

软件课程设计（Ⅱ）

总报告

|  |  |  |  |
| --- | --- | --- | --- |
| **学 生：** |  | **学 号：** |  |
| **学 院：** | 计算机科学与工程学院 | | |
| **专 业：** | 软件工程 | | |
| **指导教师：** |  | | |

**前言**

基于上学期所学的“编译原理”课程的内容，自己设计一门编程语言并对其进行实现。本次课程设计模仿“C++”语言进行设计。使用“VS Code”软件，C++语言进行开发，环境为Windows11，基于个人使用的电脑。

# 词法分析

## 设计思想

首先自己确定一个合理且有效的三型文法，输入该文法生成对应的NFA，计算出NFA的Move矩阵。再将NFA通过子集法简化成DFA。之后自行编写一段符合该文法的源代码输入进去，通过DFA识别出源代码中所有字符的类型并且输出成一个三元组token列表，即输出格式为“<行号,类型,’内容’>”。

输入的源代码通过将字符映射到一个字母的方式输出到LexicalOutput.txt中为做语法分析做准备，标识符映射为0，常量映射为1，#映射为\*，双目运算符映射为2，其他不变。

## 数据结构

FILE \*file：打开要进行词法分析的源代码文件。

ofstream output：输出语法分析时需要使用的中间文件。

NFA Move[100][100]：Move矩阵。

NFA newSet[100]：新出现的NFA。

int dfa[200][200]：判断dfa数组。

bool inFinalState[200]：判断是否为终态的数组。

char state[50]：状态集。

char alpha[50]：字母表。

char final[50]：终态集。

const char keyword[20][10]：关键字数组。

const char delimiter[20]：界符数组。

const char moOperator[10]：单目运算符数组。

const char biOperator[10][3]：双目运算符数组。

## 函数

char func(char str[])：将关键字映射到一个字母表示，便于语法分析使用。

bool isState(char a)：判断a是否为状态集中的状态。

bool isAlpha(char a)：判断a是否为字母表中的字母。

bool isFinal(char a)：判断a是否为终态集中的终态。

void createNFA()：通过三型文法构造NFA。

void showNFA()：输出NFA，测试时用于检查NFA是否正确。

void createMove()：通过NFA构造Move矩阵。

void showMove()：输出Move矩阵，测试时用于检查Move是否正确。

bool stateInSet(char a,NFA temp)：查询状态a是否在temp的状态中。

void getClosure(NFA &temp)：求ε-closure(move(Ti,alpha))边。

bool containY(NFA temp)：查询temp是否是终态。

int isIn(NFA temp)：在已有的状态集中查找有没有与temp重复的，有就返回重复的编号，否则返回-1。

void NFAtoDFA()：将NFA通过子集法转换为DFA。

bool isPassedDFA(char str[])：判断str是否符合DFA。

bool isKeyword(char str[])：判断str是否为关键字。

bool isDelimiter(char a)：判断a是否为界符。

bool isMO(char a)：判断a是否为单目运算符。

bool isBO(char str[])：判断str是否为双目运算符。

void getResult()：输出三元组token列表和用于语法分析的中间文件LexicalOutput.txt。

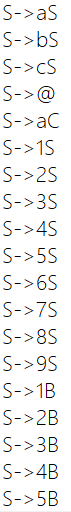
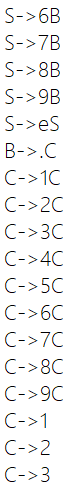
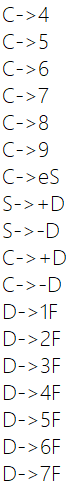
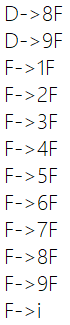
## 文法说明

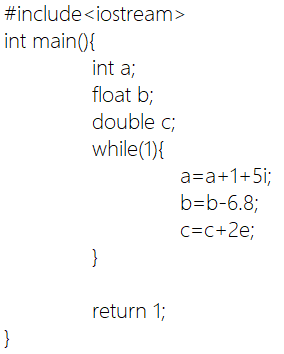
在代码文件data/ LexicalFormula中。

S为文法的开始符号。

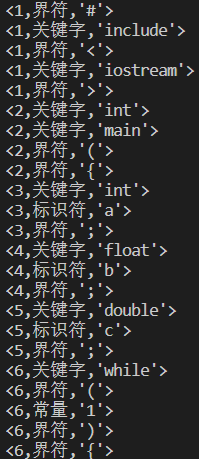
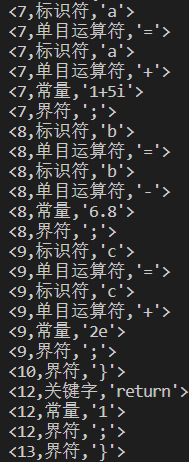
S为标识符，B为浮点数，C为整型数字或复数的实部或科学计数法中的e，D和F共同组成复数的虚部。

## 输入以及输出结果

输入的三型文法：

输入的源代码：

词法分析后输出的三元组token列表：

为便于语法分析输出的中间文件：

# 语法分析

## 设计思想

首先写好一个合理且有效的二型文法文件，输入后通过逐行读取的方式读取到grammar数组中为后续语法分析做准备工作。计算该文法的FIRST集输出。再求该文法的LR(1)项目集族，即拓广文法后求闭包再用转换函数逐步求出所有LR(1)项目集族。然后求出该文法的ACTION表和GOTO表。最后将词法分析输出的LexicalOutput.txt文件输入，对此文件进行扫描即可得到分析源代码的分析过程。

## 数据结构

bool isV[200]：记录终结符和非终结符存在与否与位置。

int sizeItem[200]：记录项目集内文法长度。

bool first[200][200]：记录FIRST集存在与否与位置。

char grammar[200][200]：记录二型文法。

int lenGrammar[200]：记录二型文法长度。

char Vt[200]：记录终结符。

char Vn[200]：记录非终结符。

stack<int> state：分析过程中的状态。

stack<char> sign：分析过程中的符号

ifstream grammarFile：二型文法的输入文件。

ifstream inputFile：词法分析输出的中间文件。

ofstream itemsFile：项目集族的输出文件。

ofstream firstsetFile：FIRST集的输出文件。

ofstream actionFile：分析表的输出文件。

ofstream procedureFile：分析过程的输出文件。

Item item[200][200]：表示项目集族。

Trans trans[200]：表示转换表。

Action actionTable[200][200]：表示分析表。

int lenItem[200]：项目集的长度。

char buffer[200]：缓冲区。

int lenActTable[200]：项目表的长度。

## 函数

void getGrammar()：读取二型文法。

void showGrammar()：输出二型文法。

bool isInVn(char a)：判断a是否为非终结符。

void ouputFirst()：输出FIRST集。

void getFirst()：计算FIRST集。

void getSearch(Item temp)：计算temp的向前搜索符。

bool isIn(Item temp,int t)：判断t是否在temp的项目集中。

void getClosure(int t)：计算t的闭包。

int isInItem()：查看项目集是否重复，重复返回重复数，否则返回0。

void getItemSet()：输出项目集。

void getAction()：输出ACTION表。

void writeStack(int x)：使用栈处理，便于最后分析过程的输出。

void getResult()：计算结果和分析过程。

## 文法分析

在代码文件data/GrammarFormula中。

鉴于拓广文法中“S’->S”的“S’”需要使用字符串存储，便于后续处理我把拓广文法改写为了S->P。

P表示测试代码的包含头文件命令行。

I表示头文件。

A表示函数。

B表示函数返回值类型。

C表示主函数。

D表示数据类型。

G表示标识符。

E表示函数中的各种指令代码。

T为定义标识符语句。

N为定义标识符。

Z为分号。

R为函数返回值语句。

W为while指令。

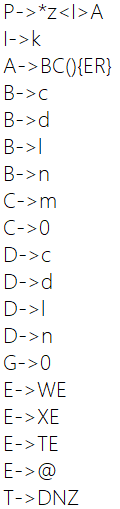
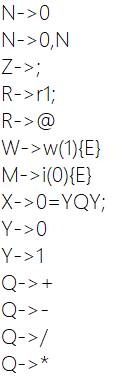
M为if指令。

X为标识符赋值语句。

Y为标识符或常量。

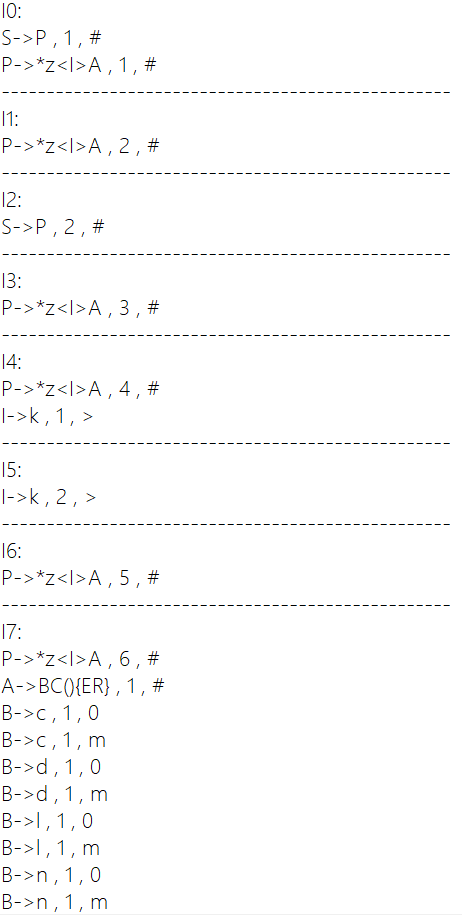
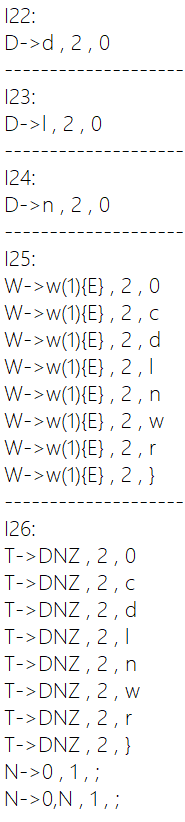
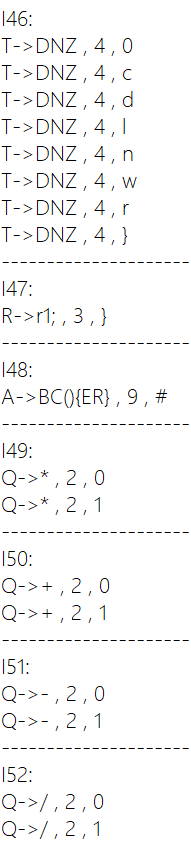
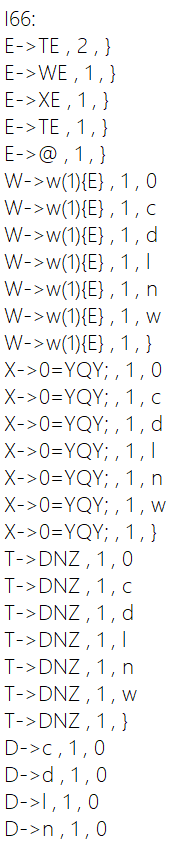
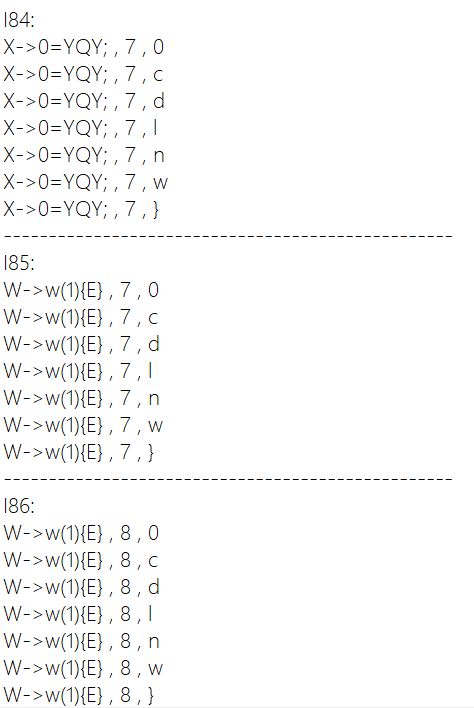
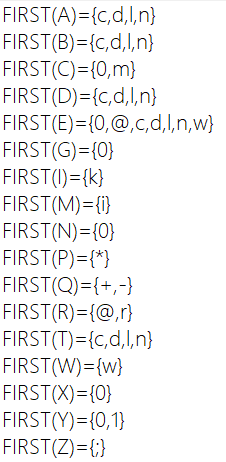
Q为符号。

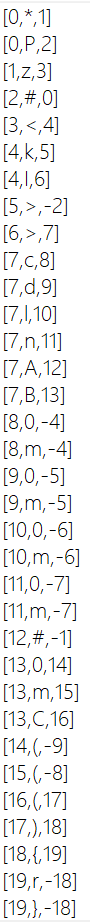
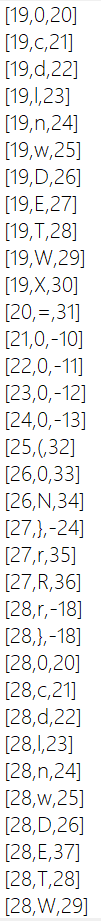
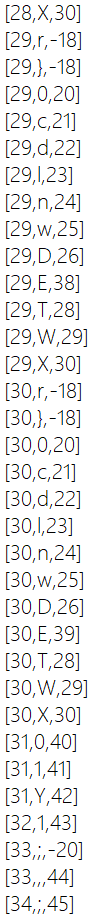
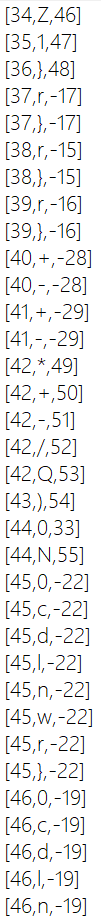
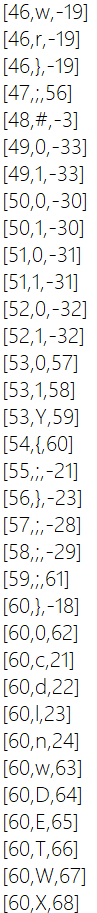
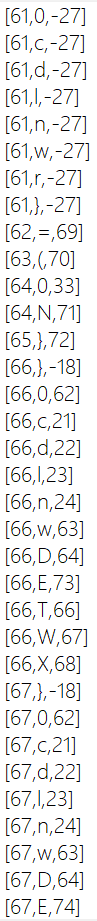
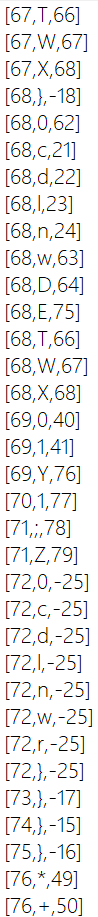
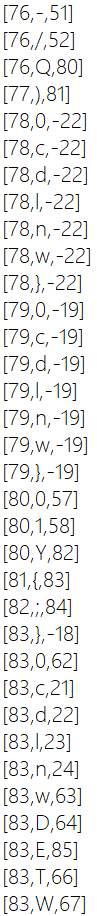
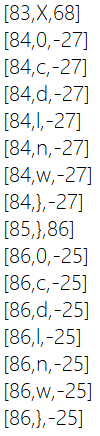
## 输入以及输出结果

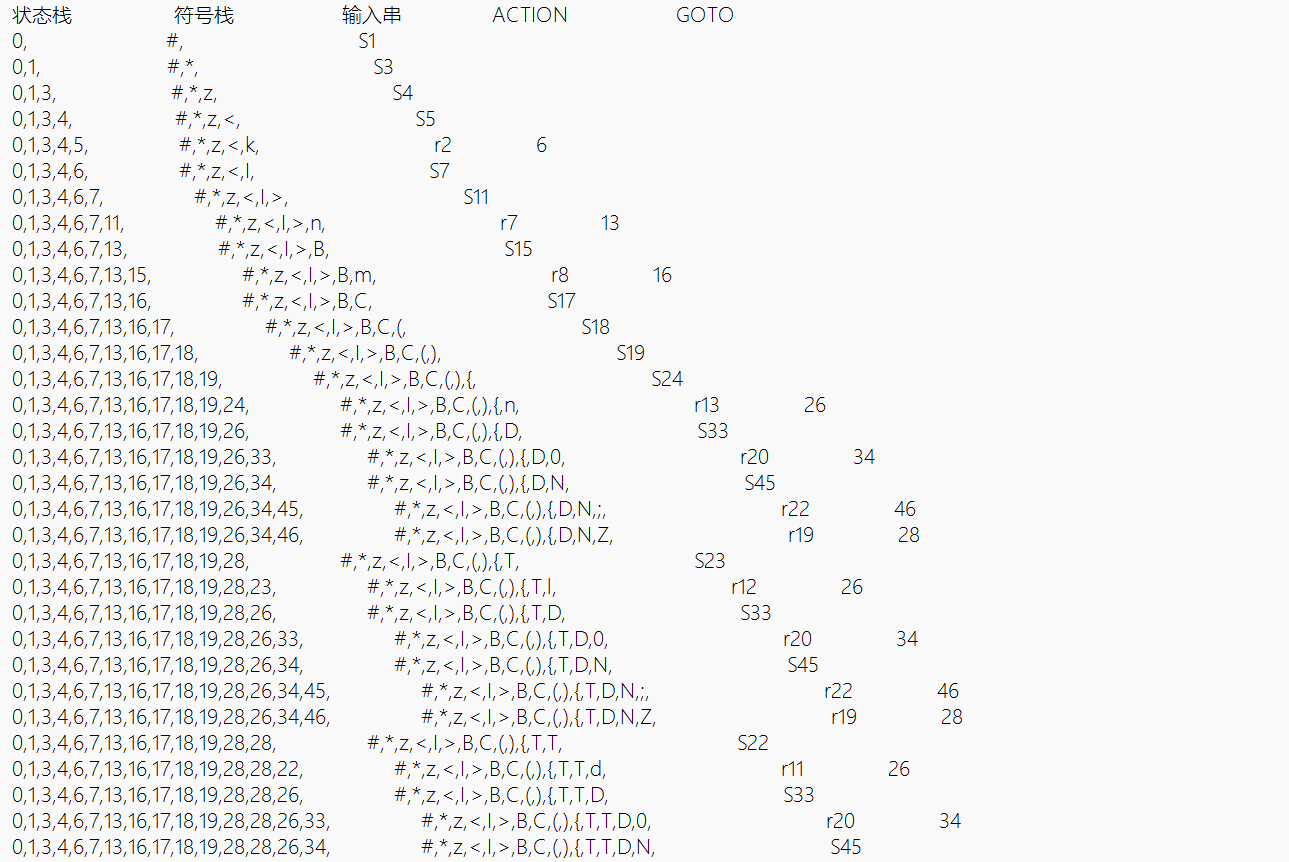
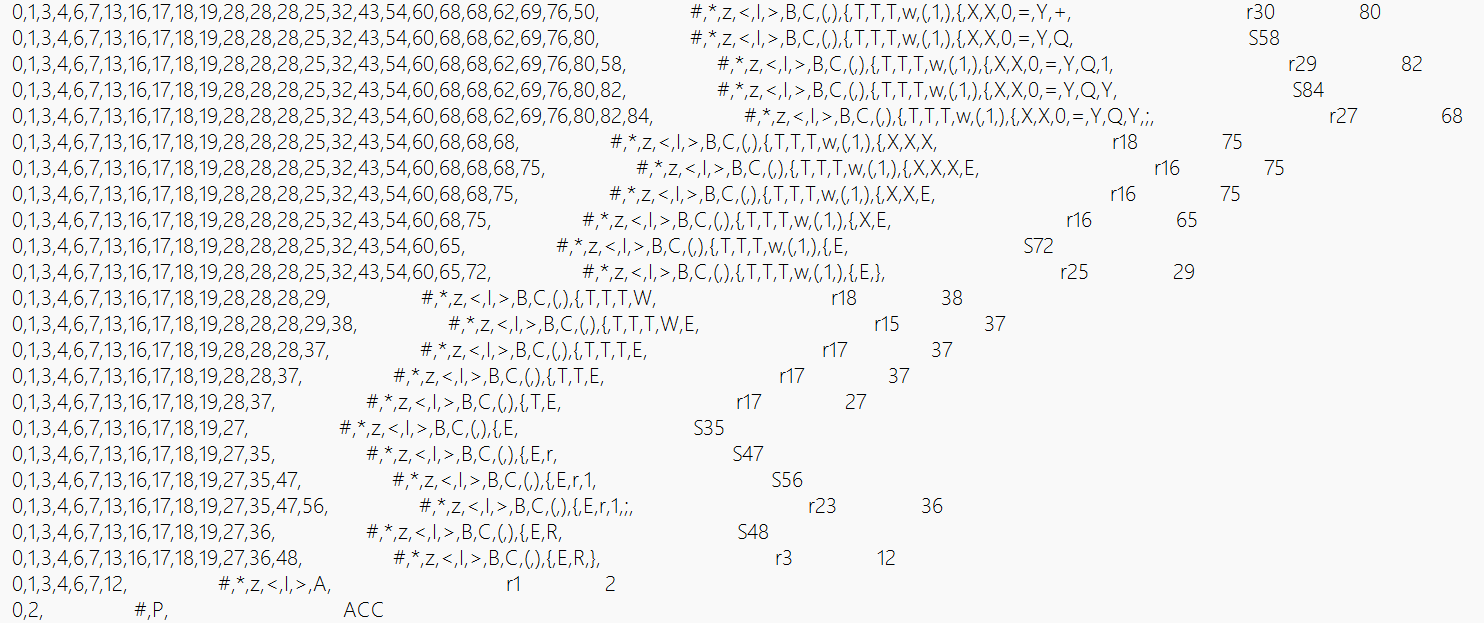
输入的二型文法：

输入的词法分析输出的中间文件：

输出的FIRST集：

输出的项目集族（由于太多只展示一部分）：

输出的分析表（未以表格形式呈现）：

输出的分析过程：

语法分析后的输出：

# 个人小结