

## 一、实验目的与方法

### 1. 实验目的

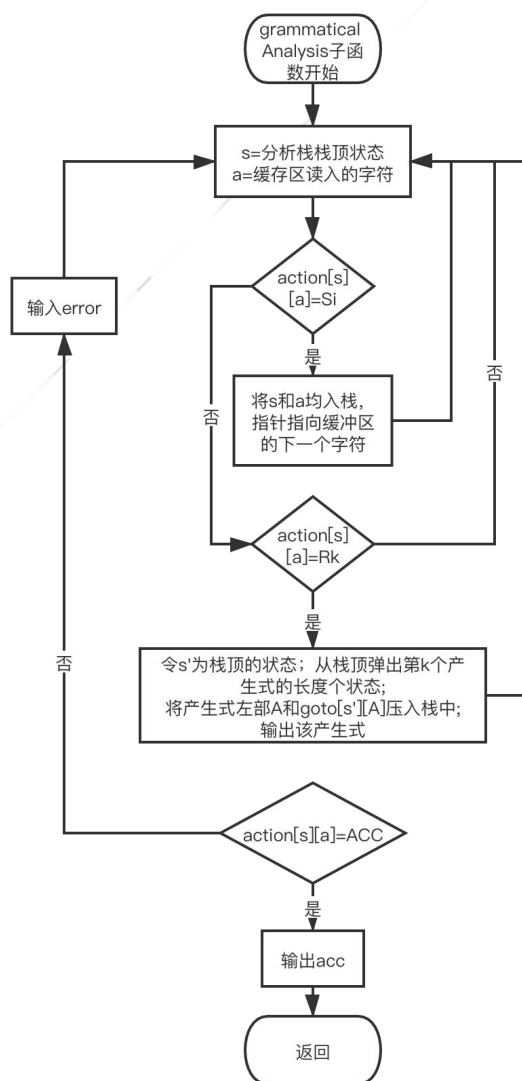
深入了解语法分析程序实现原理及方法。理解 LR(1)分析法是严格的从左向右扫描和自底向上的语法分析方法。

### 2. 实验语言

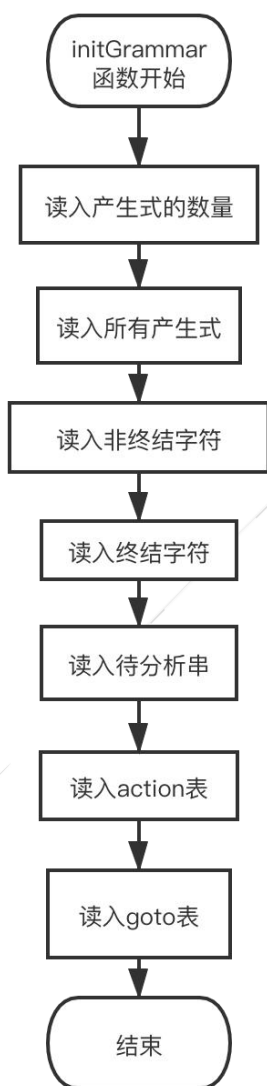
所使用的语言为 C++语言；操作系统环境为 macOS Mojave；软件环境为 CLion。

## 二、实验总体流程与函数功能描述

### 1. 语法分析程序 void grammaticalAnalysis()



## 2. 根据读入文件初始化数据结构 void initGrammar()



3. 判断 ch 是否是终结符 int isInT(char ch)  
若是，返回下标加一；若不是，返回 0。
4. 判断 ch 是否是非终结符 int isInN(char ch)  
若是，返回下标加一；若不是，返回 0。

## 三、 实验内容

## 1. 拓广文法 G

S→A  
A→A+T  
A→T  
T→T\*F  
T→F  
F→(A)  
F→a



状态	ACTION						GOTO		
	+	*	(	)	id	\$	E	T	F
0			shift 4		shift 5		1	2	3
1	shift 6					accept			
2	reduce E -> T	shift 7				reduce E -> T			
3	reduce T -> F	reduce T -> F				reduce T -> F			
4			shift 11		shift 12		8	9	10
5	reduce F -> id	reduce F -> id				reduce F -> id			
6			shift 4		shift 5			13	3
7			shift 4		shift 5				14
8	shift 15			shift 16					
9	reduce E -> T	shift 17		reduce E -> T					
10	reduce T -> F	reduce T -> F		reduce T -> F					
11			shift 11		shift 12		18	9	10
12	reduce F -> id	reduce F -> id		reduce F -> id					
13	reduce E -> E + T	shift 7				reduce E -> E + T			
14	reduce T -> T * F	reduce T -> T * F				reduce T -> T * F			
15			shift 11		shift 12			19	10
16	reduce F -> ( E )	reduce F -> ( E )				reduce F -> ( E )			
17			shift 11		shift 12				20
18	shift 15			shift 21					
19	reduce E -> E + T	shift 17		reduce E -> E + T					
20	reduce T -> T * F	reduce T -> T * F		reduce T -> T * F					
21	reduce F -> ( E )	reduce F -> ( E )		reduce F -> ( E )					

#### 4. LR(1)分析表的存储结构

##### ① Action 表

```
pair<int, int> action[INIT_LEN][INIT_LEN];
```

action[0-21][0-5](即对应表格里的 22 行, 6 列)。对于表格中的一项, 对应 action[i][j]=pair<int, int>。

第一个系数表示分析动作: 若是 S, 则第一个系数为 1 若是 R; 则第一个系数为 2, 若是 ACC, 则第一个系数为 3

第二个系数表示转移状态或者产生式序号。

##### ② Goto 表

```
int goton[INIT_LEN][INIT_LEN];
```

gton[0-21][1-3](即对应表格里的 22 行, 3 列)。对于表格中的一项, 对应二维数组中的一项。

#### 5. 其他数据结构

##### ① 产生式结构体

```
/* 产生式结构体, 左部符号和右部字符串 */
struct Production {
    char left;
    vector<char> rights;
    /* 重载 == */
    bool operator==(Production& rhs) const {
        if (left != rhs.left)
            return false;
        for (int i = 0; i < rights.size(); i++) {
            if (i >= rhs.rights.size())
                return false;
            if (rights[i] != rhs.rights[i])\
                return false;
        }
        return true;
    }
};
```

##### ② 文法结构体

```
/* 文法结构体 */
struct Grammar {
    int num; // 产生式数量
    vector<char> T; // 终结符
    vector<char> N; // 非终结符
    vector<Production> prods; //产生式
}grammar;
```

### ③ 待分析串

```
/* 待分析串 */
string str;
```

### ④ 分析栈

```
/* 分析栈 */
stack<pair<int, char>> ST; // first是状态, second是符号
```

## 6. 主要模块的算法功能

该实验的主要模块为语法分析子函数，其流程图已在第二部分给出。

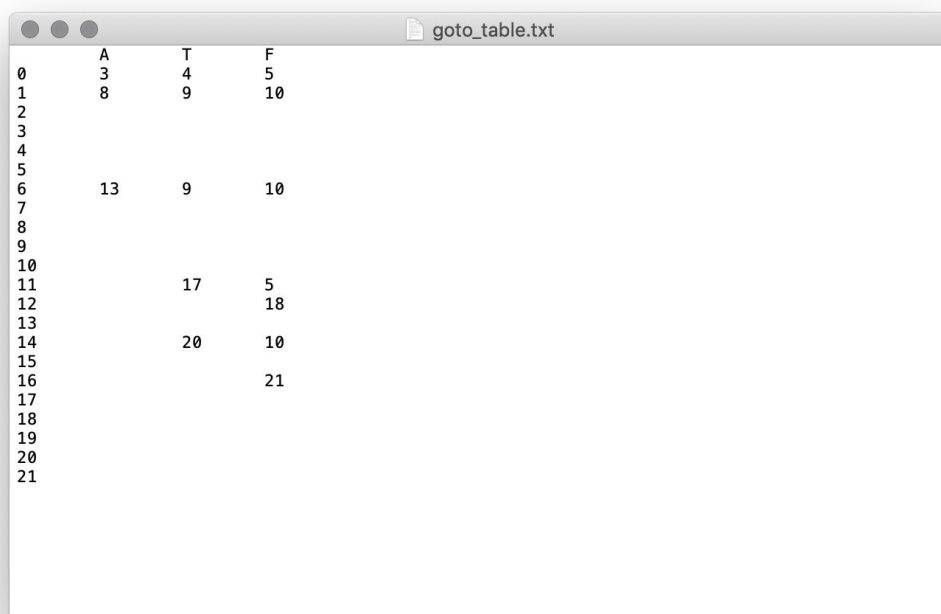
## 7. 实验中用到的特色方法或设计技巧

在读入分析表的部分我将产生式进行编号，整理成如下的预测分析表作为输入。

### ① Action 表

	+	*	(	)	a	\$
0			S1		S2	
1			S6		S7	
2	R6	R6				R6
3	S11					ACC
4	R2	S12				R2
5	R4	R4				R4
6			S6		S7	
7	R6	R6		R6		
8	S14			S15		
9	R2	S16		R2		
10	R4	R4		R4		
11			S1		S2	
12			S1		S2	
13	S14			S19		
14			S6		S7	
15	R5	R5				R5
16			S6		S7	
17	R1	S12				R1
18	R3	R3				R3
19	R5	R5		R5		
20	R1	S16		R1		
21	R3	R3		R3		

### ② Goto 表

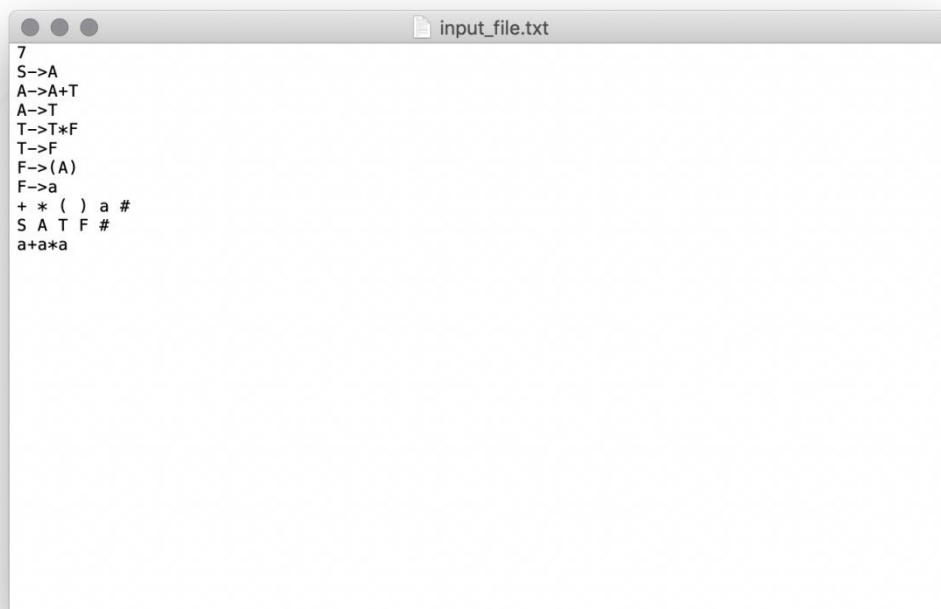


	A	T	F
0	3	4	5
1	8	9	10
2			
3			
4			
5			
6	13	9	10
7			
8			
9			
10			
11		17	5
12			18
13			
14		20	10
15			
16			21
17			
18			
19			
20			
21			

#### 四、实验结果与分析

##### ① 实验输入

除了要读入预测分析表之外，还需要读入文法以及待分析的字符串。

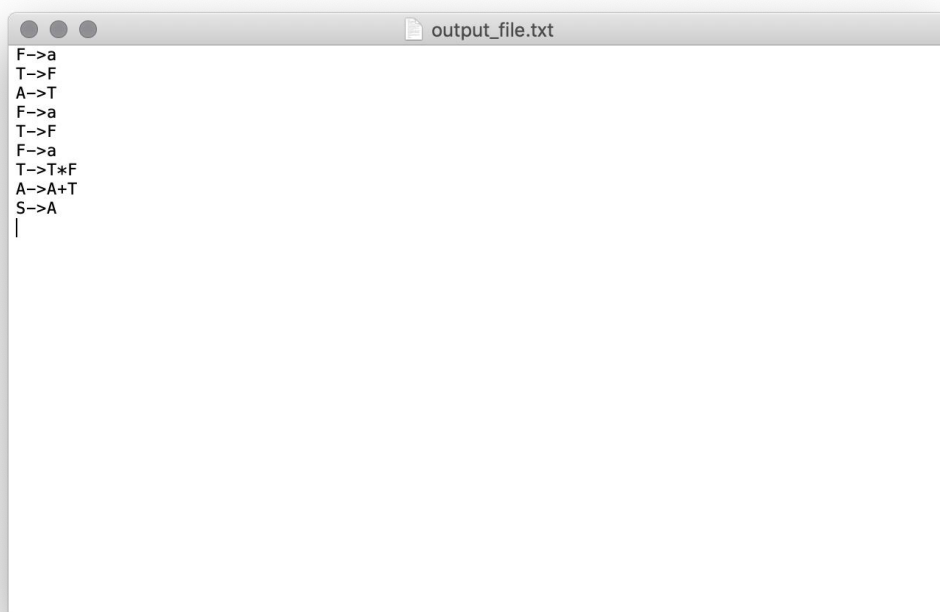


```
7
S->A
A->A+T
A->T
T->T*F
T->F
F->(A)
F->a
+ * ( ) a #
S A T F #
a+a*a
```

第一行表示有 7 个产生式。接下来的 7 行代表了 7 个产生式。第 9 行表

示终结字符（其中 a 表示 ID）。第 10 行表示非终结字符。最后一行表示待分析的字符串。

## ② 实验输出



```
F->a
T->F
A->T
F->a
T->F
F->a
T->T*F
A->A+T
S->A
|
```

其对应的语法分析树如下：

