北京科技大学实验报告

学院： 计算机与通信工程学院　　专业： 计算机科学与技术　　班级： 计 184

姓名： 王丹琳　　学号： 41824179　　实验日期： 2021 年 6 月 2 日

实验名称：词法分析

实验目的：

(1) 巩固有限自动机理论与正规文法，正规式三者之间的关系和相关的知识点。

(2) 了解词法分析程序具体的实现方法及所用算法的原理。

(3) 编写相关的词法分析程序实现对 C++语言程序文件的词法分析，输出 token 序列表。

实验仪器：

电脑：ThinkPad T480

操作系统：Windows 10

实验原理：

词法分析：逐个读入源程序字符并按照构词规则切分成一系列单词；

词法分析程序主要任务是 1.读源程序 2.产生单词符号，转换成 token 表示。

令 letter_ = [A-Za-z_]，digit = [0-9]。

(1) 关键字

"auto", "double", "int", "struct", "break", "else", "long", "switch", "case", "enum", "register","typedef", "char", "extern", "return", "union", "const", "float", "short", "unsigned", "continue", "for", "signed", "void","default", "goto", "sizeof", "volatile", "do", "if", "while", "static", "main", "String"

(2) 运算符

'+', '-', '*', '=', '<', '>', '&', '|', '~', '^', '!', '(', ')', '[', ']', '{', '}', '%', ';', ',', '#'

(3) 界符

',', ';', '[', ']', '(', ')', '{', '}'

(4) 标识符

letter_(letter_|digit)*

(5) 无符号常数

digit digit* (\.digit digit*)? ([Ee][+-]? digit digit*)?

(6) 注释

/* */型注释：/*(.|\r\n)**/

(7) 单个字符（char）

\'(.)\'

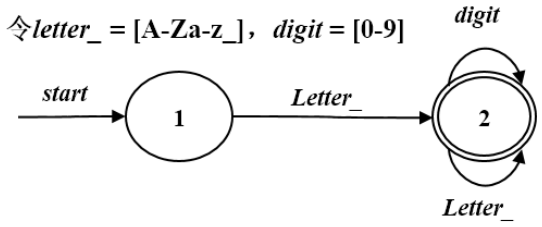
(8) 字符串（String）

\"(.*?)\"

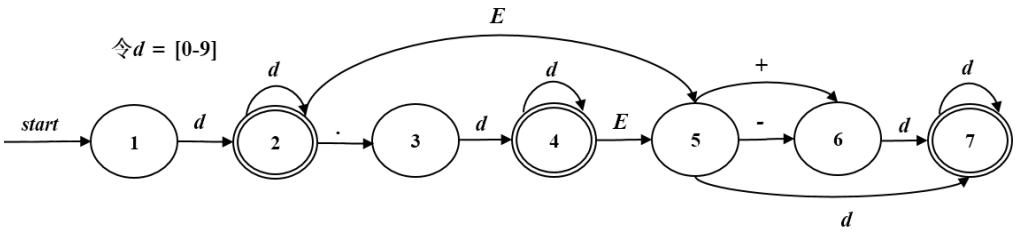
1. 各类单词的转换图

(1) 标识符

标识符:由大小写字母、数字、下划线组成，但必须以字母和下划线开头

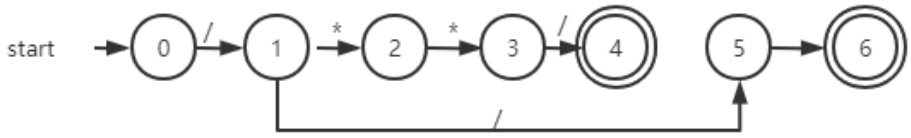


(2) 无符号常数



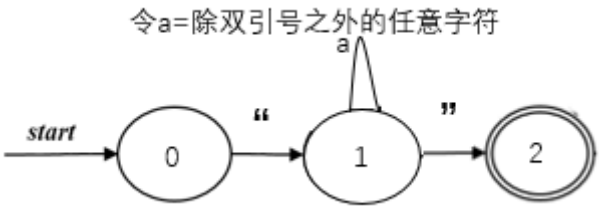
(3) 注释

/* */型注释



(4) 字符串

令 a 表示除了”以外的其他字符，状态 1 输入 a 进入状态 1

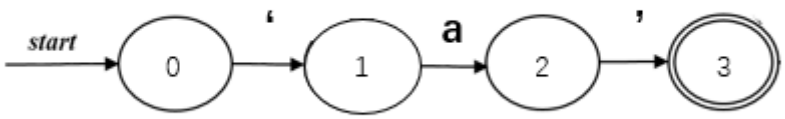


(5) 单个字符

令 a 表示除了\其他的字符

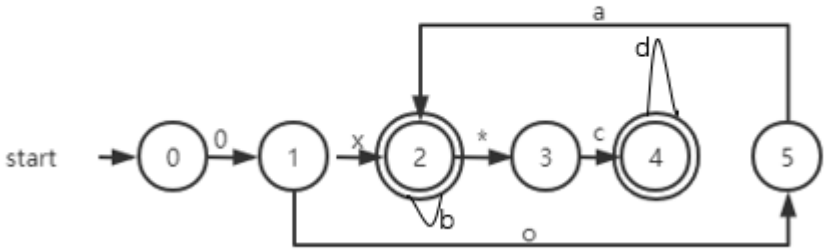
可以接收’a’这样的字符，也可以接收’\n’这样的字符

令a=任意字符

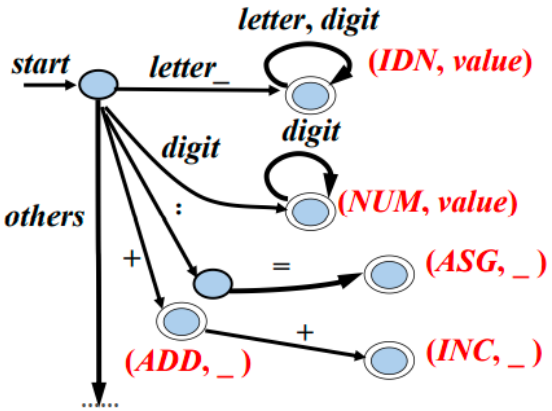


- 八进制、十六进制

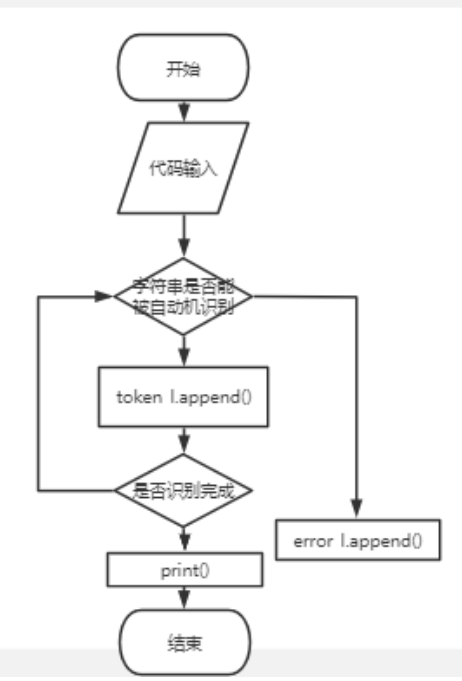
a表示1~7
b表示0~7
c表示1~9. a~f
d表示0~9. a~f



总的识别 Token 的 DFA：



实验内容与步骤：



常量的定义（对于这些部分直接进行判断即可，无需自动机进行判断）

```
#运算符
operator = ['+', '-', '*', '=', '<', '>', '&', '|', '~', '^', '!', '(',
 ')', '[', ']', '{', '}', '%', ';', ',', '#']
#界符
boundary = [',', ';', '[', ']', '(', ')', '{', '}']
#关键字
keywords = [ "auto", "double", "int", "struct", "break", "else", "long"
, "switch", "case", "enum", "register",
        "typedef", "char", "extern", "return", "union", "const", "float
", "short", "unsigned", "continue", "for", "signed", "void",
        "default", "goto", "sizeof", "volatile", "do", "if", "while", "
static", "main", "String"]
```

判断一个字符是否是一个数字

```
def isDigit(charactor):
    return charactor.isdigit()
```

判断一个字符是否是一个字母

```
def isLetter(charactor):
    return charactor.isalpha()
```

判断一个字符串是不是一个关键字

```
def isBasicWord(string):
    return string in basicWordTable
```

判断一个字符是不是一个分隔符

```
def isSeparator(charactor):
    return charactor in separatorTable
```

判断一个字符串是不是一个操作符

```
def isOperator(string):
    return string in operatorTable
```

判断一个字符串是不是可能是一个操作符

```
def isLikeOperator(string):
    return string in "".join(operatorTable)
```

是否空白符

```
def isSpace(charactor):
    return charactor in " \n\r\t"
```

DFA 判断部分（分别针对于标识符，无符号常数，/* */型注释，字符串，单个字符）

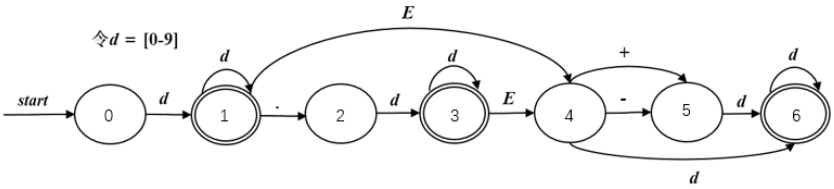
- 无符号常数

  is_digitDFA 函数输入为填入的代码，返回 true 或是 false

  起初状态为 0 号状态，对代码每个字符进行扫描，转换关系如下

	0	1	2	3	4	5	6
0		d					
1		d	.		e		
2				d			
3				d	e		
4						-	
5							
6							

该转换关系与状态转换图有所出入，如第一行：该关系意为 0 号状态遇到 d=[0…9]转到 1 号状态，且遇到其他输入皆无任何状态转换。




```
def is_digitDFA(code):
        state = 0 #state 表示该自动机的状态
        is_float = False

        for i in range(len(code)):
            DFA_state =  digitDFA[state]
            ch = code[i]
            if ch == '.' or ch == 'e':
                is_float = True
            if isDigit(ch):
                if "d" in DFA_state:
                    state = DFA_state.index("d")
                else:
```

```
                    error_message = "状态" + str(state) + "不能接受字符
" + ch
                    return (False, error_message)
            elif (ch == 'e' or ch == '.' or ch == '-'):
                if ch in DFA_state:
                    state = DFA_state.index(ch)
                else:
                    error_message = "状态" + str(state) + "不能接受字符
" + ch
                    return (False, error_message)
            else:
                error_message = "有不合法字符"
                return (False, error_message) #出现了不合法的字符


        if str(state)not in final_state:
            error_message = "自动机停止状态不再最终状态"
            return (False,error_message)
        else:
            return (True, is_float)
```

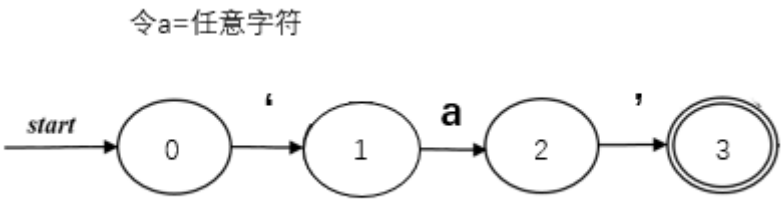
- 单个字符

is_charDFA 函数输入为填入的代码，返回 true 或是 false

起初状态为 0 号状态，对代码每个字符进行扫描，转换关系如下

	0	1	2	3
0		'		
1			a	
2				'
3				

该转换关系与状态转换图有所出入，如第一行：该关系意为 0 号状态遇到’转到 1 号状态，且遇到其他输入皆无任何状态转换。




```
def is_charDFA(self, code):
        state = 0
        for i in range(len(code)):
            ch = code[i]
            DFA_state = self.__charDFA[state]
            if "a" in DFA_state:
                state = 2
            elif ch in DFA_state:
```

```
                state = DFA_state.index(ch)
            else:
                error_message = "状态" + str(state) + "不能接受字符" + ch
                return (False, error_message)

        if str(state) not in  __final_state:
            error_message = "自动机停止状态不再最终状态"
            return (False, error_message)
        else:
            return (True, None)
```

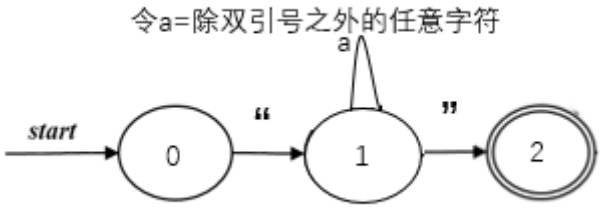
- 字符串

  is_stringDFA 函数输入为填入的代码，返回 true 或是 false

  起初状态为 0 号状态，对代码每个字符进行扫描，转换关系如下

	0	1	2
0		"	
1		a	"
3		#	

该转换关系与状态转换图有所出入，如第一行：该关系意为 0 号状态遇到”转到 1 号状态，且遇到其他输入皆无任何状态转换。




```
def is_stringDFA(self, code):
        state = 0

        for i in range(len(code)):
            ch = code[i]
            DFA_state = self.__stringDFA[state]

            if "a" in DFA_state and ch != "\"":
                state = 1
            elif ch in DFA_state:
                state = DFA_state.index(ch)
            else:
                error_message = "状态" + str(state) + "不能接受" + ch
                return (False, error_message)

        if str(state) in  _final_state:
            return (True, None)
```

```
        else:
            error_message = "自动机停止状态不再最终状态"
            return (False, error_message)
```

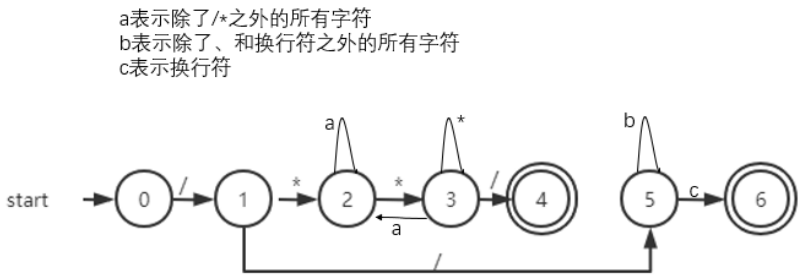
- 注释（/**/）

  is_commentDFA 函数输入为填入的代码，返回 true 或是 false

  起初状态为 0 号状态，对代码每个字符进行扫描，转换关系如下

	0	1	2	3	4	5	6
0		/					
1			*			/	
2			a	*			
3			a	*	/		
4							
5						b	c
6							

该转换关系与状态转换图有所出入，如第一行：该关系意为 0 号状态遇到/转到 1 号状态，且遇到其他输入皆无任何状态转换。




```
def is_commentDFA(self, code):
        state = 0

        for i in range(len(code)):
            ch = program_fraction[i]
            DFA_state = self.__commentDFA[state]

            if ch in DFA_state and ch != "c" and ch != '#':
                state = DFA_state.index(ch)
            elif "a" in DFA_state and ch not in ["/", "*"]:
                state = DFA_state.index("a")
            elif "b" in DFA_state and ch != "/" and ch != "\n":
                state = DFA_state.index("b")
            elif "c" in DFA_state and ch == "\n":
                state = DFA_state.index("c")
            else:
                error_message = "状态" + str(state) + "不能接受字符" + ch
                return (False, error_message)
```

```
        if str(state) not in self.__final_state:
            error_message = "自动机停止状态不再最终状态"
            return (False, error_message)
        else:
            return (True, None)
```

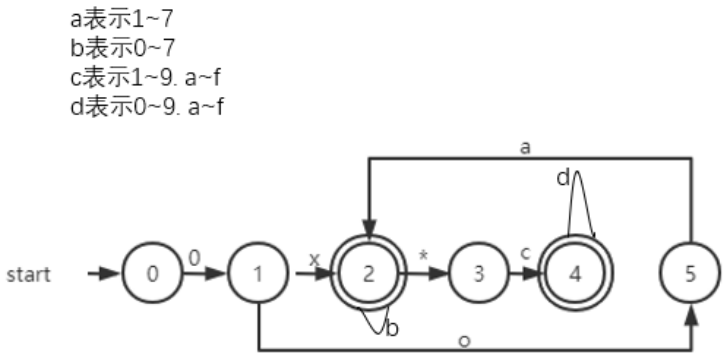
- 八进制、十六进制

  is_O_H_DFA 函数输入为填入的代码，返回 true 或是 false

  起初状态为 0 号状态，对代码每个字符进行扫描，转换关系如下

	0	1	2	3	4	5
0		0				
1				x		o
2			b			
3					c	
4					d	
5			a			

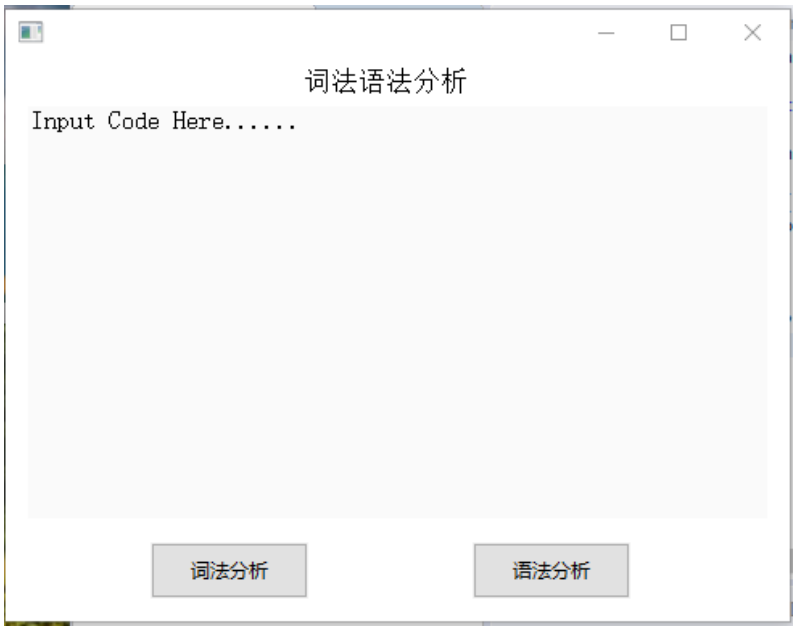
该转换关系与状态转换图有所出入，如第一行：该关系意为 0 号状态遇到数字 0 转到 1 号状态，且遇到其他输入皆无任何状态转换。




```
def is_O_H_DFA(self, code):
        state = 0
        is_octal = False

        for i in range(len(code)):
            c = code[i]
            DFA_state = self.__O_H_DFA[state]
            if state == 2:
                is_octal = True

            if c in DFA_state:
                state = DFA_state.index(c)
            elif "a" in DFA_state and c >= "1" and c <= "7":
                state = DFA_state.index("a")
            elif "b" in DFA_state and c >= "0" and c <= "7":
                state = DFA_state.index("b")
```

```
            elif "c" in DFA_state and ((c >= "1" and c <= "9") or (c >=
 "a" and c <= "f")):
                state = DFA_state.index("c")
            elif "d" in DFA_state and ((c >= "0" and c <= "9") or (c >=
 "a" and c <= "f")):
                state = DFA_state.index("d")
            else:
                error_message = "状态" + str(state) + "不能接受字符" + c
                return (False, error_message, is_octal)

        if str(state) in self.__final_state:
            return (True, None ,is_octal)
        else:
            error_message = "自动机停止状态不再最终状态"
            return (False, error_message, is_octal)
```

GUI 界面：



通过 PyQt5 设计 GUI 界面（使用 QtDesigner 进行可视化设计，然后将生成的.ui 文件转换成.py 文件）

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import QObject, pyqtSlot

class Ui_MainWindow(QObject):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 600)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
```

```
        self.gridLayout.setObjectName("gridLayout")
        self.splitter_3 = QtWidgets.QSplitter(self.centralwidget)
        self.splitter_3.setOrientation(QtCore.Qt.Vertical)
        self.splitter_3.setObjectName("splitter_3")
        self.splitter = QtWidgets.QSplitter(self.splitter_3)
        self.splitter.setOrientation(QtCore.Qt.Horizontal)
        self.splitter.setObjectName("splitter")
        self.widget = QtWidgets.QWidget(self.splitter)
        self.widget.setObjectName("widget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.widget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.label = QtWidgets.QLabel(self.widget)
        self.label.setObjectName("label")
        self.horizontalLayout.addWidget(self.label)
        self.lineEdit = QtWidgets.QLineEdit(self.widget)
        self.lineEdit.setObjectName("lineEdit")
        self.lineEdit.setText("test_file/test_file.txt")
        self.horizontalLayout.addWidget(self.lineEdit)
        self.pushButton = QtWidgets.QPushButton(self.widget)
        self.pushButton.setObjectName("pushButton")
        self.horizontalLayout.addWidget(self.pushButton)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.codeDisplay = QtWidgets.QTextEdit(self.widget)
        self.codeDisplay.setObjectName("codeDisplay")
        self.verticalLayout.addWidget(self.codeDisplay)
        self.tokenBrowser = QtWidgets.QTextBrowser(self.splitter)
        self.tokenBrowser.setObjectName("tokenBrowser")
        self.splitter_2 = QtWidgets.QSplitter(self.splitter_3)
        self.splitter_2.setOrientation(QtCore.Qt.Horizontal)
        self.splitter_2.setObjectName("splitter_2")
        self.errorbrowser = QtWidgets.QTextBrowser(self.splitter_2)
        self.errorbrowser.setObjectName("errorbrowser")
        self.tokenListBrowser = QtWidgets.QTextBrowser(self.splitter_2)
        self.tokenListBrowser.setObjectName("tokenListBrowser")
        self.gridLayout.addWidget(self.splitter_3, 0, 0, 1, 1)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 22))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
```

```
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"
))
        self.label.setText(_translate("MainWindow", "文件"))
        self.pushButton.setText(_translate("MainWindow", "修改确定"))

        self.pushButton.clicked.connect(self.SelectSlot)
        self.lineEdit.returnPressed.connect(self.ReturnPressed)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

实验数据：

测试代码如图所示：

实验数据处理：

对于实验数据，我们可以发现对于一二行，代码不存在词法错误，对于第三行 float test5 = 1.563 词法分析，自动机会发现错误，同理第四行 float test6 = 10.23e-10、第五行 float test7 = 1.1.1 也会发现错误。

对于第六行 char test8 = '\t，我们可以发现缺了’，故自动机应识别失败，同理第七行 String test9 = "asdfg。

对于第八行/*test10，缺少*/，且在对于自动机转换时，规定了/*识别后不存在/*的转换，故自动机识别失败。

实验结果与分析：

部分 token 结果

token	类型	值	行号
int	关键字	int	1
test_1	标识符	test_1	1
,	运算符	,	1
_test2	标识符	_test2	1
;	运算符	;	1
String	关键字	String	2
test4	标识符	test4	2
=	运算符	=	2
.....			

实验结果如预期所想一般。对于第三行 float test5 = 1.563 进行词法分析，输出：试图将 1.563 解析为数字型常量发生错误!与之前分析一样。同理第四行 float test6 = 10.23e-10、第五行 float test7 = 1.1.1 错误相同，输出也类似。

对于第六行 char test8 = '\t 和第七行 String test9 = "asdfg 输出类似都是：试图将'\t（"asdfg）解析为字符型常量发生错误!

对于第八行/*test10，缺少*/，且在对于自动机转换时，规定了/*识别后不存在/*的转换，故输出为：试图将/*test10/*test11*/解析为注释发生错误!

北京科技大学实验报告

学院： 计算机与通信工程学院　　专业： 计算机科学与技术　　班级： 计 184

姓名： 王丹琳　　学号： 41824179　　实验日期： 2021 年 6 月 2 日

实验名称：语法分析

实验目的：

(1) 巩固下推自动机理论与自上而下的语法分析，两者之间的关系与相关的知识点。

(2) 了解语法分析程序具体的实现方法及所涉及算法的原理。

(3) 编写相关的语法分析程序实现对 C++语言程序文件的语法分析。

实验仪器：

电脑：ThinkPad T480

操作系统：Windows 10

实验原理：

LL1 文法：从文法的开始符号出发，反复使用文法的产生式，寻找与输入符号串匹配的推导。

文法参考了此篇博客：https://blog.csdn.net/pinbodexiaozhu/article/details/25394417

部分文法如下：

```
S' -> S
S -> func funcs
funcs -> func funcs
funcs -> $
func -> type IDN ( args ) func_body
type -> int
type -> short
type -> long
type -> char
```

```
type -> float
type -> double
type -> void
type -> struct
type -> unsigned type
args -> type IDN arg
args -> $
arg -> , type IDN arg
arg -> $
func_body -> ;
func_body -> block
……
```

实验内容与步骤：

1. 构造预测分析表

   由于该文法已经满足 LL（1）文法的条件，故不需要提取左公因式或消除左递归。

1.1 求出每条产生式的 SELECT 集

计算每个产生式右部的 FIRST（β）集，为了简化算法，先分别将非终

结符，终结符初始化为空的列表和自身（非终结符的 FIRST 集确是它本身）。紧接着，分别求出单个符号 FIRST 集：此处我主要是参考了网上的一篇博客，感觉他的思路真的是巧妙！思路如下：

遍历每个产生式，从左到右遍历产生式的左部的每个字符，把该字符的 FIRST 集不重复地加到产生式左部的 FIRST 集中去，若含有$，则把下一个字符的 FIRST 集也不重复地加到产生式左部的 FIRST 集中去，直到不含有$或是遍历完了。

```
  def getFirst(self):

        #为所有的非终结符定义一个空的 list
        for i in range(len(self.__nonterminals)):
            self.__firsts[self.__nonterminals[i]] = []

        #如果 X 是一个终结符，那么 first(X) = {X}
        for i in self.__terminals:
            self.__firsts[i] = [i]

        self.__firsts["$"] = ["$"]

        while(True):
            previous_firsts = deepcopy(self.__firsts)

            #遍历每一个产生式
            for i in self.__productions:
                left = i.return_left()#产生式左部
                rights = i.return_right()#产生式

                #对于右侧的每一个元素
                for right in rights:
                    if right != "$":
                        for j in self.__firsts[right]:
                            if j in self.__firsts[left]:
                                continue
                            else:
                                self.__firsts[left].append(j)#非终结符的
first 集加入【】

                    else:
                        if "$" not in self.__firsts[left]:
                            self.__firsts[left].append("$")
```

```
                    if self.isCanBeNull(right):
                        continue
                    else:
                        break

            if previous_firsts == self.__firsts:
                 break
```

然后求出产生式右部的 FIRST 集

```
def getListFirst(self, list):
        if list != []:
            result = deepcopy(self.__firsts[list[0]])
            for l in range(len(list)):
                if "$" not in self.__firsts[list[l]]:#可能的 first 都被找到了
                    return result
                else:
                    if l != len(list) - 1:#借鉴 csdn 上的方法
                        for j in self.__firsts[list[l + 1]]:
                            if j not in self.__firsts[list[l]]:
                                result.append(j)
            return result
        else:
            return []
```

求 FOLLOW 集：

将放到 follow（S）中，其中 S 是开始符号，#将放到 follow（S）中，其中 S 是开始符号，而#是输入右端的结束标记。

如果存在一个产生式 A→αBβ，那么 first（β）中除 ε 之外的所有符号都在 follow（B）中。

如果存在一个产生式 A→αB，或存在产生式 A→αBβ（β 可以是单个非终结符，或数个非终结符的组合，β 或许是 CD）且 first（β）包含 ε，那么 follow（A）中的所有符号都在 follow（B）中。

```
def getFollow(self):
        #初始化每一个非终结符的 terminal 集合
        for i in self.__nonterminals:
            self.__follows[i] = []

        #使用#代替 ppt 算法中的$
        self.__follows[self.__nonterminals[0]].append("#")
```

```
        while(True):
            previous_follows = deepcopy(self.__follows)

            #遍历每一个产生式
            for production in self.__productions:
                left = production.return_left()#非终结符
                rights = production.return_right()#产生式

                for right_index in range(len(rights)):
                    right = rights[right_index]
                    #follow 集和终结符无关
                    if right in self.__terminals or right == "$":
                        continue
                    #求这个 right 之后的 rights 的 first 集
                    first_after_this_right = self.getListFirst(rights[r
ight_index + 1:])
                    if first_after_this_right != []:
                        #这个 right 之后的 first 集中除$外，都是 follow{right}
中的部分
                        for i in first_after_this_right:
                            if i not in self.__follows[right] and i !=
"$":
                                self.__follows[right].append(i)

                    #如果这个是产生式右边的最后一个或者这个 right 后面的产生式
里面有$，那么 Follow{left}中的内容一定会出现在 follow{right}中
                    if (right_index == len(rights) - 1) or ("$" in firs
t_after_this_right):
                        for i in self.__follows[left]:
                            if i not in self.__follows[right]:
                                self.__follows[right].append(i)

            if previous_follows == self.__follows:
                break
```

1.2　按照 SELECT 集把产生式填入分析表，生成预测分析表

算法：

①若 a∈FIRST(α)，则把 A→α 放入 M[A,a]中

②若 ε∈FIRST(α)，则对每个终结符 b∈ FOLLOW(A),将 A→ε 放入 M[A,b]中

③用空表示出错

遍历 production 中的元素，按照格式合成测试分析表：

```
def create_AnalyseTable(self):
    file_path = "file/Analyse_table.txt"
    file = open(file_path, "w")
    temp = []
    for production in self.__productions:
        line = ""
        left = production.return_left()
        rights = production.return_right()

        line = line + left
        line = line + "#"

        for select_list_item in production.select:
            line = line + str(select_list_item)
            temp.append(line)
            line = line + " ->"

            for right in rights:
                line = line + " " + str(right)
            line += "\n"
            file.write(line)
            line = ""
            line = line + left
            line = line + "#"

    for production in self.__productions:
        left = production.return_left()
        for follow in self.__follows[left]:
            line2 = ""
            line2 = line2 + left + "#" + follow
            if line2 not in temp:
                line2 = line2 + "->" + " synch\n"
                file.write(line2)
    file.close()
```

得到了如下结果

```
S'#int -> S
S'#short -> S
S'#long -> S
S'#char -> S
S'#float -> S
```

S'#double -> S

S'#void -> S

S'#struct -> S

S'#unsigned -> S

S#int -> func funcs

S#short -> func funcs

S#long -> func funcs

.....

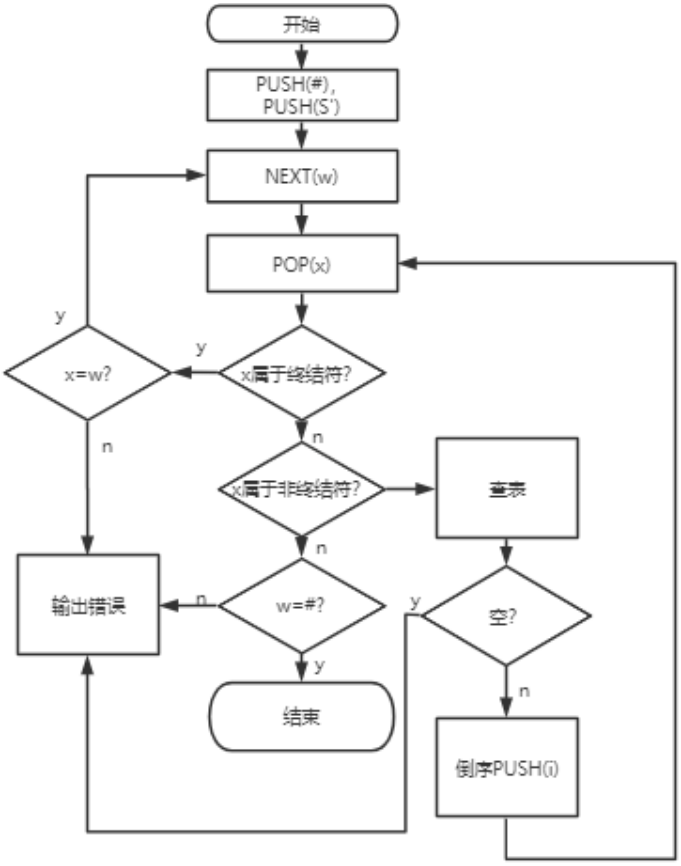
同时存入.xlsx 文件

	Id	NUM	#include<	>	#include"	"	ngnamespa	int	float	double
P			1		1					
PC			2		2					
DL								10	10	10
HL			3		4					
NL							7			
H1			5		5		6			
N1							8	9	9	9
D								13	13	13
D+										

LL(1)算法实现：

利用刚刚所生成的预测分析表作为预测条件，利用栈作为分析栈进行具体推导。首先，初始化空栈，压入'@'表示栈底（为了避免混淆，将‘@’代替‘#’），同时将指针指向字符串第一个字符。

然后，开始符号进栈。根据栈顶符号、指针指向的字符在 SELECT 集合找到对应的产生式。将栈顶符号出栈，将产生式“逆序”压栈。一直这么循环，直到所指字符与栈顶字符相同，此时该字符匹配成功。此时需要，将后指针移向下一个字符，同时将栈顶符号弹栈。继续根据栈顶符号、指针指向的字符在 SELECT 集合找到对应的产生式，逆序压栈，相同再出栈。直到栈为空，字符串字符全部匹配完。但是，若不能全部匹配完，字符串匹配不成功。




```
def LL_1_Analyse(input_list, analyse_table_path, grammar_Analyse, line_
number_1):
    tree_input_file = open("file/tree_input.txt", "w")
    error_1 = [] #存储打印出来的错误信息
    production_1 = [] #存储打印出来的产生式
    input_list.append("#") #添上标记为输入尾巴的井号
    stack = Stack()
    #初始符号压入栈
    stack.push("@") #为了避免混淆，初始符号从 ppt 里面的$改为了这里的@
    file = open(analyse_table_path, "r")
    line = file.readline()
    hash_key_index = line.index("#")
    stack.push(line[:hash_key_index])
    file.close()

    X = stack.top()
    ip = 0

    def search_analyse_table(stack_top,ip):
        file = open(analyse_table_path, "r")
        for line in file.readlines():
```

```
            if stack_top + "#" + input_list[ip] == line[0:line.index("-
")].strip():
                return (True, line)
        file.close()
        return (False, None)

    while(X != "@"):
        if X == input_list[ip]:
            stack.pop()
            ip += 1
        #栈顶终结符和输入符号不匹配，弹出栈顶的终结符

        elif X in grammar_Analyse.get_terminals():
            print(str(line_number_l[ip]) + ":栈顶终结符和输入符号不匹
配，弹出栈顶终结符" + stack.top())
            error_l.append(str(line_number_l[ip]) + ":栈顶终结符和输入符
号不匹配，弹出栈顶终结符" + stack.top())
            stack.pop()

        else:
            search_result, line = search_analyse_table(X, ip)

            if search_result:
                if line[-1] == "\n":
                    line = line[:-1]
                if "synch" not in line:
                    arrow_index = line.index("-")
                    hash_key = line.index("#")
                    line = line[:hash_key] + line[arrow_index:]
                    print(line)
                    tree_input_file.write(line + "\n")
                    production_l.append(line)
                    stack.pop()
                    line_right = line[line.index(">") + 1 :].strip()
                    rights = line_right.split(" ")
                    if rights[0] != "$":
                        for i in range(len(rights) - 1, -1, -1):
                            stack.push(rights[i])
                else:
                    #TODO: 打印错误信息和行号
                    #M<A,a>是 synch，弹出栈顶的非终结符 A 并继续
                    print(str(line_number_l[ip]) + ":弹出栈顶终结符
" + stack.top())
```

```
                    error_l.append(str(line_number_l[ip]) + ":弹出栈顶终
结符" + stack.top())
                    stack.pop()
            else:
                #M<A,a>是空，表示检测到错误，忽略输入符号 a

                print(str(line_number_l[ip]) + ": 忽略
" + input_list[ip])
                error_l.append(str(line_number_l[ip]) + ": 忽略
" + input_list[ip])
                ip += 1
        X = stack.top()
    return error_l, production_l
```

主函数：

首先需要点击词法分析按钮：利用实验一的 Latex_analys()进行词法分析，获得 token 序列，然后点击语法分析按钮：调用 LL_1_Analyse()函数进行语法分析：

```
    token_l = []
    line_number_l = []

    for i in token:
        tmp = i.split("\t")[1]
        if tmp == "IDN":
            token_l.append("IDN")
        elif tmp == "FLOAT":
            token_l.append("FLOAT")
        elif tmp == "INT10":
            token_l.append("INT10")
        elif tmp == "INT8":
            token_l.append("INT8")
        elif tmp == "INT16":
            token_l.append("INT16")
        elif tmp == "STR":
            token_l.append("STR")
        elif tmp == "CHAR":
            token_l.append("CHAR")
        else:
            token_l.append(i.split("\t")[0])

        line_number_l.append(i.split("\t")[-1])
    # print(line_number_l)
    # print(token_l)
    grammar_analyser = Grammar_Analyser()
```

```
    error_l, production_l = LL_1_Analyse(token_l, "file/Analyse_table.t
xt", grammar_analyser, line_number_l)
```

界面设计：

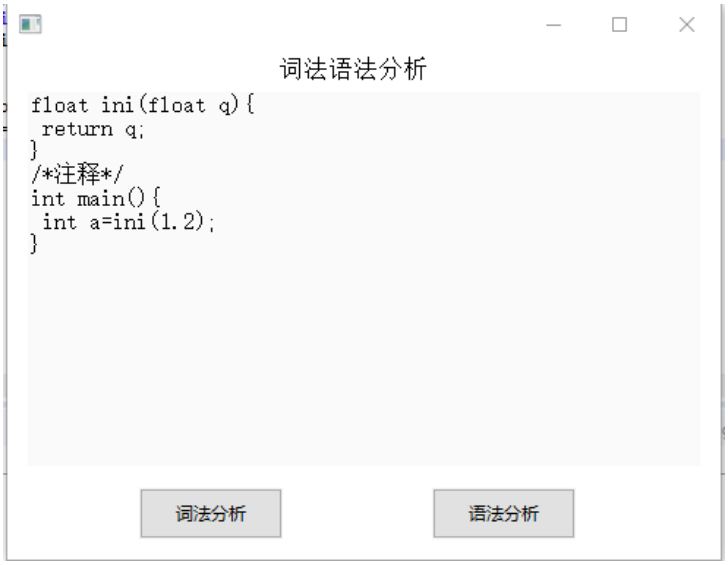
通过 PyQt5 设计 GUI 界面（使用 QtDesigner 进行可视化设计，然后将生成的.ui 文件转换成.py 文件）

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import QObject, pyqtSlot
from latex_analyse.Latex import Latex_analyse_main
from Grammar_Analyse import Grammar_Analyser
from Parser import LL_1_Analyse

class Ui_MainWindow(QObject):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 600)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName("gridLayout")
        self.splitter_3 = QtWidgets.QSplitter(self.centralwidget)
        self.splitter_3.setOrientation(QtCore.Qt.Vertical)
        self.splitter_3.setObjectName("splitter_3")
        self.splitter = QtWidgets.QSplitter(self.splitter_3)
        self.splitter.setOrientation(QtCore.Qt.Horizontal)
        self.splitter.setObjectName("splitter")
        self.layoutWidget = QtWidgets.QWidget(self.splitter)
        self.layoutWidget.setObjectName("layoutWidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.label = QtWidgets.QLabel(self.layoutWidget)
        self.label.setObjectName("label")
        self.horizontalLayout.addWidget(self.label)
        self.lineEdit = QtWidgets.QLineEdit(self.layoutWidget)
        self.lineEdit.setObjectName("lineEdit")
        self.lineEdit.setText("file/test")
        self.horizontalLayout.addWidget(self.lineEdit)
        self.pushButton = QtWidgets.QPushButton(self.layoutWidget)
        self.pushButton.setObjectName("pushButton")
        self.horizontalLayout.addWidget(self.pushButton)
```

```
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.codeDisplay = QtWidgets.QTextEdit(self.layoutWidget)
        self.codeDisplay.setObjectName("codeDisplay")
        self.verticalLayout.addWidget(self.codeDisplay)
        self.tokenBrowser = QtWidgets.QTextBrowser(self.splitter)
        self.tokenBrowser.setObjectName("tokenBrowser")
        self.splitter_2 = QtWidgets.QSplitter(self.splitter_3)
        self.splitter_2.setOrientation(QtCore.Qt.Horizontal)
        self.splitter_2.setObjectName("splitter_2")
        self.errorbrowser = QtWidgets.QTextBrowser(self.splitter_2)
        self.errorbrowser.setObjectName("errorbrowser")
        self.gridLayout.addWidget(self.splitter_3, 0, 0, 1, 1)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 22))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        self.pushButton.clicked.connect(self.SelectSlot)
        self.lineEdit.returnPressed.connect(self.ReturnPressed)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"
))
        self.label.setText(_translate("MainWindow", "文件"))
        self.pushButton.setText(_translate("MainWindow", "修改"))

    @pyqtSlot()
    def SelectSlot(self):
        pass

    @pyqtSlot()
    def ReturnPressed(self):
        pass


# if __name__ == "__main__":
#     import sys
#     app = QtWidgets.QApplication(sys.argv)
```

```
#     MainWindow = QtWidgets.QMainWindow()
#     ui = Ui_MainWindow()
#     ui.setupUi(MainWindow)
#     MainWindow.show()
#     sys.exit(app.exec_())
```

实验数据：



实验数据处理：

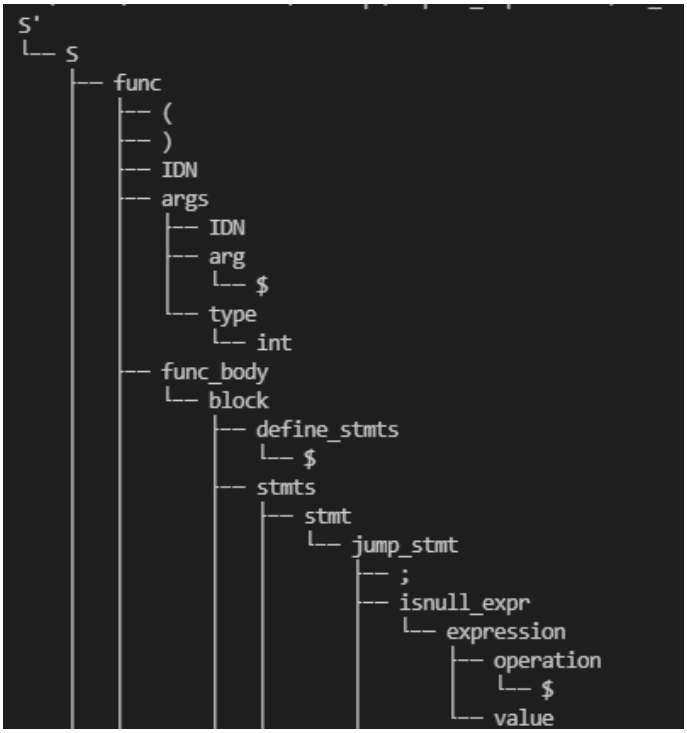
读取指定的文件，先进行词法分析过程，并将做好词法分析的 token 流存储到队列中，并进行语法分析。比对栈顶内容与读头下内容，再通过查询预测分析表和产生式表获得一步推导，将推导结果压到栈中，并重复这一过程直到出现错误或者完全推导结束。

点击词法分析按钮完成词法分析获得 token，然后点击语法分析按钮进行语法分析。

实验结果与分析：

可以发现，产生式与所输入的代码构造方法相同。

同时在控制台输出语法树：



对比文法，可以发现打印出来的语法树符合我们所输入的代码。但是使用的自上而下的分析方法，LL(1)的分析能力是很有限的，二义文法是处理不了的，只能对文法做出强行的规定，所以对于可以分析的语法就受到一定的限制。本此实验实现的 LL(1)语法所用的文法不具有二义性，不含有左递归或者是左公因式，分析比较局限，但是其实还是比较复杂的，可以对基本的头文件声明、名字空间使用声明、函数声明、变量初始化、分支语句、循环语句、返回语句、运算表达式（加减乘除）、表达式比较（关系云算法）等进行分析处理。