这是 lab2 文档部分，将详细回答 lab2 有关的 4 个问题

1. how you handle comments;

tiger 语言的注释是一个嵌套结构，可以看到在 scanner.h 文件中 comment_level_ 初始化为 1

```
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);
}
```

所以处理注释过程中，每次遇到/*，嵌套的层数就增加一层，每次遇到*/，嵌套的层数减少一层，当 comment_level_重新变为 1 时，说明处理完毕。
代码如下，检测到/*，comment_level_增加，进入<COMMENT>mini scanner，comment_level_返回初始值时，返回 INITIAL

```
/* COMMENT */
"/*"                {adjust(); comment_level_++; begin(StartCondition__::COMMENT);}
<COMMENT>{

  "/*"  {   adjustStr();
            comment_level_++;
        }
  .|\n  {
            adjustStr();
        }
  "*/"  {
            adjustStr();
            comment_level_--;
            if(comment_level_==1){
               begin(StartCondition__::INITIAL);
            }
        }
}
```

2. how you handle strings;

处理字符串一个主要任务是将 string literal 转换为 real string value，string literal 和 real string value 的主要区别就是 /,“,^前面有没有/。
主要思路就是检测到 ” 后进入<STR> ，当再次遇到 ” 时返回 INITIAL，string_buf_存储已转换的 real string value，最后调用 setMatched()设置当前匹配值
查看 tiger 语言规范，发现下面几种处理 string 的特殊情况：

\n	Newline
\t	Tab
\"	Double quote
\\	Backslash
\^c	Control-c, where c is one of @A...Z[\]^_.
\ddd	The character with ASCII code ddd (three decimal digits)
\···\	Any sequence of whitespace characters (spaces, tabs, newlines, returns, and formfeeds) surrounded by \s is ignored. This allows string constants to span multiple lines by ending and starting each with a backslash.

\n , \t , \" , \\处理比较简单，将转换后的 real string value 加到 string_buf_就行，
\ddd 是用 ASCII 码表示得字符，所以要把后面的三位数转换为对应的 char 才行：

```
\\[0-9][0-9][0-9]   {
                adjustStr();
                std::string str=matched();
                string_buf_+=char((str[3]-'0')+(str[2]-'0')*10+(str[1]-'0')*100);

            }
```

具体就是要将后三位数的十进制 int 值算出来，转换为 char 类型，就是对应的字符了
\...\ 这种情况下中间是 space，tab，newline，formfeed 的组合，tiger 语言规范中说这些是要 ignore 的，所以只需要调整 pos，不需要做别的操作：

```
\\[ \n\t\f]+\\  {
                adjustStr();
            }
```

3. error handling;

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

这部分代码用来处理 illegal input，报一个 illegal token 的错

4. end-of-file handling;

查看 flexc++的文档，看到如下图：

- <<EOF>>
  matches `end-of-file';

所以看到<<EOF>>时，代表着文件处理完毕，直接 return，如图

```
/*    end   */
<<EOF>> return 0;
```