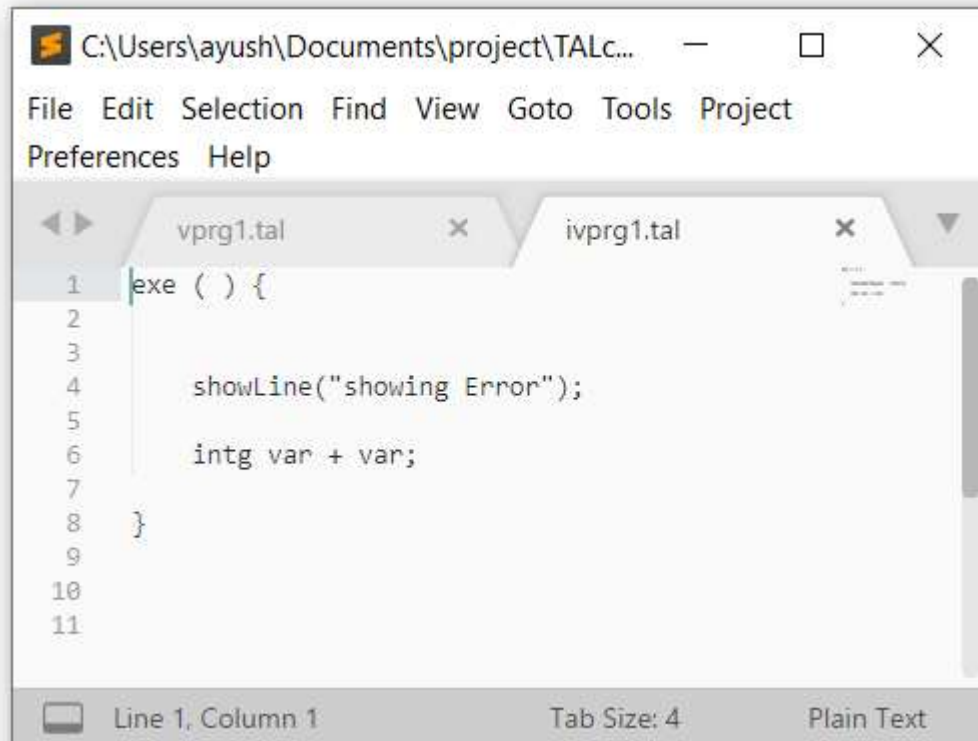
Output:



```
C:\WINDOWS\system32\cmd.exe  
ENTER FILE NAME TO COMPILE:  
vprg1.tal  
  
ACCEPTED!!!!  
  
PRESS ANY KEY TO EXIT.....
```

lvprg1.tal

Input:

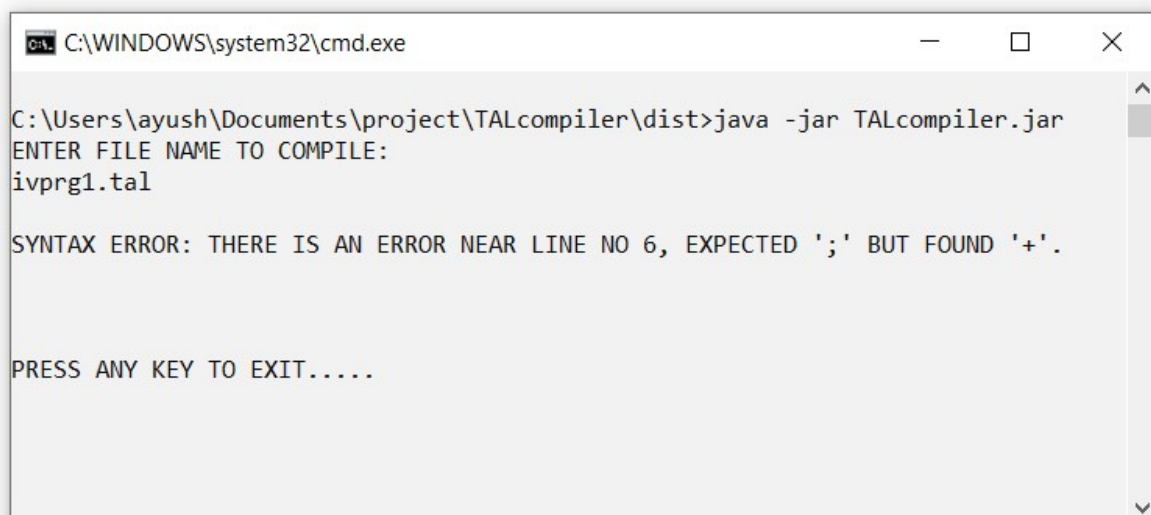


The screenshot shows a text editor window titled "C:\Users\ayush\Documents\project\TALc...". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. There are two tabs open: "vprg1.tal" and "lvprg1.tal". The "lvprg1.tal" tab is active, showing the following code:

```
1 exe ( ) {  
2  
3  
4     showLine("showing Error");  
5  
6     intg var + var;  
7  
8 }  
9  
10  
11
```

The status bar at the bottom indicates "Line 1, Column 1", "Tab Size: 4", and "Plain Text".

Output:

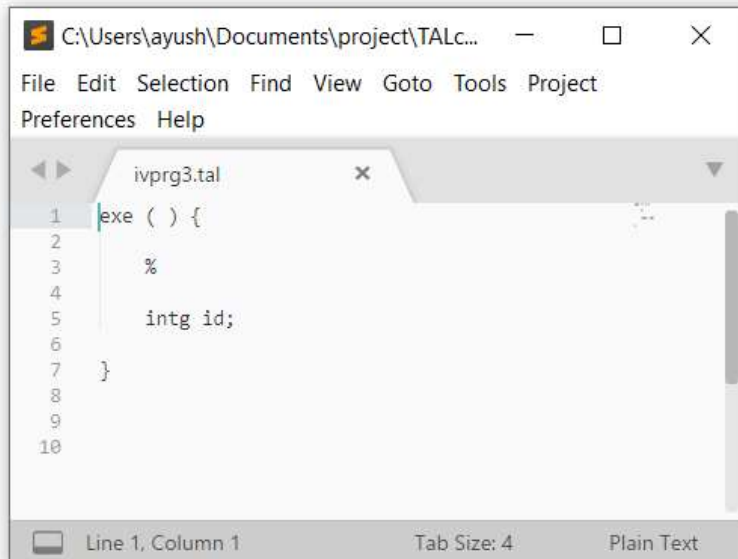


The screenshot shows a command prompt window titled "C:\WINDOWS\system32\cmd.exe". The command prompt displays the following text:

```
C:\Users\ayush\Documents\project\TALcompiler\dist>java -jar TALcompiler.jar  
ENTER FILE NAME TO COMPILE:  
lvprg1.tal  
  
SYNTAX ERROR: THERE IS AN ERROR NEAR LINE NO 6, EXPECTED ';' BUT FOUND '+'.  
  
PRESS ANY KEY TO EXIT.....
```

Invprg3.tal

Input:

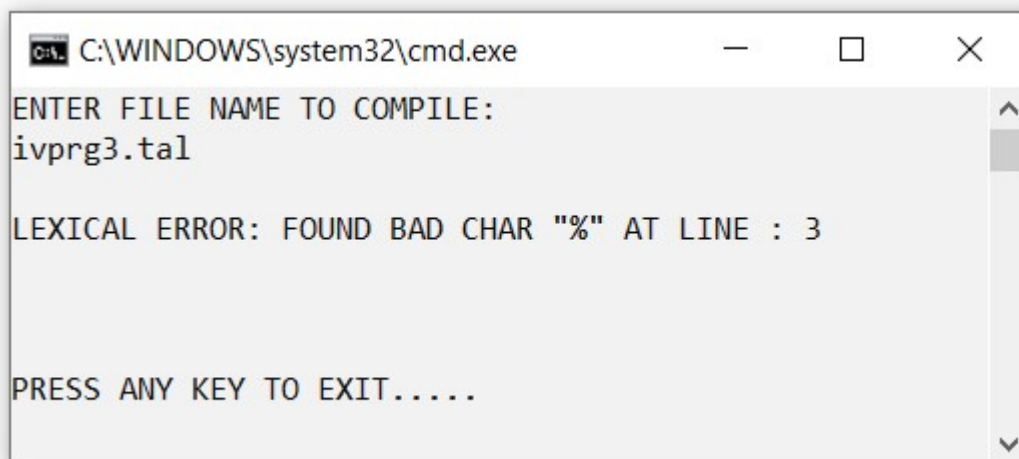


The screenshot shows a text editor window titled 'C:\Users\ayush\Documents\project\TALc...'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The editor displays the file 'ivprg3.tal' with the following code:

```
1 exe ( ) {  
2  
3     %  
4  
5     intg id;  
6  
7 }  
8  
9  
10
```

The status bar at the bottom indicates 'Line 1, Column 1', 'Tab Size: 4', and 'Plain Text'.

Output:



The screenshot shows a Windows command prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The output is as follows:

```
ENTER FILE NAME TO COMPILE:  
ivprg3.tal  
  
LEXICAL ERROR: FOUND BAD CHAR "%" AT LINE : 3  
  
PRESS ANY KEY TO EXIT.....
```