dOvs - lexical analysis

Group 9
Miran Hasanagić - 20084902
Jakob Graugaard Laursen - 20093220
Steven Astrup Sørensen - 201206081

September 15, 2015

1 Introduction

2 How did we implement the lexer

This lexer is implemented by using the existing tool ML-Lex. With this tool it is possible to specify which regular expression shall map to which specific tokens. At first the support for all the basic constructs in the Tiger language where implemented. For example this includes to recognise the keyword. Also the support for non nested comments was implemented in this first version of the lexer.

Following the book by using the ML-Lex tool was ...

2.1 How did we implement nested comments

Implementing the nested comments functionality was harder than the first version of the lexer without this functionality. In order to implement this functionality, the possibility to use states in ML-Lex was used. In this case a COMMENT state was used. Additionally, a counter was used in order to handle nested comments, and detected when we jump out of the COMMENT state. This is accomplished by the following code:

```
”/*”=> (commentLevel := !commentLevel+1; YYBEGIN COMMENT; continue());

<COMMENT>”*/”=> (commentLevel := !commentLevel-1; if !commentLevel < 1 then
YYBEGIN INITIAL else (); continue());
<COMMENT>.        => (continue());
```

This code basically shows that for each /* we meet, the counter is incremented by one. Afterwards, for each */ the counter is decremented by one. When the counter is at zero, we jump out of the COMMENT state. Additionally, this counter value is used in order to detect unclosed comments, and issue an error. In this case we use the function eof, which is called at the EOF. If we reach the EOF and the comment counter value is not zero, then an error is issued.

2.2 How did we implement escape codes

3 Problems experienced

If you encountered any other problems describe how you solved them

4 5 tiger programs

5 interesting tiger programs that you find test your lexer in a good way.

4.1