Lab2: Lexical Analysis —— Doc

学号：522031910415　姓名：周家乐　日期：2024/10/18

本次lab使用flexc++来实现了Tiger language的lexical scanner，主要处理了字符串（String）和Comment（注释）的状态。以下是对本次lab工作的解释：

Scanner.h：

我保留了原scanner.h文件，并没有做更多修改，使用int comment_level_变量来记录注释的层次，最外层的注释comment_level_的值为1。

scanner.h

```
#ifndef TIGER_LEX_SCANNER_H_
#define TIGER_LEX_SCANNER_H_

#include <algorithm>
#include <cstdarg>
#include <iostream>
#include <string>

#include "scannerbase.h"
#include "tiger/errormsg/errormsg.h"
#include "tiger/parse/parserbase.h"

class Scanner : public ScannerBase {
public:
  Scanner() = delete;
  explicit Scanner(std::string_view fname, std::ostream &out = std::cout)
      : ScannerBase(std::cin, out), comment_level_(1), char_pos_(1),
        errormsg_(std::make_unique<err::ErrorMsg>(fname)) {
    switchStreams(errormsg_->infile_, out);
  }

  /**
   * Output an error
   * @param message error message
   */
  void Error(int pos, std::string message, ...) {
    va_list ap;
    va_start(ap, message);
    errormsg_->Error(pos, message, ap);
```

```
30      va_end(ap);
31    }
32
33    /**
34     * Getter for `tok_pos_`
35     */
36    [[nodiscard]] int GetTokPos() const { return errormsg_->GetTokPos(); }
37
38    /**
39     * Transfer the ownership of `errormsg_` to the outer scope
40     * @return unique pointer to errormsg
41     */
42    [[nodiscard]] std::unique_ptr<err::ErrorMsg> TransferErrormsg() {
43      return std::move(errormsg_);
44    }
45
46    int lex();
47
48  private:
49    int comment_level_; //记录当前的注释层级，因为注释可以嵌套，所以要确保注释在最外层
50    std::string string_buf_;
51    int char_pos_;
52    std::unique_ptr<err::ErrorMsg> errormsg_;
53
54    /**
55     * NOTE: do not change all the funtion signature below, which is used by
56     * flexc++ internally
57     */
58    int lex_();
59    int executeAction_(size_t ruleNr);
60
61    void print();
62    void preCode();
63    void postCode(PostEnum_ type);
64    void adjust();
65    void adjustStr();
66  };
67
68  inline int Scanner::lex() { return lex_(); }
69
70  inline void Scanner::preCode() {
71    // Optionally replace by your own code
72  }
73
74  inline void Scanner::postCode(PostEnum_ type) {
75    // Optionally replace by your own code
76  }
77
```

```
78 inline void Scanner::print() { print_(); }
79
80 inline void Scanner::adjust() {
81   errormsg_->tok_pos_ = char_pos_;
82   char_pos_ += length();
83 }
84
85 inline void Scanner::adjustStr() { char_pos_ += length(); }
86
87 #endif // TIGER_LEX_SCANNER_H_
88
```

tiger.lex

1. 我怎么处理comments 注释

   在tiger.lex文件中，有针对注释的代码：

   我在下列代码中添加了更详细的注释，问题解释见代码注释

   tiger.lex comment

```
1  /* comment */
2
3  /* 当扫描到 /* 时，第一层注释开启 ，进入COMMENT状态，并增加comment_level_ */
4  "/*" {
5    adjust();
6    comment_level_++;
7    begin(StartCondition_::COMMENT);
8  }
9
10 <COMMENT> {
11   /* 在COMMENT状态下，遇到 /*，意味着遇到了注释嵌套，需要增加comment_level_ */
12   "/*" {
13     adjust();
14     comment_level_++;
15   }
16   /*
17   在COMMENT状态下，遇到 * /，意味着遇到了注释嵌套，需要增加comment_level_ */
18   "*/" {
19     adjust();
20     comment_level_--;
21     /*如果层次恰好是初始的1，则注释结束，回到INITIAL状态*/
22     if (comment_level_ == 1)
23       begin(StartCondition_::INITIAL);
24   }
25  /* 遇到换行符时要给出Newline的信息 */
26   \n {
```

```
    adjust();
    errormsg_->Newline();
  }
  /*其他情况则继续扫描*/
  . {
    adjust();
  }
}
```

2. 我如何处理strings 字符串

字符串的处理需要考虑到转义符的处理，以下是代码解释：

tiger.lex strings

```
/* string literal */

/* 当扫描到双引号” 时，字符串开始 ，进入STR状态 */
\" {
  adjust();
  begin(StartCondition_::STR);
}
<STR> {
  /* 除了双引号和反斜杠转义字符之外的字符都收集起来 */
  ([[:print:]]{-}[\"\\]{+}[[:space:]])+ {
    adjustStr();
    string_buf_ += matched();
  }

  /*如果再次遇到双引号“，则需要回到INITIAL状态，
  使用setMatched()将string_buf_中的内容设置为当前匹配的字符串值;
  之后清空string_buf_等待下一次字符串扫描
  返回STRING枚举值，表示成功匹配到一个字符串
  */
  \" {
    adjustStr();
    begin(StartCondition_::INITIAL);
    setMatched(string_buf_);
    string_buf_.clear();
    return Parser::STRING;
  }
  /*当遇到反斜杠（\）时，表示转义序列的开始，此时进入到IGNORE状态*/
  \\ {
    adjustStr();
    begin(StartCondition_::IGNORE);
  }
}

 /*以下为转义序列处理，处理的转义字符来源于课本附录中的Tiger语言参考手册*/
```

```
<IGNORE> {
  /*遇到 \n， 对string_buf_加上换行符，同时回到STR状态*/
  "n" {
    adjustStr();
    string_buf_ += "\n";
    begin(StartCondition_::STR);
  }
  /*遇到 \t， 对string_buf_加上制表符，同时回到STR状态*/
  "t" {
    adjustStr();
    string_buf_ += "\t";
    begin(StartCondition_::STR);
  }
   /*遇到 \\， 对string_buf_加上反斜杠符，同时回到STR状态*/
  \\ {
    adjustStr();
    string_buf_ += "\\";
    begin(StartCondition_::STR);
  }
   /*遇到 \” 双引号字符， 对string_buf_加上双引号字符，同时回到STR状态*/
  \" {
    adjustStr();
    string_buf_ += "\"";
    begin(StartCondition_::STR);
  }
   /*遇到具有ASCII码ddd（三个十进制数字）的单个字符， 对string_buf_加上这个字符，同时回到STR状态*/
  [[:digit:]]{3} {
    adjustStr();
    string_buf_ += (char) atoi(matched().data());
    begin(StartCondition_::STR);
  }

   /*遇到 \f___f\ 即一个或多个以上组成的格式化序列，此序列将被忽略，同时回到STR状态*/
  [[:space:]]+\\ {
    adjustStr();
    begin(StartCondition_::STR);
  }

  /*以下表示遇到控制字符，对string_buf_加上这个字符（使用ASCII码），同时回到STR状态*/
  "^C" {
    adjustStr();
    string_buf_ += (char) 3;
    begin(StartCondition_::STR);
  }
  "^E" {
    adjustStr();
    string_buf_ += (char) 5;
```

```
    begin(StartCondition_::STR);
  }
  "^I" {
    adjustStr();
    string_buf_ += (char) 9;
    begin(StartCondition_::STR);
  }
  "^L" {
    adjustStr();
    string_buf_ += (char) 12;
    begin(StartCondition_::STR);
  }
  "^M" {
    adjustStr();
    string_buf_ += (char) 13;
    begin(StartCondition_::STR);
  }
  "^O" {
    adjustStr();
    string_buf_ += (char) 15;
    begin(StartCondition_::STR);
  }
  "^P" {
    adjustStr();
    string_buf_ += (char) 16;
    begin(StartCondition_::STR);
  }
  "^R" {
    adjustStr();
    string_buf_ += (char) 18;
    begin(StartCondition_::STR);
  }
  "^S" {
    adjustStr();
    string_buf_ += (char) 19;
    begin(StartCondition_::STR);
  }
}

```

3. 错误处理

除去COMMENT STR IGNORE INITIAL这几种正确状态，如果走到了还剩余的其他字符，则为错误，所以错误状态优先级放在最后处理

```
/* illegal input */
. {adjust(); errormsg_->Error(errormsg_->tok_pos_, "illegal token");}
```

```
3 /* end of file*/
```