

dOvs - lexical analysis

Group 9

Miran Hasanagić - 20084902

Jakob Graugaard Laursen - 20093220

Steven Astrup Sørensen - 201206081

September 15, 2015

1 Introduction

This report describes the approach in order to develop the lexer, which is the first component in the compilation process. First, it provides an overview of the concrete solutions to specific problems, such as handling nested comments, escape code and multi-line strings. Additionally, it provides an overview of problems encountered and interesting experiments during the development process. Finally, five tiger programs are provided test

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 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 Escape codes

The escape codes where implemented when the lexer recognises that it is in the `STRING` state. Afterwards when a backslash is found, it check the following character/sequence in order to recognise if it is legal inside a string according to the tiger language specification.

When inside the **STRING** state, the three basic basics chars are handled by:

```
<STRING> "\\n"|"\\t"|"\\\" => (addToCurString yytext;continue());
```

Additionally, the three decimals after backslash are handled the following way:

```
<STRING> "\\\"{digit}{3} => (addToCurString (digitEsc yypos yytext);continue());
```

There is also some error handling for the decimal in order to ensure that there are exact three digits after the backslash:

```
<STRING> "\\\"{digit}{1,2} => (ErrorMsg.error yypos (yytext ^ \" is an illformed  
ascii decimal escape code. ascii decimal escape code must be of the form  
\\\\ddd, with 0 <= ddd <= \" ^ (Int.toString maxAsciiCode) ^ \" , and d a digit  
between 0 and 9\"); continue());
```

At last the escape control chars where handled in order to convert these to their respective escape characteres. This achieved by recognising the character `\^`. Then a function handles to find if the control character is a valid one, and provide the correct mapping using the function **handleCtrl**:

```
<STRING> "\\^\". => (addToCurString (handleCtrl yytext yypos);continue());
```

3 Multiline strings

4 Problems experienced

If you encountered any other problems describe how you solved them

5 5 tiger programs

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

5.1