

软件课程设计II实验报告

学 院： 计算机科学与工程学院

班 号： 9191069501

姓 名： 江楠

学 号： 919106840417

指导老师： 王永利

2022 年 4月

目录

[一、实验要求 1](#_Toc100933082)

[二、语言和文法 1](#_Toc100933083)

[2.1语言的设计 1](#_Toc100933084)

[2.2文法的设计 2](#_Toc100933085)

[三、程序实现 6](#_Toc100933086)

[3.1语言与环境 6](#_Toc100933087)

[3.2词法分析器 6](#_Toc100933088)

[3.3语法分析器 8](#_Toc100933089)

[3.4语义分析器 12](#_Toc100933090)

[四、实验结果 14](#_Toc100933091)

[4.1词法分析结果 14](#_Toc100933092)

[4.2语法分析结果 16](#_Toc100933093)

[4.3语义分析结果 17](#_Toc100933094)

## 一、实验要求

本次实验要求设计一个词法分析器、一个语法分析器和一个语义分析器，具体的如**表1**所示。

表1 实验要求清单表

|  |  |  |  |
| --- | --- | --- | --- |
|  | 任务1：词法分析器 | 任务2：语法分析器 | 任务3：语义分析器 |
| 输入 | 1）3型语法的文本文件   1. 源代码文本文件 | 1. 2型文法的文本文件 2. 任务1输出的token表 | 1）2型文法的文本文件  2）任务1输出的token表 |
| 输出 | 1. token表 | 1）YES/NO（是否符合文法） | 1）四元式序列 |
| 要求 | 1）使用NFA、DFA  2）可识别科学计数法  3）可检查常量和标识符的合法性 | 1）若有语法错误，则标出行号并给出大致的错误原因 | 1）可以简单以表达式计算语义为例 |

## 二、语言和文法

本次实验完成的程序支持输入各种文法与其配套的语言代码，以实现token表生成、语法分析和语义分析等操作。特别的，在提交的项目文件中给出的实验样例中语言和文法的设计呈现如下。

### 2.1语言的设计

#### 2.1.1语言的词法

本次实验仿造c/c++设计了一种语言。对于关键字、运算符和界符的采用字符串匹配方法，在程序对应set中加入需要的单词即可，目前程序中已有的如**表2**所示

表2关键字、运算符与界符表

|  |  |
| --- | --- |
| 关键字 | include, int, real, return, main, iostream, string, char, while, cstring |
| 运算符 | +, -, \*, /, %, ~, &, |, ^, =, >, <, ++, --, &&, ||, <<, >>, +=, -=, &=, |=, <=, >= |
| 界符 | , ( ) { } ; # |

对于标识符和常量采用形式化描述工具正规文法进行定义，在程序中识别正规文法生成NFA、DFA进行合法性判断。目前支持数学常量（含整数、小数、科学计数法等）、字符常量和字符串常量。标识符遵守首字符为字母或下划线的规则。

#### 2.1.2语言的语法

本次实验所设计的语言的语法类似于c/c++，在除include语句外都需要以分号结束。目前支持导入库语句、函数定义语句、while循环语句、算术计算、逻辑计算和赋值语句，具体的如**表3**所示。

表3 语法说明表

|  |  |  |
| --- | --- | --- |
| 序号 | 语法 | 说明 |
| 1 | #include lib | 导入名为lib的库 |
| 2 | type0 name(type1 a0, type2 a1, …){3~8} | 定义名为name的函数，形参个数不限；大括号内为函数内语句，可使用符合3~10号语法的语句 |
| 3 | return 5 | 函数返回语句，返回值可为符合3~10号语法的语句 |
| 4 | while(5){3~8} | while循环 |
| 5 | 算数运算 op 算数运算…; | op为逻辑运算符，为n（≥1）个算数运算的连接 |
| 6 | type name0 = 算数运算, name1 = 算数运算, …; | 定义变量，初始化赋值可省略 |
| 7 | name = 算数运算; | 赋值运算 |
| 8 | 算数运算； | 运算符、圆括号、标识符和常量组成的算数运算（包括仅标识符和仅常量的情况） |

### 2.2文法的设计

#### 2.2.1词法分析

为实现对五类单词字符中标识符和常量的识别，设计了如下文法（即\lexical\input\grammar.txt中的文法）。

S -> T

S -> -T

S -> +T

S -> P

S -> -P

S -> +P

S -> cQ

S -> eQ

S -> iQ

T -> dA

A -> dA

A -> .B

A -> eC

A -> +D

A -> $

B -> dE

E -> dE

E -> eC

E -> +D

E -> $

C -> dG

C -> +F

C -> -F

F -> dG

G -> dG

G -> +D

G -> $

D -> P

D -> -P

D -> +P

P -> dH

H -> dH

H -> .I

H -> eJ

H -> i

I -> dK

K -> dK

K -> eJ

K -> i

J -> dM

J -> +L

J -> -L

L -> dM

M -> dM

M -> i

Q -> cQ

Q -> dQ

Q -> eQ

Q -> iQ

Q -> $

其中d表示0~9的数字，c表示大写字母A~Z、小写字母a~z和下划线，$表示空（），大写字母为状态，其余字符的字面值即为输入字符。

#### 2.2.2语法分析

为实现语法分析，设计了如下文法（即\lexical\input\grammar.txt中的文法）。

M

M -> IF

I -> #ilI

I -> $

F -> TN(P){B}F

F -> $

T -> t

T -> c

T -> s

T -> a

N -> m

N -> 1

P -> T1Q

P -> $

Q -> ,T1Q

Q -> $

B -> w(E){B}B

B -> TK;B

B -> O;B

B -> rX;B

B -> $

E -> GfE

E -> G

K -> 1=XL

K -> 1L

L -> ,1=XL

L -> ,1L

L -> $

G -> X

G -> 1=X

G -> 1dX

G -> XeX

G -> (G)

O -> 1=X

O -> 1dX

O -> X

X -> J

X -> JoX

X -> J+X

X -> J-X

J -> H

J -> (X)

H -> 1

H -> 2

H -> p1

H -> -2

H -> -1

其中首行字母代表初态，f表示逻辑运算符，d表示赋值运算符，e表示比较运算符，o表示双目运算符，p表示单目运算符，l表示库关键字，$表示空（），1、2分别代表标识符和常量，大写字母为状态，其余字符的字面值即为输入字符。

#### 2.2.3语义分析

在语义分析中以表达式计算语义为例，设计了如下文法即/ \lexical\input\grammar.txt中的文法）。

E

E -> E+T { E := E + T }

E -> E-T { E := E - T }

E -> T { E := T }

T -> T\*F { T := T \* F }

T -> T/F { T := T / F }

T -> F { T := F }

F -> (E) { F := E }

F -> 1 { F := 1 }

F -> 2 { F := 2 }

其中首行字母代表初态，接下来每行中大括号包括的部分为语义，其中1、2分别代表标识符和常量，大写字母为状态，其余字符的字面值即为输入字符。

## 三、程序实现

### 3.1语言与环境

算法采用c/c++实现，在Windows10中用vs code开发，具体的如**表4**所示。

表4语言与环境表

|  |  |
| --- | --- |
| 语言 | c++14 |
| 编译器 | MinGW GCC 8.1.0 |
| 操作系统 | Windows10 |
| 编辑软件 | Visual Studio Code 1.59.1 |
| CPU | Intel CORE i7-9750H |

### 3.2词法分析器

#### 3.2.1宏与全局变量

词法分析器的宏与全局变量如**表7**所示。

表5 词法分析器宏与全局变量表

|  |  |
| --- | --- |
| 宏/全局变量 | 说明 |
| DIGITAL | 0~9的数字在文法中对应的符号 |
| LETTER | 大小写字母和下划线在文法中对应的符号 |
| EPSILON | 空（）在文法中对应的符号 |
| TERMINAL | 终态符号 |
| INITIAL\_STATUS | 初态在文法中对应的符号 |
| MAX\_STATUS | 状态数上限 |
| MAX\_INPUT\_LETTER | 输入字符的范围 |
| KEYWORD | 关键字种别编号 |
| IDENTIFIER | 标识符种别编号 |
| CONSTANT | 常量种别编号 |
| OPERATOR | 运算符种别编号 |
| DELIMITER | 界符种别编号 |
| const string grammar | 文法文件的路径 |
| const string codetext | 代码文件的路径 |
| const string tokens | tokens表的输出路径 |
| const string codetext\_simple | 专用于语义分析的代码文件的路径 |
| const string tokens\_simple | 用于语义分析的tokens表的输出路径 |
| const set<string> keyword | 关键字集合 |
| const set<char> opt\_ch | 单个字符的运算符集合 |
| const set<char> opt\_doubl | 相同两个字符的运算符和带运算的赋值运算符的运算部分的集合 |
| const set<char> delimiter | 界符集合 |
| const set<char> separator | 无意义的分隔符集合 |

#### 3.2.2函数说明

在词法分析器中设计了FA自定义类，实现对正规文法的读取和nfa到dfa的转换。具体的词法分析器函数说明见**表6**。

表6词法分析器函数说明表

|  |  |
| --- | --- |
| 函数 | 说明 |
| bool is\_letter(char ch) | 判断ch是否为字母或下划线。 |
| bool is\_digital(char ch) | 判断ch是否为数字。 |
| FA::void show\_dfa() | 输出dfa到控制台。 |
| FA::void show\_nfa() | 输出nfa到控制台。 |
| FA::void show\_input\_letter() | 输出输入符号表到控制台。 |
| FA::void show\_terminals() | 输出终态集到控制台。 |
| FA::bool cin\_nfa() | 从文件grammar输入正规文法，输入成功返回true，否则返回false。 |
| FA::void to\_dfa() | 将nfa转化成dfa。 |
| bool analyze(const vector<vector<int>> &dfa,const set<int> &terminals, const string &codetext, const string &tokens) | 根据codetext中的代码和dfa开始词法分析，输出结果到tokens文件。若文件打开失败或出现非法单词返回false，否则返回true。 |

#### 3.2.3程序流程

在analysis\_lexical.cpp词法分析器的主函数中，输入、分析以及输出的过程如**图1**所示。

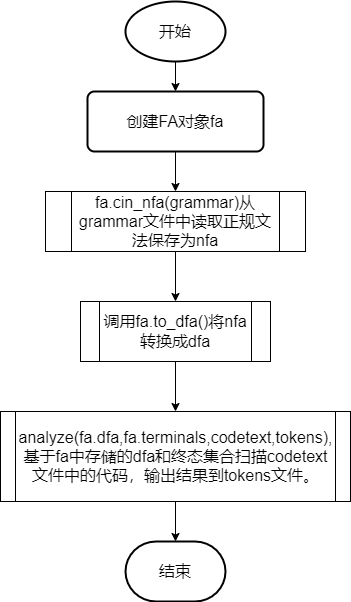


图1词法分析流程图

### 3.3语法分析器

#### 3.3.1宏与全局变量

语法分析器的宏与全局变量如**表7**所示。

表7词法分析器宏与全局变量表

|  |  |
| --- | --- |
| 宏/全局变量 | 说明 |
| LOGIC\_CONNECT\_OPT | 逻辑运算符文法中对应的符号 |
| ASSIGN\_OPT | 赋值运算符在文法中对应的符号 |
| CONPARE\_OPT | 比较运算符在文法中对应的符号 |
| BINO\_OPT | 双目运算符在文法中对应的符号 |
| MONO\_OPT | 单目运算符在文法中对应的符号 |
| LIB | 库关键字在文法中对应的符号 |
| EPSILON | 空（）在文法中对应的符号 |
| INITIAL\_SEARCH | 句子括号 |
| INITIAL\_STATE | 增广文法所用的初态 |
| KEYWORD | 关键字种别编号 |
| IDENTIFIER | 标识符种别编号 |
| CONSTANT | 常量种别编号 |
| OPERATOR | 运算符种别编号 |
| DELIMITER | 界符种别编号 |
| const string grammar | 文法文件的路径 |
| const string codetext | 输出codetext\_modified的路径 |
| const string tokens\_file | 词法分析器结果tokens表的路径 |
| const string projects\_file | 项目集规范族的输出路径 |
| const string analysis | 分析过程的输出路径 |
| vector<map<char,map<int, map<int, set<char>>>>> projects | 项目集规范族 |
| map<int, map<char, pair<char, int>>> action | action表 |
| vector<pair<int, char>> codetext\_modified | 根据mapping映射过后的代码串 |
| map<char, vector<string>> grammar\_map | 文法表 |
| map<char, set<char>> first\_set | FIRST集 |
| map<int, map<char, int>> go\_to | goto表 |
| map<string, char> mapping | 关键字、运算符的映射表 |

其中文法表元素形如grammar\_map[state][gm\_idx]存储的是以state为左部的第gm\_idx条文法的右部字符串。

项目集规范族元素形如projects[idx][state][gm\_idx][pos]存储的是idx号项目集中原点位置为pos的在grammar\_map中以state为左部的第gm\_idx条文法语句的向前搜索符集合。

action表元素形如action[cur\_idx][ch]存储的是cur\_idx项目集在输入ch时的操作。在规约项操作时存储的为规约语句的左部state和gm\_idx，以从grammar\_map查找归约语句；移进操作时存储的为-1与移进状态。

goto表元素形如goto[cur\_idx][state]存储的是cur\_idx项目集在非终结符state入栈时应移入状态栈的项目集编号。

#### 3.3.2函数说明

表8语法分析器函数说明表

|  |  |
| --- | --- |
| 函数 | 说明 |
| void generate\_mapping() | 生成关键字、运算符到文法中对应字符的映射。 |
| bool preprocess\_tokens(const string &codetext, const string &tokens\_file) | 将token\_file中的token通过mapping处理成codetext\_modified，并存入codetext文件中。 |
| void show\_codetext\_m() | 输出codetext\_modified到控制台。 |
| bool get\_grammar(const string &grammar) | 从文件grammar输入2型文法，输入成功返回true，否则返回false。 |
| void show\_grammar() | 输出文法到控制台。 |
| bool is\_upper(char ch) | 判断ch是否为大写字母。 |
| void get\_first\_set() | 根据文法得到FIRST集。 |
| void show\_first\_set() | 输出FIRST集到控制台。 |
| void get\_closure(map<char,map<int, map<int, set<char>>>> &np) | 获取项目及np的闭包。 |
| int same\_project(map<char,map<int, map<int, set<char>>>> &np) | 判断是否出现过相同的项目集，若出现过则返回对应项目集下标，否则返回-1。 |
| void get\_projects() | 获得项目集规范族。 |
| void show\_projects() | 输出项目集规范族到控制台。 |
| bool write\_projects(const string &projects\_file) | 输出项目集规范族到文件projects\_file。 |
| bool analyze(const string &analysis) | 开始词法分析，输出分析过程到analysis文件。若满足文法则返回true，否则返回false并输出可能的错误。 |

#### 3.3.3程序流程

在analysis\_phaser.cpp语法分析器的主函数的流程如**图2**所示。

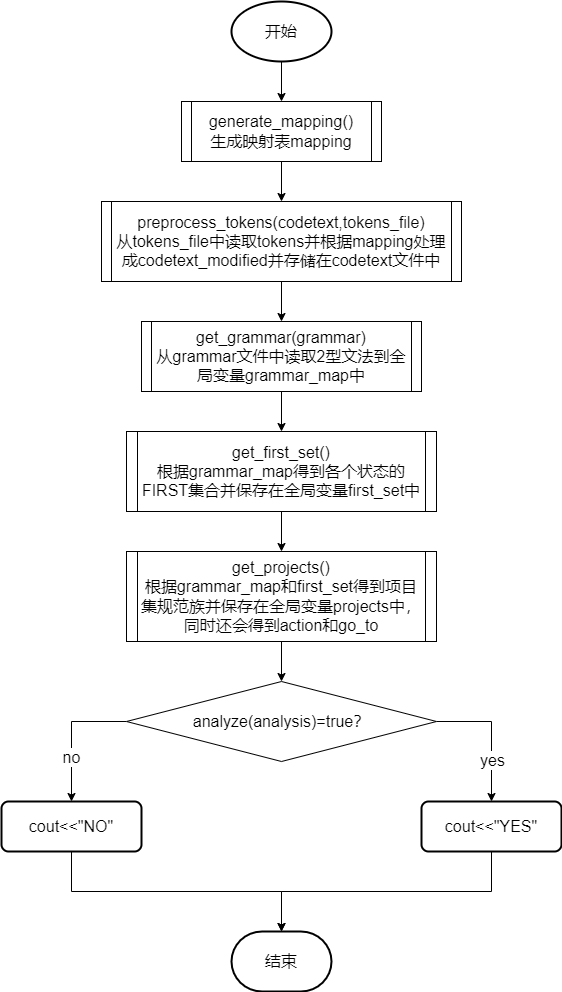


图2语法分析流程图

### 3.4语义分析器

#### 3.4.1宏与全局变量

表9词义分析器宏与全局变量表

|  |  |
| --- | --- |
| 宏/全局变量 | 说明 |
| EPSILON | 空（）在文法中对应的符号 |
| INITIAL\_SEARCH | 句子括号 |
| INITIAL\_STATE | 增广文法所用的初态 |
| KEYWORD | 关键字种别编号 |
| IDENTIFIER | 标识符种别编号 |
| CONSTANT | 常量种别编号 |
| OPERATOR | 运算符种别编号 |
| DELIMITER | 界符种别编号 |
| const string grammar | 文法文件的路径 |
| const string tokens\_file | 专用于语义分析的tokens表的路径 |
| const string projects\_file | 项目集规范族的输出路径 |
| const string analysis | 四元式的输出路径 |
| vector<map<char,map<int, map<int, set<char>>>>> projects | 项目集规范族 |
| map<int, map<char, pair<char, int>>> action | action表 |
| vector<string> tokens | tokens值的表 |
| vector<pair<int, char>> codetext\_modified | 根据mapping映射过后的代码串 |
| map<char, vector<string,string>> grammar\_map | 文法表 |
| map<char, set<char>> first\_set | FIRST集 |
| map<int, map<char, int>> go\_to | goto表 |

其中文法表元素形如grammar\_map[state][gm\_idx].first存储的是以state为左部的第gm\_idx条文法的右部字符串，grammar\_map[state][gm\_idx].second存储的为对应的语义。其余同名全局变量与语法分析中一致。

#### 3.4.2函数说明

表10 语义分析器函数说明表

|  |  |
| --- | --- |
| 函数 | 说明 |
| string to\_str(char ch) | 将char类型变量转换成string类型。 |
| bool preprocess\_tokens(const string &tokens\_file) | 将token\_file中的token处理成codetext\_modified，并报存token到tokens中。 |
| bool get\_grammar(const string &grammar) | 从文件grammar输入文法，输入成功返回true，否则返回false。 |
| void show\_grammar() | 输出文法到控制台。 |
| bool is\_upper(char ch) | 判断ch是否为大写字母。 |
| void get\_first\_set() | 根据文法得到FIRST集。 |
| void show\_first\_set() | 输出FIRST集到控制台。 |
| void get\_closure(map<char,map<int, map<int, set<char>>>> &np) | 获取项目及np的闭包。 |
| int same\_project(map<char,map<int, map<int, set<char>>>> &np) | 判断是否出现过相同的项目集，若出现过则返回对应项目集下标，否则返回-1。 |
| void get\_projects() | 获得项目集规范族。 |
| void show\_projects() | 输出项目集规范族到控制台。 |
| bool analyze(const string &analysis) | 开始语义分析，输出四元式到analysis文件。若满足文法则返回true，否则返回false并输出可能的错误。 |

#### 3.4.3程序流程

在analysis\_semantic.cpp语义分析器的主函数的流程如**图3**所示。

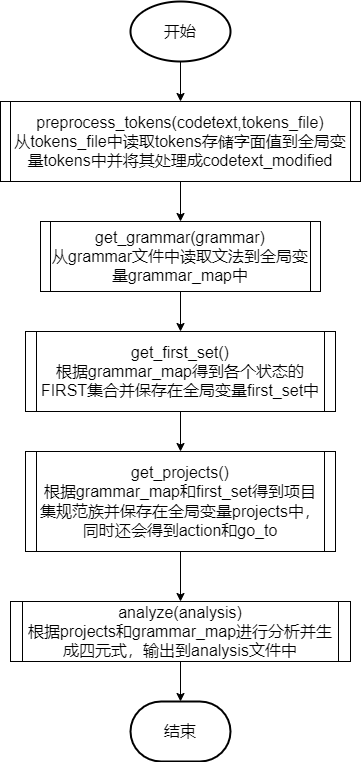


图3语义分析流程图

## 四、实验结果

### 4.1词法分析结果

词法分析中有两份输入代码以及其对应的输出tokens表。

|  |  |
| --- | --- |
| 输入代码1： | 输出1： |
| #include iostream  #include cstring  int myfunc(int a, int b){  return a \* b;  }  int main(){  string str = "aaa\\5555\"cz23x";  char ch = '\\', ch2 = '\'';  real a\_b\_g\_0 = -1e-9+1.0i;  real \_ab = 17.0 + a\_b\_g\_0;  real ab = \_ab;  int i = 0;  while(i < 10 ){  ++i;  }  int e = i \* \_ab;  e += i;  return 0;  } | (1 , # , 4 )  (1 , include , 0 )  (1 , iostream , 0 )  (2 , # , 4 )  (2 , include , 0 )  (2 , cstring , 0 )  (4 , int , 0 )  (4 , myfunc , 1 )  (4 , ( , 4 )  (4 , int , 0 )  (4 , a , 1 )  (4 , , , 4 )  (4 , int , 0 )  …（此处省略55行）  (15 , 10 , 2 )  (15 , ) , 4 )  (15 , { , 4 )  (16 , ++ , 3 )  (16 , i , 1 )  (16 , ; , 4 )  (17 , } , 4 )  (21 , } , 4 ) |

|  |  |
| --- | --- |
| 输入代码2： | 输出2： |
| a + b \* 10 + x - ( y / 20.1e9+10i ) | (1 , a , 1 )  (1 , + , 3 )  (1 , b , 1 )  (1 , \* , 3 )  (1 , 10 , 2 )  (1 , + , 3 )  (1 , x , 1 )  (1 , - , 3 )  (1 , ( , 4 )  (1 , y , 1 )  (1 , / , 3 )  (1 , 20.1e9+10i , 2 )  (1 , ) , 4 ) |

其中输入代码1即\lexical\input\codetext.txt中的代码，输出1即\lexical \output\tokens.txt中的tokens；输入代码2即\lexical\input\ codetext \_simple.txt中的代码，输出1即\lexical\output\tokens\_simple.txt中的tokens。

### 4.2语法分析结果

输入tokens表即词法分析中的输出1，分析过程保存于\phaser\output\ analysis.txt，项目集规范族保存于\phaser \output\projects.txt，映射后的token序列保存于\phaser\output \codetext\_modified.txt。由于输出过长截取部分，如**图4~6**所示。

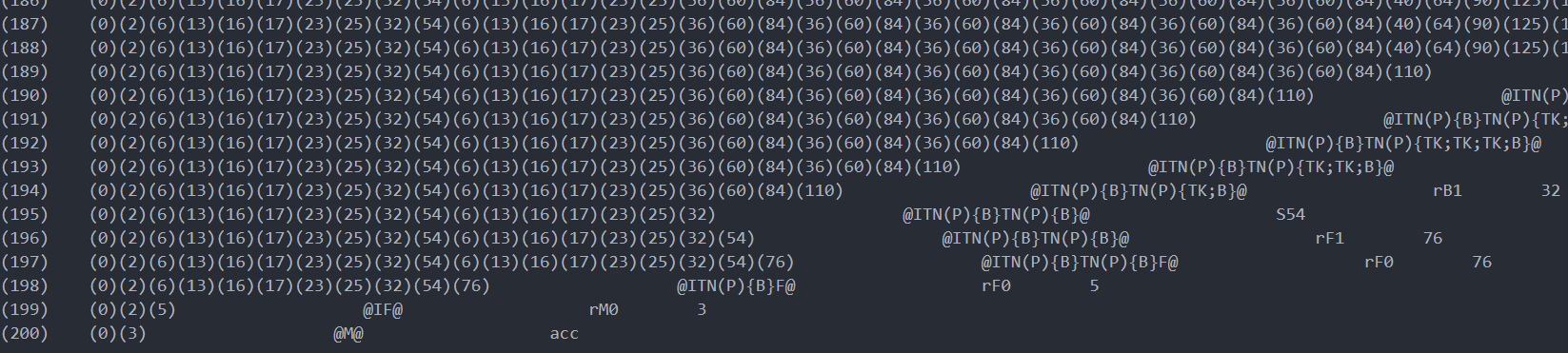


图4语法分析过程

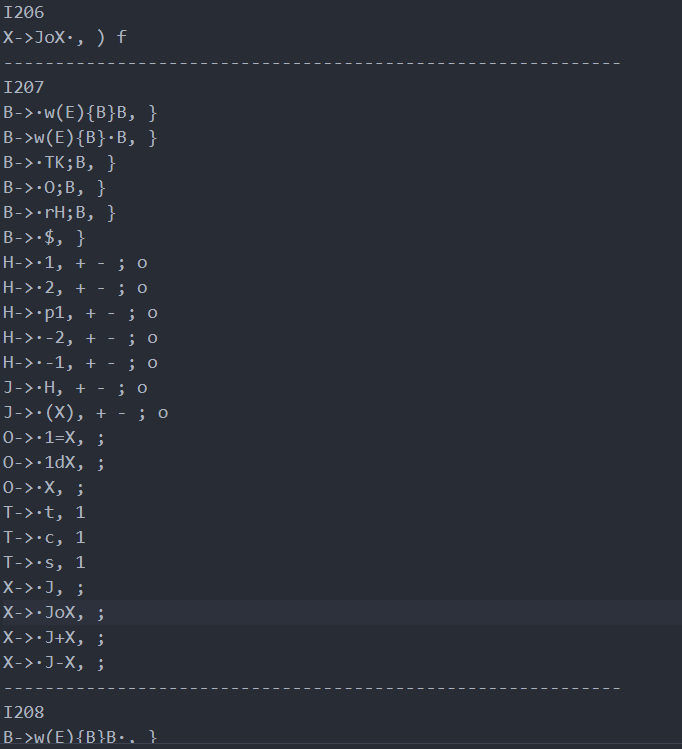


图5项目集规范族



图6映射后的token序列

语法错误示例：将输入代码中的iostream变为iostrea，在分析结果中会输出报错如**图7**所示。

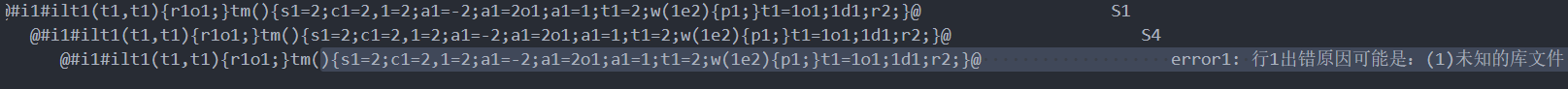


图7报错示例

### 4.3语义分析结果

输入tokens表即词法分析中的输出2，输出四元式保存于\semantic\output\ TAC.txt。具体的输出内容如下所示。

F1:=a

T1:=F1

E1:=T1

F2:=b

T2:=F2

F3:=10

T3:=T2\*F3

E2:=E1+T3

F4:=x

T4:=F4

E3:=E2+T4

F5:=y

T5:=F5

F6:=20.1e9+10i

T6:=T5/F6

E4:=T6

F7:=E4

T7:=F7

E5:=E3-T7