

# 软件课程设计 2 报告



学院： 计算机科学与工程学院

---

班级： 9191062301

---

学号： 919106840547

---

姓名： 朱竑典

---

日期： 2022 年 5 月 4 日

---

# 目录

词法分析程序.....	3
一、 设计要求.....	3
二、 设计思路.....	3
三、 数据结构及函数.....	4
四、 运行环境.....	6
五、 实例展示.....	7
LR(1) 语法分析程序.....	11
一、 设计要求.....	11
二、 设计思路.....	11
三、 数据结构及函数.....	11
四、 运行环境.....	17
五、 实例展示.....	17
心得体会.....	21

# 词法分析程序

## 一、设计要求

创建一个词法分析程序，该程序支持分析常规单词。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：一个文本文档，包括一组 3° 型文法（正规文法）的产生式；一个源代码文本文档，包含一组需要识别的字符串（程序代码）。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），可检查整数常量的合法性，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要求的规则。

## 二、设计思路

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、数据结构及函数

#### 1. 数据结构

`vector<string> keyword` 存储用户定义的关键字。

`vector<string> inchar` 存储规定的可输出字符。

`vector<string> type` 存储识别出的单词类别。

`vector<string> token` 存储源文件中分离的单词。

`vector<int> tokenLine` 存储单词行号。

`vector<string> _type = { "O","I","A", "C", "B", "L","K" }、`

`vector<string> to_type = { "operator", "identifier", "const", "const",  
"const", "limiter", "keyword" }` 存储定义的单词类别及描述。

`char str_file[1000]` 用于保存读取文件的内容。

`string ch` 用于保存输出 LOG 的内容。

`string keywordState` 定义关键字的符号。

`string symbolState` 定义标识符的符号。

`struct NFA_sturct {`

`string now;`

`string input;`

`string next;`

`};` 定义 NFA 三元组 "当前状态|接收字符|下一个状态"。

`struct DFA {`

`string startState;`

`vector<string> endState;`

```
map<string, string> f;
```

};定义 DFA：开始状态、结束状态、转换关系。

```
struct NFA {
```

```
    string startState;
```

```
    vector<string> endState;
```

```
    NFA_sturct f[200];
```

};定义 NFA：开始状态、结束状态、转换关系。

```
struct closure {
```

```
    string name;
```

```
    vector<string> t;
```

};定义状态集合的闭包。

## 2. 函数

**bool** compare(**vector**<**string**> v1, **vector**<**string**> v2); 比较两个 **string**数组中的内容是否一致。

**bool** operator == (**const** **closure**& c1, **const** **closure**& c2); 操作符重载，用于**closure**的比较。

**vector**<**string**> split(**const** **string**& s, **const** **string**& seperator); 用于字符串分割。

**void** readKeywords(); 读取KEYWORD文件内容。

**void** readInChar(); 读取INCHAR文件内容。

**void** readGrammar(); 读正规文法入NFA。

**void** readSource(); 读取源文件内容。

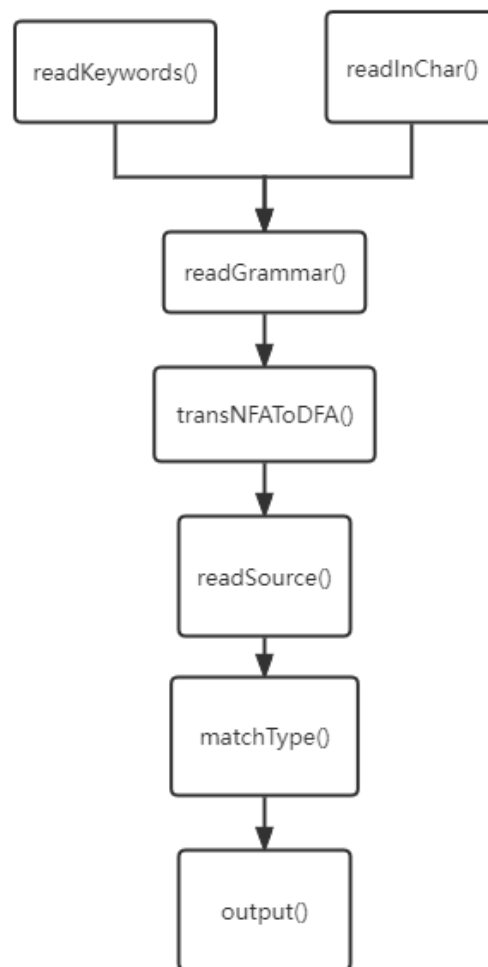
`void transNFAToDFA();` 执行NFA转DFA的操作。

`void matchType();` 匹配token类型。

`void output();` 输出结果。

### 3. 调用流程

main函数：



## 四、运行环境

使用系统：Windows 11

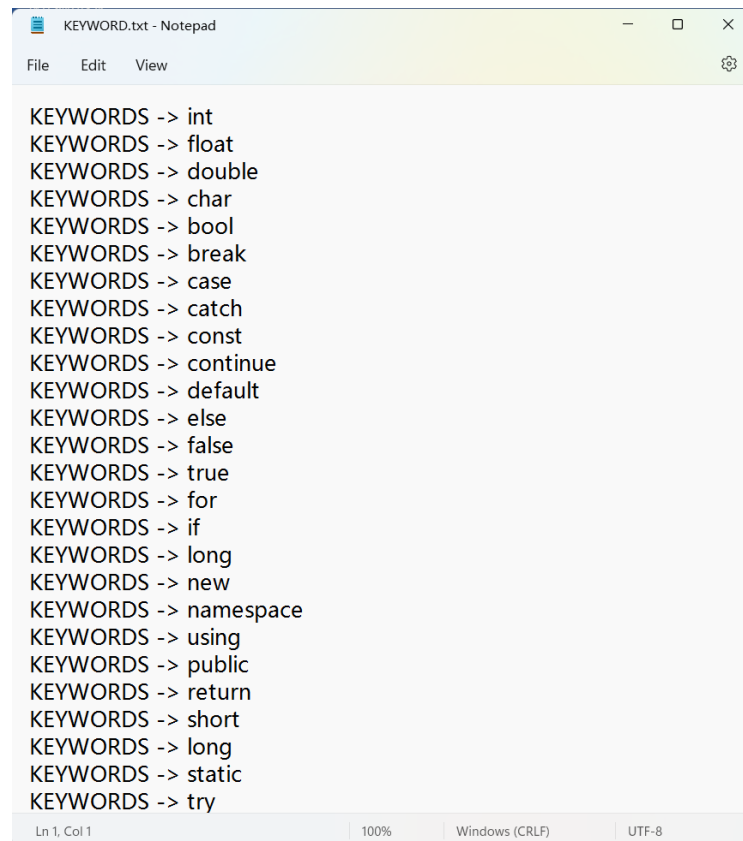
编程语言：C++

编程工具：Visual Studio 2022

## 五、实例展示

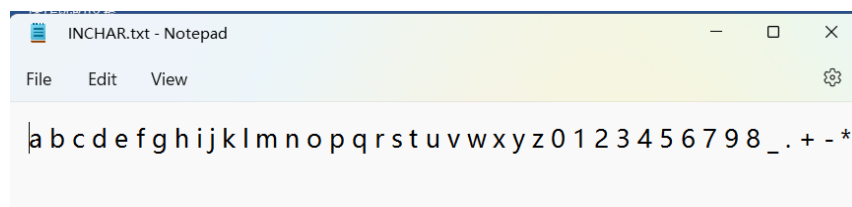
### 1. txt 文件展示

KEYWORD.txt 中定义了该语言的关键字。其文件内容如下例：



```
KEYWORDS -> int
KEYWORDS -> float
KEYWORDS -> double
KEYWORDS -> char
KEYWORDS -> bool
KEYWORDS -> break
KEYWORDS -> case
KEYWORDS -> catch
KEYWORDS -> const
KEYWORDS -> continue
KEYWORDS -> default
KEYWORDS -> else
KEYWORDS -> false
KEYWORDS -> true
KEYWORDS -> for
KEYWORDS -> if
KEYWORDS -> long
KEYWORDS -> new
KEYWORDS -> namespace
KEYWORDS -> using
KEYWORDS -> public
KEYWORDS -> return
KEYWORDS -> short
KEYWORDS -> long
KEYWORDS -> static
KEYWORDS -> try
```

INCHAR.txt 中定义了该语言所能输入的字符。注意该语言中所有可以使用的字符都应在 INCHAR.txt 中定义。在编译过程中，如果发现未在 INCHAR.txt 中定义的字符将会输出“ERROR”并结束。其文件内容如下例：

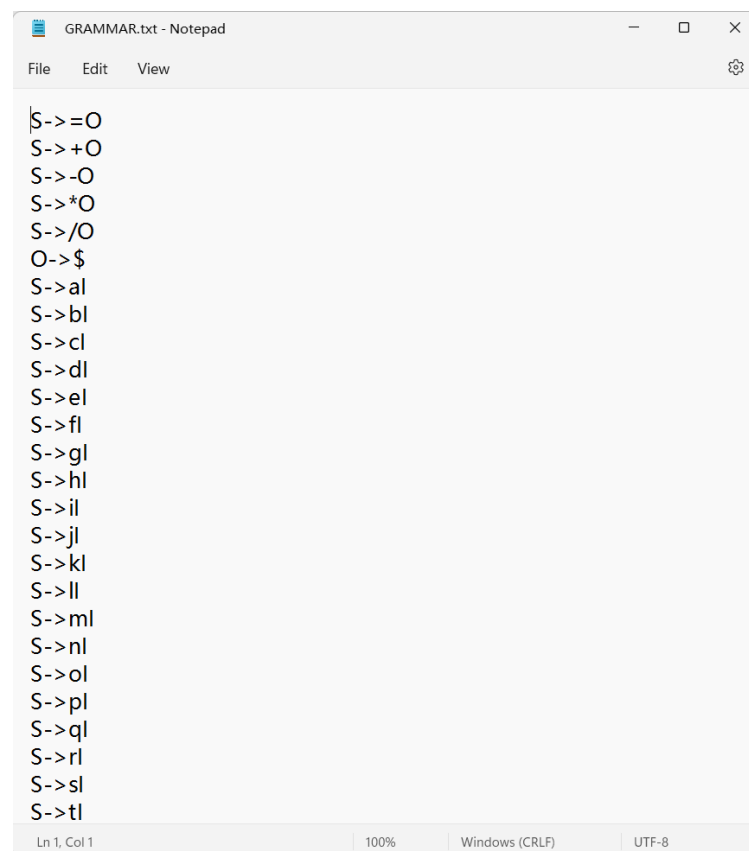


```
abcdefghijklmnopqrstuvwxyz0123456789_ . + - *
```

GRAMMAR.txt 中定义了该语言的文法，该例中设计了一个三型文法，分别用各个终态表示一类单词。对应关系和描述如下：

符号	类型	备注
O	运算符	有 = 、 + 、 - 、 * 、 / 5 个符号
I	标识符	以字母开头，后面可以跟字母、数字、下划线
A	整数常数	整数（如：1 ， 22 ， 333）
C	小数常数	小数型常数（如：1.1 ， 2.22e+8）
G	复数	复数（如：1+i, 2+2i）
L	界符	有 # 、 ; 、 , 、 ( 、 ) 、 { 、 } 、 < 、 >

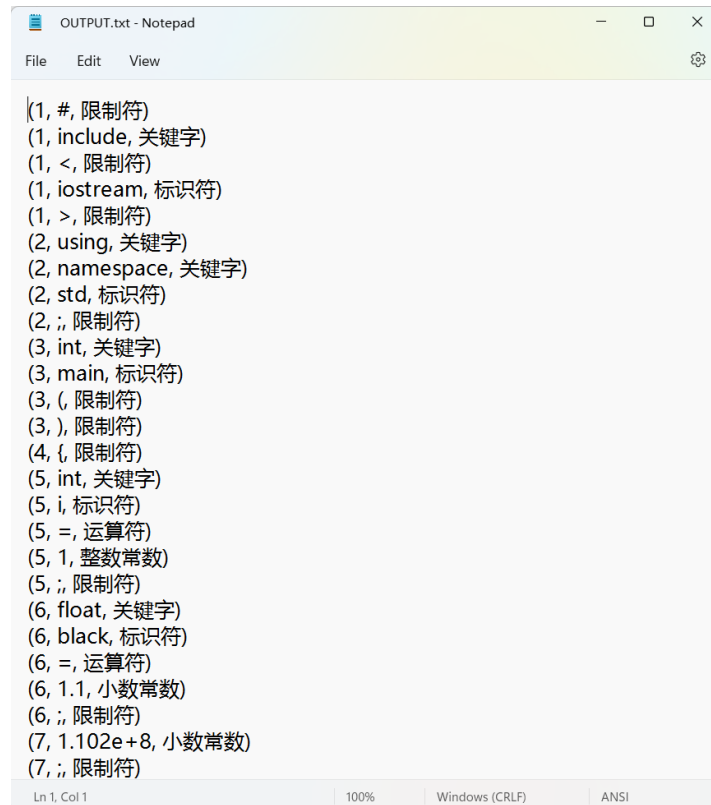
文件内容如下例：



```
S->=O
S->+O
S->-O
S->*O
S->/O
O->$
S->aI
S->bI
S->cI
S->dI
S->eI
S->fI
S->gI
S->hI
S->iI
S->jI
S->kI
S->lI
S->mI
S->nI
S->oI
S->pI
S->qI
S->rI
S->sI
S->tI
```

分析的结果会保存一份到 OUTPUT.txt ， 如下图：

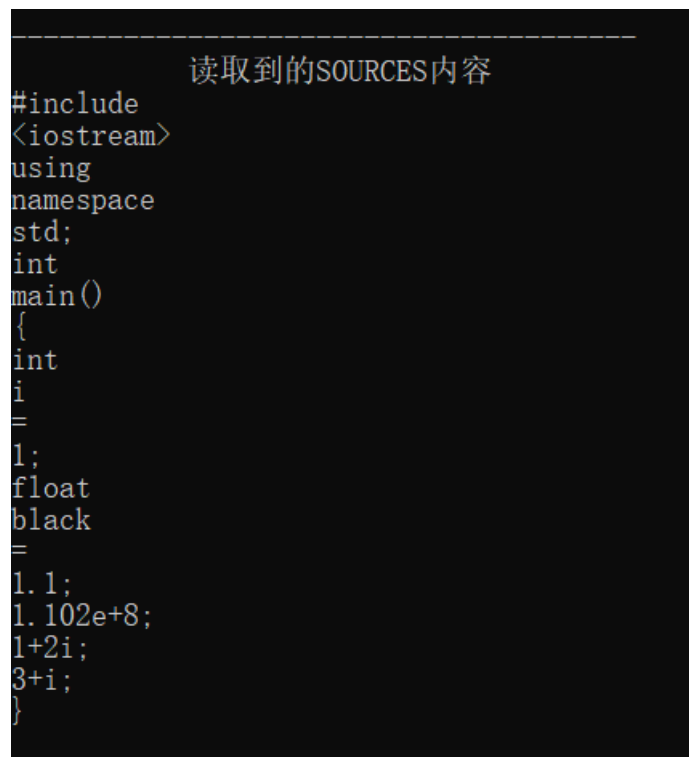




```
OUTPUT.txt - Notepad
File Edit View
(1, #, 限制符)
(1, include, 关键字)
(1, <, 限制符)
(1, iostream, 标识符)
(1, >, 限制符)
(2, using, 关键字)
(2, namespace, 关键字)
(2, std, 标识符)
(2, ;, 限制符)
(3, int, 关键字)
(3, main, 标识符)
(3, {, 限制符)
(3, }, 限制符)
(4, {, 限制符)
(5, int, 关键字)
(5, i, 标识符)
(5, =, 运算符)
(5, 1, 整数常数)
(5, ;, 限制符)
(6, float, 关键字)
(6, black, 标识符)
(6, =, 运算符)
(6, 1.1, 小数常数)
(6, ;, 限制符)
(7, 1.102e+8, 小数常数)
(7, ;, 限制符)
Ln 1, Col 1 100% Windows (CRLF) ANSI
```

## 2. 运行结果

按照分割符读入，并分析 token 类别。



```
-----
读取到的SOURCES内容
#include
<iostream>
using
namespace
std;
int
main()
{
int
i
=
1;
float
black
=
1.1;
1.102e+8;
1+2i;
3+i;
}
```

输出 token 列表，并存储到 OUTPUT.txt 中：

### 分析出的词法结果

```
(1, #, 限制符)
(1, include, 关键字)
(1, <, 限制符)
(1, iostream, 标识符)
(1, >, 限制符)
(2, using, 关键字)
(2, namespace, 关键字)
(2, std, 标识符)
(2, ;, 限制符)
(3, int, 关键字)
(3, main, 标识符)
(3, (, 限制符)
(3, ), 限制符)
(4, {, 限制符)
(5, int, 关键字)
(5, i, 标识符)
(5, =, 运算符)
(5, 1, 整数常数)
(5, ;, 限制符)
(6, float, 关键字)
(6, black, 标识符)
(6, =, 运算符)
(6, 1.1, 小数常数)
(6, ;, 限制符)
(7, 1.102e+8, 小数常数)
(7, ;, 限制符)
(8, 1+2i, 复数常数)
(8, ;, 限制符)
(9, 3+i, 复数常数)
(9, ;, 限制符)
(10, }, 限制符)
```

生成的词法结果已保存到'./OUTPUT.txt'

请按任意键继续. . .

# LR(1) 语法分析程序

## 一、 设计要求

程序有两个输入：1) 一个是文本文档，其中包含 2° 型文法（上下文无关文法）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。程序的输出包括：YES 或 NO（源代码字符串符合此 2° 型文法，或者源代码字符串不符合此 2° 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。建议能演示语法处理的中间过程。

## 二、 设计思路

语法分析程序的推荐处理逻辑：根据用户输入的 2° 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）。

## 三、 数据结构及函数

### 1. 数据结构

`vector<string> Vn` 语法非终结符。

`vector<string> Vt` 语法终结符。

```
struct grammar{  
    string left; //文法左部  
    string right; //文法右部
```

```
int num;        //语法序号

int length;     //归约长度

}; 定义语法。
```

```
struct project{

    int num;        //第几条语法

    int position;    //圆点位置

    vector<char>  search;  //向前搜索符(即 First 集)

}; 定义项目。
```

```
struct project_set {

    string name;     //项目集名称

    vector<project> t;  //定义项目集中的项目数组 t

}; 定义项目集。
```

`vector<grammar> grammars;` 语法集合。

`vector<project_set> sets;` 项目集集合。

`map<string, string> m;` key为当前状态+输入符号，value为执行的  
动作。

`vector<string> states` 状态栈。

`vector<string> symbols` 符号栈。

`vector<string> in;` 输入语句。

## 2. 函数

`bool comp(vector<project> v1, vector<project> v2);` 比较两项目集  
是否相同。

`bool comp(vector<char> v1, vector<char> v2);` 比较两字符串是否相同。

`bool operator == (const project& p1, const project& p2);` 重载操作符，判断两项目是否相同。

`bool operator == (const project_set& c1, const project_set& c2);` 重载操作符，判断两项目集是否相同。

`vector<char> getFirst(vector<grammar> yufa, string s, vector<string> Vt);` 求得 First 集合。

`void read_Grammar();` 读入文法。

`void getClosure(project_set &clo);` 求得当前项目集闭包。

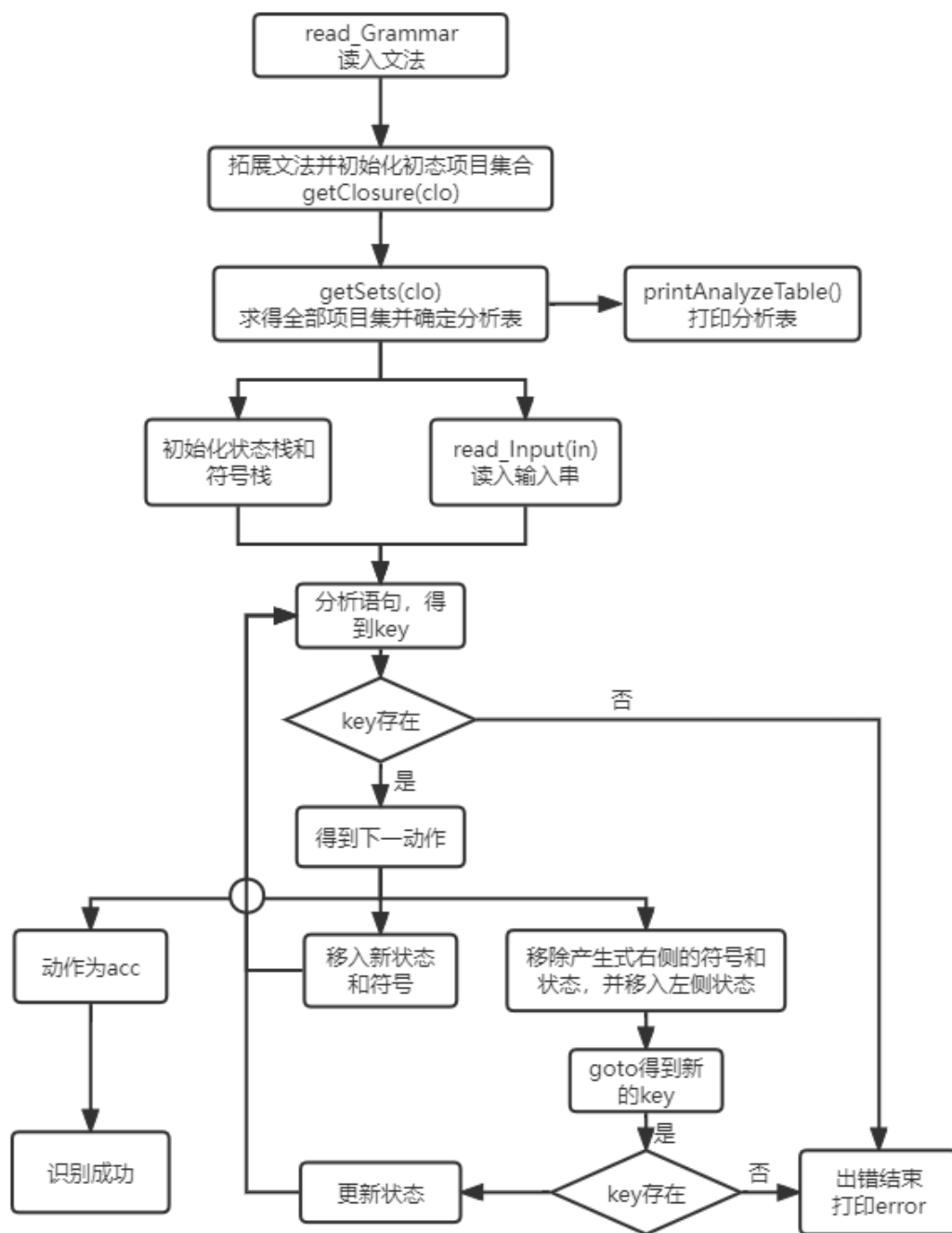
`void getSets(project_set& clo);` 求得所有项目集和构造分析表。

`void printAnalyzeTable();` 打印分析表。

`void read_Input(vector <string> &in);` 读入输入串。

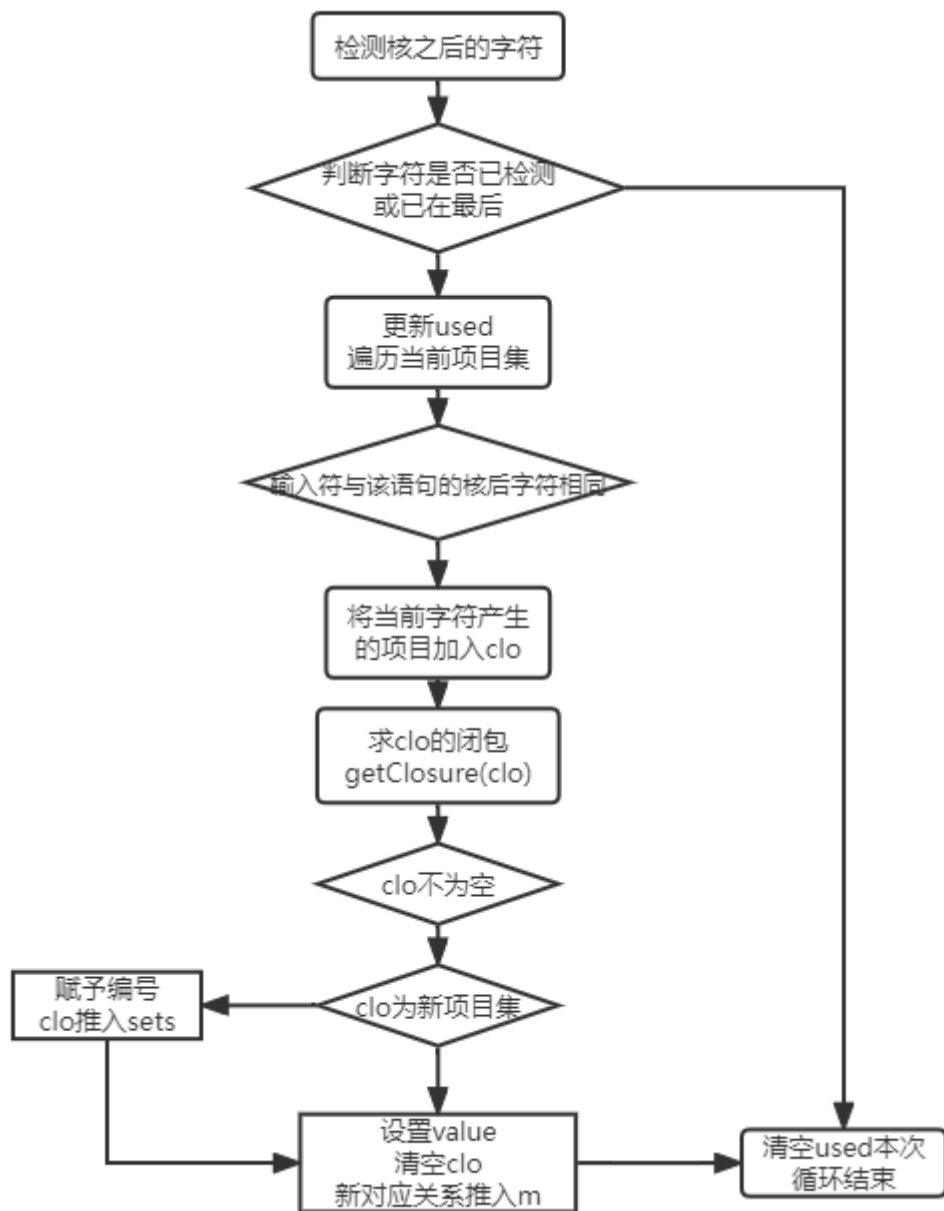
### 3. 函数调用

main 函数：

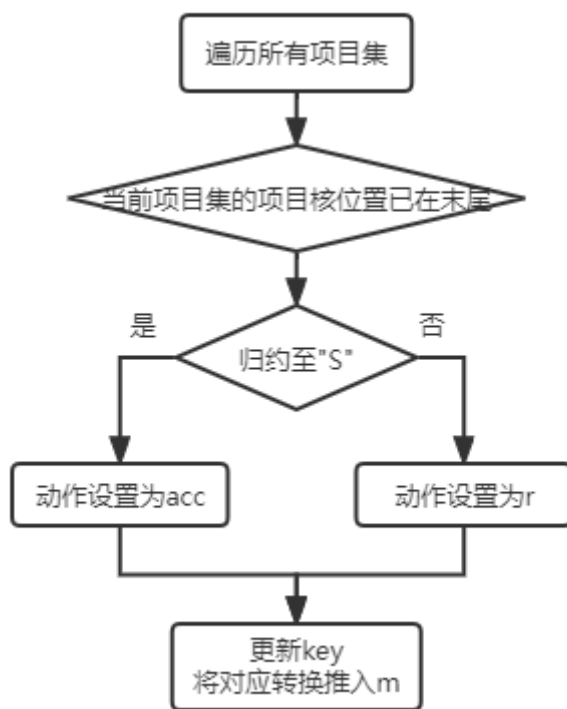


getSets 函数：

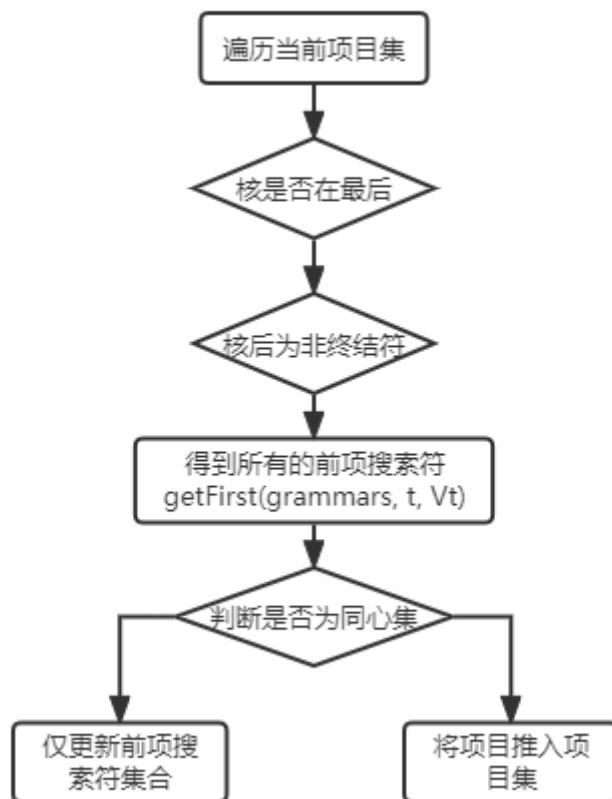
得到所有项目集，已经项目集间的移进关系：



自此，开始处理项目集间的归约关系：



getClosure 函数:





## 四、 运行环境

使用系统：Windows 11

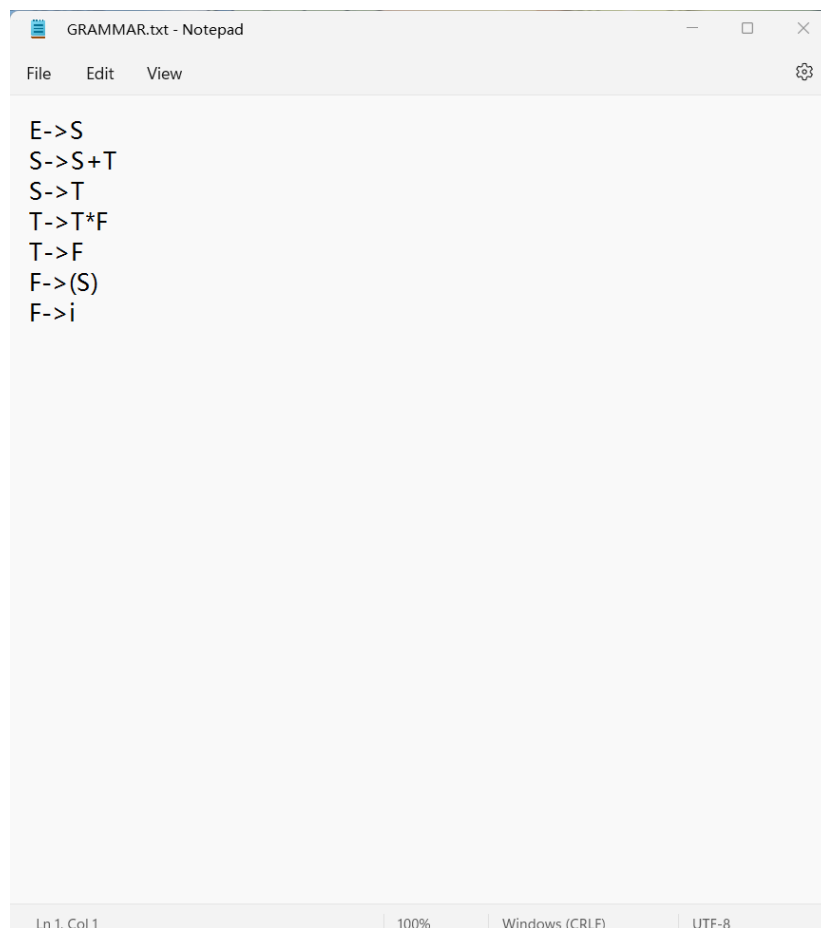
编程语言：C++

编程工具：Visual Studio 2022

## 五、 实例展示

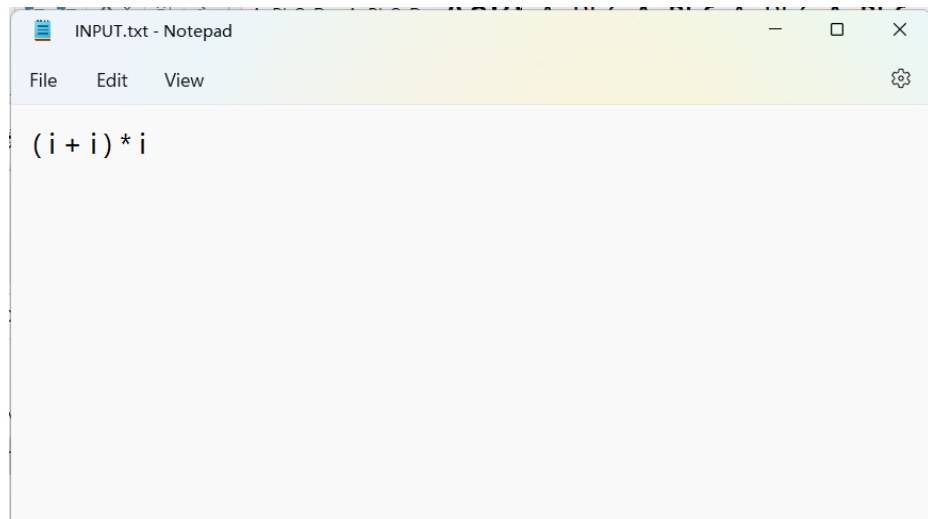
### 1. txt 展示：

该 txt 内容为用户定义的语法文法，例如：



```
GRAMMAR.txt - Notepad
File Edit View
E->S
S->S+T
S->T
T->T*F
T->F
F->(S)
F->i
Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8
```

该 txt 内容为用户定义的语法输入，例如：



## 2. 运行结果

输入文法展示为:

```
输入的文法为:  
0   E→S  
1   S→S+T  
2   S→T  
3   T→T*F  
4   T→F  
5   F→(S)  
6   F→i
```

项目集展示为:

```

I0
E->S [·的位置S] ,#
S->S+T [·的位置S] ,# +
S->T [·的位置T] ,# +
T->T*F [·的位置T] ,# + *
T->F [·的位置F] ,# + *
F->(S) [·的位置()] ,# + *
F->i [·的位置i] ,# + *
I1
E->S [·的位置] ,#
S->S+T [·的位置+] ,# +
I2
S->T [·的位置] ,# +
T->T*F [·的位置*] ,# + *
I3
T->F [·的位置] ,# + *
I4
F->(S) [·的位置S] ,# + *
S->S+T [·的位置S] ,) +
S->T [·的位置T] ,) +
T->T*F [·的位置T] ,) + *
T->F [·的位置F] ,) + *
F->(S) [·的位置()] ,) + *
F->i [·的位置i] ,) + *
I5
F->i [·的位置] ,# + *
I6
S->S+T [·的位置T] ,# +
T->T*F [·的位置T] ,# + *
T->F [·的位置F] ,# + *
F->(S) [·的位置()] ,# + *
F->i [·的位置i] ,# + *
I7
T->T*F [·的位置F] ,# + *
F->(S) [·的位置()] ,# + *
F->i [·的位置i] ,# + *
I8
F->(S) [·的位置)] ,# + *
S->S+T [·的位置+] ,) +
I9
S->T [·的位置] ,) +
T->T*F [·的位置*] ,) + *
I10
T->F [·的位置] ,) + *
I11
F->(S) [·的位置S] ,) + *
S->S+T [·的位置S] ,) +
S->T [·的位置T] ,) +
T->T*F [·的位置T] ,) + *
T->F [·的位置F] ,) + *
F->(S) [·的位置()] ,) + *
F->i [·的位置i] ,) + *
I12
F->i [·的位置] ,) + *
I13
S->S+T [·的位置] ,# +
T->T*F [·的位置*] ,# + *
I14
T->T*F [·的位置] ,# + *
I15
F->(S) [·的位置] ,# + *
I16
S->S+T [·的位置T] ,) +
T->T*F [·的位置T] ,) + *
T->F [·的位置F] ,) + *
F->(S) [·的位置()] ,) + *
F->i [·的位置i] ,) + *
I17
T->T*F [·的位置F] ,) + *
F->(S) [·的位置()] ,) + *
F->i [·的位置i] ,) + *
I18
F->(S) [·的位置)] ,) + *
S->S+T [·的位置+] ,) +
I19
S->S+T [·的位置] ,) +
T->T*F [·的位置*] ,) + *
I20
T->T*F [·的位置] ,) + *
I21
F->(S) [·的位置] ,) + *

```

分析表展示为:

LR(1)分析表:										
	(	)	*	+	i	#	E	S	T	F
0	S4				S5			1	2	3
1				S6		acc				
2			S7	r2		r2				
3			r4	r4		r4				
4	S11				S12			8	9	10
5			r6	r6		r6				
6	S4				S5				13	3
7	S4				S5					14
8		S15		S16						
9		r2	S17	r2						
10		r4	r4	r4						
11	S11				S12			18	9	10
12		r6	r6	r6						
13			S7	r1		r1				
14			r3	r3		r3				
15			r5	r5		r5				
16	S11				S12				19	10
17	S11				S12					20
18		S21		S16						
19		r1	S17	r1						
20		r3	r3	r3						
21		r5	r5	r5						

输入语句展示为:

输入的语句为:  
#(i+i)\*i#

分析过程如下:

分析过程为:				
步骤	状态栈	符号栈	输入符	动作
1	0	#	(i+i)*i#	S4
2	0 4	#(	i+i)*i#	S12
3	0 4 12	#+(	+i)*i#	r6
4	0 4 10	#+(F	+i)*i#	r4
5	0 4 9	#+(T	+i)*i#	r2
6	0 4 8	#+(S	+i)*i#	S16
7	0 4 8 16	#+(S+	i)*i#	S12
8	0 4 8 16 12	#+(S+i	)i#	r6
9	0 4 8 16 10	#+(S+F	)i#	r4
10	0 4 8 16 19	#+(S+T	)i#	r1
11	0 4 8	#+(S	)i#	S15
12	0 4 8 15	#+(S)	*i#	r5
13	0 3	#+F	*i#	r4
14	0 2	#+T	*i#	S7
15	0 2 7	#+T*	i#	S5
16	0 2 7 5	#+T*i	#	r6
17	0 2 7 14	#+T*F	#	r3
18	0 2	#+T	#	r2
19	0 1	#+S	#	acc
语法识别成功 请按任意键继续. . .				

若输入语句错误, 如 i + i ) \* i, 结果如下:

分析过程为:				
步骤	状态栈	符号栈	输入符	动作
1	0	#	i+i)*i#	S5
2	0 5	#i	+i)*i#	r6
3	0 3	#F	+i)*i#	r4
4	0 2	#T	+i)*i#	r2
5	0 1	#S	+i)*i#	S6
6	0 1 6	#S+	i)*i#	S5
7	0 1 6 5	#S+i	)*i#	
ERROR: 当前状态下错误的输入导致出错!				
请按任意键继续. . .				

## 心得体会

### 一、对实验原理有更深入的理解

通过该课程设计，掌握了什么是编译程序，编译程序工作的基本过程及其各阶段的基本任务，熟悉了编译程序总流程框图，了解了编译程序的生成过程、构造工具及其相关的技术对课本上的知识有了更深入的理解，课本上的知识是机械的，表面的。通过把该算法的内容，算法的执行顺序在计算机上实现，把原来以为很深奥的书本知识变的更为简单，对实验原理有更深入的理解。

### 二、对该理论在实践中的应用有深刻的理解

通过把该算法的内容，算法的执行顺序在计算机上实现，知道和理解了该理论在计算机中是怎样执行的，对该理论在实践中的应用有深刻的理解。

### 三、激发了学习的积极性

通过该课程设计，全面系统的理解了编译原理程序构造的一般原理和基本实现方法。把死板的课本知识变得生动有趣，激发了学习的积极性。把学过的计算机编译原理的知识强化，能够把课堂上学的知

识通过自己设计的程序表示出来，加深了对理论知识的理解。以前对与编译原理的认识是模糊的，概念上的；现在通过自己动手做实验，对计算机编译原理的认识更加深刻。在这次课程设计中，我按照实验指导的思想来完成，加深了理解文件系统的内部功能及内部实现，培养实践动手能力和程序开发能力的目的。