



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 | 2018211958 孙淼 |
| 任 课 教 师 | 李宏芒老师 |
| 实验指导教师 | 李宏芒老师 |
| 实验地点 | 家中 |
| 2019~2020 学年第二学期 | |

## 实验一 词法分析设计

**一、实验目的**

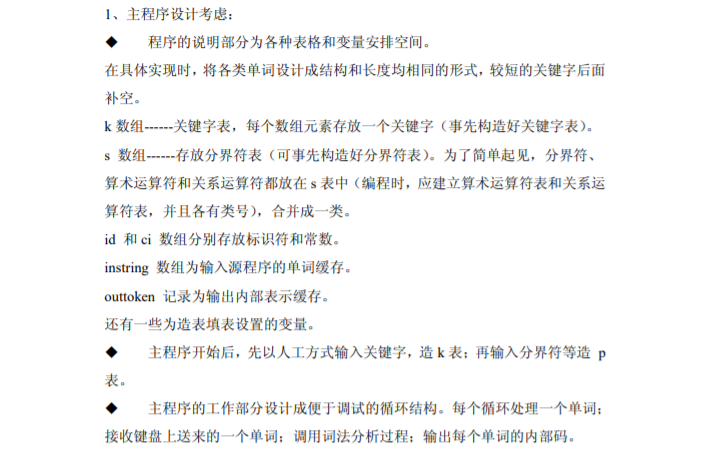
通过本实验的编程实践，使学生了解词法分析的任务，掌握词法分析程序设计的原理和构造方法，使学生对编译的基本概念、原理和方法有完整的和清楚的理解，并能正确地、熟练地运用。

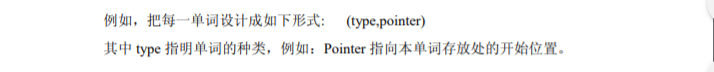
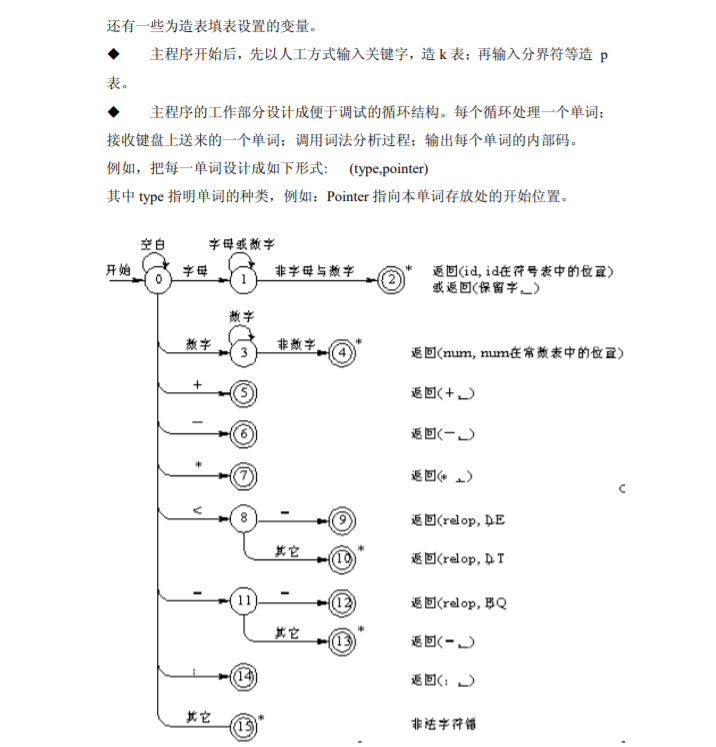
**二、实验内容**

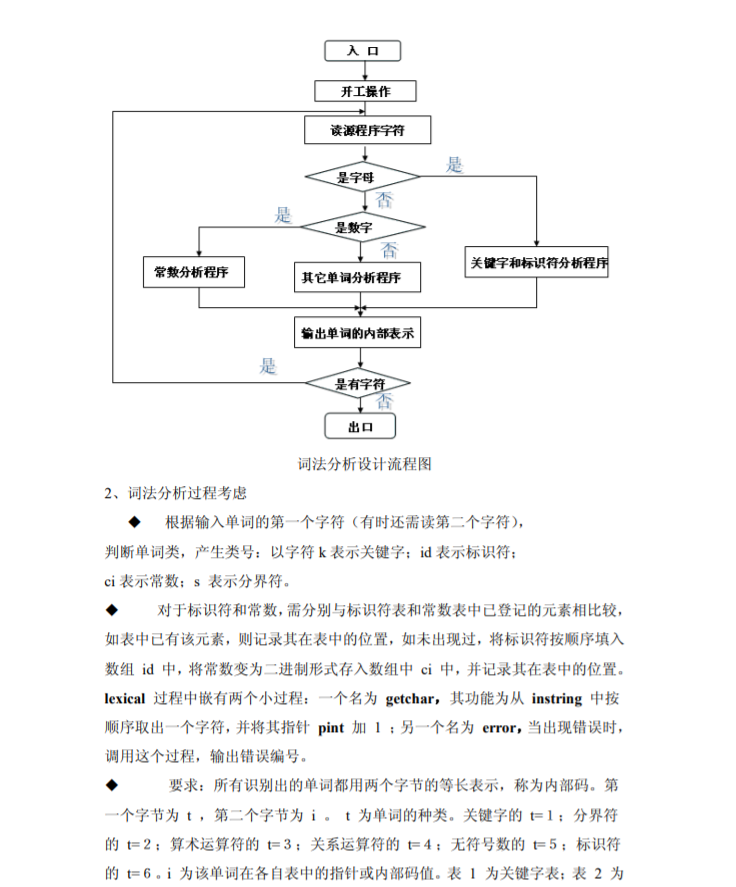
用C++语言实现对C语言子集的源程序进行词法分析。通过输入源程序从左到右对字符串进扫描和分解，依次输出各个单词的内部编码及单词符号自身值，若遇到错误则显示“Error”，然后跳过错误部分继续显示；同时进行标识符登记符号表的管理。

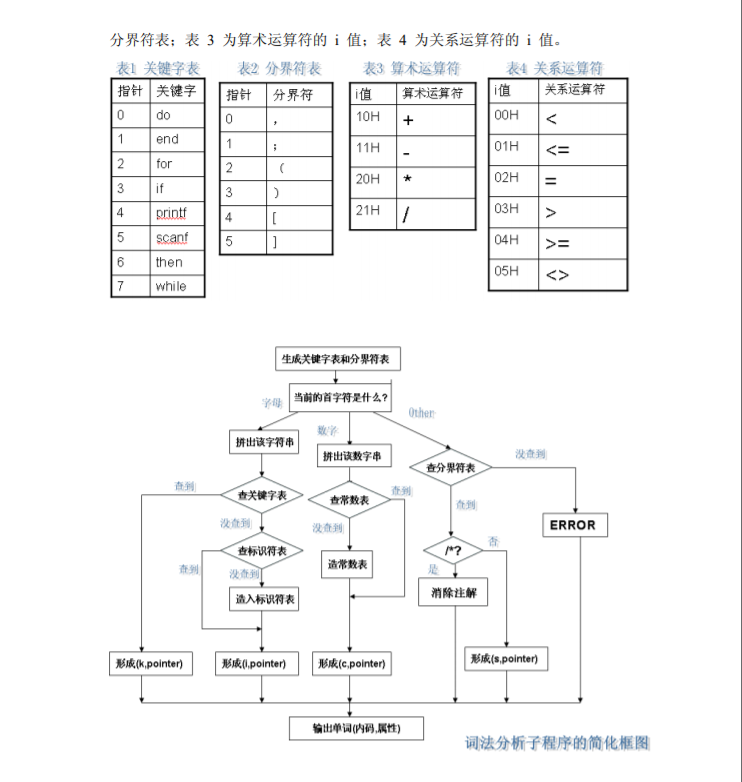
以下是实现词法分析设计的主要工作：

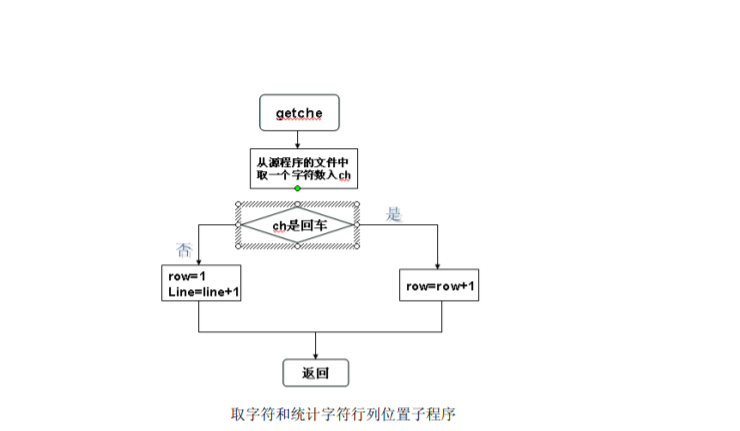
1. 从源程序文件中读入字符。
2. 统计行数和列数用于错误单词的定位。
3. 删除空格类字符，包括回车、制表符空格
4. 按拼写单词（关键字、标识符、常数、运算符、关系运算符、分界符号），并用（内码，属性）二元式表示。（属性值——token的机内表示）
5. 如果发现错误则报告出错
6. 根据需要是否填写标识符表供以后各阶段使用

**三、词法分析实验设计思想及算法**

****

****

****

**四、实验要求**

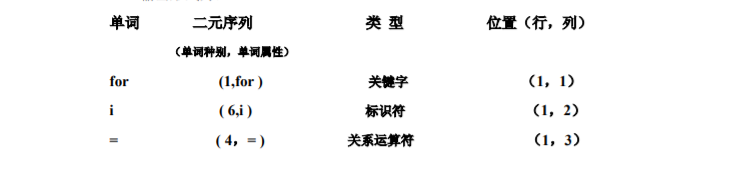
1.编程时注意编程风格：空格的使用，注释的使用，缩进的使用等。

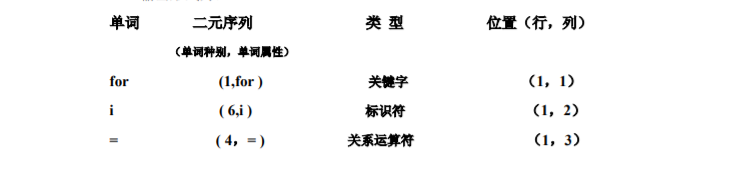
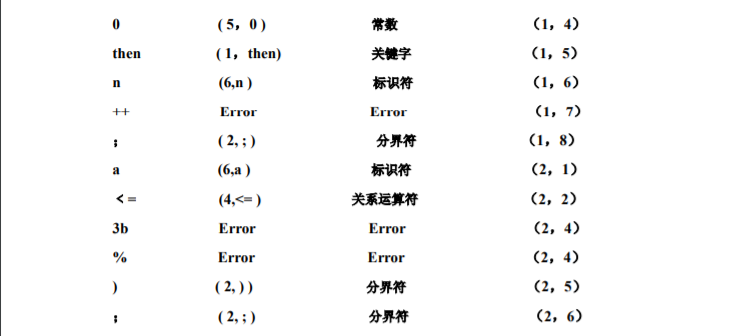
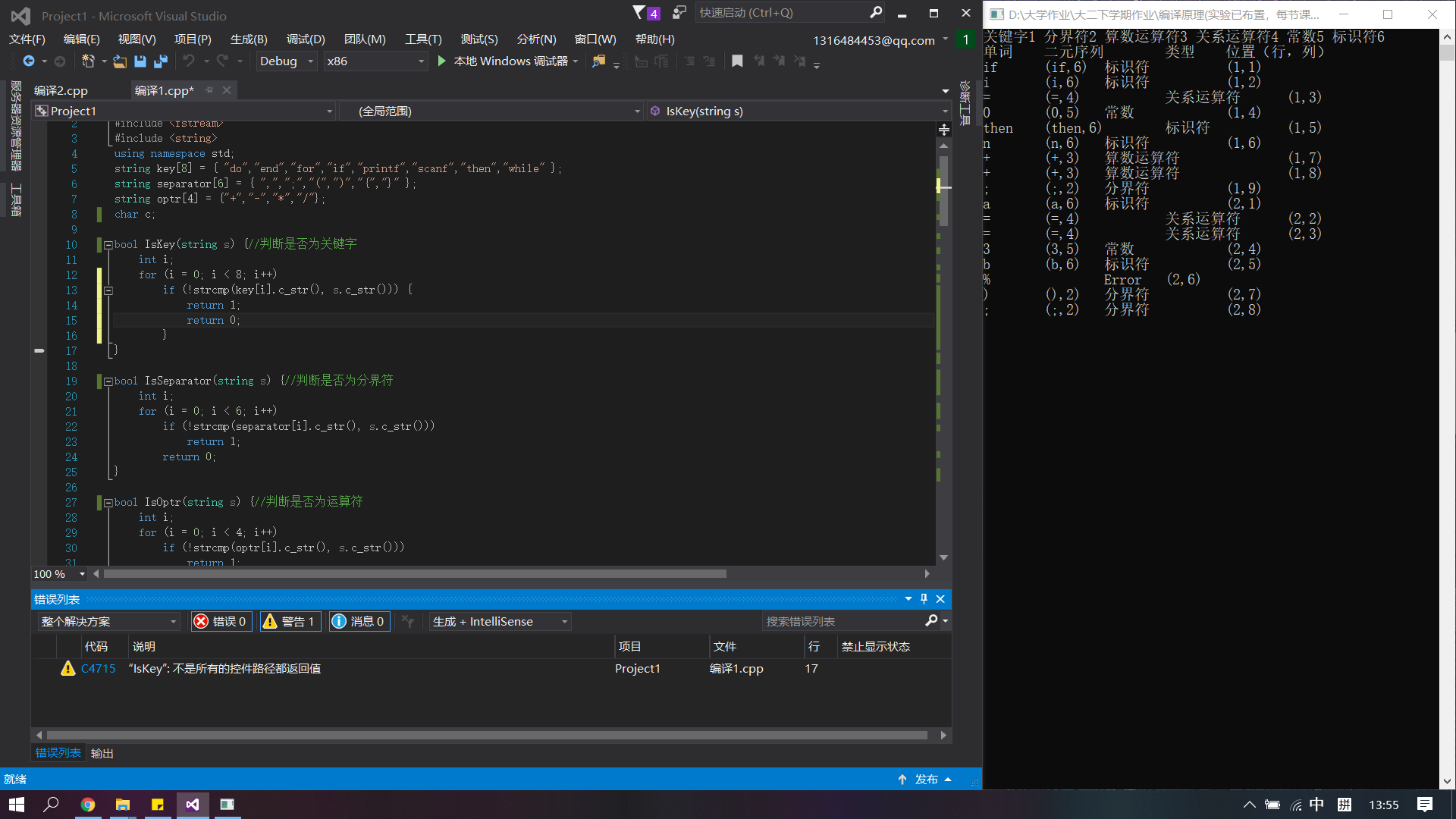
2.将标识符填写的相应符号表须提供给编译程序的以后各阶段使用。

3.根据测试数据进行测试，测试实例应包括以下三个部分：

4.词法分析程序设计要求输出示例：

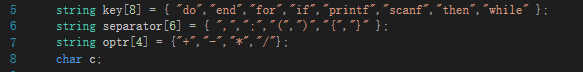
输出形式为：

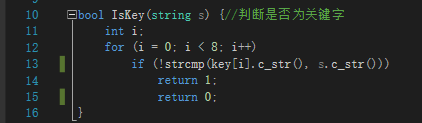


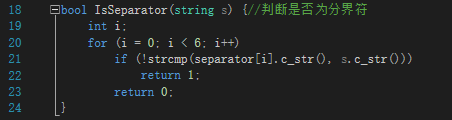
******五、实验步骤**

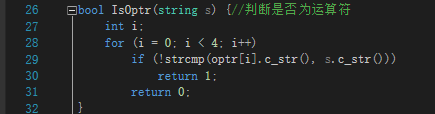
代码共159行，实现了词法分析：

实现对C语言子集的源程序进行词法分析。通过输入源程序从左到右对字符串进扫描和分解，依次输出各个单词的内部编码及单词符号自身值，若遇到错误则显示“Error”，然后跳过错误部分继续显示；同时进行标识符登记符号表的管理。

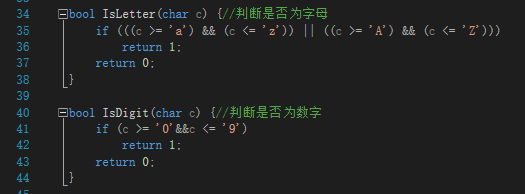
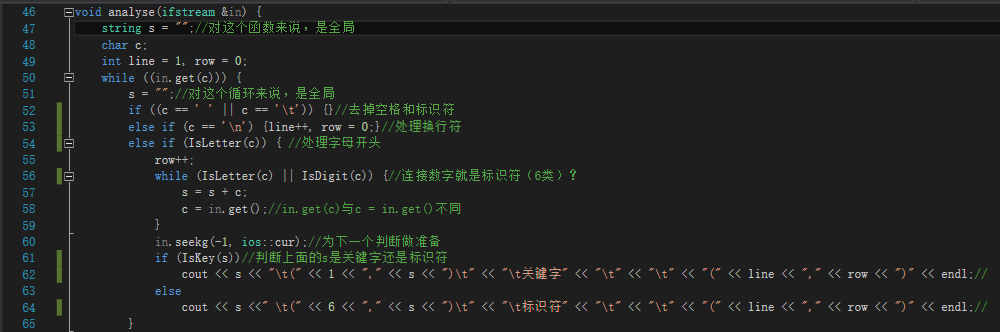


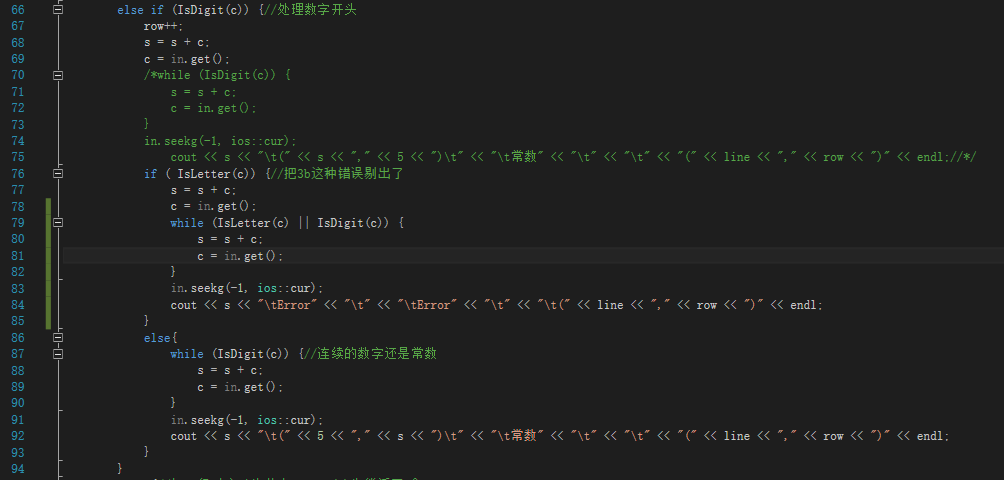
自定义三个数组，分别将给定的关键字，分界符，运算符都存进去，便于下一步设计函数。

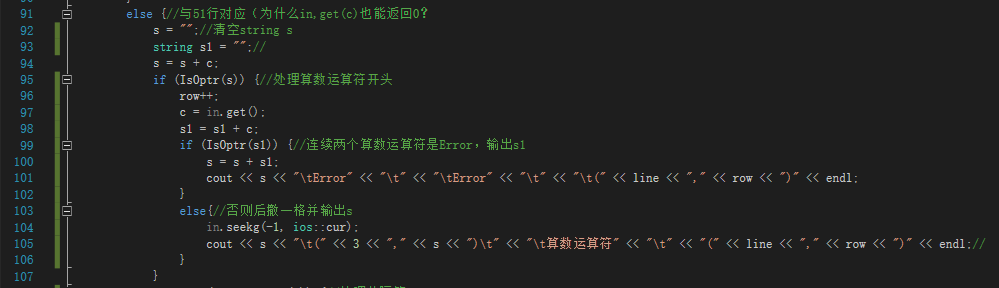
此为判断关键词函数，返回bool逻辑值，通过内置的strcmp（）实现了对字符串是否相等的判断。

此为判断分界符函数，返回bool逻辑值，，通过内置的strcmp（）实现了对分界符是否相等的判断。

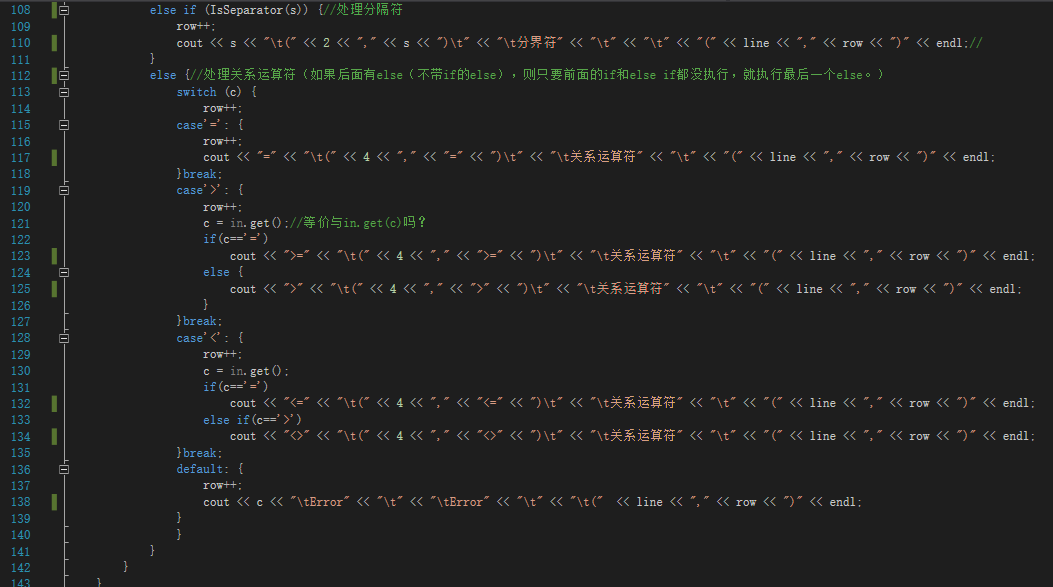
此为判断运算符函数，返回bool逻辑值，，通过内置的strcmp（）实现了对运算符是否相等的判断

原理同上

这段代码通过if…else if…实现了对空格，制表符的处理，实现了对换行符的处理，然后开始处理字母开头的词，通过嵌套while函数再多读进字符，进一步使用if…else…将字母开头的词分为了关键词还是标识符，并实现对结果的输出。

这段代码先通过if将数字串任何地方出现字母的判为Error，然后对连续的数字仍然判定为常数字。

这段代码同上段代码的原理，也是先用if将连续两个运算符如++剔除，然后在else中实现对正常形式的运算符的输出。

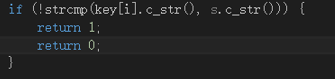
这段代码通过switch()case:选择的方式来对多个关系运算符进行处理，通过比照实验要求可知，在以>,<开头的关系运算符时，都会存在其他情况，于是在他们的case中，加入if…else…实现对<=,<>,>=的判断并输出。

总结如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| Bool IsKey(string s); | 一个通过s=s+c;  c=in.get();读入的字符串s | Bool类型，即是否是关键字 | 判断是否为关键字 |
| bool IsSeparator(string s)； | 一个通过s=s+c;  c=in.get();读入的字符串s | Bool类型，即是否是分界符 | 判断是否为分界符 |
| bool IsDigit(char c)； | 一个通过c=in。get();读入的字符c | Bool类型，即是否是数字 | 判断是否为数字 |
| void analyse(ifstream &in) | 通过ifstream读入的文件的输入流in | 无 | 复杂的内容，通过调用上面几个布尔函数配合各种分支语句，以及switch判断关系运算符，实现了在159行完成词法分析的功能，具体内容见截图与相应的分析 |

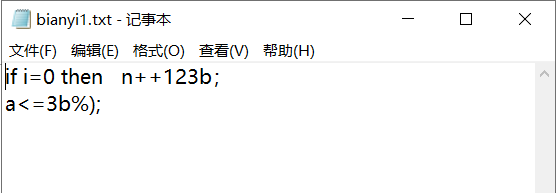
初测试结果如下：结果中总是把if，then识别为非关键字而且判断为6类。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 |
| 警告 | C4715 | “IsKey”: 不是所有的控件路径都返回值 | Project1 | d:\大学作业\大二下学期作业\编译原理(实验已布置，每节课看书ppt)\实验\project1\project1\编译1.cpp | 17 |  |

后来发现是在

此处多打了一个{}，导致两个return被框在一起，也就是if判断为否的时候没有对应的return了 ，这再一次说明了代码编写的严谨性，一个小错就导致后面都不对。

实验结果如下：

文件bianyi1.txt

运行结果如下：

**六、实验总结**

1.switch函数的使用可以大大简化函数的复杂度，代码思路也清晰很多

2. in.seekg(-1, ios::cur);语句很关键，多处需要多读一个字符的地方都需要这个函数来回退一个光标，来为下一步分析打基础。

3.自己在验收前对词法分析的理解还不够深刻，经过老师一番点拨，自己又重新研读了ppt，领悟了很多，学到了很多，发现自己在思考问题方面还不够全面，容易犯想当然的错误，。

## 实验二 LL(1)分析法

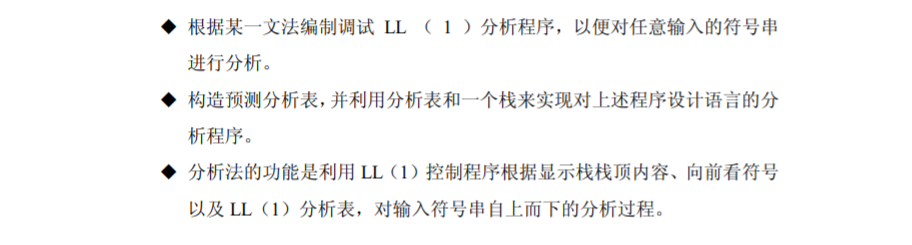
1. **实验目的**

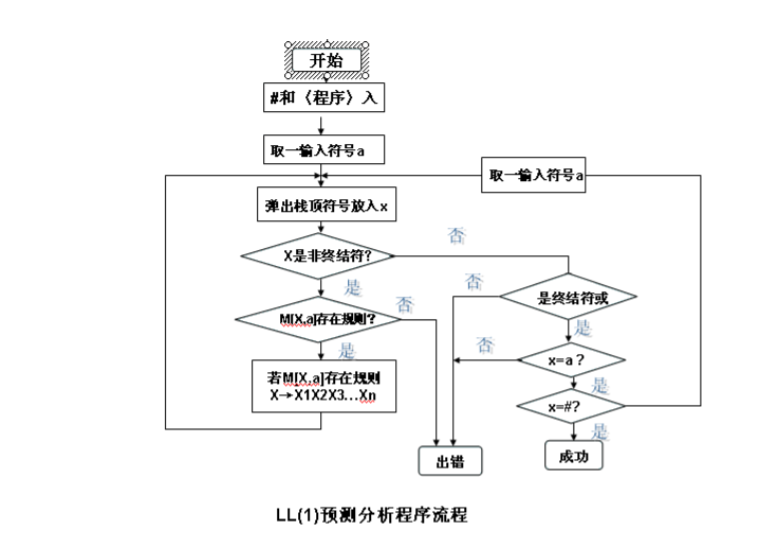
通过完成预测分析法的语法分析程序，了解预测分析法和递归子程序法的区

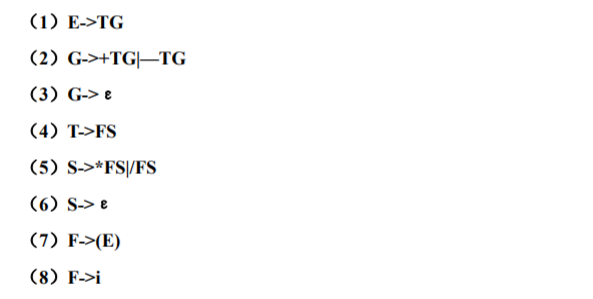
别和联系。使学生了解语法分析的功能，掌握语法分析程序设计的原理和构造方

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. **实验内容**
2. **LL（1）分析法实验设计思想及算法**
3. 定义部分：定义常量、变量、数据结构。
4. 初始化：设立LL（1）分析表、初始化变量空间（包括堆栈、结构体、数组、临时变量等）；
5. 控制部分：从键盘输入一个表达式符号串；
6. 利用LL（1）分析算法进行表达式处理：根据LL（1）分析表符号串进行堆栈（或其他）操作，输出分析结果，如果遇到错误则显示错误信息。

**四、实验要求**

1. 编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。
2. 如果遇到错误的表达式，应输出错误提示信息。
3. 对下列文法，用LL（1）分析法对任意输入的符号串进行分析：

输出的格式如下：

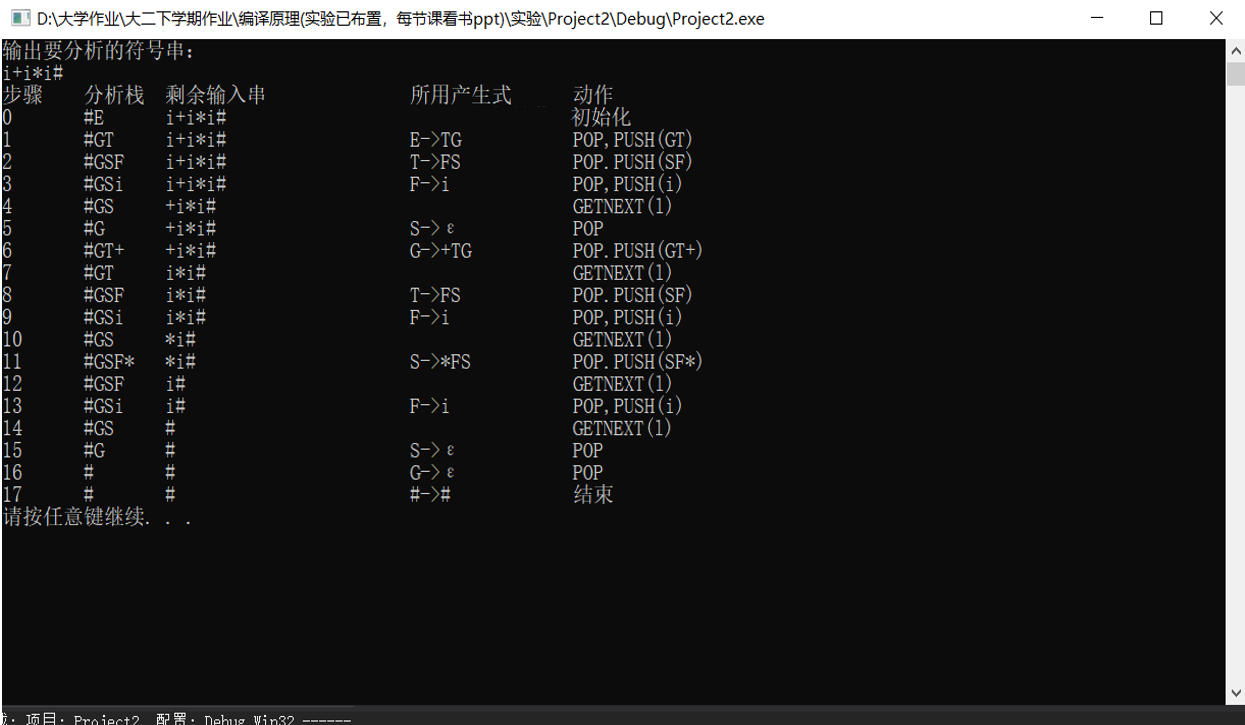
* 1. **实验步骤**

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：**

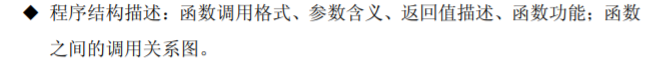
对于文法G的每个非终结符A构造FOLLOW(A)的办法是，连续使用下面的规则，直至每个FOLLOW不再增大为止：

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；

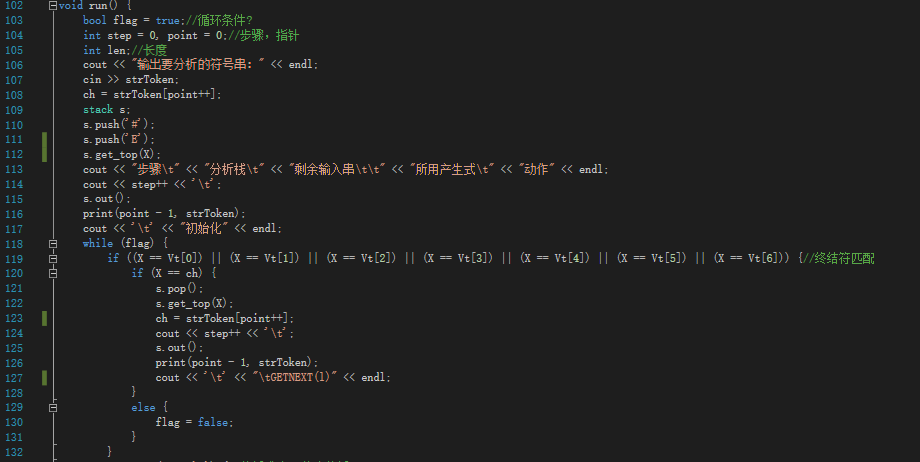
3. 若A→αB是一个产生式，或A→αBβ是一个产生式而β→ε (即ε∈FIRST(β))，则把FOLLOW(A)加至FOLLOW(B)中。

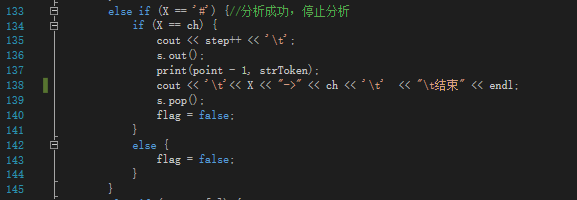
**文法1：**

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

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| stack::stack() | 无 |  | 构造函数，初始化一个栈 |
| bool stack::empty() const | 无 | 若判断为空栈，则返回true若不为空栈，则返回false | 判断是否为空栈 |
| bool stack::full() const | 无 | 若判断为满栈，则返回true，若不为满栈，则返回false | 判断是否为满栈 |
| bool stack::get\_top(char &c)const | 地址 | 逻辑值 | 若空栈，返回false，若不空，将栈顶元素赋值给c，返回ture |
| bool stack::push(const char c) | 字符c | 逻辑值 | 若满栈，返回false，否则压c进栈，返回ture |
| bool stack::pop() | 无 | 逻辑值 | 若空栈，返回false，否则，出栈顶元素 |
| void stack::out()) | 无 | 逻辑值 | 输出栈内元素 |
| int length(char \*c) | 字符指针 | 整型值 | 返回字符串长度 |
| void print(int i, char \*c) | 无 | 整型值，字符指针 | 打印从i至后的内容 |
| void run() | 无 | 无 | 复杂的循环对待规约的字符串进行处理 |

进入run()函数，输入待规约字符串后，对ch赋值为待规约字符串的首位，并对point+1为1，建栈，先入后出压入#和非终结符E，取栈顶元素E给X，…一系列输出后，进入循环，当X为终结符时，判断是否为ch，若是，终结符匹配，若否，出错；

若X=ch=#，则分析完毕，停止分析，若X=#！=ch,出错。

若X=Vn[0]=E，先判断是否对应分析表中的是否都是error，若不是，则将X=E出栈，对len赋值为产生式右端长度-1，并倒序将其压栈，进入下一步骤，输出栈内元素，规约式(-1),所用产生式，动作，再取栈顶为下一步循环做准备。后面的循环都和这个原理类似，不再赘述。

**实验二的周二补充：对first和follow集以及预测分析表的自动生成**

**文法要求：**

（1）从文件读入，每条产生式占用一行

（2）文法为LL(1)文法

从文件中读入文法，从键盘上输入待分析的符号串，采用 LL(1)分析算法判断该符号串是否为该文法的句子。

**实验思路**：

首先实现集合FIRST(X)构造算法和集合FOLLOW(A)构造算法，再根据FIRST和FOLLOW集合构造出预测分析表，并对指定的句子打印出分析栈的分析过程，判断是否为该文法的句子。

**实验原理：**

（1）first集的算法思想

如果产生式右部第一个字符为终结符，则将其计入左部first集

如果产生式右部第一个字符为非终结符执行以下步骤

求该非终结符的first集

将该非终结符的非$first集计入左部的first集

若存在$，则将指向产生式的指针右移

若不存在$，则停止遍历该产生式，进入下一个产生式

若已经到达产生式的最右部的非终结符，则将$加入左部的first集

处理数组中重复的first集中的终结符

（2）follow集的算法思想

对于文法G中每个非终结符A构造FOLLOW(A)的办法是,连续使用下面的规则,直到每个FOLLOW不在增大为止.

对于文法的开始符号S,置#于FOLLOW(S)中;

若A->aBb是一个产生式,则把FIRST(b)\{ε}加至FOLLOW(B)中;

若A->aB是一个产生式,或A->aBb是一个产生式而b=>ε(即ε∈FIRST(b))则把FOLLOW(A)加至FOLLOW(B)中

（3）生成预测分析表的算法思想

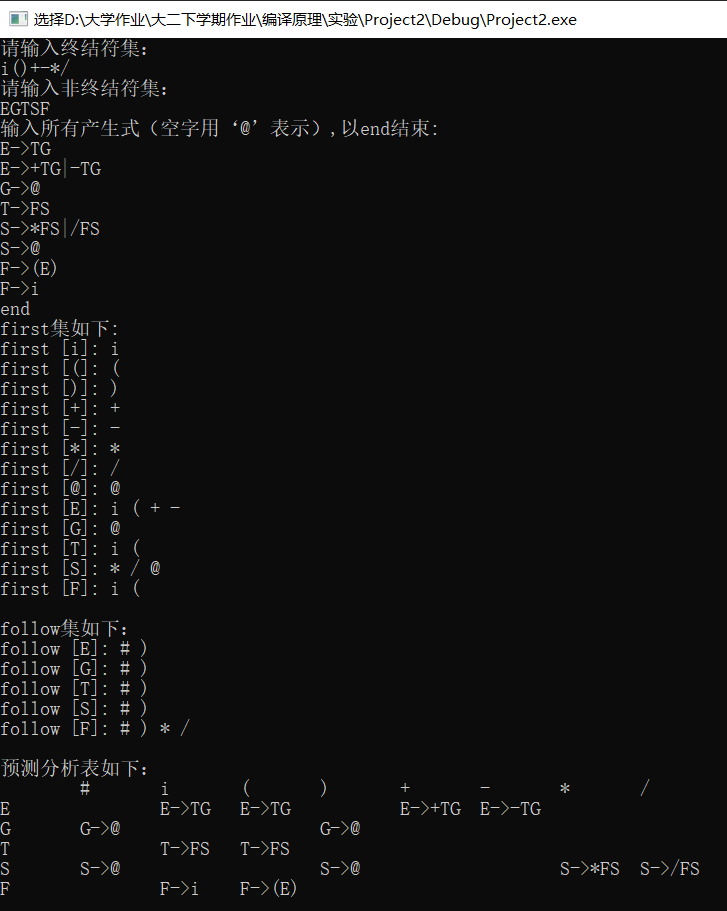
构造分析表M的算法是：

对文法G的每个产生式A->a执行第二步和第三步;

对每个终结符a∈FIRST(a),把A->a加至M[A,a]中;

若ε∈FIRST(a),则把任何b∈FOLLOW(A)把A->a加至M[A,b]中;

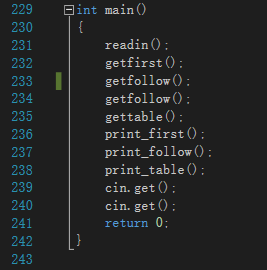
把所有无定义的M[A,a]标上出错标志.

**实验结果截图：**

**程序结构描述：函数调用格式，参数含义，返回值描述，函数功能。**

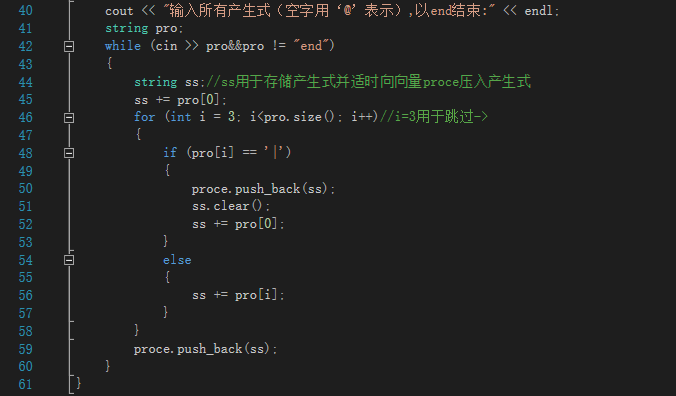
|  |  |  |  |
| --- | --- | --- | --- |
| 函数名称及调用格式返回类型 | 参数含义 | 返回值描述 | 函数功能 |
| void readin() | 无 | 无 | 将输入的终结符，非终结符读入map getnum中和数组get\_char中，将table初始化，将产生式进入到proce向量中， |
| void set\_union(string &a, string b) | 一个字符类型引用a和一个字符串b | 无 | 取a,b并集赋值给a |
| string get\_f(int vn, int & has\_0) | int型vn和int型引用has  \_0 | 字符串 | 将该非终结符的非空集的first集计入该左部非终结符的的first集 |
| void getfirst() | 无 | 无 | 调用get\_f补充,求first集合 |
| void getfollow() | 无 | 无 | 求follow集合 |
| void gettable() | 无 | 无 | 得预测分析表 |
| string get\_proce(int i) | 整型值 | 字符串 | 获得向量proce内每个对应的产生式 |
| void print\_first() | 无 | 无 | 输出first集合 |
| void print\_follow() | 无 | 无 | 输出follow集合 |
| void print\_table() | 无 | 无 | 输出预测分析表 |

**详细的算法描述（程序执行流程图）**

运行主程序

对终结符和非终结符进行输入并读入，实现通过getnum能得到字符对应的索引数，通过get\_char能得到索引数对应的字符，如终结符#对应1，1对应终结符#。

对产生式进行输入并读入，实现每个产生式的右侧都进入向量proce，遇到|则分开入。

对两个字符串求并集。

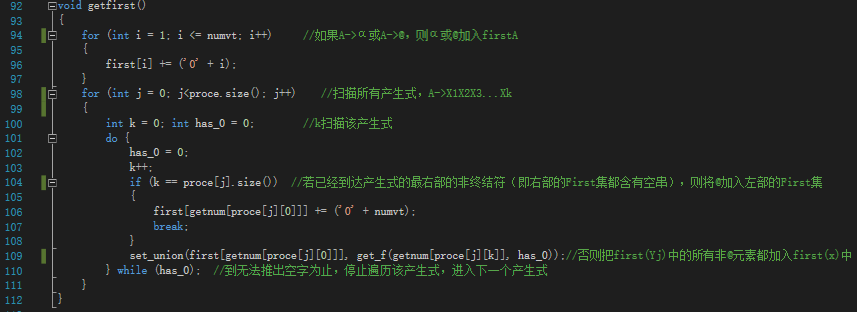
然后开始求first集合，按照算法：

1. 若X∈VT，则FIRST(X)＝{X}。

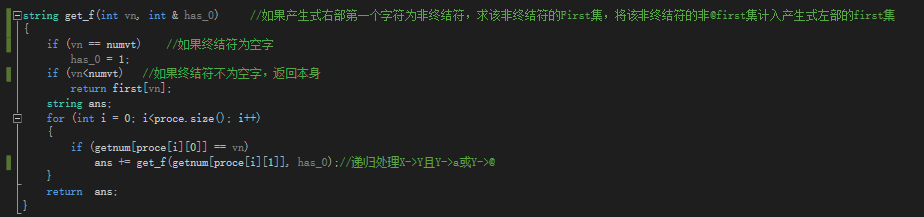
2. 若X∈VN，且有产生式X→a…，则把a加入到FIRST(X)中；若X→ε也是一条产生式，则把ε也加到FIRST(X)中。

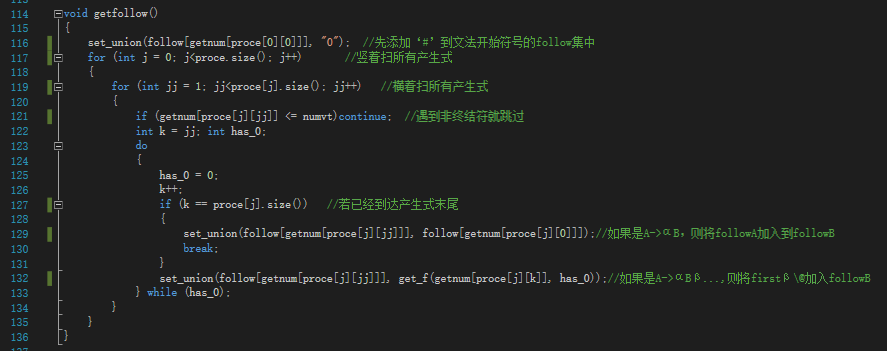
3.

若X→Y…是一个产生式且Y∈VN，则把FIRST(Y)中的所有非ε-元素都加到FIRST(X)中；

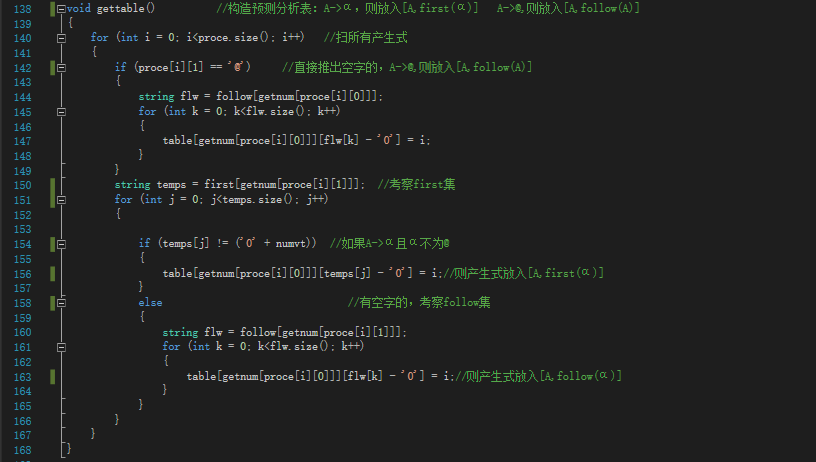
若X→Y1Y2…Yk是一个产生式，Y1，…，Yi-1都是非终结符，而且，对于任何j，1≤j≤i-1，FIRST(Yj)都含有ε(即Y1…Yi-1ε)， 则把FIRST(Yi)中的所有非ε-元素都加到FIRST(X)中；特别是，若所有的FIRST(Yj)均含有ε，j＝1，2，…，k，则把ε加到FIRST(X)中。

其中调用了get\_f,用递归的思想处理了如果产生式右部第一个字符为非终结符，求该非终结符的First集，将该非终结符的非$first集计入左部的first集的情况，get\_f与getfirst共同构成了对first集的自动生成。

接下来进行getfollow ();按照算法：

1. 对于文法的开始符号S，置#于FOLLOW(S)中；
2. 着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有限步推得ε (即ε∈FIRST(β))， 则把FOLLOW(A)加至FOLLOW(B)中

之后进行gettable();按照算法：

1. 对文法G的每一个产生式A->α执行第二步和第三步；
2. 对每个终结符a∈FIRST(α),把A->α加至M[A,a]中；
3. 若ε∈FIRST(α)，则对任何b∈FOLLOW(A)把A->α加至M[A,b]中。
4. 最后进行集合和预测分析表的输出。

**六、实验总结**

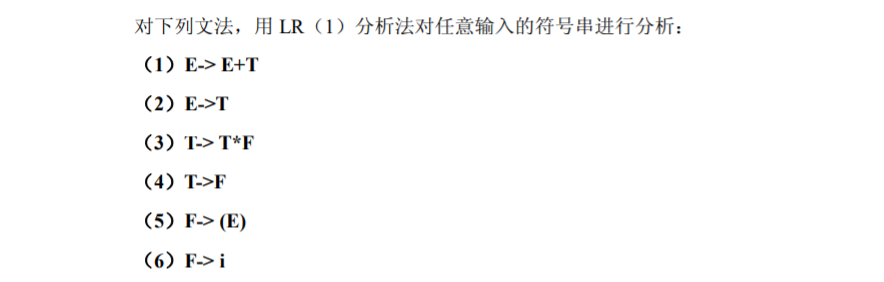
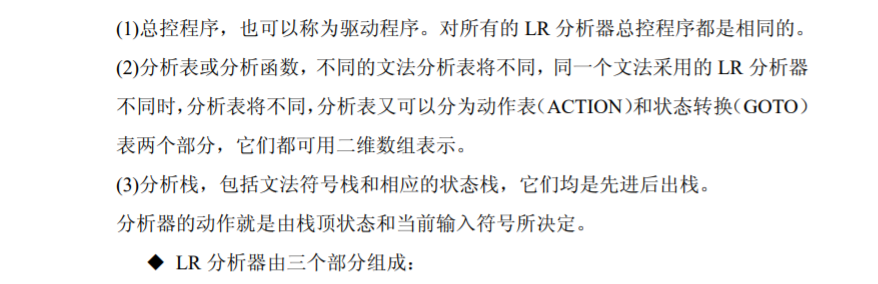
1. 验收时老师提醒我多按照书本上的思路来写，之前我也发现设计实验二只要按照老师上课时的ppt的思路来写，难度不是很大。

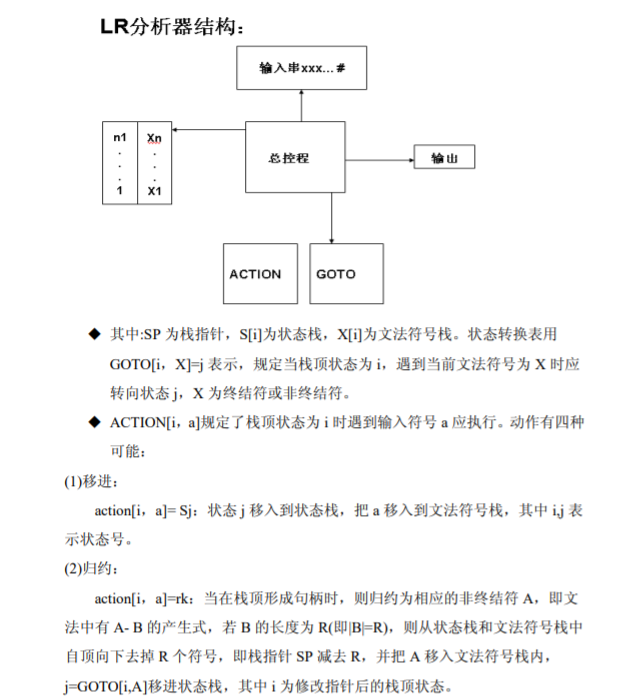
2. 关键是之前数据结构掌握的还不够好，导致在处理求“如果产生式右部第一个字符为非终结符，求该非终结符的First集”时，采用递归的思想的时候用了不久时，惭愧。

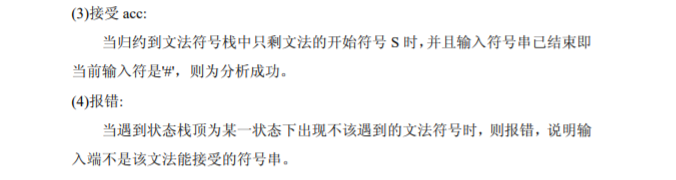
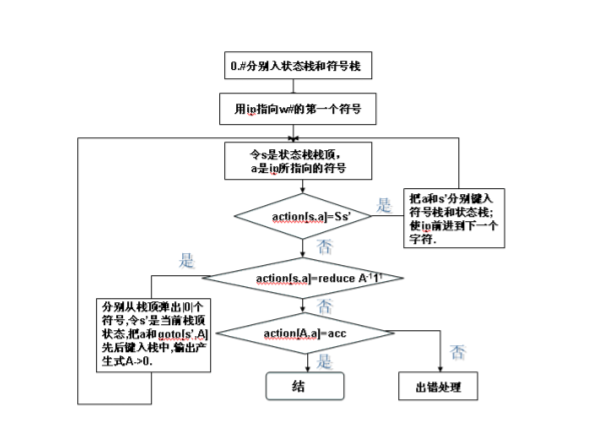
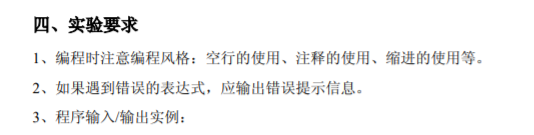
## 实验三 LR(1)分析法

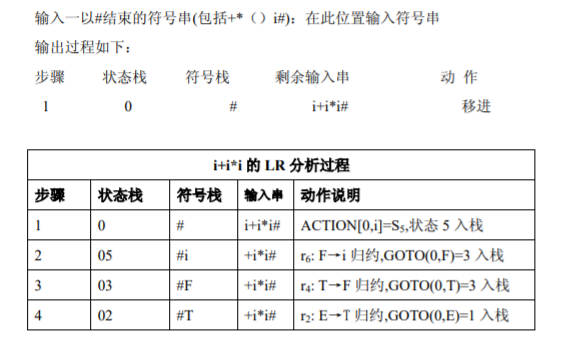
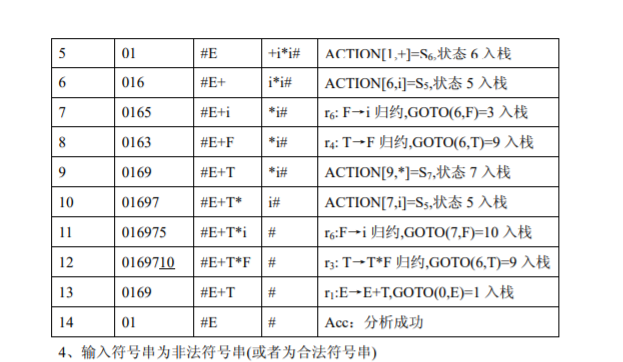
1. **实验目的**

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. **实验内容**
2. **LR（1）分析法实验设计思想及算法**

****

****





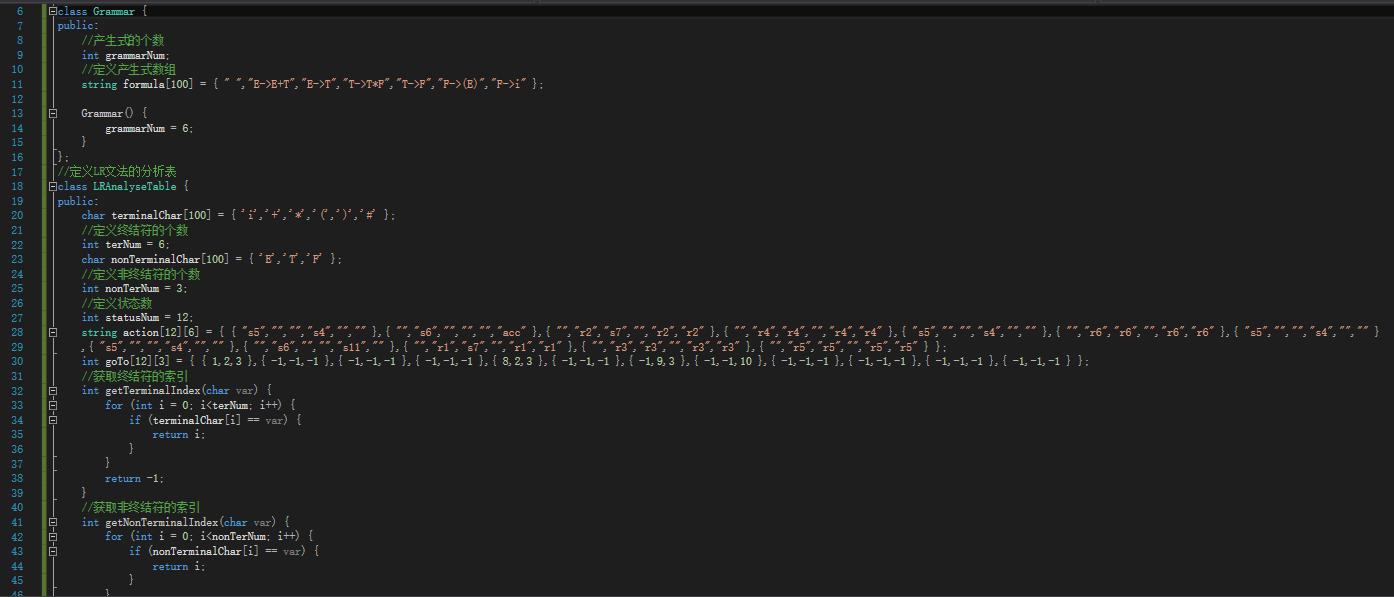
**五 、实验步骤**

**老师说我验收的代码很蠢（确实写的很垃圾…），痛定思痛，重新写了一个构造预测分析表的总控程序：（验收程序的原实验报告删除，下面是重新写的版本）**

采用直接构造了分析表，然后对栈中元素进行读取分析，根据状态栈和输入串的情况决定下一步做什么，分别进行规约和移进，具体以输入串为i+i\*i#为例子，进行分析。

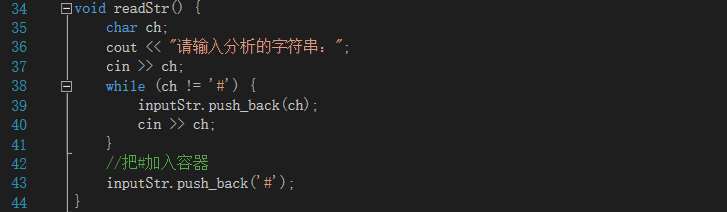
|  |  |  |  |
| --- | --- | --- | --- |
| 函数名称及调用格式返回类型 | 参数含义 | 返回值描述 | 函数功能 |
| int getTerminalIndex(char var) | 产生式字符 | 整型数 | 获取终结符的索引 |
| int getNonTerminalIndex(char var) | 产生式字符 | 整型数 | 获取非终结符的索引 |
| void readStr() { | 无 | 无 | 读取输入的字符串 |
| string vectTrancStr(int i) | 整型数 | 字符串 | 对栈容器进行输出,i=0,返回status中的字符串,i=1,返回sign中的字符串，i=2返回inputStr中的字符串 |
| void LRAnalyse() { | 无 | 无 | 总控，对输入的字符串进行分析 |

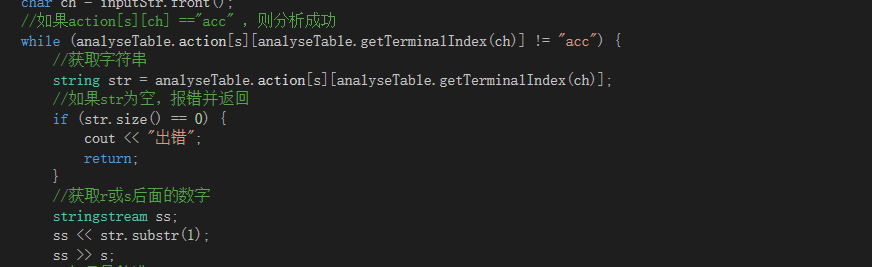
**详细的算法描述（程序执行流程图）**

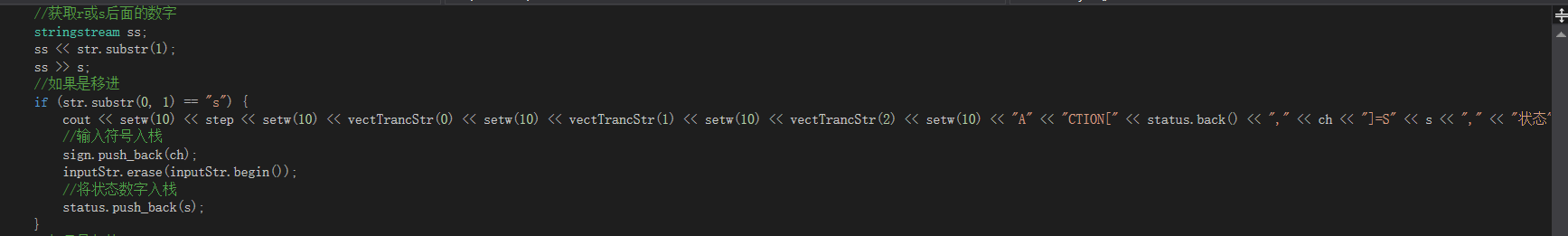
在头文件中定义了产生式数组，再在LR文法分析表的类里面定义了终结符的个数，非终结符的个数，状态数，获取终结符的索引和非终结符的索引的函数。

源文件中，直接在main()函数中调用了readStr()和LRAnalyse();

运行readStr(),实现把字符ch加入容器，其中#最后单独进入。

LRAnalyse中先对“步骤，状态栈，符号栈，输入串，动作说明”进行输出

然后对“分析成功”和“出错”的情况进行判定。

继续判断，如果是移进，进行对应的处理：输入符号入栈，状态数字入栈。

继续判断，如果是规约，进行相应的处理：先获取第S个产生式，再将产生式转化为字符数组，再获取产生式的首字符，符号栈的出栈次数，进行反向迭代后修改s，对符号栈和状态栈进行出栈，最后对产生式的开始符号入栈，并进栈新的状态。

最后对分析成功进行输出。

实验结果：

1. 限于编代码的水平，只实现了总控，但是对LR(1)分析法还是加深了理解。
2. 这种对应分析表进行输出的程序其实只要对何时移进，何时规约掌握了就不难
3. 看到其他同学都用java做出了漂亮的界面，我很羡慕，经过这次实验，我也感受到写C++程序的时候思路很容易乱，下定决心以后要用java编程了，毕竟java的语法结构相对C++来说比较简单，也去掉了C++中容易出错也比较难掌握的的指针和多继承，用接口代替了多继承的机制，减小了类之间关系的复杂度。

**4.老师在验收时提到的会导致分数不高的问题都解决了，希望老师能手下留情，谢谢老师！**