第一次实验

实验目的

通过编写PL0语法的词法分析器，加深对词法分析原理的理解，学会一般的词法分析程序设计步骤。

具体要求：

1. 分析所给的PL0文法，确定关键字，分界符，运算符，常量。
2. 对输入的代码段进行分析，输出每个单词的分类，并输出该单词的值，整数必须用二进制表示
3. 常数默认为整数，必须区分分界符和运算符。
4. 运算符必须区分单目运算符和双目运算符。
5. 对不规范的单词有错误提示。
6. 要求文件输入。
7. 常数里可以识别小数。（选做）
8. 界面友好，可视化操作。（选做）

实验步骤

1. 分析PL0语法，找出语法所有的终结符，按关键词、运算符、标识符、常量
2. 定义词法分析程序的单词类别，这里自己分成了6类
    - **KEYWORD** 关键词
    - **INDENTIFIER** 标识符
    - **DELIMITER** 分界符
    - **SINGLE_OPERATOR** 单字符运算符
    - **DOUBLE_OPERATOR** 双字符运算符
    - **NUMBER** 数字
3. 设计词法分析程序，这里自己使用python语言实现，词法分析本身是正则表达式的匹配问题，定义相关正则表达式并按照优先级依次匹配，一轮匹配中出现多个符合的字符串则选择长度较长的字符串，定义正则表达式匹配串如下

    ```
    r'(?P<BLANK>\s)',
    r'(?P<KEYWORD>const|var|procedure|if|then|else|while|do|call|begin|end|repeat|until|read|
    write|odd)',
    r'(?P<IDENTIFIER>)[A-Za-z][A-Za-z0-9]*',
    r'(?P<NUMBER>\d+(\.\d+)?)',
    r'(?P<DELIMITER>\(|\)|\.|,|;)',
    r'(?P<SINGLE_OPERATOR>\+|-|\*|/|=|<|>)',
    r'(?P<DOUBLE_OPERATOR>:=|<>|<=|>=)',
    r'(?P<COMMENT>/\*[\s\S]*\*/)'
    ```

    BLANK用于辅助识别空格等不输出的字符，对于字符串**constint**，正则表达式首先会匹配为关键词**const**，然后会匹配为标识符**constint**，根据匹配最长原则，最后识别为标识符

    对于未匹配的情况，表明出现了不该出现的单词，需要进行出错处理，本程序在设计时记录了识别的行号和列号，在识别单词出错时输出出错的位置同时终止程序
4. 界面设计，自己采用flask搭了一个简易的网站，输入采用文件输入或者用户手动输入，输出以表格展示

程序分析

程序访问：http://compile.lkc1621.xyz/lexer

这里只给出词法分析器的代码进行分析，完整程序请看文件夹。

```
#! env python3
# -*- coding: UTF-8 -*-

import re
from exceptions import *
from collections import namedtuple

lexicon = [
    r'(?P<BLANK>\s)',
    r'(?
P<KEYWORD>const|var|procedure|if|then|else|while|do|call|begin|end|repeat|until|read|write|odd)'
,
    r'(?P<IDENTIFIER>)[A-Za-z][A-Za-z0-9]*',
    r'(?P<NUMBER>\d+(\.\d+)?)',
    r'(?P<DELIMITER>\(|\)|\.|,|;)',
    r'(?P<SINGLE_OPERATOR>\+|-|\*|/|=|<|>)',
    r'(?P<DOUBLE_OPERATOR>:=|<>|<=|>=)',
    r'(?P<COMMENT>/\*[\s\S]*\*/)'
]
Token = namedtuple('Token', 'type, value')


class LexerEngine:
    def __init__(self):
        self.file = ''
        self.lexicon = [re.compile(x) for x in lexicon]

    def load_file_by_path(self, file_path):
        with open(file_path) as f:
            self.file = f.read()

    def load_file_by_content(self, content):
        self.file = content

    def dec2bin(self, dec_num):
        if '.' in dec_num:
            num = dec_num.split('.')
            res = str(bin(int(num[0]))).split('0b')[1]
            num[1] = '0.' + num[1]
            temp = float(num[1])
            bins = []
            while temp:
                temp *= 2
                if temp >= 1.0:
                    bins.append('1')
                else:
                    bins.append('0')
```

```
                temp -= int(temp)
            res = res + '.' + ''.join(bins)
        else:
            res = str(bin(int(dec_num))).split('0b')[1]
        return res

    def get_token(self):
        cur = 0
        pos = [1, 1]
        length = len(self.file)
        while cur < length:
            token_length = 0
            for pattern in self.lexicon:
                match = re.match(pattern, self.file[cur:])
                if match and (token_length == 0 or match.end() > token_length):
                    token = Token(match.lastgroup, match.group())
                    token_length = len(token.value)
            if token_length == 0:
                raise LexerError(pos=tuple(pos))
            pos[1] += token_length
            cur += token_length
            if token.type == 'BLANK':
                if token.value == '\n':
                    pos[0] += 1
                    pos[1] = 1
                continue
            elif token.type == 'COMMENT':
                temp = token.value.split('\n')
                if len(temp) != 1:
                    pos[0] = pos[0] + len(temp) - 1
                    pos[1] = len(temp[-1]) + 1
                continue
            else:
                yield token

    def complete_token(self):
        res = list()
        try:
            for index, token in enumerate(self.get_token()):
                if token.type == 'NUMBER':
                    res.append({
                        'state': 'normal',
                        'type': token.type,
                        'value': self.dec2bin(token.value)
                    })
                else:
                    res.append({
                        'state': 'normal',
                        'type': token.type,
                        'value': token.value
                    })
        except LexerError as e:

            res.append({
```

```
                'state': 'error',
                'message': e.message
            })
        finally:
            return res

    def print_token(self):
        try:
            for index, token in enumerate(self.get_token()):
                if token.type == 'NUMBER':
                    print("{0:<15}{1}".format(token.type, self.dec2bin(token.value)))
                else:
                    print("{0:<15}{1}".format(token.type, token.value))
        except LexerError as e:
            print(e.message)


def main():
    lexer = LexerEngine()
    file_path = "../doc/test.txt"
    lexer.load_file_by_path(file_path)
    lexer.print_token()


if __name__ == '__main__':
    main()
```

整个词法分析程序封装成一个**LexerEngine**类，正则表达式使用**(?P...)**进行不同类型的分组，读取采用文件输入，**file**变量为文件内容。

load_file_by_path为根据文件路径输入，**load_file_by_content**为根据文件内容输入。

get_token为词法分析关键函数，它将文件内容按行处理，设置**cur**作内容指针，表明当前读取了几个字符，**pos**表明当前读取的行列数。每次进行正则表达式匹配，从当前字符位置往后匹配，使用**re.match**，这会匹配第一个符合条件的字符串，当一轮匹配中出现多个符合条件的字符串时，选择最长的字符串，这种情况只出现在**标识符字符串内容开头为关键字**或者**双字符运算符**的时候。若一轮匹配中没有出现符合条件的字符串，说明文件中出现了不规范的单词，程序报错。当识别到换行符是，要设置**pos**指向下一行的第一个字符，即行数加1，列数设置为1。**print_token**为**token**输出程序，当识别出单词类型为**NUMBER**时要转换为二进制数。

complete_token 返回字典格式的**token**，用于前端js处理显示。

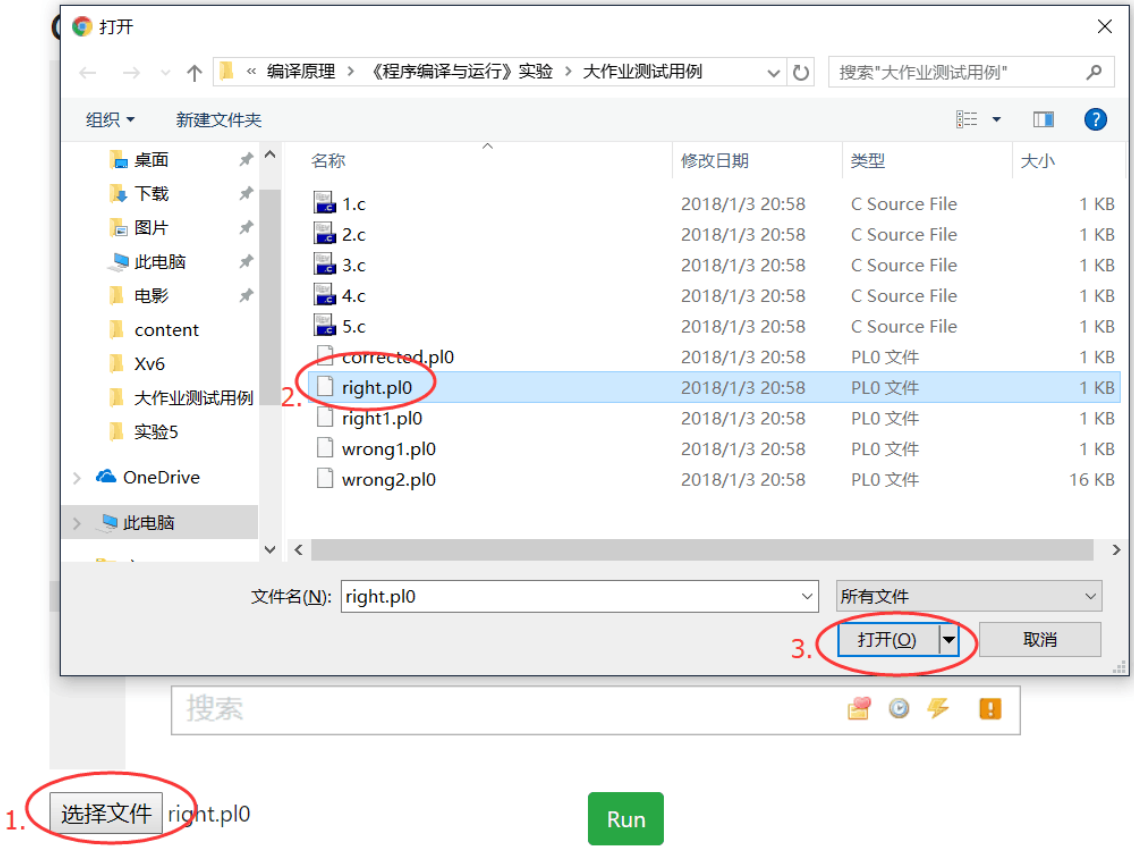
dec2bin为十进制浮点数转换为二进制浮点数的程序，逻辑不再赘述。

程序说明

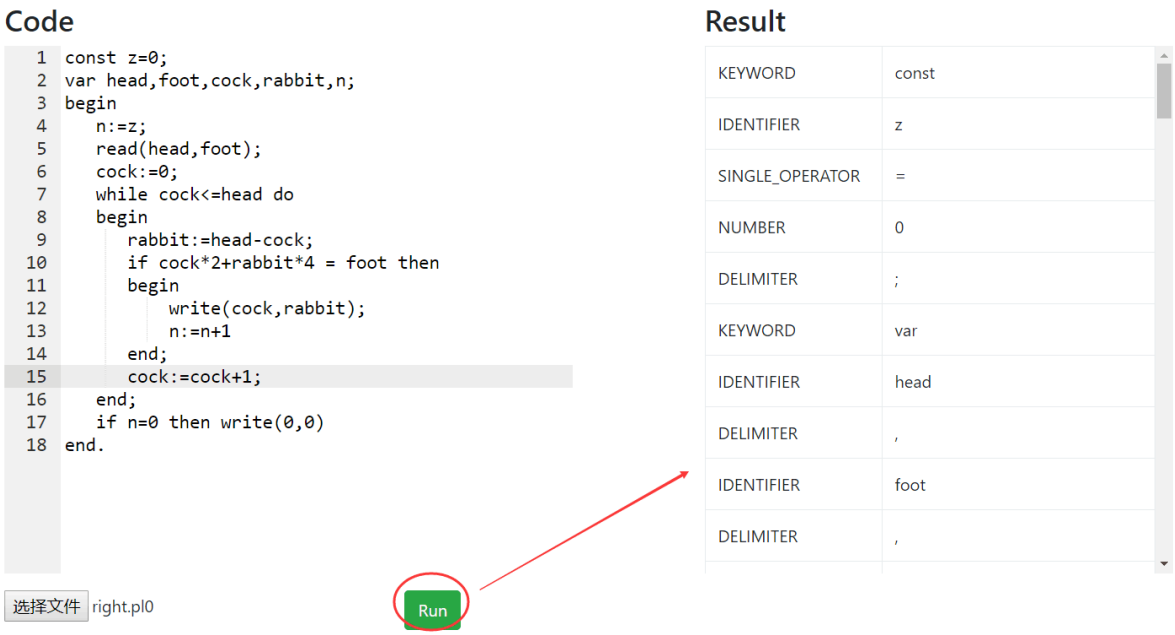
1. 程序采用多符一类。
2. 对于类似”1.1.1.1“，”123a“，仅有单边注释“/*”等，不在此词法分析程序报错，留待语法分析程序报错。
3. 程序处理注释格式为“/.../”

程序使用

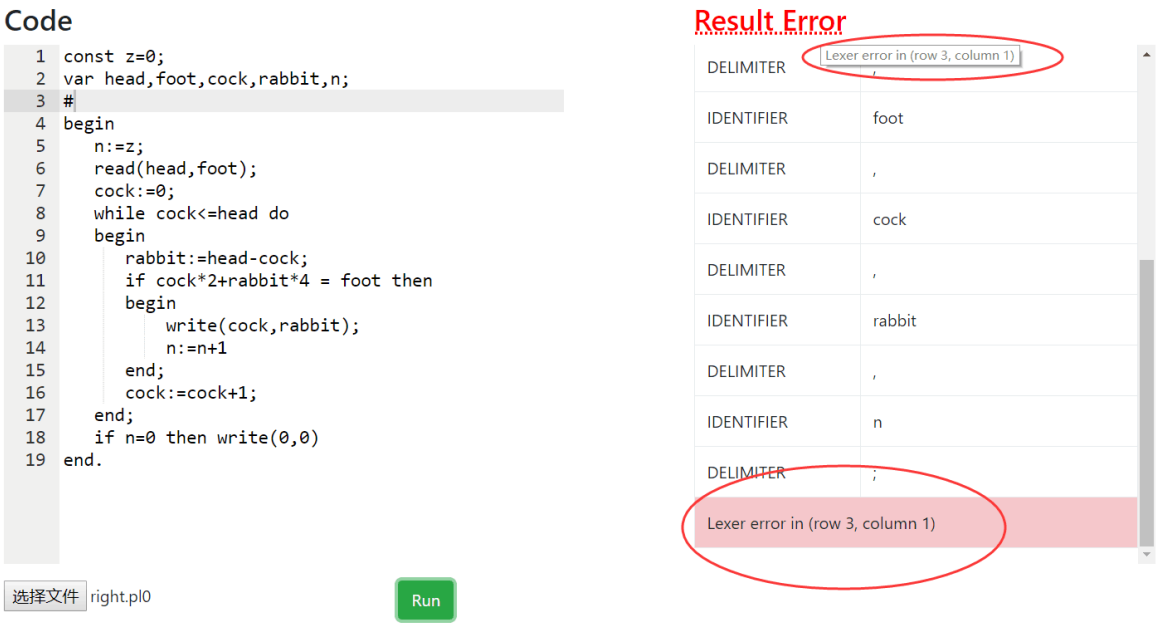
1. 在Code输入框内输入Pascal代码，可手动输入也可文件导入。



2. 点击RUN按钮，右方Result表格内会输出词法分析结果，若显示”Rusult Error“说明出现了不规范的单词。



3. 程序报错提示



实验感想

通过本次实验，自己加强了对词法分析原理的理解，同时也认识到词法分析实质上是一个正则表达式匹配的过程，通过定义正则表达式的优先级和匹配最长的原则完成单词的识别。本次实验也加强了自己对python的掌握，速成了flask和bootstrap实现一个简单的web页面，避免了重复造轮子。