南京理工大学

软件课设（II）

智能科学与技术专业

姓名：高宏艳

学号：9191906840202

指导老师：王永利

目录

1.前言

1.1 实验目的

《编译原理》是计算机专业的一门重要的专业课程，其中包含大量软件设计细想。通过课程设计，实现一些重要的算法，或设计一个完整的编译程序模型，能够进一步加深理解和掌握所学知识，对提高自己的软件设计水平具有十分重要的意义。

1.2 开发环境

- 操作系统：Windows10
- 编程语言：C++
- 编译器：VsCode+Code Runner

2.词法分析器

2.1 题目要求

您必须使用 DFA（确定性有限自动机）或 NFA（不确定性有限自动机）来实现此程序。 程序有两个输入：

1. 一个文本文档，包括一组 3 语法（正规文法）的产生式；
2. 一个源代码文本文档，包含一组需要识别的字符串.程序的输出是一个 token（令牌）表，该表由 5 种 token 组成：关键词，标识符，常量，限定符和运算符。

词法分析程序的推荐处理逻辑 根据用户输入的正规文法，生成 NFA，再确定化生成 DFA，根据 DFA 编写识别 token 的程序，从头到尾从左至右识别用户输入的源代码，生成 token 列表（三元组：所在行号，类别，token 内容）。

要求

- 词法分析程序可以 准确识别 科学计数法形式的常量（如 0.314E+1），复数常量（如 10+12i）
- 可检查整数产量的合法性，标识符的合法性（首字符不能为数字等）
- 尽量符合真实常用高级语言（如 C++、Java 或 python）要求的规则

2.2 实现思路

本次实验中，采用了 DFA 来实现词法分析器，从左至右读取用户输入的“程序文本”，根据用户输入的正规文法识别 token 序列，生成三元组列表。

1.2.1 词法分析器分析过程以及相关数据结构

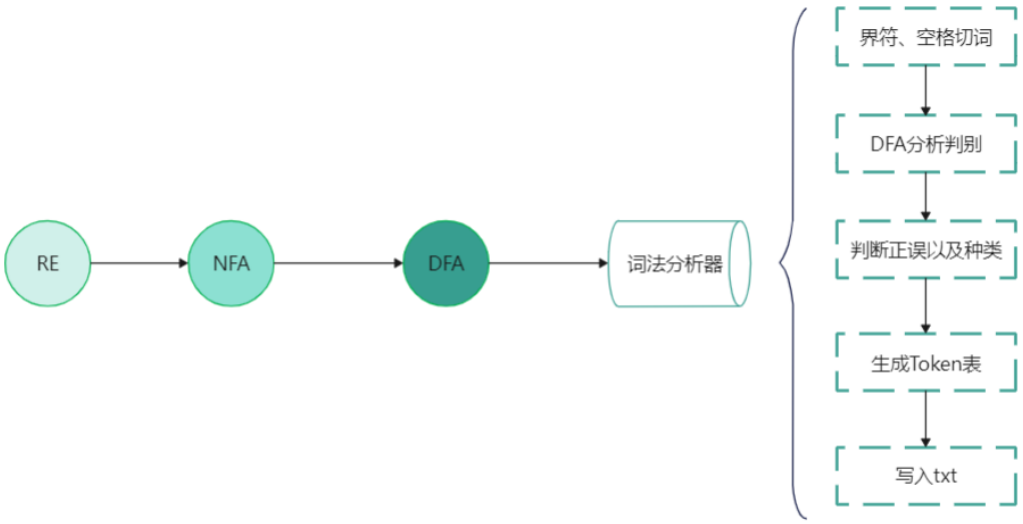


图 1 词法分析过程

从 lex_grammar.txt 中读取预先写好的文法，将其转换为 NFA，存储 NFA 的数据结构为三维结构体，结构体定义如下：

```
struct NFA{
    string start="START";
    map<string,vector<Twostate> >relations;//中间桥梁是 i 的关系
    vector<string>ends;
};
```

其中 Twostate 表示结点之间的连接关系，定义如下：

```
struct Twostate{
    string a;
    string b;
};
```

则 relations 存储的是以 key 连接起来的两两之间的状态集合。

程序中_init_()函数读取 txt 中的正规文法并转换成 NFA。

用子集法将 NFA 转换成 DFA，转换过程与课本中子集法转换过程是一致的，最终得到 DFA 用结构体存储如下：

```
struct DFA{
    set<string>start;
    vector<set<string> >closures;//状态形成的闭包
    vector<string>middle;
    vector<int>ends;
    int rels[200][200];//闭包有自己的代号
    DFA(){
        //初始化
        for(int i=0;i<200;i++){
            for(int j=0;j<200;j++){
                rels[i][j]=-1;
            }
        }
    }
};
```

程序中用 toDFA()将 NFA 转换成 DFA，moving(set<string>closure,string sign)得到当前闭包移动一个符号后可以抵达的下一个状态。

得到 DFA 后，读取 input.txt 文件，利用词法分析函数 Lex()对源码文件进行词法分析得到 Token 表，写入 output.txt

2.3 主要函数

2.3.1 RE 转 NFA

Init()函数读取两个文件，一个存放三型文法（词法规则）一个存放关键词，在读取文件数据的同时完成对全局变量 NFA 以及 keywords 的初始化。

```
1.  void _init_(){
2.      /*初始化得到 NFA 与关键词*/
3.      //读取关键词
4.      ifstream input;
5.      input.open(KFILE);
6.      if(!input.is_open()){
7.          printf("FAILED TO LOAD KEYWORDS! ");
8.      }else{
```

```
        while(input.getline(buffer,maxn-1)){
            string keyword=getRight(buffer);

            keywords.push_back(keyword);
        }
    }
    input.close();
    //大写的字母表示状态，''内的表示字符,小写的指代一类字符
    input.open(GFILE);
    if(!input.is_open()){
        printf("FAILED TO LOAD GRAMMAR!");
    }else{
        while(input.getline(buffer,maxn-1)){
            vector<string>temp=splitG(buffer);
            if(temp.size()==2){
                //TODO:这里默认 F->final
                nfa.ends.push_back(temp[0]);
                if(temp[1]!="final"){
                    temp[1]=getFromSign1(temp[1]);
                    middles.insert(temp[1]);

                }
            }else{
                //其余的情况是 F->fA
                //这里分特殊字符，或者统称
                if(temp[1]!="letter"&&temp[1]!="number"){
                    //这里需要对‘-’做特殊处理
                    temp[1]=getFromSign1(temp[1]);
                }
                middles.insert(temp[1]);
                Twostate two;
                two.a=temp[0];two.b=temp[2];
                nfa.relations[temp[1]].push_back(two);
            }
        }
    }
}
```

2.3.2 NFA 转 DFA

toDFA()将 NFA 转换成 DFA，过程与课本提供的伪代码一致：

```
void toDFA(){
    /*将 NFA 转化成 DFA*/
    set<set<string> >worked_closure;//已经标记过的 closure
```

```
4.      stack<set<string> >Closure;
5.      set<string>clo;//当前的 clo
6.      clo.insert(nfa.start);
7.      Closure.push(clo);
8.      //中间符号
9.      for(set<string>::iterator it=middles.begin();it!=middles.end();it
    ++){
10.         dfa.middle.push_back(*it);
11.     }
12.     //将第一个加入 dfa 中
13.     dfa.start=clo;
14.     dfa.closures.push_back(clo);//编号为 0
15.     int id,tid;
16.     //在处理的过程中需要更新 dfa
17.     while(!Closure.empty()){
18.         clo=Closure.top();
19.         Closure.pop();
20.         if(worked_closure.count(clo)!=0){
21.             //说明已经做过了
22.             continue;
23.         }else{
24.             worked_closure.insert(clo);
25.         }
26.         id=find(dfa.closures.begin(),dfa.closures.end(),clo)-
    dfa.closures.begin();//返回查询的下标
27.         if(isEnd(clo)){
28.             dfa.ends.push_back(id);
29.         }
30.         for(int i=0;i<dfa.middle.size();i++){
31.             string sign=dfa.middle[i];
32.             set<string>new_closure=moving(clo,sign);
33.             if(new_closure.size()==0){
34.                 continue;
35.             }
36.             if(worked_closure.count(new_closure)==0){//之前没有出现过这
    种闭包，则加入到待检测闭包中
37.                 Closure.push(new_closure);
38.                 if(count(dfa.closures.begin(),dfa.closures.end(),new_
    closure)==0)//原来也没有
39.                     dfa.closures.push_back(new_closure);
40.             }
41.             tid=find(dfa.closures.begin(),dfa.closures.end(),new_clos
    ure)-dfa.closures.begin();
42.
```

```
        }
    }
}
```

2.3.3 词法分析器

Lex(codefile)读取源码的 txt 文件并且对源码进行分析，主要思路是先用界符、空格对源码进行切割分词，再将切好的词送入 DFA 中看是否正常终止，对异常终止的情况，仍然可以生成对应的 token 但是需要标记上 false.

```
void Lex(string codefile){
    //处理
    ifstream input;
    input.open(codefile);
    if(!input.is_open()){
        printf("FAILED TO LOAD CODEFILES! ");
    }else{
        //按照分割符号读入
        int line=0;
        while(input.getline(buffer,maxn-1)){
            //按照空格、界符将字符串划分
            line++;
            vector<string>words=Splitby_tab_limit(buffer,line);
              //处理切好的词
            words2tokens(words,line);
        }
    }
}
```

2.3.4 其他函数

函数	说明
`set<string> moving(set<string>closure,string sign)`	Moving()状态转移
`bool isEnd(set<string>closure)`	是否是终态
`vector<string> Splitby_tab_limit(string str,int line)`	切词
`void words2tokens(vector<string> words,int line)`	将 string 转成最终的 token

2.4 实验结果

2.4.1 输入

正规文法 lex_grammar.txt	源码

[START] --> ['_'][A]
[START] --> [letter][A]
[A] --> [letter][A]
[A] --> [number][A]
[A] --> ['_'][A]
[A] --> [final]
[START] --> ['+'][B]
[START] --> ['-'][B]
[START] --> [number][B]
[START] --> [number][C]
[B] --> [number][B]
[B] --> ['.'][C]
[B] --> [number][C]
[C] --> [number][C]
[C] --> [final]
[B] --> [final]
[START] --> ['+'][D]
[START] --> ['-'][D]
[START] --> [number][D]
[D] --> [number][D]
[D] --> ['.'][E]
[D] --> [number][E]
[E] --> [number][E]
[E] --> ['E'][F]
[E] --> ['e'][F]
[F] --> ['+'][G]
[F] --> ['-'][G]
[F] --> [number][G]
[G] --> [number][G]
[G] --> [final]
[START] --> ['+'][H]
[START] --> ['-'][H]
[START] --> [number][H]
[H] --> [number][H]
	using namespace std;
	int main(){
	12+13i;
	int k==-12.46e78;
	89_;//或者” _89”
	cout<<x+k;
	return 0;
	}

[H] --> ['.'][I] [H] --> [number][I] [I] --> [number][I] [I] --> ['+'][J] [I] --> ['-'][J] [J] --> [number][J] [J] --> ['.'][K] [J] --> [number][K] [K] --> [number][K] [K] --> ['i'][L] [L] --> [final] [START] --> ['='][M] [M] --> [final] [M] --> ['='][X] [X] --> [final] [START] --> ['<'][N] [Y] --> [final] [N] --> ['<'][Y]	

1.4.2 中间生成

NFA:

```
Start Point:START
Show Relations:******************************
middle: +
START--B
START--D
F--G
START--H
I--J
middle: -
START--B
START--D
F--G
START--H
I--J
middle: .
B--C
D--E
H--I
J--K
middle: <
START--N
N--Y
middle: =
START--M
M--X
middle: E
E--F
middle: _
START--A
A--A
middle: e
E--F
```

DFA 处理结果：

问题 输出 调试控制台 终端

```
K--K
Show End Points:******************************
ACBGLMXYStart Points:
START
Closures:
id=0 :START
id=1 :B D H
id=2 :N
id=3 :M
id=4 :A
id=5 :B C D H
id=6 :C E I
id=7 :B C D E H I
id=8 :J
id=9 :F
id=10 :G
id=11 :K
id=12 :J K
id=13 :L
id=14 :X
id=15 :Y
Relations:
{START }--+:{B D H }
{START }---:{B D H }
{START }--<:{N }
{START }--=:{M }
{START }--_:{A }
{START }--letter:{A }
{START }--number:{B C D H }
{B D H }--.:{C E I }
{B D H }--number:{B C D E H I }
{N }--<:{Y }
{M }--=:{X }
{A }--_:{A }
{A }--letter:{A }
{A }--number:{A }
{B C D H }--.:{C E I }
{B C D H }--number:{B C D E H I }
{C E I }--+:{J }
{C E I }---:{J }
{C E I }--E:{F }
```

1.4.3 输出

可以看到，对于源码中的词，程序可以正确识别。输出利用的 C++语法高亮来显示 token 三元组，对于错误的词用红色进行标注。

问题 输出 调试控制台 终端

```
------------------------------

TOKENS                          LINE                            TYPE
using                           1                               keyword
namespace                                       1                  keyword
std                             1                               sign
;                               1                               limited
int                             2                               keyword
main                            2                               keyword
(                               2                               limited
)                               2                               limited
{                               2                               limited
12+13i                          3                               const
;                               3                               limited
int                             4                               keyword
k                               4                               sign
==                              4                               operator
-12.46e78                                       4                  const
;                               4                               limited
89_                             5                               FALSE!
;                               5                               limited
cout                            6                               keyword
<<                              6                               operator
x                               6                               sign
+                               6                               operator
k                               6                               sign
;                               6                               limited
return                          7                               keyword
0                               7                               const
;                               7                               limited
}                               8                               limited
current: 2022-04-15 21:15:06
Total Time Cost:1
```

图 2.1 词法分析结果（含错误变量示例）

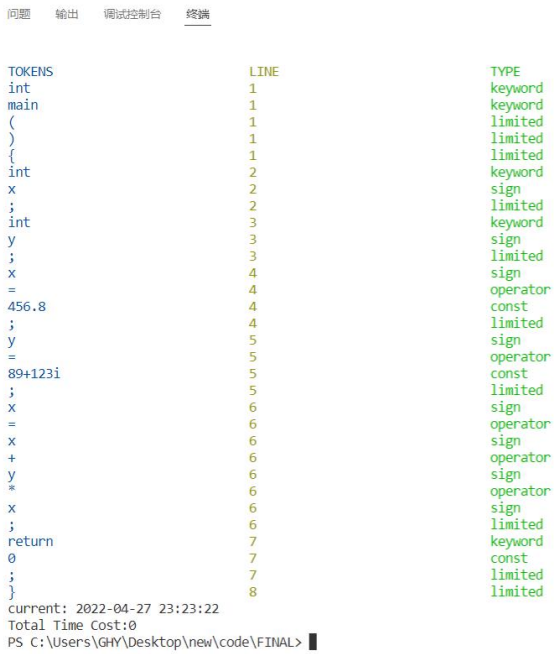


图 2.1 词法分析结果(正确源码示例)

3. 语法分析器

3.1 题目要求

创建一个使用 LL(1) 方法或 LR(1) 方法的语法分析程序。

程序有两个输入

1. 一个是文本文档，其中包含 2º 型文法（上下文无关文法）的产生式集
2. 任务 1 词法分析程序输出的（生成的）token 令牌表。程序的输出包括：YES 或 NO（源代码字符串符合此 2º 型文法，或者源代码字符串不符合此 2º 型文法）；错误提示文件，如果有语法错标示出错行号，并给出大致的出错原因。

语法分析程序的推荐处理逻辑：根据用户输入的 2º 型文法，生成 Action 及 Goto 表，设计合适的数据结构，判断 token 序列（用户输入的源程序转换）是否符合 LR1 文法规则，识别 token 序列，输出结果。

3.2 实现思路

3.2.1 语法分析器分析过程以及相关数据结构

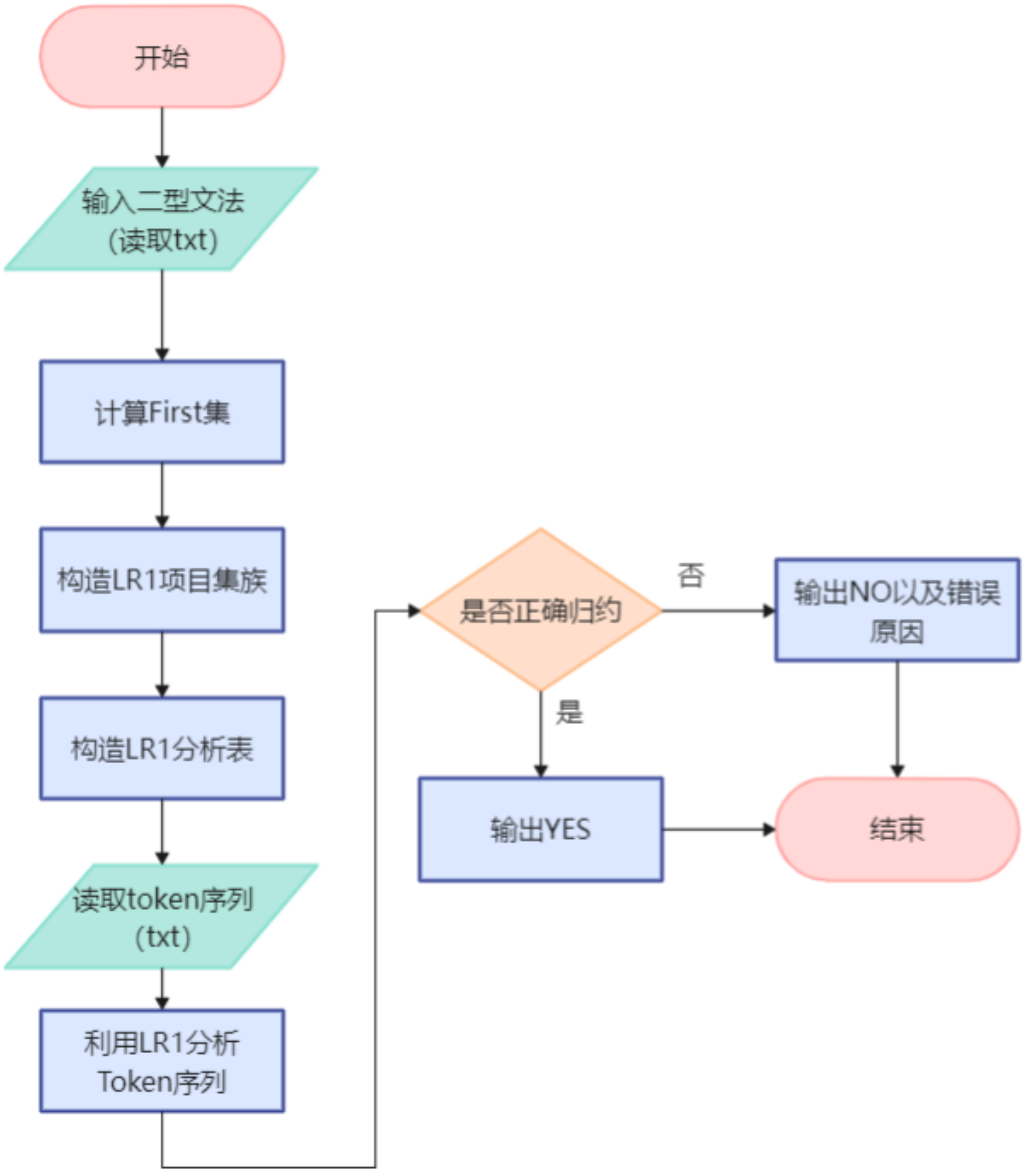


图 3 语法分析过程

预先写好的文本文件 Syn_grammar.txt 中存有语法规则二型文法，读取该文件，保存在存放语法规则的结构体中：

```
struct Grammar{//保存语法规则
    string Left;
    vector<string>Rights;
};
```

计算 First 集，具体实现过程为：不停的扫描每个文法，其首个字符是终结符，则加入到对应的非终结符的 first 集中，如果首个字符是非终结符，则扫描

该非终结符的 first 集，把他们全部加入到当前非终结符的 first 集中，为空，则扫描下一个字符。直到某一次没有发生任何改动，说明 first 集计算完成。

求 LR（1）项目集族。该计算过程和书上的求法一致，从 S‘->S 开始，构造第一个项目集，通过求闭包，构造转换函数，期间注意项目集内部元素的重复和项目集之间的重复，保证不重复，可以求出所有的项目集。

构造 LR（1）分析表，结构体如下：

```
struct Table_Item{//LR(1)分析表
    int STATE;
    map<string,int>ACTION;
    map<string,int>GOTO;
};
```

这里 ACTION 和 GO 表我用了两个 map 分别表示，同时为了方便规约项目用负数编号，在后续编程中便于判别。

3.3 主要函数

3.3.1 计算 First 集合

在初始化的_init_()过程中计算 First 集合

```
void _init_(){
    ifstream input;
    input.open(grammarfile);
    if(!input.is_open()){
        printf("FAILED TO LOAD GRAMMARS\n");
    }else{
        while(input.getline(buffer,maxn-1)){
            grammars.push_back(getLineGrammar(buffer));
        }
    }
    int cnt=1;
    while(cnt){
        cnt=0;
        for(set<string>::iterator it=Vn.begin();it!=Vn.end();it++){
            if(getFirst_init(*it)==false){
                cnt++;
            }
        }
    }
}
```

计算 First 集合的主要函数 getFirst_init(string vn)

```
bool getFirst_init(string vn){
    set<string>ans;
    set<string>nu;
```

```
    nu.insert("final");
    bool flag=true;

    for(int i=0;i<grammars.size();i++){
       if(grammars[i].Left==vn){
           string f=grammars[i].Rights[0];
           int len=grammars[i].Rights.size();
           int cnt=0;
           while(cnt<len){
               if(isVT(f)){
                    ans.insert(f);
                    break;
                }else if(first_set.find(f)!=first_set.end()){

                    if(toNull.find(f)==toNull.end()){

                    set<string>temp;
                    set_difference( first_set[f].begin(), first_set[f].end(),nu.begin(), nu.end(),inserter( temp, temp.begin() ) );
                    ans.insert(temp.begin(),temp.end());
                    cnt++;
                    }else{

                        ans.insert(first_set[f].begin(),first_set[f].end());
                        break;
                    }
                }else{
                    flag=false;
                    break;
                }
           }
           if(cnt>=len){

               ans.insert("final");
           }
       }
    }
    if(flag){
        first_set[vn]=ans;
    }
    return flag;

}
```

3.3.2 求 LR(1)项目集族

构造 LR(1)项目集族的函数 **build_LR():**

```
1.  void build_LR(){
2.
3.      Item item;
4.      item.Left=grammars[0].Left;
5.      item.Rights=grammars[0].Rights;
6.      item.pos=0;
7.      item.Tail="final";
8.      Closure clo;
9.      clo.items.insert(item);
10.     clo.state=0;
11.     getClosure(clo);
12.     closures.insert(clo);
13.     stack<Closure>st;
14.     st.push(clo);
15.     Closure temp_clo;
16.     int cnt=0;
17.     while(!st.empty()){
18.         clo=st.top();
19.         st.pop();
20.         int nid=clo.state;
21.         int pid;
22.
23.         set<string>strs;
24.         for(set<Item>::iterator it=clo.items.begin();it!=clo.items.en
    d();it++){
25.             string nx=getNext(*it);
26.             if(nx!="[FINAL]"&&nx!="[DANGER]"){
27.                 strs.insert(nx);
28.             }
29.         }
30.         //Go,
31.         printf("**********************next str:%d\n",strs.size());
32.         for(set<string>::iterator it=strs.begin();it!=strs.end();it++
    ){
33.             temp_clo=Go(clo,*it);
34.             getClosure(temp_clo);
35.             if(temp_clo.items.size()==0){
36.
37.                 continue;
```

```
            }

            set<Closure>::iterator cit=find(closures.begin(),closures.end(),temp_clo);
            if(cit!=closures.end()){

                pid=(*cit).state;
            }else{

                cnt++;
                pid=cnt;
                temp_clo.state=cnt;
                closures.insert(temp_clo);
                st.push(temp_clo);
            }

            printf("%d %d\n",pid,nid);
            rels[nid].push_back(pid);
            relations[nid][pid].v.insert(*it);
        }
        printf("ok\n");

    }
}
```

3.3.3 求 LR(1)分析表

基于 LR(1) 项目集族构造 LR(1) 分析表的函数 build_Table()：

```
void build_Table(){

    for(set<Closure>::iterator it=closures.begin();it!=closures.end();it++){
        Table_Item titem;

        int nid=(*it).state;
        titem.STATE=nid;
        for(int i=0;i<rels[nid].size();i++){
            int j=rels[nid][i];
            for(set<string>::iterator it=relations[nid][j].v.begin();it!=relations[nid][j].v.end();it++){
                string str=*it;
                if(isVT(str)){
                    titem.ACTION[str]=j;
```

```
                }else{
                    titem.GOTO[str]=j;
                }
            }
        }

        for(set<Item>::iterator tit=(*it).items.begin();tit!=(*it).items.end();tit++){
            if((*tit).pos==(*tit).Rights.size()){//A->a
                int rid=getRid((*tit).Left,(*tit).Rights);
                titem.ACTION[(*tit).Tail]=rsign+rid;
            }
        }
        table.tabel_item.push_back(titem);
    }
}
```

3.3.4 其他函数

函数	说明
`Grammar getLineGrammar(string str)`	读取二型文法
`bool isVT(string str)`	是否是 VT
`bool isVn(string str)`	是否是 VN
`Closure Go(Closure clo,string str)`	GO
`set<string> getFirst(vector<string>ba)`	获取对应的 First 集合
`void getClosure(Closure &clo)`	获取对应的集合
`int getRid(string Left,vector<string>Rights)`	获取 R 值
`int getGotoId(int state_id,string str)`	获取 goto 表对应的值

3.4 实验结果

3.4.1 输入

Syn_token.txt	Syn_token_false.txt	语法规则.txt
2 int keyword	2 int keyword	[START] --> [X]
2 main keyword	2 main keyword	[X] --> ['int']['main']['(']['(']['{'][BLOCK]['}']
2 (limited	2 (limited	[BLOCK] --> [const][';']
2) limited	2) limited	[BLOCK] --> [const][';'][BLOCK]
2 { limited	2 { limited	[BLOCK] --> [FORMULA][';'][BLOCK]
3 12+13i const	3 12+13i const	[FORMULA] --> ['int'][sign]['='][F]
3 ; limited	3 ; limited	[F] --> [E]['+'][F]
4 int keyword	4 int keyword	[F] --> [E]
4 k sign	4 k sign	[E] --> [N]['*'][N]
4 = operator	4 = operator	[E] --> [N]
4 -12.46e78 const	4 -12.46e78 const	[N] --> [sign]
5 ; limited	5 ; limited	[N] --> [const]
6 int keyword	6 int keyword	[BLOCK] --> [RETURN]
6 y sign	6 y sign	[RETURN] --> ['return'][const][';']
6 = operator	6 = operator	[RETURN] --> [final]
6 x sign	6 x sign	[X] -->[final]
6 + operator	6 + operator	
6 k sign	6 k sign	
6 ; limited	6 ; limited	
7 return keyword	7 return keyword	
7 0 const	7 0 const	
7 ; limited	8 } limited	
8 } limited		

3.4.2 中间生成

处于篇幅限制，这里仅展示部分截图。
1、项目集生成

```
-----------------------------------show----------------------------------------
************************************34
0:
START : X                       |final
X : final                       |final
X : 'int','main','(',')','{',BLOCK,'}'                  |final
1:
X : 'int','main','(',')','{',BLOCK,'}'                  |final
2:
START : X                       |final
3:
X : 'int','main','(',')','{',BLOCK,'}'                  |final
4:
X : 'int','main','(',')','{',BLOCK,'}'                  |final
5:
X : 'int','main','(',')','{',BLOCK,'}'                  |final
6:
BLOCK : RETURN                  |'}'
BLOCK : const,';'                       |'}'
BLOCK : FORMULA,';',BLOCK                       |'}'
BLOCK : const,';',BLOCK                 |'}'
FORMULA : 'int',sign                    |';'
FORMULA : sign,'=',F                    |';'
FORMULA : 'int',sign,'=',F                      |';'
RETURN : final                  |'}'
RETURN : 'return',const,';'                     |'}'
X : 'int','main','(',')','{',BLOCK,'}'                  |final
7:
FORMULA : 'int',sign                    |';'
FORMULA : 'int',sign,'=',F                      |';'
8:
RETURN : 'return',const,';'                     |'}'
```

2、LR(1)分析表生成：

```
-------------------------------------------------------------
0:
ACTION:
['int']:1
GOTO:
[X]:2
-------------------------------------------------------------
1:
ACTION:
['main']:3
GOTO:
-------------------------------------------------------------
2:
ACTION:
[final]:-1000
GOTO:
-------------------------------------------------------------
3:
ACTION:
['(']:4
GOTO:
-------------------------------------------------------------
4:
ACTION:
[')']:5
GOTO:
-------------------------------------------------------------
5:
ACTION:
['{']:6
GOTO:
-------------------------------------------------------------
6:
ACTION:
['int']:7
['return']:8
[const]:12
[sign]:13
GOTO:
```

3.4.3 输出

输入是 syn_token.txt：YES

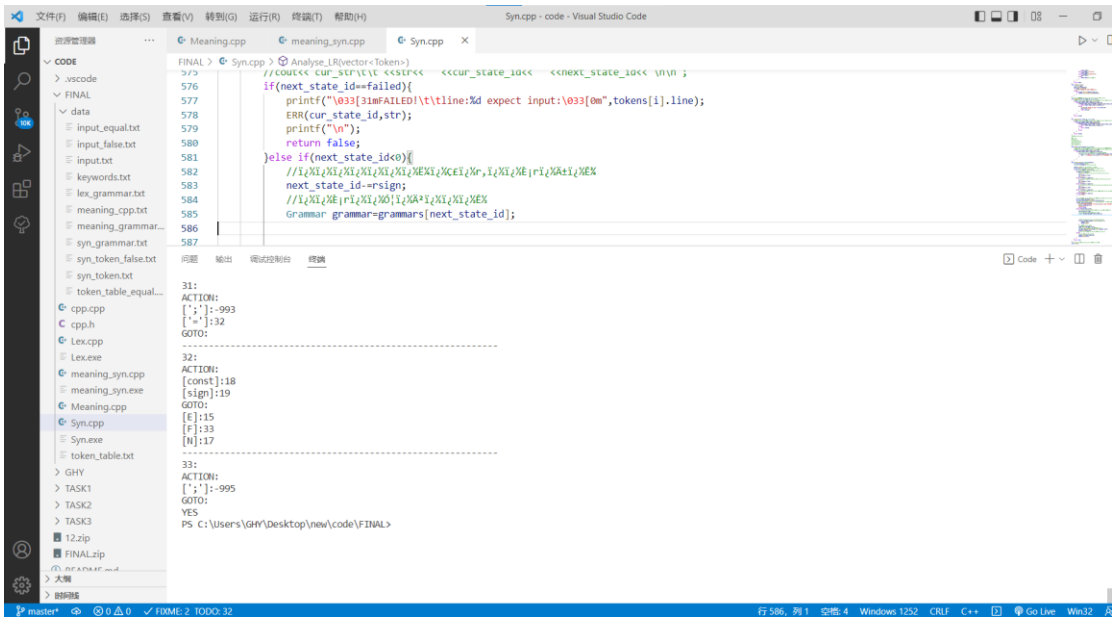


图 4.1 语法分析正确 token 序列截图

输入是 syn_token_false.txt： NO，并给出错误原因分析

```
29 Failed to find '}'!
FAILED!         line:8 expect input:';'

NO
PS C:\Users\GHY\Desktop\new\code\FINAL>
```

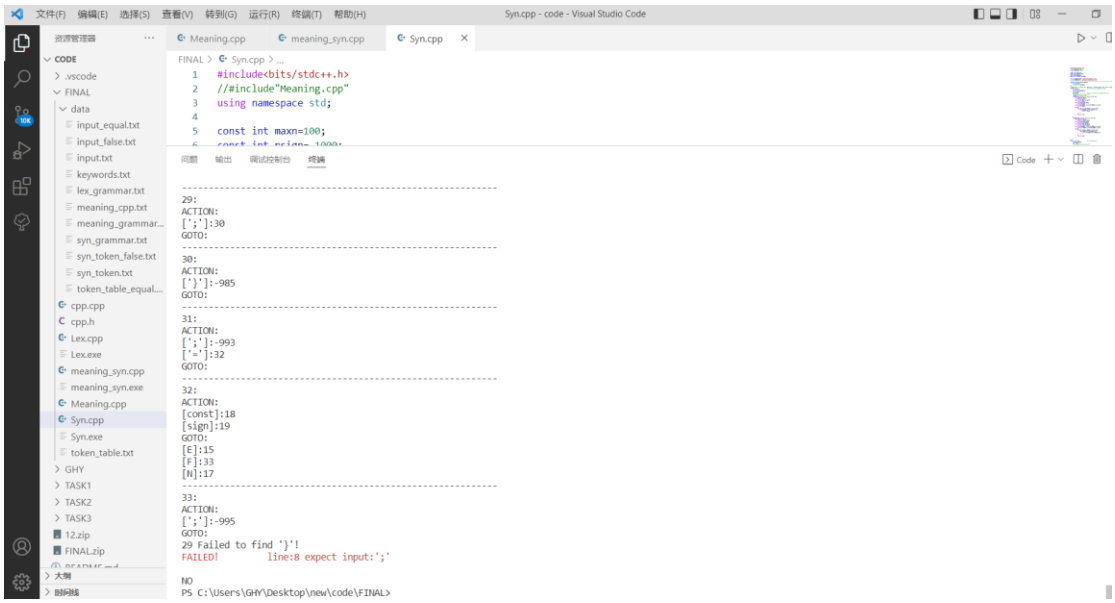


图 4.2 语法分析错误 token 序列截图

4. 语义分析器

4.1 题目要求

创建符合属性文法规则的语义分析程序。 程序有两个输入

1. 一个是文本文档，其中包含 2º 型文法（上下文无关文法+属性文法，包含语义规则注释，可以简单以表达式计算语义为例）的产生式集合；
2. 任务 1 词法分析程序输出的（生成的）token 令牌表。 程序输出：四元式序列，可以利用优化技术生成优化后的四元式序列 提示：也可以利用 Flex 工具设计功能更加丰富的语义分析程序。

4.2 实现思路

基于语法分析，分析当前目录下 input.txt 文件中的语义并生成中间代码。实现的主要功能有：

- 算术表达式的语义识别
- 简单赋值语句的语义识别
- 算术表达式的四元式生成
- 简单赋值语句的四元式生成

4.2.1 语义分析器分析过程以及相关数据结构

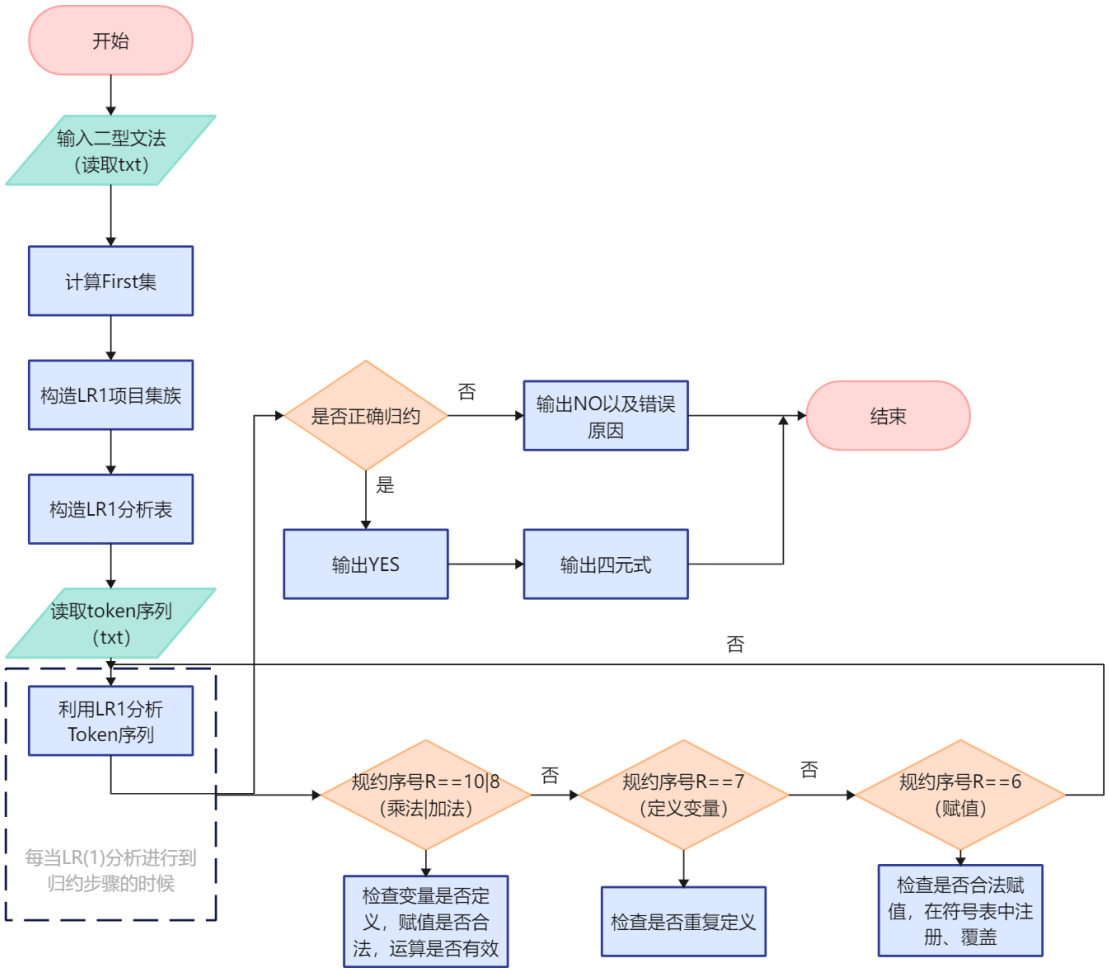


图 5 语义分析器分析过程

任务三的函数流程总体上与任务二一致，唯一的区别在于，本次实验在 LR(1)分析的过程中，同时进行了语义分析，生成符号表以及四元组序列，对错误语义进行报错提示，对于正确的句法成功输出四元式语句。

其中符号表的结构体定义如下：

```
struct SignTable{
//符号表
string name="";//变量名称
string type;
string val;
};
```

四元式的结构体定义如下：

```
  struct Quaternary{
      string op;
      string arg1;
```

```
        string arg2;
        string result;
    };
```

4.3 主要函数

4.3.1 语义分析函数

LR(1)每次规约的时候，需要将当前规约的序号 R 输入给语义分析函数 meaning_process()进行语义分析，此函数在 Meaning.cpp 下，属于主程序 Meaning_syn.cpp 外部引用文件.

```
Quaternary meaning_process(int id,stack<Token>&tokens,vector<SignTable>&signtable){
    Quaternary temp;
    temp.op="";
    switch(id){
        case 7:{
            //[FORMULA] --> ['int'][sign]定义语句
            tokens.pop();
            string name=tokens.top().word;
            tokens.pop();
            string type=tokens.top().word;
            cout<<"7:"<<name<<" "<<type<<endl;
            tokens.pop();
            int p=lookup(name,signtable);
            if(p!=-1){
                //说明之前被注册过了
                ERROR(0,name);
            }else{
                //之前没有注册，现在注册
                SignTable ns;
                ns.name=name;
                ns.type=type;
                signtable.push_back(ns);
            }
            break;
        }
        case 12:{
            //赋值语句中的一些: [N] --> [sign]

            break;
```

```
30.         }
31.         case 10:{
32.             //[E] --> [N]['*'][E]
33.             tokens.pop();
34.             Token arg1=tokens.top();
35.             tokens.pop();
36.             tokens.pop();
37.             Token arg2=tokens.top();
38.             tokens.pop();
39.             bool flag=true;
40.             if(arg1.type!=3){
41.                 int p=lookup(arg1.word,signtable);
42.                 if(p==-1){
43.                     //如果没有注册就参与运算
44.                     ERROR(2,arg1.word);
45.                     flag=false;
46.                 }else if(signtable[p].val==""){
47.                     //如果注册了但是没有赋值就参与运算
48.                     ERROR(1,arg1.word);
49.                     flag=false;
50.                 }
51.
52.             }
53.             if(arg2.type!=3){
54.                 int p=lookup(arg2.word,signtable);
55.                 if(p==-1){
56.                     //如果没有注册就参与运算
57.                     ERROR(2,arg2.word);
58.                     flag=false;
59.                 }else if(signtable[p].val==""){
60.                     //如果注册了但是没有赋值就参与运算
61.                     ERROR(1,arg2.word);
62.                     flag=false;
63.                 }
64.
65.             }
66.             if(flag){
67.                 string a,b;
68.                 if(arg1.type==3){
69.                     a=arg1.word;
70.                 }else{
71.                     int p=lookup(arg1.word,signtable);
72.                     a=signtable[p].val;
73.                 }
```

```
74.                 if(arg2.type==3){
75.                     b=arg2.word;
76.                 }else{
77.                     int p=lookup(arg2.word,signtable);
78.                     b=signtable[p].val;
79.                 }
80.                 SignTable res=MUL(a,b);
81.                 signtable.push_back(res);
82.                 Token nt;
83.                 nt.word=res.name;
84.                 nt.type=4;
85.                 tokens.push(nt);
86.                 temp=emit(res.name,arg1.word,arg2.word,"*");
87.             }
88.
89.             break;
90.         }
91.         case 8:{
92.             //[F] --> [E]['+'][F]
93.             //TODO: 这边有点问题，继续规约的时候，不可以直接去除规约符号
94.
95.             if(tokens.top().word==";"){
96.                 tokens.pop();
97.             }
98.             Token arg1=tokens.top();
99.             tokens.pop();
100.                tokens.pop();
101.                Token arg2=tokens.top();
102.                tokens.pop();
103.                bool flag=true;
104.                if(arg1.type!=3){
105.                    int p=lookup(arg1.word,signtable);
106.                    if(p==-1){
107.                        //如果没有注册就参与运算
108.                        ERROR(2,arg1.word);
109.                        flag=false;
110.                    }else if(signtable[p].val==""){
111.                        //如果注册了但是没有赋值就参与运算
112.                        ERROR(1,arg1.word);
113.                        flag=false;
114.                    }
115.
116.                }
117.                if(arg2.type!=3){
```

```
118.                    int p=lookup(arg2.word,signtable);
119.                    if(p==-1){
120.                        //如果没有注册就参与运算
121.                        ERROR(2,arg2.word);
122.                        flag=false;
123.                    }else if(signtable[p].val==""){
124.                        //如果注册了但是没有赋值就参与运算
125.                        ERROR(1,arg2.word);
126.                        flag=false;
127.                    }
128.
129.                }
130.                if(flag){
131.                    string a,b;
132.                    if(arg1.type==3){
133.                        a=arg1.word;
134.                    }else{
135.                        int p=lookup(arg1.word,signtable);
136.                        a=signtable[p].val;
137.                    }
138.                    if(arg2.type==3){
139.                        b=arg2.word;
140.                    }else{
141.                        int p=lookup(arg2.word,signtable);
142.                        b=signtable[p].val;
143.                    }
144.
145.                    SignTable res=ADD(a,b);
146.                    signtable.push_back(res);
147.                    Token nt;
148.                    nt.word=res.name;
149.                    nt.type=4;
150.                    tokens.push(nt);
151.                    temp=emit(res.name,arg1.word,arg2.word,"+");
152.                }
153.
154.
155.                break;
156.            }
157.            case 6:{
158.                //[FORMULA] --> [sign]['='][F]赋值语句
159.                if(tokens.top().word==";"){
160.                    tokens.pop();
161.                }
```

```
162.
163.                 string val=tokens.top().word;
164.
165.                 tokens.pop();
166.                 string a=tokens.top().word;
167.                 tokens.pop();
168.                 string name=tokens.top().word;
169.                 cout<<name<<"|"<<val<<"|"<<a<<endl;
170.                 tokens.pop();
171.                 int p=lookup(name,signtable);
172.                 if(p==-1){
173.                     ERROR(3,name);
174.                 }else{
175.                     //注册值或者更改值
176.                     signtable[p].val=val;
177.                     temp=emit(val,name,"_","=");
178.                 }
179.                 break;
180.             }
181.             case 11:{
182.                //[E] --> [N]
183.
184.                 break;
185.             }
186.             case 9:{
187.                 //[F] --> [E]
188.
189.                 break;
190.             }
191.             default:{
192.                 //看语法，grammar 前面有几个就打几个 7 12 10 8 6
193.                 break;
194.             }
195.         }
196.         return temp;
197.     }
```

4.3.2 其他函数

函数名称	功能
`SignTable MUL(string a,string b)`	乘法语义分析
`SignTable ADD(string a,string b)`	加法语义分析

`void ERROR(int type,string name)`	定义不同类型的报错
`Quaternary emit(string result,string arg1,string arg2,string op)`	生成四元式
`string newtemp()`	返回一个新的临时变量
`int lookup(string name,vector<SignTable>signtable)`	查找符号表，返回对应的下标，若不在则返回-1

4.4 实验结果

4.4.1 输入

实验中进行语义分析的文法产生式为：

<赋值语句>　　标识符 = <表达式>
<表达式>　　<项> {+<项> }
<项>　　<因子> {*<因子>}
<因子>　　标识符 | 数字

为了方面叙述，我们用下面的式子进行替代。
则文法产生式简化为：

(1) [FORMULA] --> ['int'][sign]['='][F]
(2) [FORMULA] --> [sign]['='][F]
(3) [FORMULA] --> ['int'][sign]
(4) [F] --> [E]['+'][F]
(5) [F] --> [E]
(6) [E] --> [N]['*'][E]
(7) [E] --> [N]
(8) [N] --> [sign]
(9) [N] --> [const]

那么，其文法的每一个产生式的语义子程序可写为：

(1) [FORMULA] --> ['int'][sign]['='][F]{p=lookup(sign),if(p!=-1)error(0);else (resign(sign), emit("=",F.value, sign,_))}
(2) [FORMULA] --> [sign]['='][F]{ p=lookup(sign),if(p==-1)error(3);else emit("=",F.value, sign,_)}
(3) [FORMULA] --> ['int'][sign]{ p=lookup(sign),if(p!=-1)error(0);else (resign(sign))}
(4) [F] --> [E]['+'][F]{if (E.type=='sign')(p=lookup(sign),if(p==-1)error(3););if(F.type=='sign')(p=lookup(sign),if(p==-1)error(3);)

if(ADD(E,F))emit("+",ADD(E,F),E,F)else error()}

(5) [F] --> [E]

(6) [E] --> [N]['*'][E]{ if (E.type=='sign')(p=lookup(sign),if(p==-1)error(3););if(F.type=='sign')(p=lookup(sign),if(p==-1)error(3);) if(MUL(E,F))emit("*",MUL(E,F),E,F)else error()}

(7) [E] --> [N]{}

(8) [N] --> [sign]{}

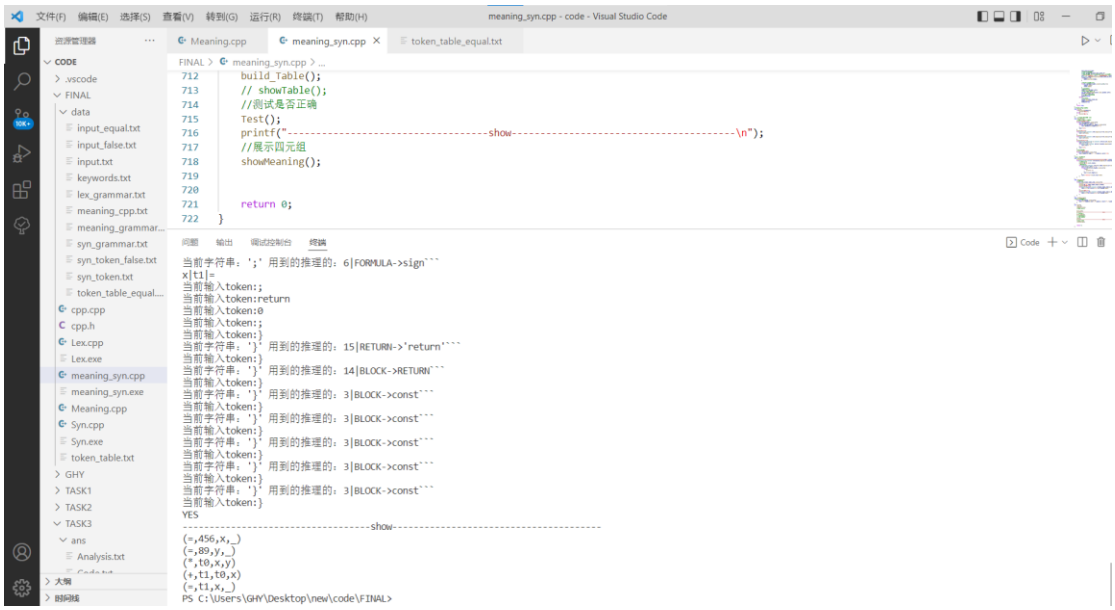
(9) [N] --> [const]{}

声明：由于语义的申明写入 txt 中读取较为繁琐，所以实验中语义分析规则用文件 Meaning.cpp 申明

表 输入文件示意

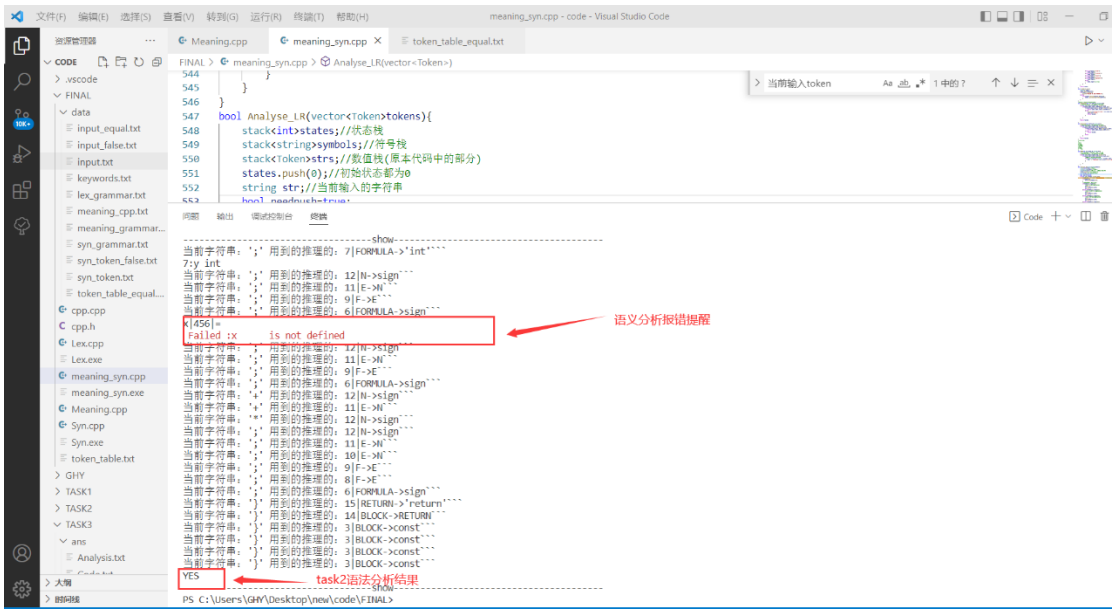
Token_right.txt	Token_err.txt	Grammar.txt
1 int keyword	1 int keyword	[START] --> [X]
1 main keyword	1 main keyword	[X] --> ['int']['main']['(']['(']['{'][BLOCK]['}']
1 (limited	1 (limited	[BLOCK] --> [const][';']
1) limited	1) limited	[BLOCK] --> [const][';'][BLOCK]
1 { limited	1 { limited	[BLOCK] --> [FORMULA][';'][BLOCK]
2 int keyword	3 int keyword	[FORMULA] --> ['int'][sign]['='][F]
2 x sign	3 y sign	[FORMULA] --> [sign]['='][F]
2 ; limited	3 ; limited	[FORMULA] --> ['int'][sign]
3 int keyword	4 x sign	[F] --> [E]['+'][F]
3 y sign	4 = operator	[F] --> [E]
3 ; limited	4 456 const	[E] --> [N]['*'][E]
4 x sign	4 ; limited	[E] --> [N]
4 = operator	5 y sign	[N] --> [sign]
4 456 const	5 = operator	[N] --> [const]
4 ; limited	5 89 const	[BLOCK] --> [RETURN]
5 y sign	5 ; limited	[RETURN] --> ['return'][const][';']
5 = operator	6 x sign	[RETURN] --> [final]
5 89 const	6 = operator	[X] -->[final]
5 ; limited	6 x sign	
6 x sign	6 + operator	
6 = operator	6 y sign	
6 x sign	6 * operator	
6 + operator	6 x sign	
6 y sign	6 ; limited	
6 * operator	7 return keyword	
6 x sign	7 0 const	
6 ; limited	7 ; limited	
7 return keyword	8 } limited	
7 0 const		
7 ; limited		
8 } limited		

4.4.2 输出



```
YES
------------------------------------show----------------------------------------
(=,456,x,_)
(=,89,y,_)
(*,t0,x,y)
(+,t1,t0,x)
(=,t1,x,_)
PS C:\Users\GHY\Desktop\new\code\FINAL>
```

图 6.1 正确 Token 序列输出四元组截图




```
当前字符串：';' 用到的推理的：6|FORMULA->sign```
x|456|=
 Failed :x      is not defined
```

图 6.2 错误 token 序列报错提醒

5. 实验总结

本次实验历时十周，因为大三下学期忙于升学事务，同时疫情肆虐导致很多事情的节奏和自己最初安排的有所出入，最终实验部分零零总总耗费了两周的时间。从一开始查找资料、翻阅教材，到设计数据结构，复现书上的算法，过程中不仅复习了编译原理的 NFA/DFA、LR(1)算法还有 C++结构体中运算符重构等基础知识。

因为是智能专业，大二之后接触比较多的是 Python，这次实验选择用 C++，一方面是为了准备暑期的机试，另一方面也想挑战自己自主编程减少对开发包的依赖。

值得一提的是，本次耗费时间最久的是 Task2，主要是在构建语法的时候，为了自己审阅方便，我用的都是类似**[BLOCK] --> [const][';'][BLOCK]**结构,这就意味着需要用 string 存储每一个符号，无形中增加了后续编程的难度；同时在设计项目集中每一个 Item 结构体的时候，由于没有对结构体中的运算符重载，导致调用 STL 中的 set 时候，命令行报了万行的错误（第一次体会到了报错说明比源码长的痛苦）。这部分花了两天的时间去找资料，从如何让 vscode 命令行完整显示错误信息到如何解决'<''='不能正常工作，bug 解决的时候很有成就感。

实验的部分过程记录在我个人搭建的博客上，保险起见没有公布自己的完整源码，但是记录了一些 debug 的过程以及感想感悟，以及自己查阅的资料。

任务	网址
Task1	博客 1
Task2	博客 2
Task3	博客 3

本次实验收获颇丰，代码逻辑部分也许还存在不足，时间复杂度和空间复杂度还有改进的空间。

尽管如此，仍然很感谢这次课设实验，进一步提高了自己的 coding 能力和 debug 能力，同时时间安排非常人性化和宽裕，让我有时间进行一些自主的 idea 的探索。