


# 语法分析器报告

## 摘要:

本次语法分析采用的是 LR(1) 分析方法，词法分析直接使用了上次的语法分析器，不过加了一层对输出的处理，使得输出出来是语法分析所需要的结构体。本程序所支持的语法在 grammar.txt 文档中，如下：



```
grammar.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
$S->$Program$LoopProgram
$LoopProgram->$Program$LoopProgram
$LoopProgram->@~
$Program->$Type@ID@($Parameter@)$StatementBlock
$Type->@int
$Type->@void
$Parameter->$OneParameter$LoopParameter
$OneParameter->@~
$OneParameter->@int@ID
$LoopParameter->@,$OneParameter$LoopParameter
$LoopParameter->@~
$StatementBlock->@{($InternalDeclaration$StatementString@)}
$StatementString->$StateMent$LoopStatement
$StatementString->@~
$LoopStatement->@~
$LoopStatement->$StateMent$LoopStatement
$StateMent->$IfStatement
$StateMent->$WhileStatement
$StateMent->$ForStatement
$StateMent->$ReturnStatement
$StateMent->$CoutStatement
$CoutStatement->@cout@<<$CoutStr$LoopCout@;
$CoutStr->$Expression
$CoutStr->@"@strid@"
$LoopCout->@<<$CoutStr$LoopCout
$LoopCout->@~
$StateMent->$Assignment
$ForStatement->@for@($ForOne$ForTwo@,$ForThree@)$StatementBlock
$ForOne->$IVD
$ForTwo->$Expression
$ForThree->@ID$ForT@=$Expression
$ForT->@+
$ForT->@-
$ForT->@~
$IfStatement->@if@($Expression@)$StatementBlock$ElseStatement
$Expression->$Addition$LoopAddition
$LoopAddition->@relon$Addition$LoopAddition
第 77 行, 第 17 列 100% Windows (CRLF) UTF-8
```

一共 77 行，\$开头的是非终结符，@开头的是终结符，@~当做空终结符。规定必须由\$S 在第一行开头当做整个语法的起始，除了要求外还拓展了一定的语法，下面有介绍。程序运行的逻辑大致为在初始化绘制程序界面的时候，读取 grammar.txt 中的语法，然后直接依据此文法建立一个项目集，然后再建立一个 ACTION\_GOTO 表，而后绘制完成界面，依据文本框输入的内容，首先进行词法分析，然后转化为需要的结构体数据，然后借助一开始建立的 ACTION\_GOTO 表进行语法分析，在分析的过程中，结合对空字符串的理解，在程序中规定，如果在这一个状态下依据指向的字符没有在 ACTION\_GOTO 表中找到对应的动作，那么就尝试将@~即空字符串带入分析，看是否能获得对应的动作。最后分析如果成功，返回相应的信息，并且跳出 success 的弹窗，如果错误，就进行相关信息的报错，同时将所有分析到的信息输出。

整体来看，程序具有一定的灵活性，直接修改 grammar.txt 中的语法，就可以进行添加，拓展程度高，对于一些较为常见的语法进行了分析，可视化程度较高，对于分析过程的信息都进行了详细的输出，对于错误信息也有一定的反馈能力。语法分析采用 class 的结构编写，具有较强的可读性。

# 一、运行和开发环境

运行环境: window 10

开发环境: QT5.14.2

## 二、能识别的单词、能分析的语法、扩充的功能

### 1.能识别的保留字单词和运算符如下:

```
//全局变量, 保留字表
static char reserveWord[WORDLEN][20] = {
    "include","using","namespace","std","define",
    "main","bool","auto", "break", "case", "char", "const", "continue",
    "cout","cin","_getch","default", "do", "double", "else", "enum", "extern", "while",
    "float", "for", "goto", "if", "int", "long","string",
    "register", "return", "short", "signed", "sizeof", "static",
    "struct", "switch", "typedef", "union", "unsigned", "void",
    "volatile","system"
};
//运算符表
static char oneOpera[ONEOPLEN] = {
    '+', '-', '*', '/', '<', '>', '=',
    ';', '(', ')', '^', '|', '\\', '\\', '#', '&',
    '|', '%', '~', '[', ']', '{', '\\',
    '}', '.', ':', ':', '\\?',
};
static char twoOpera[TWOOPLEN][3] = {
    "<=", ">=", "==",
    "!=", "&&", "||", "<<", ">>"
};
```

### 2. 能分析的语法

题目要求的全部:

**Program** ::= <类型> < ID>'( ' )<语句块>  
 <类型> ::= int | void  
 <ID> ::= 字母(字母|数字)\*  
 <语句块> ::= '{' <内部声明> <语句串>'}'  
 <内部声明> ::= 空 | <内部变量声明>; <内部变量声明>  
 <内部变量声明> ::= int <ID> (注: {}中的项表示可重复若干次)  
 <语句串> ::= <语句> { <语句> }  
 <语句> ::= <if语句> | <while语句> | <return语句> | <赋值语句>  
 <赋值语句> ::= <ID> = <表达式>;  
 <return语句> ::= return [ <表达式> ] (注: []中的项表示可选)  
 <while语句> ::= while '(' <表达式> ')' <语句块>  
  
 <if语句> ::= if '(' <表达式> ')' <语句块> [ else <语句块> ] (注: []中的项表示可选)  
 <表达式> ::= <加法表达式> { relop <加法表达式> } (注: relop -> <|<=>|>|=|!=>)  
 <加法表达式> ::= <项> {+ <项> | -<项>}  
 <项> ::= <因子> { \* <因子> | / <因子> }  
 <因子> ::= ID | num | '(' <表达式> ')'

额外扩充的语法:

1. Program 可以包含多个函数
2. 添加了 for 语句. eg:

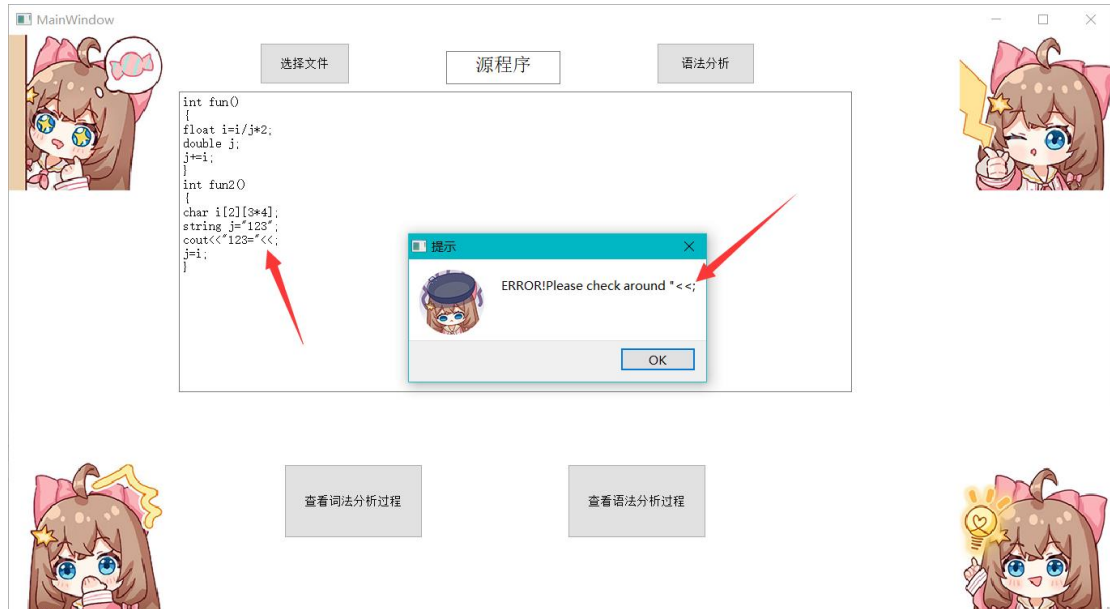
```

for(int i=0;i<8;i+=2)
{
    j=i+1;
    if(a>(b+c))
    {
        j=a+(b*c+1);
    }
    else
    {
        j=a;
    }
}
  
```

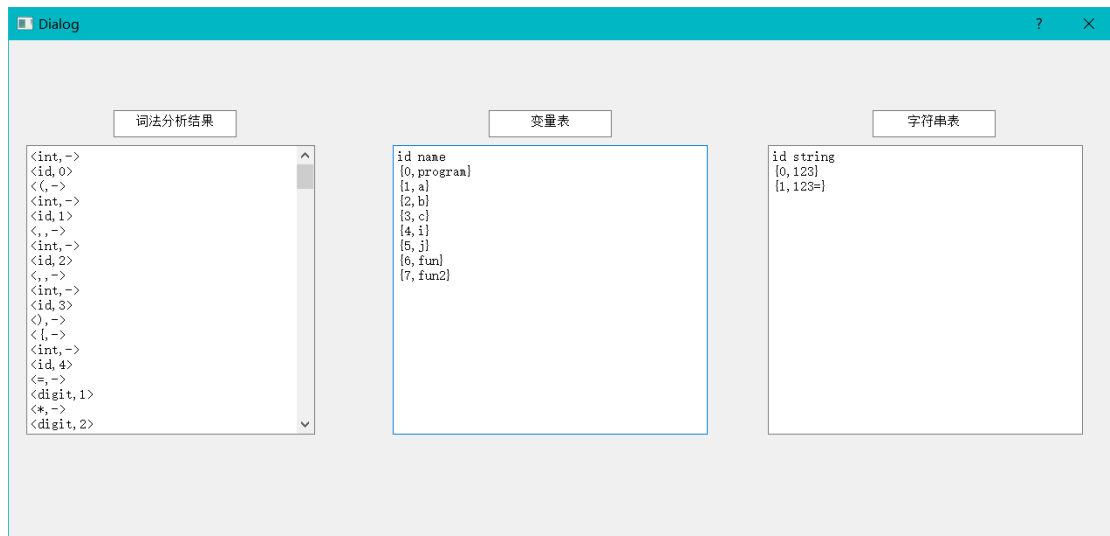
3. 定义语句可以直接赋值. eg: int a=0;
4. 添加了简便赋值语句. eg: a+=1;
5. 可以定义数组. eg: char i[2][3\*4];
6. 可以使用字符串赋值 string. eg: string s="12345 Hello World"
7. 支持分析 cout 语句. eg: cout<<"The result of i+j is"<<i+j;

### 3. 扩充的功能

1. 扩充了一部分如上说明的语法
2. 具有一定的报错功能，能够将错误附件的单词显示出来。



3. 能够查看语法分析过程中产生的语法分析结果、变量表和字符串表



4. 能够查看语法分析过程中产生的所有项目集，和 ACTION\_GOTO 表，以及整个规约过程。

Dialog

项目集

expression	index	forward_char
S->ProgramLoopProgram #		
Program->	TypeID(Parameter)	StatementBlock int void ~
Type->	int ID	
Type->	void ID	
-----1-----		
S->ProgramLoopProgram 1 #		
LoopProgram->	ProgramLoopProgram 0 #	
Program->	TypeID(Parameter)	StatementBlock 0 int void ~
Type->	int 0 ID	
Type->	void 0 ID	
LoopProgram->	0 #	
-----2-----		
Program->	TypeID(Parameter)	StatementBlock 1 int void ~
-----3-----		
Type->	int 1 ID	
-----4-----		
Type->	void 1 ID	

ACTION\_GOTO表

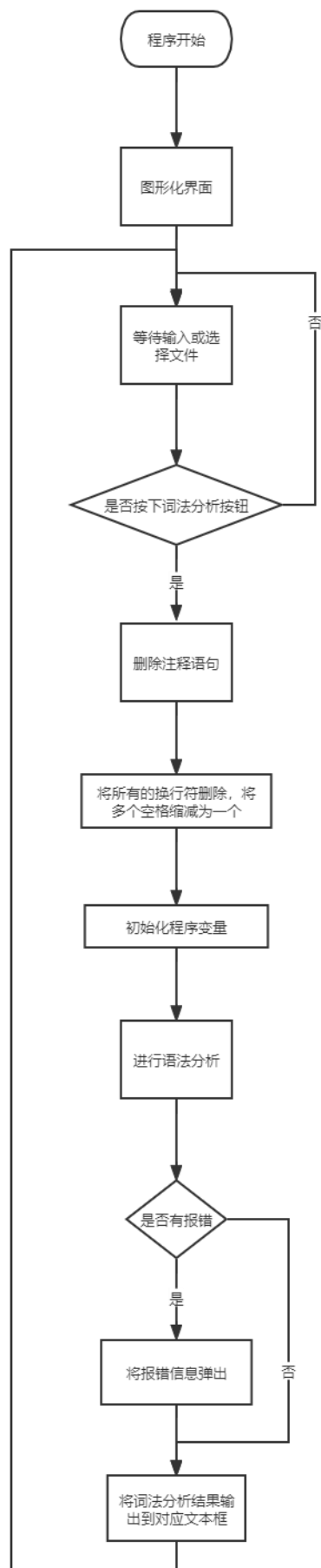
itemset_id	action
0	G-Program-1 G-Type-2 S-int-3 S-void-4
1	G-LoopProgram-5 G-Program-6 G-Type-2 S-int-3 S-void-4 S-~7
2	S-ID-8
3	R-ID-4
4	R-ID-5
5	R-#-0
6	G-LoopProgram-9 G-Program-6 G-Type-2 S-int-3 S-void-4 S-~7
7	R-#-2
8	S-(-10
9	R-#-1
10	G-Parameter-11 G-OneParameter-12 S-~-13 S-int-14
11	S-)-15
12	G-LoopParameter-16 S-,-17 S-~-18
13	R-,-7 R-~-7
14	S-ID-19

规约过程

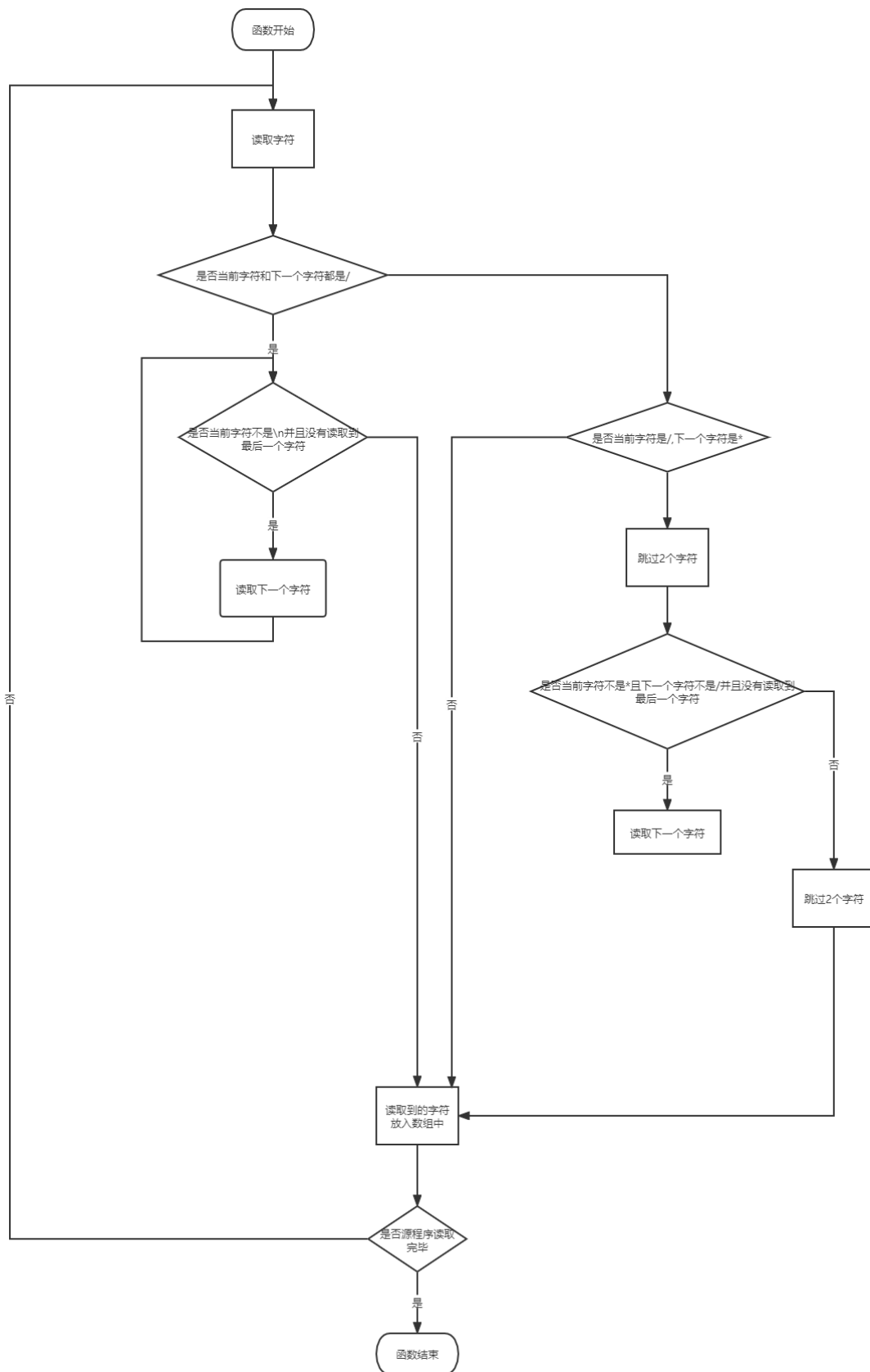
```
[, InternalDeclaration, StateMent, ID, AT, =, Addition, LoopAddition----, ], #
716----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 34, 46, 72, 110, 142----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StateMent, ID, AT, =, Expression----, }, #
717----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 34, 46, 72, 110, 142, 169----Program, Program, Type, ID, (, Parameter, ),
{, InternalDeclaration, StateMent, ID, AT, =, Expression, ;----}, #
718----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 34, 45----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StateMent, Assignment----}, #
719----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 34, 58----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StateMent, StateMent----}, #
720----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 34, 58, 57----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StateMent, StateMent, }----}, #
721----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 34, 58, 85----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StateMent, StateMent, LoopStatement----}, #
722----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 34, 56----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StateMent, LoopStatement----}, #
723----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 33----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StatementString----}, #
724----0, 1, 6, 2, 8, 10, 11, 15, 21, 23, 33, 55----Program, Program, Type, ID, (, Parameter, ), {, InternalDeclaration, StatementString, }----#
725----0, 1, 6, 2, 8, 10, 11, 15, 20----Program, Program, Type, ID, (, Parameter, ), StatementBlock----#
726----0, 1, 6, 6----Program, Program, Program----#
727----0, 1, 6, 6, 7----Program, Program, Program, #----#
728----0, 1, 6, 6, 9----Program, Program, Program, LoopProgram----#
729----0, 1, 6, 9----Program, Program, LoopProgram----#
730----0, 1, 5----Program, LoopProgram----#
Success!
```

# 分析算法的主程序框图

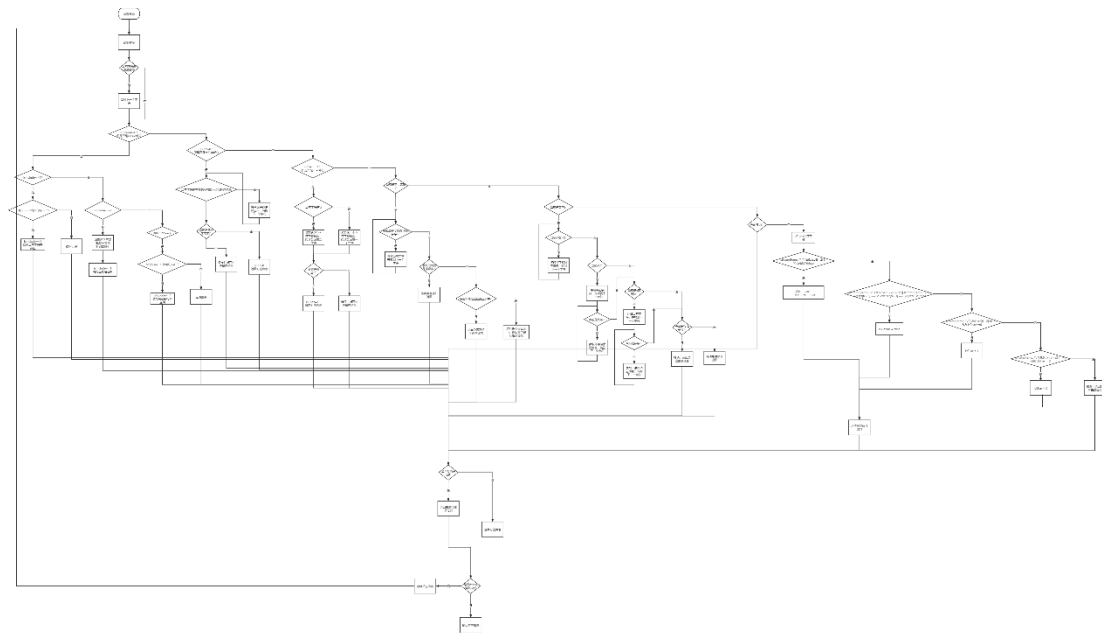
## 词法分析器部分



删除语句的函数流程图



## 语法分析的函数流程图



(附件有清晰图片)



# 语法分析器部分

本次语法分析器的构建采用 class 类进行构建，如下：

```
class syntactic_parser
{
private :
    string* txt;//语法规则对应的文本
    int txt_len;//语法规则对应的文本的长度
    STR* p;//所有的语法规则
    int p_len;//语法规则的长度
    V* Vt;//所有的终结符
    V* Vn;//所有的非终结符
    int Vt_len;//终结符的数量
    int Vn_len;//非终结符的数量
    FF* first;//所有非终结符对应的first集合
    int first_len;//first集合的长度
    Itemset* itemsets;//所有的项目集
    int itemset_len=0;//所有项目集的长度
    ALL_ACTION_GOTO* aag;//ACTION_GOTO表
    V* aTxt;//待分析的程序经过词法分析器分析后得到的结果

    bool is_In_Vt(V a);//是否是终结符
    bool is_In_Vn(V a);//是否是非终结符
    void Letter_First(V a);//计算a的first集合
    bool isInFirst(FF a, V b);//b是否在a的first集合中
    bool isInV(V* a, int a_len, V b);//b是否在集合a中
    int getVnIndex(V a);//a在非终结符表中的位置
    int getSTRIndex(STR a);//得到a在语法规则表中的位置
    void getOtherItem(Itemset& a, int a_i);//得到项目集中的其他项目
    void getOtherForward(Itemset& a, int a_i, int n_i);//得到其他的展望集合
    void changeOnce(int i);//第i个项目集拓展出其他的项目集
    int isInOtherItemset(Item a);//是否在其他已有的项目集中
    bool isItemSame(Item a, Item b);//2个项目是否相同
    void copyItem(Item origin, Item& target);//复制一份项目
    int getAction(int state,V a,int& action,int& target);//根据ACTION_GOTO表得到在当前状态，当前字符下应产生的动作
    string errorMsgChange(V a);//报错信息的v的转化

public:
    int aTxt_len;//词法分析器分析后得到的结果的长度
    int is_lex_error;//语法分析是否有错
    string lex_res, lex_variable, lex_string_table, lex_error_msg;//词法分析产生词法分析的结果、变量表、字符串表、词法分析错误信息
    string syn_itemset, syn_action_goto, syn_process;//语法分析的产生的项目集、ACTION_GOTO表、语法分析的过程
    syntactic_parser();//建立的初始化
    void read_file(string path);//读取文件
    void getaTxt(char* txt);//调用词法分析器从文本中得到要分析的程序
    void get_Vt_Vn();//得到终结符和非终结符
    void get_First();//得到First集合
    void get_Itemset();//得到项目集
    void getInitItemset();//得到初始项目集
    void changeTogetOtherItem();//得到其他的项目集
    void createACTION_GOTO();//创建ACTION_GOTO表
    void Analysis(int& synIsError,string& synErrorMsg);//语法分析
    void init();//初始化一遍关键变量
};
```

自定义了一些结构体：

```
struct V//终结符或非终结符
{
    string name;//符号名字
    bool type;//0代表非终结符，1 代表终结符
    string realName;//记录真实的变量名和字符串名
    bool operator==(const V b) const//重构==，判断2个V是否相等
    {
        return this->name == b.name && this->type == b.type;
    }
};
struct STR//推导式
{
    V left;//左边符号
    V right[MAX];//右边符号
    int right_len = 0;//右边符号的长度
};
struct FF//first集合
{
    V name;//对应的非终结符名字
    V first_list[MAX];//first列表
    int first_list_len = 0;//first列表的长度
};
struct Item//项目
{
    STR expression;//推导式
    V forward[LM];//展望字符
    int forward_len = 0;//展望字符长度
    int index = 0;//当前右部坐标
};

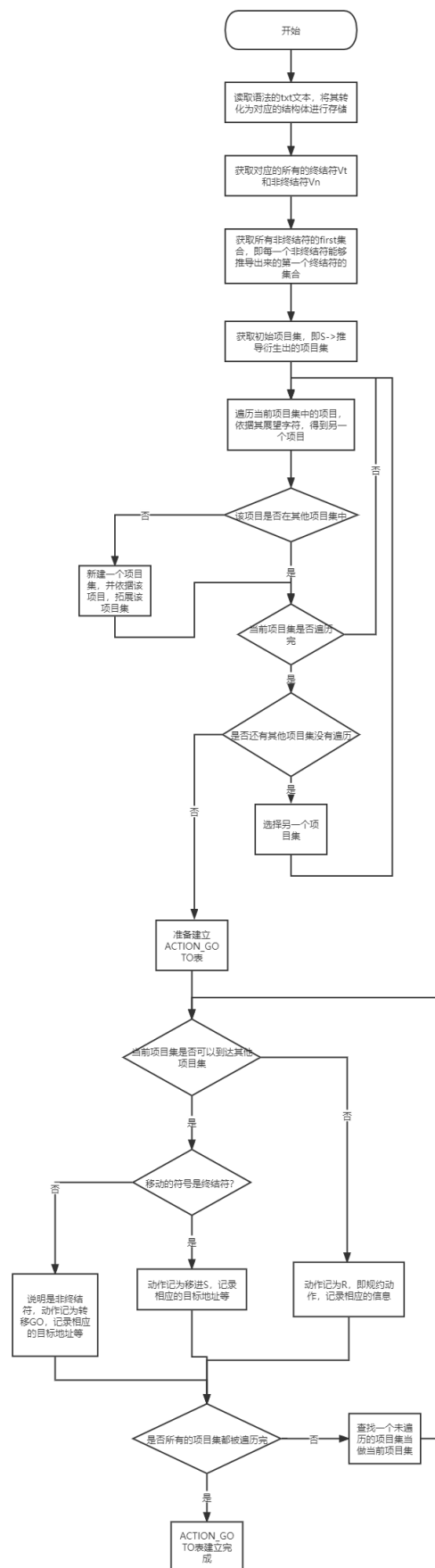
struct itemChange//项目的移动信息
{
    int target;//移动目标
    V method;//借助的符号
};

struct Itemset//项目集
{
    Item item[LM];//项目列表
    int item_len = 0;//项目列表长度
    itemChange change[LM];//该项目集转换到其他项目集的方式和目标
    int change_len = 0;//change数组的有效长度
};

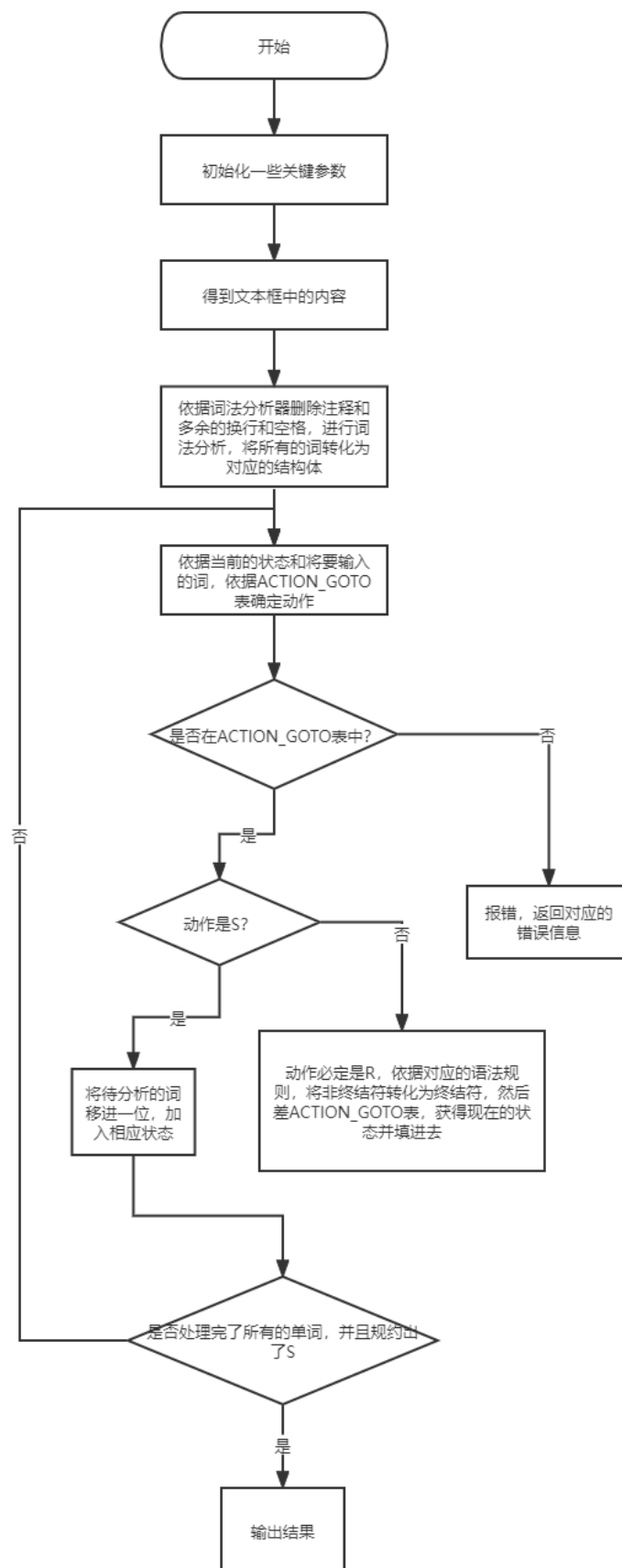
struct ACTION_GOTO//ACTION_GOTO表的一个单元
{
    V method;//借助的符号
    int action;//动作，go:0 s:1 r:2
    int target;//目标
};

struct ALL_ACTION_GOTO//ACTION_GOTO表的一行
{
    ACTION_GOTO* ag;//这一行中的所有有效单元
    int len = 0;//这一行中的所有有效单元的长度
};
```

语法分析前需要依据语法规则建立好 ACTION\_GOTO 表，流程如下：

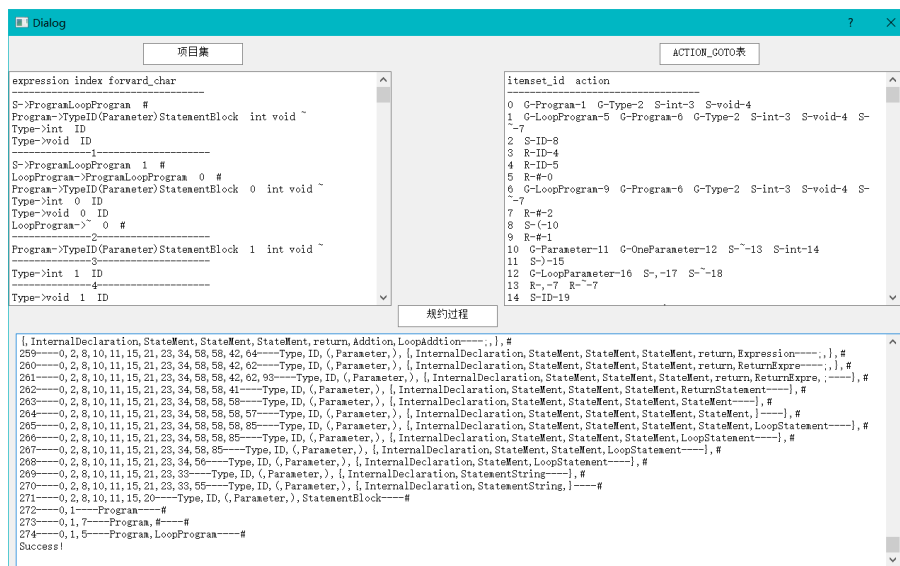
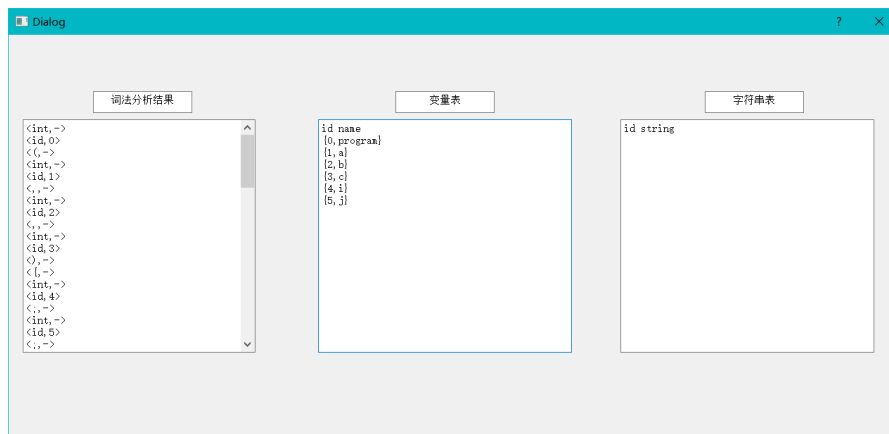


语法分析时的流程图如下：



# 运行结果截图

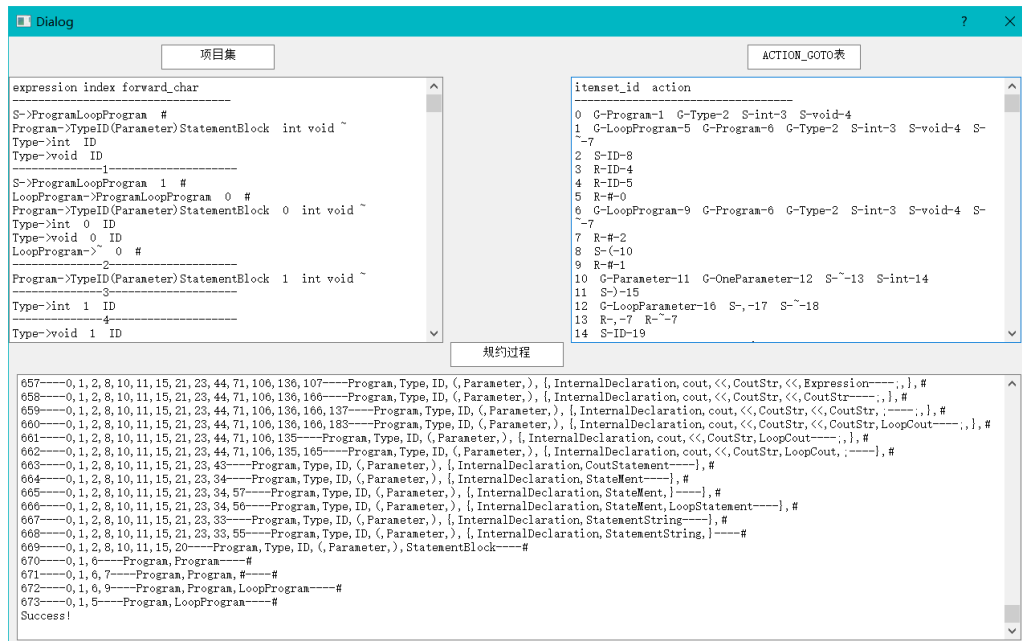
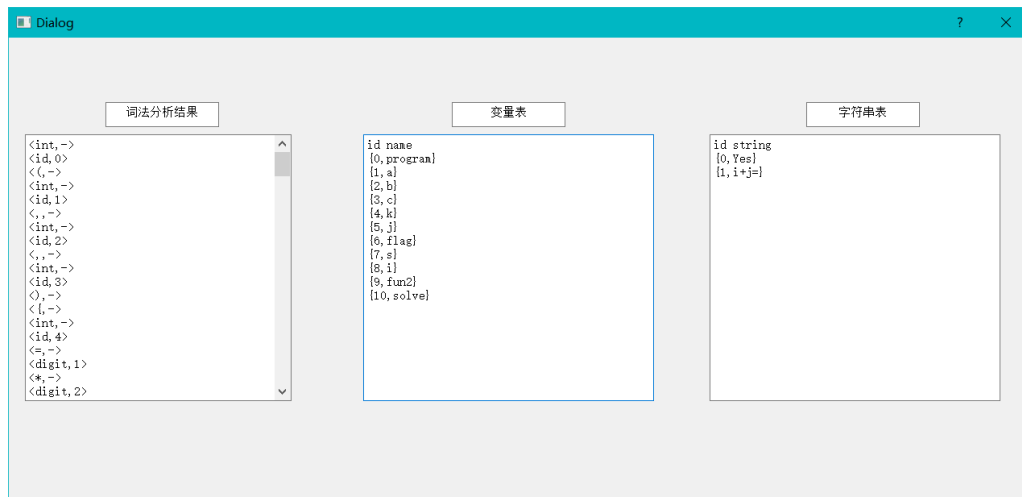
## PPT 示例程序：



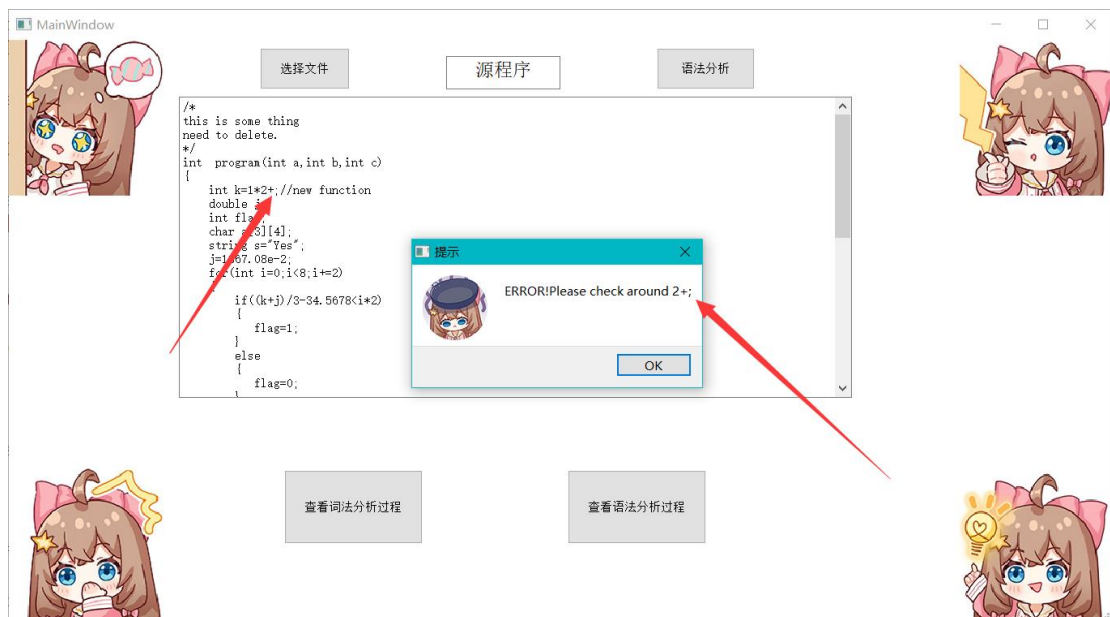
## 自己构建的较为复杂的程序分析结果：

程序代码：

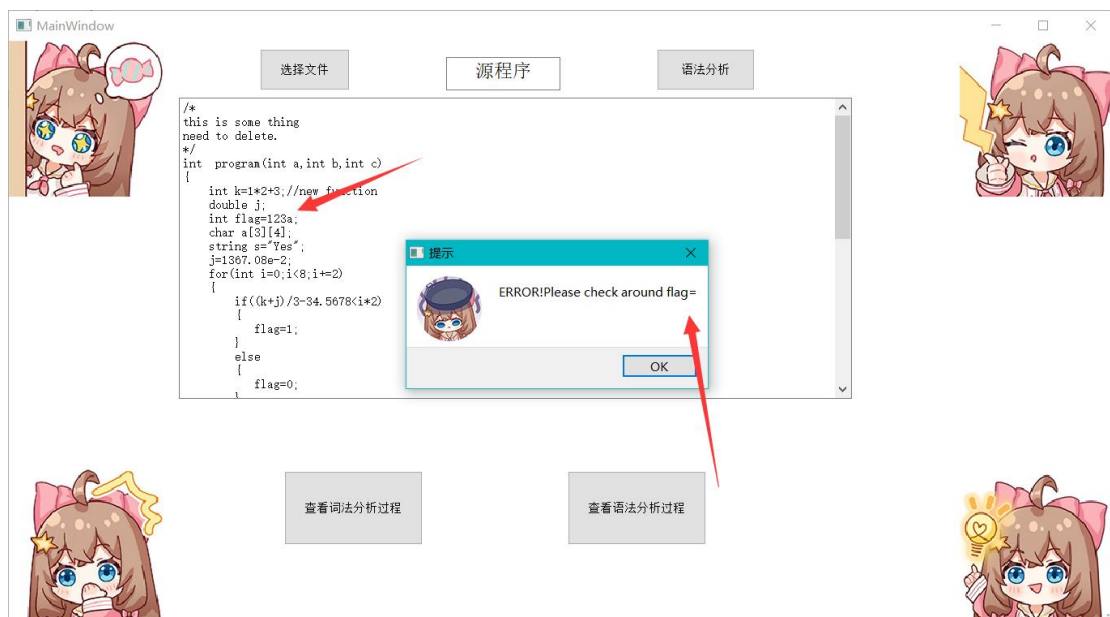
```
/*
this is some thing
need to delete.
*/
int program(int a,int b,int c)
{
    int k=1*2+3;//new function
    double j;
    int flag;
    char a[3][4];
    string s="Yes";
    j=1367.08e-2;
    for(int i=0;i<8;i+=2)
    {
        if((k+j)/3-34.5678<i*2)
        {
            flag=1;
        }
        else
        {
            flag=0;
        }
        i-=1;
    }
    if(a>(b+c))
    {
        j=a+(b*c+1);
    }
    else
    {
        j=a;
    }
    while(i<=100)
    {
        i=j*2;
    }
    return i;
}
//the next program
int fun2()
{
    float i=3.3;
    float j=4.4;
    string solve="i+j=";
    cout<<solve<<i+j;
}
```



## 报错测试 1:



## 报错测试 2:





报错测试 3:

