



编译原理实验报告

|  |  |
| --- | --- |
| 专 业 班 级 | 计算机科学与技术18-2班 |
| 学生姓名及学号 | 2018217692 牛洪亮 |
| 任 课 教 师 | 李宏芒老师 |
| 实验指导教师 | 李宏芒老师 |
| 实验地点 | 计算机楼第五机房 |
| 2020 ~2021 学年第一学期 | |

## 实验一 词法分析设计

**一、实验目的**

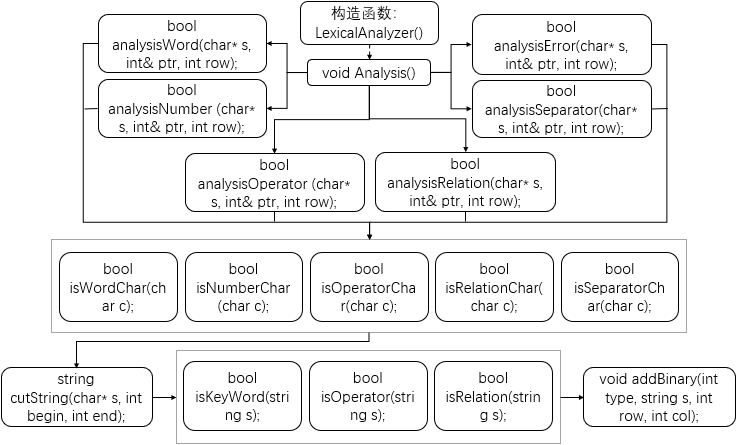
通过本实验的编程实践，使学生了解词法分析的任务，掌握词法分析程序设计的原理和构造方法，使学生对编译的基本概念、原理和方法有完整的和清楚的理解，并能正确地、熟练地运用。

**二、功能描述**

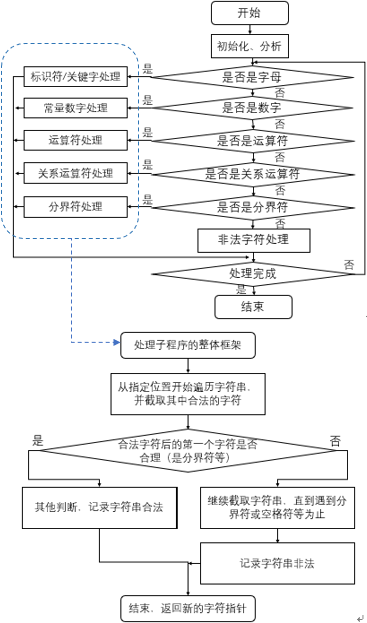
1. 可手动输入多行字符并读入进行分析
2. 统计行数和列数用于错误单词的定位
3. 删除空格类字符，包括回车、制表符空格
4. 按拼写单词（关键字、标识符、常数、运算符、关系运算符、分界符号），并用（内码，属性）二元式表示
5. 如果发现错误则报告出错
6. 根据需要是否填写标识符表供以后各阶段使用
7. 识别注释
8. 判别浮点数和整数

**三、程序结构描述**

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名称及调用格式返回类型 | 参数含义 | 返回值描述 | 函数功能 |
| LexicalAnalyzer(const char\* fileDir); | 相关参数文件目录的相对路径（空时默认为“./sign/”） | 无 | 构造函数，获取关键字、运算符、关系运算符、分隔符 |
| void analysis(const char\* fileName); | 要分析的源代码的文件相对路径（空时默认“./program/main.cpp”） | 无 | 获取输入串，分析每一串字符，并记录行号，根据每一串字符的第一个字符调用不同的函数 |
| bool analysisWord(char\* s, int& ptr, int row); | s是当行的字符串的首地址，ptr是分析部分首地址的位置，row是当前字符串的行号 | 返回值为能否正确分析出一串标识符或关键字 | 字母分析，将由字母和数字构成的字符串进行提取，分析为标识符或者关键字并记录 |
| bool analysisNumber(char\* s, int& ptr, int row); | s是当行的字符串的首地址，ptr是分析部分首地址的位置，row是当前字符串的行号 | 返回值为能否正确分析出一串数字 | 常量分析，对数字开头的输入串进行分析，并记录数字，若数字中夹杂或后接字母将报错 |
| bool analysisOperator(char\* s, int& ptr, int row); | s是当行的字符串的首地址，ptr是分析部分首地址的位置，row是当前字符串的行号 | 返回值为能否正确分析出一串运算符 | 运算符分析，对运算符开头的输入串进行分析，并记录判断该运算符是否合法 |
| bool analysisRelation(char\* s, int& ptr, int row); | s是当行的字符串的首地址，ptr是分析部分首地址的位置，row是当前字符串的行号 | 返回值为能否正确分析出一串关系运算符 | 关系运算符分析，对关系运算符开头的输入串进行分析，并记录判断该关系运算符是否合法 |
| bool analysisSeparator(char\* s, int& ptr, int row); | s是当行的字符串的首地址，ptr是分析部分首地址的位置，row是当前字符串的行号 | 返回值为能否正确分析出一串分隔符 | 分隔符分析，并记录分隔符 |
| bool analysisError(char\* s, int& ptr, int row); | s是当行的字符串的首地址，ptr是分析部分首地址的位置，row是当前字符串的行号 | 返回值恒为false，保持形式统一 | 记录由非法字符串开头的一串字符为错位 |
| bool isWordChar(char c); | 输入任意一个字符 | 返回该字符是否是字母（a-z、A-Z） | 判断输入的字符是否是字母 |
| bool isNumberChar(char c); | 输入任意一个字符 | 返回该字符是否是数字（0-9） | 判断输入的字符是否是数字 |
| bool isOperatorChar(char c); | 输入任意一个字符 | 返回输入的字符是否是运算符中使用的字符 | 判断输入的字符是否是运算符中会用到的字符 |
| bool isRelationChar(char c); | 输入任意一个字符 | 返回输入的字符是否是关系运算符中使用的字符 | 判断输入的字符是否是关系运算符中会用到的字符 |
| bool isSeparatorChar(char c); | 输入任意一个字符 | 返回输入的字符是否是分界符中使用的字符 | 判断输入的字符是否是分界符中会用到的字符 |
| string cutString(char\* s, int begin, int end); | s是原字符串的首地址指针，begin指出了需剪切出字符串的开始位置，end指出了需剪切的字符串的结束位置（不包含end） | 返回s中从begin开始到end之间的字符串 | 从一个长字符串剪切出指定位置的字符串并返回字符串 |
| bool isSpace(char c); | 输入任意一个字符 | 返回该字符是否是空格符（“ ”、“\t”等） | 判断输入的字符是否是空格符 |
| bool isKeyWord(string s); | 输入任意一串字符串 | 返回该字符串是否是关键字 | 判断一串字符串是否是关键字 |
| bool isOperator(string s); | 输入任意一串字符串 | 返回该字符串是否是运算符 | 判断一串字符串是否是运算符 |
| bool isRelation(string s); | 输入任意一串字符串 | 返回字符串是否是关系运算符 | 判断一串字符串是否是关系运算符 |
| void addBinary(int type, string s, int row, int col); | 字符串类型，字符串，行号，列号 | 无 | 记录一个单词的类型及位置 |



**四、详细的算法描述**



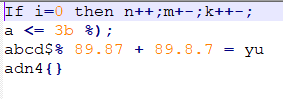
**补充：**

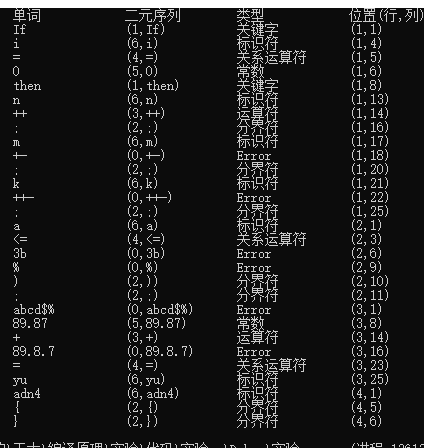
1. 对数字读取若其中有一位"."进行判别为浮点数，若有若干个则判别为"Error"。
2. 若字母后接数字判别为标识符，若数字后面接字母不符合变量命名规则判别为"Error"。
3. 可以通过向文件中添加更多规则的运算符、分界符来匹配更多运算符，如++、--等
4. 对关键字的匹配区分大小写
5. 对于字母开头、数字开头的单词的错误匹配会一直匹配到分界符、空白符或运算符、关系运算符；对于运算符、关系运算符的错误匹配只会匹配到下一个非同类字符即截止

**五、给出软件的测试方法和测试结果**

程序默认打开相对路径./sign/keyWord.sign、./sign/operaWord.sign、relatWord.sign、./sign/separatWord.sign四个文件获取相关设置参数，可以在程序中修改路径或直接编辑文件；程序默认打开./ program/main.cpp文件获取需要分析的源程序，可以在程序中修改路径或直接编辑文件

main.cpp内容：





**六、实验总结**

1. 实验的没有处理程序注释、没有形成对引号的匹配，需要进一步改进。
2. 程序在获取配置文件（关键字表、运算符表等）的时候对文件格式的要求较高，不具有很好兼容性。
3. 程序没有制作良好的交互界面。
4. 对词法的分析规则基本上与市面上流行的C语言编译器保持一致，能够较为合理地对错误地单词进行截取。
5. 将一个词法分析器构建为一个对象，便于语法分析语义分析对程序的调用。
6. 使用STL相应算法，提高程序的性能。
7. 通过这次试验加强了对语言构词法地理解，也进一步领会了编译器地词法的处理方式。

## 实验二 LL(1)分析法

1. **实验目的**

通过完成预测分析法的语法分析程序，了解预测分析法和递归子程序法的区

别和联系。使学生了解语法分析的功能，掌握语法分析程序设计的原理和构造方

法，训练学生掌握开发应用程序的基本方法。有利于提高学生的专业素质，为培

养适应社会多方面需要的能力。

1. **写出 LL（1）分析法的思想及写出符合 LL（1）分析法的文法**

对文法G的句子进行确定的自顶向下语法分析的充分必要条件是，G的任意两个具有相同左部的产生式A—>α|β 满足下列条件：

（1）如果α、β均不能推导出ε，则 FIRST(α) ∩ FIRST(β) = ∅。

（2）α 和 β 至多有一个能推导出 ε。

（3）如果 β \*═> ε，则 FIRST(α) ∩ FOLLOW(A) = ∅。

将满足上述条件的文法称为LL(1)文法。

**First构造：**

对每一文法符号X∈VT∪VN构造FIRST(X)，连续使用下面的规则，直至每个集合FIRST不再增大为止：

1. 若X∈VT，则FIRST(X)＝{X}。

2. 若X∈VN，且有产生式X→a…，则把a加入到FIRST(X)中；若X→ε也是一条产生式，则把ε也加到FIRST(X)中。

**Follow构造：**

对于文法G的每个非终结符A构造FOLLOW(A)的办法是，连续使用下面的规则，直至每个FOLLOW不再增大为止：

1. 对于文法的开始符号S，置＃于FOLLOW(S)中；

2. 若A→αBβ是一个产生式，则把FIRST(β)\{ε}加至FOLLOW(B)中；

3. 若A→αB是一个产生式，或A→αBβ是一个产生式而β→ε (即ε∈FIRST(β))，则把FOLLOW(A)加至FOLLOW(B)中。

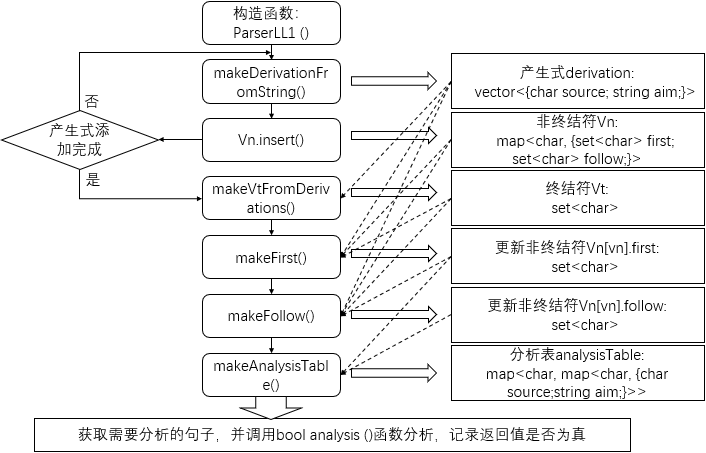
**文法1：**

1. E->TG
2. G->+TG|—TG
3. G->ε
4. T->FS
5. S->\*FS|/FS
6. S->ε
7. F->(E)
8. F->i

**文法2：**

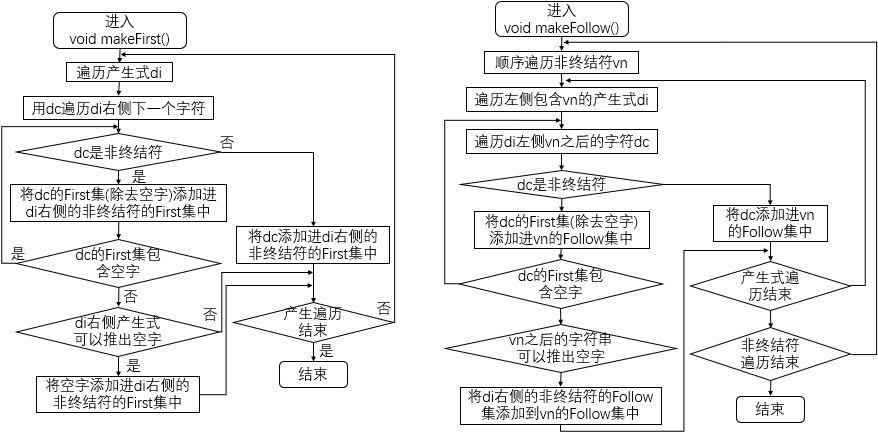
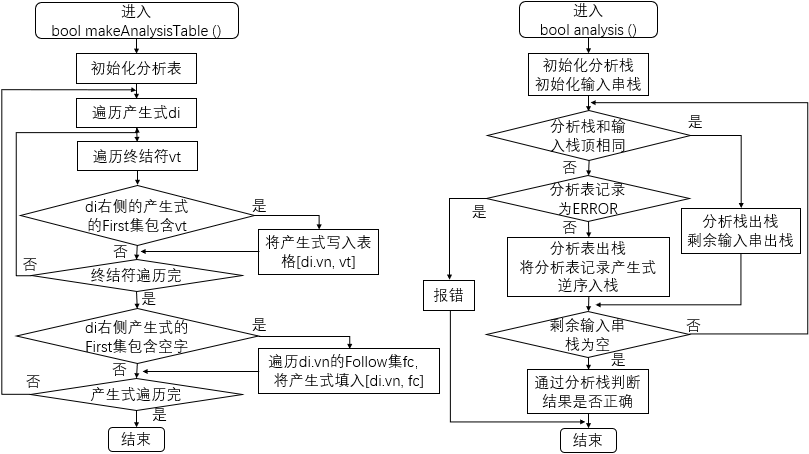
1. E→TE′
2. E′→+TE′ | ε
3. T→FT′
4. T′→\*FT′ | ε
5. F→(E) | i
6. **程序结构描述**

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名称及调用格式返回类型 | 参数含义 | 返回值描述 | 函数功能 |
| ParserLL1(const char\* fileName); | 文法文件的路径 | 无 | 构造函数，获取文法，并调用相关函数完成分析程序的初始化 |
| void makeDerivationFromString(string s); | 字符串形式的产生式s | 无 | 对字符串形式的产生式进行结构化。结构化形式：{  source：产生式左侧  aim：产生式右侧} |
| bool isVn(char c); | 任意字符 | 字符c是终结符返回true，否则返回false | 判断字符c是否是非终结符 |
| void makeVtFromDerivations() | 无 | 无 | 从产生式分解出终结符 |
| void makeFirst(); | 无 | 无 | 创建非终结符的First集 |
| void makeFollow(); | 无 | 无 | 创建非终结符的Follow集 |
| void makeAnalysisTable(); | 无 | 无 | 创建分析表 |
| string getStringFromVector(vector<char> vc); | vector<char>类型的变量 | string类型字符串 | 将vc中的内容转换为字符串 |
| void recordResult(int step, string analysis, string leave, derType useDer, int operate); | 步骤号、分析栈、剩余字符串、产生式、操作符 | 无 | 记录下分析过程 |



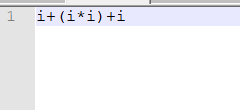
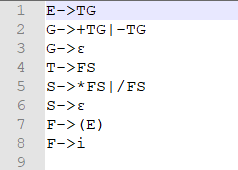
注：LL1分析器的初始化工作全部在构造函数中完成，初始化的工作为获得相关文法的First集、Follow集进而创建分析表，每一个步骤依赖于之前步骤所产生的结果。创建完对应文法的分析表后可以独立调用句子分析器，多次独立分析各种句子，便于和词法分析器、语义分析器联合工作。

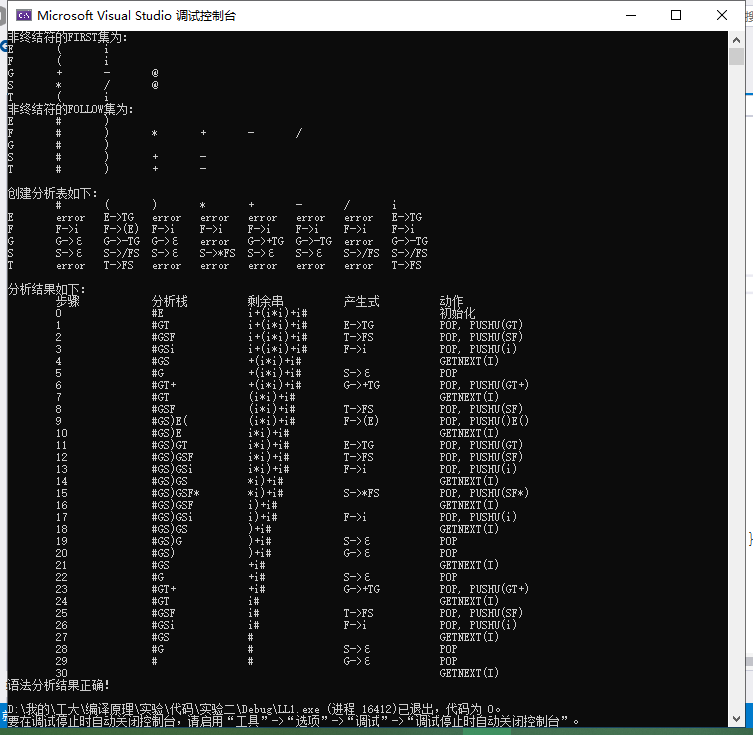
1. **详细的算法描述**



1. **给出软件的测试方法和测试结果。**

下图为默认文件中保存的文法和需要进行分析的句子：





1. **实验总结**
2. 程序在获取配置文件（关键字表、运算符表等）的时候对文件格式的要求较高，不具有很好兼容性。
3. 程序没有制作良好的交互界面，对于分析内容的修改仅限于在文件中进行。
4. 没有对文法进行左递归消除转换，需进一步加强。
5. 程序产生分析表与推到结果能保持一直，对文法的处理具有较强的兼容性。
6. 将一个语法分析器构建为一个对象，便于语义分析对程序的调用。
7. 在处理语法时，将语法的初始化与句子的分析高度分离，可以更加便捷地应用于实际生产，与其他程序的结合。
8. 程序对数据结构进行了创新，有较好的拓展性，使程序的逻辑相对更加清晰，操作更加简便。
9. 使用STL相应算法，提高程序的性能。
10. 通过这次试验加强了对语言构词法地理解，也进一步领会了编译器地词法的处理方式。

## 实验三 LR(1)分析法

1. **实验目的**

构造 LR(1)分析程序，利用它进行语法分析，判断给出的符号串是否为该文法识别的句子，了解 LR（K）分析方法是严格的从左向右扫描，和自底向上的语法分析方法。

1. **描述 LR(1)语法分析程序的设计思想**

**1.提取所有有效识别活前缀的式子：**

形式上我们说一个LR(1)项目[A→α·β, a]对于活前缀γ是有效的，如果存在规范推导其中，

1) γ＝δα；

2) a是ω的 第一个符号，或者a为#而ω为ε。

[A→α·Bβ, a]对活前缀γ＝δα是有效的，则对于每个形如B→ξ的产生式， 对任何b∈FIRST(βa)，[B→·ξ, b]对γ也是有效的。

**2.项目集I 的闭包CLOSURE(I)构造：**

1. I的任何项目都属于CLOSURE(I)。
2. 若项目[A→α·Bβ, a]属于CLOSURE(I)，B→ξ 是一个产生式，那么，对于FIRST(βa) 中的每个终结符b，如果[B→·ξ, b]原来不在CLOSURE(I)中，则把它加进去。
3. 重复执行步骤2，直至CLOSURE(I)不再增大为止。

**3.GO构造：**

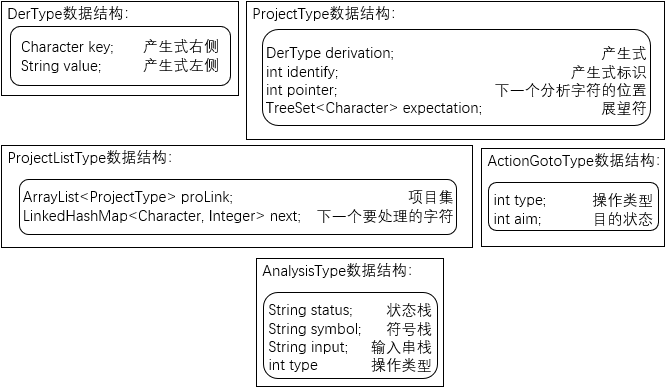
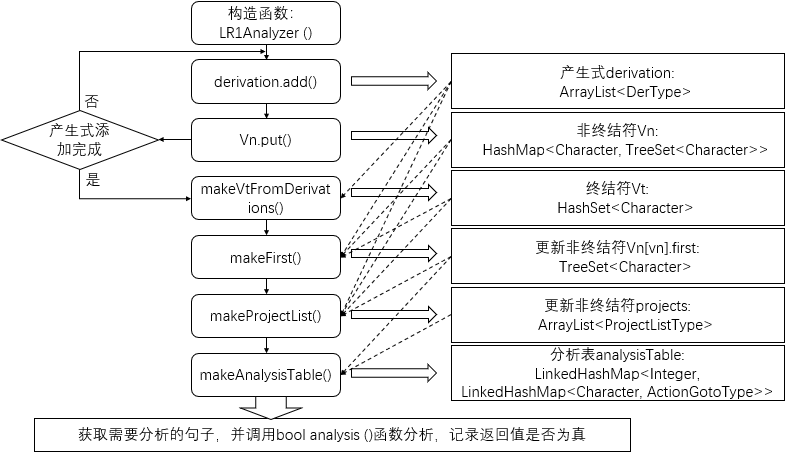
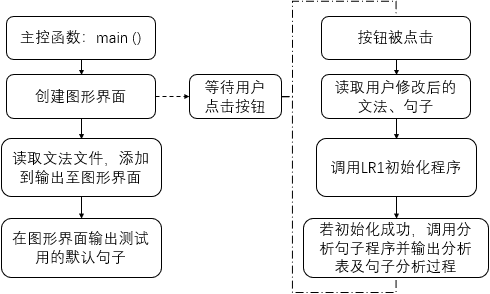
令I是一个项目集，X是一个文法符号，函数GO(I，X)定义为：GO(I，X)＝CLOSURE(J)其中J＝{任何形如[A→αX·β, a]的项目| [A→α·Xβ, a]∈I}

**4.分析表构造：**

令每个Ik的下标k为分析表的状态，令含有[S′→·S, #]的Ik的k为分析器的初态。

1. 若项目[A→α·aβ, b]属于Ik且GO(Ik, a)＝Ij， a为终结符，则置ACTION[k, a]为 “sj”。
2. 若项目[A→α·，a]属于Ik，则置ACTION[k, a]为 “rj”；其中假定A→α为文法G′的第j个产生式。
3. 若项目[S′→S·, #]属于Ik，则置ACTION[k, #]为 “acc”。
4. 若GO(Ik，A)＝Ij，则置GOTO[k, A]=j。
5. 分析表中凡不能用规则1至4填入信息的空白栏均填上“出错标志”。
6. **程序结构描述：**

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名称及调用格式返回类型 | 参数含义 | 返回值描述 | 函数功能 |
| LR1Analyzer(String grammar, String sentence) | 从程序界面获得的文法和需要分析的句子 | 无 | 构造函数，对文法进行分析，调用该类中的部分函数进行分析并构造分析表，完成文法的初始化工作 |
| boolean isVn(char v) | 任意字符v | 若v是非终结符则返回true若不是则返回false | 判断数组str1中是否存在str2 |
| void makeVtFromDerivations() | 无 | 无 | 从产生式获取终结符 |
| void makeFirst() | 无 | 无 | 创建非终结符的First集 |
| TreeSet<Character> getFirst(String s, TreeSet<Character> first) | 字符串、展望符集 | 构成First集的set集合 | 输入产生式的部分字符串，项目的展望符集，创建获得First集，用于构成新的展望符集 |
| void makeProjectList() | 无 | 无 | 构建项目集族 |
| boolean makeAnalysisTable() | 无 | 创建LR1分析表成功返回true，否则返回false | 创建LR1分析表，并在项目表有冲突的时候报错 |
| Object[] getAnalysisTable() | 无 | 返回一个只有两个元素的对象数组，第一个元素是表格头，第二个元素是表格体 | 对具有结构化的分析表数据去结构化，用二维数组表示表格，供其他结构调用，如：用户界面控制程序调用，将分析表输出至用户界面 |
| boolean analysis(StringBuilder sentence) | 输入串 | 分析输入串成功则返回true，否则false | 利用分析表对输入串进行自下而上的分析。 |
| Object[] getResultTable() | 无 | 返回一个只有两个元素的对象数组，第一个元素是表格头，第二个元素是表格体 | 对具有结构化的分析表数据去结构化，用二维数组表示表格，供其他结构调用，如：用户界面控制程序调用，将分析表输出至用户界面 |



注： LR1分析器的初始化工作全部在构造函数中完成，初始化的工作为获得相关文法的First集进而创建项目集族、分析表，每一个步骤依赖于之前步骤所产生的结果。创建完对应文法的分析表后可以独立调用句子分析器，多次独立分析各种句子，便于和词法分析器、语义分析器联合工作。

程序在物理上将Action表和Goto表合并为analysisTable表，但在逻辑上进行了区分。

1. **详细的算法描述**

(1)移进：

table[i，a]= Sj：状态 j 移入到状态栈，把a移入到文法符号栈，其中 i,j 表示状态号。

(2)归约：

table[i，a]=rk：当在栈顶形成句柄时，则归约为相应的非终结符 A，即文法中有 A- B 的产生式，若 B 的长度为 R(即|B|=R)，则从状态栈和文法符号栈中自顶向下去掉 R 个符号，即栈指针 SP 减去 R，并把 A 移入文法符号栈内，j=GOTO[i,A]移进状态栈，其中 i 为修改指针后的栈顶状态。

(3)接受 acc:

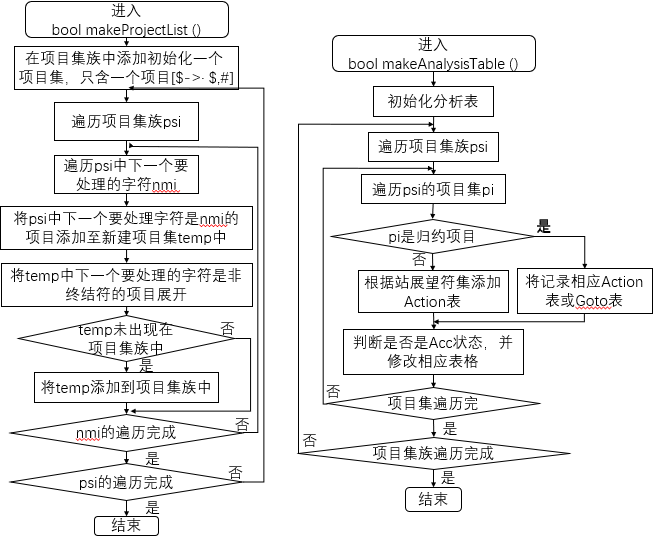
当归约到文法符号栈中只剩文法的开始符号 S 时，并且输入符号串已结束即当前输入符是'#'，则为分析成功,当table中该项为acc时分析成功接受。

(4)报错:

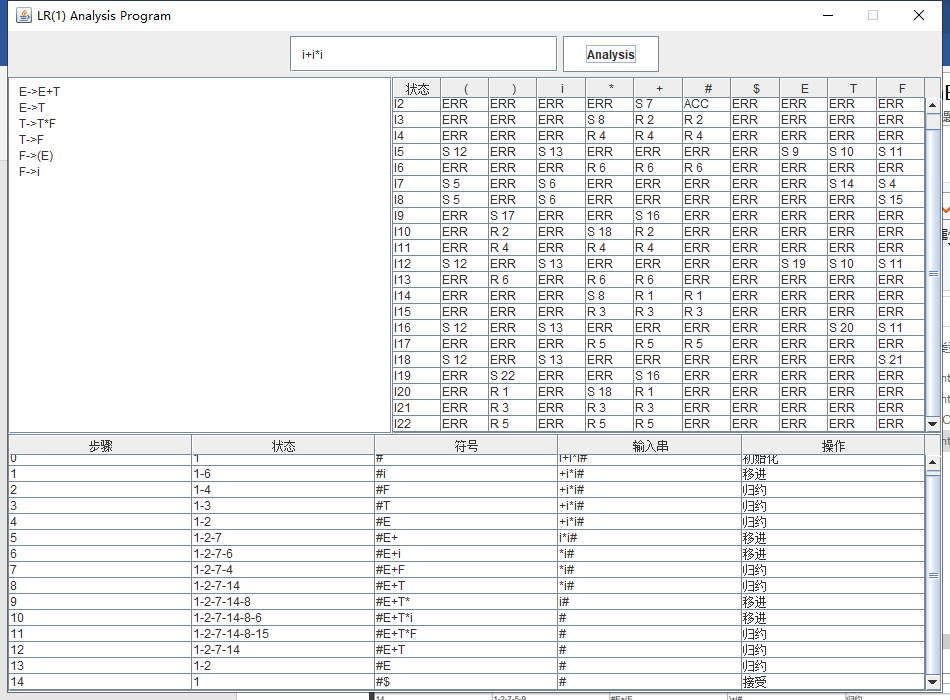
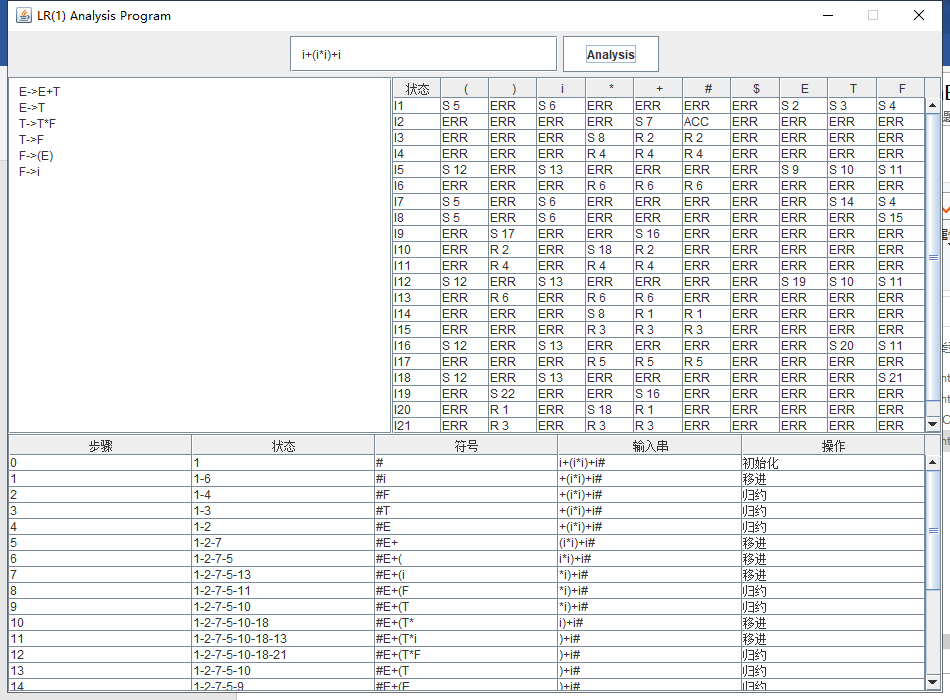
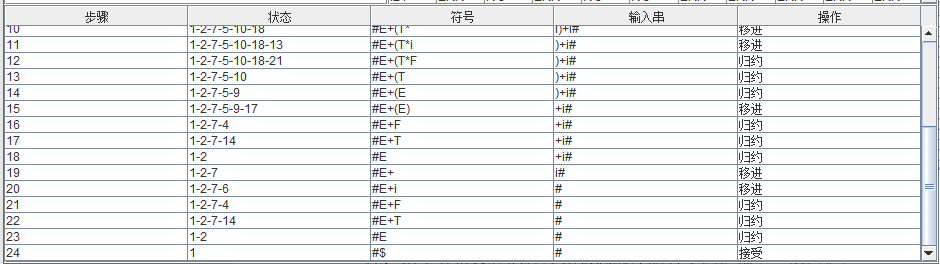
当遇到状态栈顶为某一状态下出现不该遇到的文法符号时，则报错，说明输

入端不是该文法能接受的符号串。

注：First集的创建同LL1实验，不再详细描述。



1. **给出软件的测试方法和测试结果。**

注：分析表由项目集族创建，其中项目集族中的0号项目集是[$->· $,#]，是无效项目集（为避免降低程序可读性未删除），因此在创建分析表时，跳过了0号项目集，同时为了保持状态号与项目集族的对应，取消了0号状态，因此最终生成的分析表从1号状态开始。

1. **实验总结**
2. 此次试验中对于Vn、Vt、项目、First集合等的创建均与LL1实验的数据结构、算法保持一致，仅将代码从C++迁移至Java，同时改用了更加合理的Java内置类型。
3. 实验中文法仍采取读文件的形式获取，对文件格式的要求与LL1一致，但引入了操作界面，可以直接在操作界面上修改文法内容。使用操作界面对程序的调试也带了较大的便利。但界面美观度有待改善。
4. 实验中对空字在不同位置进行了不同的转化，主要是为了适应字符编码的不同以及占用字节空间的不同。
5. 实验中默认非终结符、终结符只有一个字符，降低了程序的拓展性，结合课本后续内容，可以考虑将终结符、非终结符定义为字符串，在文法中使用空格分离，以提升程序处理能力。
6. 在创建项目集族时引入了一个项目[$->· $,#]，使得后续程序具有了相同的结构，可以放入同一个循环中直接完成，提升了代码的阅读性、简洁性。
7. 在定义数据类型时，添加了一些解释性的属性，如使用int类型区分分析表中的不同操作，并提供了一个数组用于将int数字对应为相应的文字，提升了方便了程序的修改。