编译原理实验一：词法分析与语法分析

郭松 2015301500205

系统环境

Ubuntu 17.10 (Kernel 4.13.0-16)
GCC 7.2.0
Flex 2.6.1
Bison (GNU Bison 3.0.4)

功能简介

尽力地进行语法分析。对于有一定错误的程序，也尝试生成抽象语法树。
构建抽象语法树，并给出每一个节点对应的源代码(的范围)。
在附件中的诸多.c 文件，是我自己用来测试功能的，涵盖了大部分语法点。另外，Makefile 文件是非常简单的 Make 脚本，直接调用 make 即可编译。
当程序遇到了词法错误，会产生类似于下面的输出（注意 Nya)：

```
line 4: Nya! I cannot recognize ``~'', wtf!
line 5: Nya! I can recognize ``le'', but why?
line 6: Nya? ``0x1G'' might be a wrong hex integer.
line 7: Nya? ``09'' might be a wrong oct integer.
```

当程序遇到了文法错误，会产生类似于下面的输出（注意 Meow)：

```
line 8: Meow? Illegal argument list: ``,'' expected
line 8: Meow? Unexpected token
line 11: Meow? ``;'' is expected
line 11: Meow? Unexpected token: legal expression expected
```

对于没有捕获的错误，会输出：

```
line 12: syntax error
```

功能测试

调用 make 编译完成之后，会产生一个叫"ejq_cc"的程序，即最终的可执行文件。
我在附件的 test 文件夹中准备了诸多用于测试的样例。
Ac.c 是一个简单的程序，包含了基本的语法元素
Ac_numbers.c 是一个仅包含数值字面量的程序，展示了对于数字的词法分析
Normal.c 是一个由实际环境下的程序修改得到的程序，展示了在一般情况下的表现
Dinic.c 是一个简短的近似标准的 C 语言程序，描述了用于求解最大流的 Dinic 算法，基本覆盖了所有语言点，展示了一般情况下的表现。
Error.c 是一个简短的，包含了常见错误的 C 语言程序。
Error2.c 是一个简短的，和 error.c 类似的程序，特别的，这个程序包含了未关闭的注释。

词法分析

1. 十进制整数
十进制整数不含有前导零，即如果这个数非零，那么它首位不为 0,否则其为 0,根据这个定义，可以写出：

```
([1-9][0-9]*)|0
```

其前半部分表示正数，后半部分表示 0,对于负数的表示，我们将其表示为符号后跟一个正数，这也是大部分 C 编译器的实现。

2．八进制整数

八进制整数以 0 开头，不包含有大于 7 的数码，可以有前导零，因而可以写为

```
0[0-7]+
```

3．十六进制整数

十六进制整数以 0x 开头，包含 0-9 和 a-f，不区分大小写，因而可以写为

```
0[Xx][0-9A-Fa-f]+
```

4．十进制浮点数

十进制浮点数有两种表示方法，其一为直接表示的方法，其二为科学计数法。对于普通的表示方法，小数点前后至少一部分不为空，则可以分情况考虑为

```
{INT}.[0-9]* | .[0-9]+
```

对于第二种表示方法，可以认为其是简单地在后加上了指数部分，那么指数部分可以表示为

```
[eE][+-]?{INT}
```

综合上述两种情况可以表示为

```
{FLOAT_PARTA}{FLOAT_PARTB}?
```

5．行末注释的表示

行末注释可以表示为"//".*"\n"，Flex 使用的正则表达式的"."不包含换行符，十分方便。

6．块注释的表示

因为注释不组成任何一个语法符号，所以考虑使用词法分析完成块注释的隔离。Flex 提供了“状态”这一概念，可以给自动机加入指定的状态，因此，在读入"/*"之后，我们进入 comment 状态，直到读到第一个"*/"为止，返回 INITAL 状态。这样的处理方式天然解决了嵌套注释的问题。

对于其余的正则表达式，十分简单，不再赘述。

文法分析

对于基本的文法分析，照着要求的附件翻译一下即可。对于错误处理，也只需稍加修改，在可能的地方加上 error 标记，然后处理即可，这里只介绍构建语法树的方法。

定义语法树结构体：

```
typedef struct node{
    char *desc;
    int soncnt, lineno;
    int start_lineno, start_pos, end_lineno, end_pos;
    struct node** son;
} node;
```

desc 起到了描述的功能，在日后进行修改的时候，如要添加访问标志符表等需求，只需稍加修改如添加 item 项，即可实现。

start_lineno、start_pos、end_lineno 和 end_pos 是为了描述语法树节点对应的源代码的位置。

存在的问题

由于对 Yacc 处理错误的方式尚不特别理解，在部分情况下，程序会重复输出一些错误信息。同时对于一些错误信息的输出不是特别准确。