**课 程 设 计 报 告**

设计题目：一个简单文法的编译器前端的设计与实现

班 级： 计算机1507

组长学号： 20151817

组长姓名： 史英慧

指导教师： 王会珍

设计时间：2017年11月

设计分工

组长学号及姓名：20151817 史英慧

分工：设计文法和翻译文法 语法分析 语义分析

组员1学号及姓名：20154497 梁晨

分工：词法分析

组员2学号及姓名：20154378 张宇航

分工：符号表系统

组员3学号及姓名：20154468 武博

分工：目标代码生成

**摘 要**

编译系统是计算机三大系统软件之一，有着非常重要的地位。编译程序(compiler)是一种翻译程序，它特指把某种高级程序设计语言翻译成具体计算机上的低级程序设计语言。计算机专业相关的学生已经接触过并经常使用编译器，并且在编程开发的过程中依赖编译器的可靠性和正确性，但在未学习编译原理课程之前，对编译器的工作和实现大都了解甚少。通过理论学习和上机实践，我们较为全面地系统地掌握了相关知识。

本次课程设计中我们自主设计了一个简单的C语言文法，采用递归下降子程序的分析方法，运用类C语言实现了一个简单文法的编译器前端及后端的目标代码生成部分。前端包括词法分析器的设计与实现，语法分析器的设计与实现，中间代码的生成，符号表系统的设计与实现，本文对以上部分从功能，数据结构和算法三个方面进行了详尽阐述，并给出了整个编译器程序的主要函数的流程框图和实验结果。

我们的特色是基础文法规则定义较全，如变量可连续定义，可初始化，赋值语句可实现连续赋值，数组元素可参与逻辑和算数运算，并且逻辑和算数运算较为全面，如可实现自增、自减、取负等操作，循环语句中可有break和continue语句等，除此之外，我们增加了错误处理机制。

**关键词：**编译原理，递归下降子程序，前端，目标代码

**目 录**

**摘要**

1. 概述 ....................................................................................................................... 1

2. 课程设计任务及要求 ........................................................................................... 2

2.1 设计任务....................................................................................................... 2

2.2 设计要求....................................................................................................... 3

3. 算法及数据结构 ................................................................................................... 3

3.1算法的总体思想（流程）.......................................................................... 4

3.2 词法分析模块 ........................................................................................... 4

3.2.1 功能 .................................................................................................. 4

3.2.2 数据结构 .......................................................................................... 4

3.2.3 算法 .................................................................................................. 6

3.3 语法分析模块 ........................................................................................... 8

3.3.1功能 ................................................................................................... 8

3.3.2 数据结构 ........................................................................................ 10

3.3.3算法 ................................................................................................. 10

3.3 语义分析模块 ......................................................................................... 12

3.3.1功能 ................................................................................................. 12

3.3.2 数据结构 ........................................................................................ 14

3.3.3算法 ................................................................................................. 15

3.3 符号表模块 ............................................................................................. 16

3.3.1功能 ................................................................................................. 16

3.3.2 数据结构 ........................................................................................ 16

3.3.3算法 ................................................................................................. 18

3.3 目标代码生成模块 ................................................................................. 20

3.3.1功能 ................................................................................................. 20

3.3.2 数据结构 ........................................................................................ 20

3.3.3算法 ................................................................................................. 22

4. 程序设计与实现 ................................................................................................. 23

4.1 程序流程图 ............................................................................................... 23

4.2 程序说明 ................................................................................................... 40

4.3 实验结果 ................................................................................................... 40

5. 结论 ..................................................................................................................... 48

6. 参考文献 ............................................................................................................. 48

7. 收获、体会和建议 .............................................................................................. 49

**1. 概述**

本次课程设计的编译程序主要包括词法分析、语法分析、语义分析、符号表的建立与管理和目标代码生成五大部分。编译程序的输出包括词法分析阶段的关键字表、界符表、常数表和所有标识符，语法分析阶段的分析结果，中间代码生成阶段的四元式序列和符号表系统以及目标代码生成阶段的汇编程序。除了完成设计所要求的内容之外，我们增加了错误处理机制以报告出源程序中的错误。

我们首先设计了一个简单的C语言文法，包括变量的定义及初始化、主函数定义、赋值语句、表达式、if条件语句、while循环语句以及复杂数据结构如数组等的定义和引用等，规则较全。

词法分析是编译程序进行的第一步操作，也是整个编译程序的基础。对于输入的一段源程序，词法分析器从左至右扫描源程序中的字符串，通过自动机按照确定的词法规则识别出所有正确的单词，供语法分析使用。

语法分析是编译程序的核心，在确定的语法规则的基础上，通过语法分析来判断源程序中是否存在语法错误。

语义分析是分析源程序中各种语法成分的语义特征，在原文法的基础上构造翻译文法，语法分析中插入相应的语义动作，来生成相应的四元式。

目标代码生成是编译的最后一个阶段，结合四元式的目标代码生成算法，生成汇编语言程序。

**2. 课程设计任务及要求**

**2.1 设计任务**

**一个简单文法的编译器前端的设计与实现**

1) 定义一个简单程序设计语言文法（包括变量说明语句、算术运算表达式、赋值语句；扩展包括逻辑运算表达式、If语句、While语句等）；

2) 扫描器设计实现；

3) 语法分析器设计实现；

4) 中间代码设计；

5) 中间代码生成器设计实现。

**2.2 设计要求**

1)给出一个源程序文件，作为编译器前端的输入

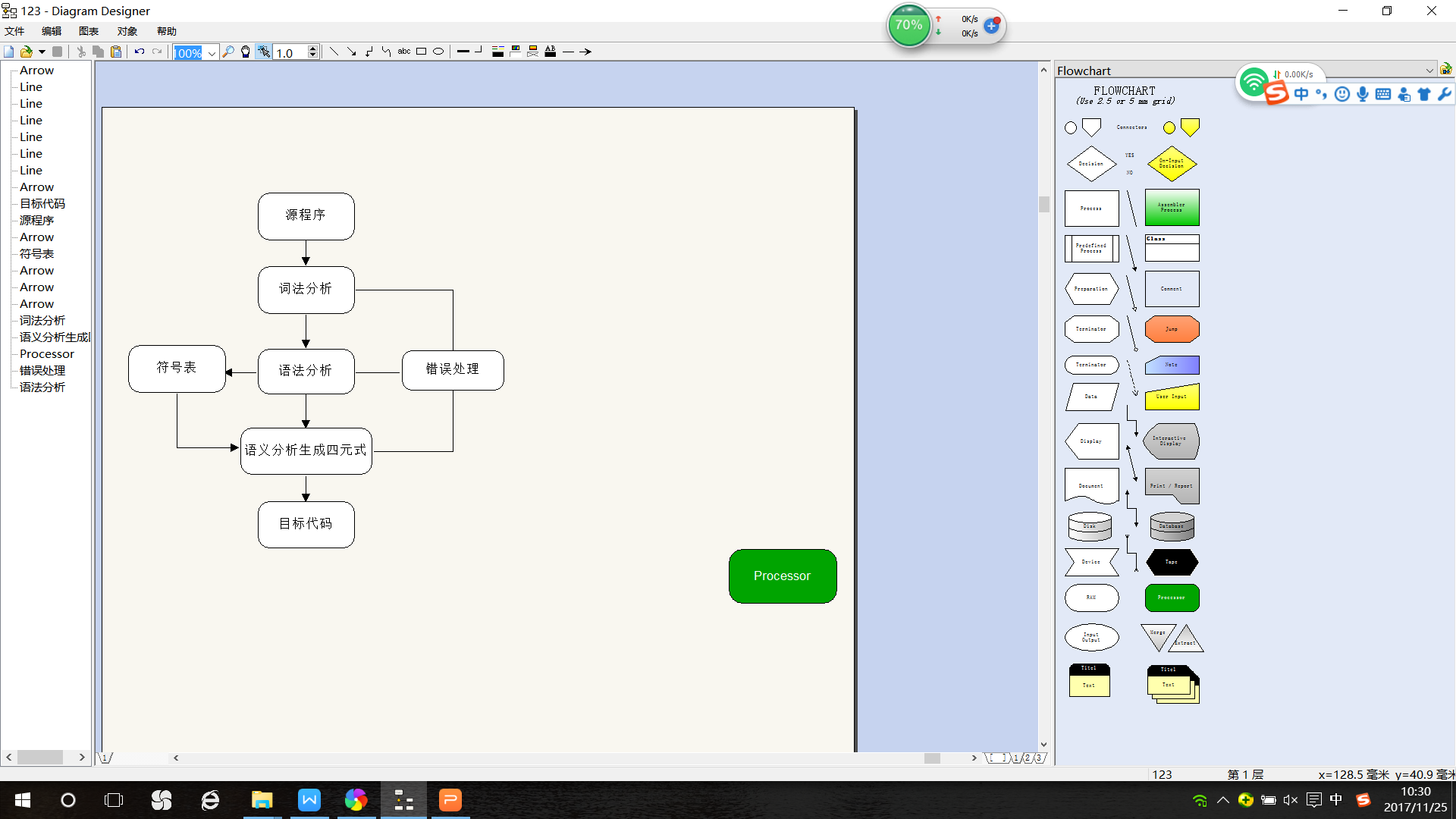
2)输出相关编译阶段的运行结果

3)词法分析阶段：Token序列；关键字表、界符表、常数表、符号表系统。

4)中间代码生成阶段：四元式序列；符号表系统。

1. **算法及数据结构**

**3.1算法的总体思想（流程）**



程序总体上遵循上图所示流程，首先词法分析器对源程序文件进行词法分析，得到token序列，遇到识别不出的字符串会报出相应的错误。然后语法分析从左至右扫描分析token序列，检查源程序中是否有语法错误，得出语法分析的结果。在语法分析的过程中，插入语义动作，建立符号表，遇到变量声明等语句，查填符号表，检查变量是否重复定义，遇到操作语句，查符号表检查变量是否定义，生成正确的四元式。最后在生成的四元式的基础上，结合目标代码生成算法，生成汇编语言程序。

**3.2 词法分析模块**

**3.2.1 功能**

自此次课程设计中，词法分析起一个基础性作用：识别程序中的单词并输出为token供后续程序使用，并进行一些简单的处理。

此次词法分析可以识别的类型如下所示：

#define IDENT\_TYPE 1 //标识符

#define INTEGER\_TYPE 2 //整形数

#define FLOAT\_TYPE 3 //浮点数

#define CHAR\_TYPE 4 //字符

#define STRING\_TYPE 5 //字符串

#define KEYWD\_TYPE 6 //关键字

#define DELIMIT\_TYPE 7 //界符

可实现一些简单的处理：

1. 常数处理机，包括整数、小数以及科学计数法的处理
2. 行注释以及段注释的处理
3. 转义字符的处理
4. 两种模式token的输出，分步输出和一次全部输出
5. 输出界符表、关键字表、常数表和标识符表
6. 错误输出

**3.2.2 数据结构**

//type of token

typedef struct

{

string tokenWord;

SType tokenType;

int tokenValue;

int tokenIndex;

}token;

//tokenizer class

class Tokenizer

{

public:

Tokenizer(const char\* srcFile);

~Tokenizer();

//judge the type of a character

CType judgeCType(const char tmpChar);

//find next token

token findNextToken(void);

//output keyword list, delimiter list, identifier list or const list

vector<string> keywdSet;

vector<string> delimitSet;

vector<string> identifierSet;

vector<string> constSet;

void outputList(vector<string> list);

//find all token and put them to a vector

vector<token> tokenSet;

void findAllToken(void);

private:

ifstream srcCode;

//current character

char crtChar;

string crtString;

token crtToken;

void pushBackNoRepeat(vector<string> &list, string str);

};

**3.2.3 算法**

Tokenizer(const char\* srcFile);

~Tokenizer();

构造与析构，通过将文件名作为参数传入构造函数实现文件流的传入，方便后续的词法分析操作。

//judge the type of a character

CType judgeCType(const char tmpChar);

用于判断字符的类型，基于模块化思想设计，具体实现如下:

//judge the type of a character

CType Tokenizer::judgeCType(const char tmpChar)

{

//is letter?

if ((tmpChar >= 'a' && tmpChar <= 'z') ||

(tmpChar >= 'A' && tmpChar <= 'Z'))

return LETTER\_TYPE;

//is number?

else if (tmpChar >= '0' && tmpChar <= '9')

return NUMBER\_TYPE;

//is space or enter?

else if (tmpChar == ' ' || tmpChar == '\f' || tmpChar == '\n' ||

tmpChar == '\r' || tmpChar == '\t' || tmpChar == '\v')

return BLANK\_TYPE;

//is forward quote?

else if (tmpChar == '\'')

return S\_QUOTE\_TYPE;

//is double quote?

else if (tmpChar == '\"')

return D\_QUOTE\_TYPE;

//is underline?

else if (tmpChar == '\_')

return UDLINE\_TYPE;

//else type

else

return OTHER\_TYPE;

}

//find next token

token findNextToken(void);

词法分析主要功能的实现函数，功能为找到下一个token，用自动机实现。先判断第一个字符的类型，然后进入相应的case进行后续处理。同时处理注释、常数处理机等操作。

#define \_err\_report do \

{ \

crtToken.tokenType = ERROR\_TYPE; \

cout << setiosflags(ios::left) << setw(7) << crtString << \

"Error token type." << endl; \

return crtToken; \

} while (0) \

宏函数，用以进行错误报告。

//output keyword list, delimiter list, identifier list or const list

void outputList(vector<string> list);

输出各表，将表的类型作为参数输入，然后将其内容以字符串形式输出。

//find all token and put them to a vector

void findAllToken(void);

获取所有token，并将其放入一个token类型的向量传给下一模块。

**3.3 语法分析模块**

**3.3.1 功能**

语法分析的功能是识别和处理比单词更大的语法单位，如：程序设计语言中的表达式、各种说明和语句乃至全部源程序，指出其中的语法错误。形式上说，语法分析是指对给定的符号串( α )，判定其是否是某文法G(Z)的句子。

本次课程设计定义的文法如下：

Program -> ExtDefList

ExtDefList -> DefList FunDef CompSt

FunDef -> int main ()

CompSt -> { DefList Statement }

Statement -> StmtList Statement | DefList Statement | ε

DefList -> Def DefList | ε

Def -> specifier DecList ; // specifier: int float char double short long signed unsigned

DecList -> Dec Dec1

Dec1 -> , Dec Dec1| ε // , 连续定义多个变量

Dec -> Va Init // Init 进行初始化

Init -> = Init1 | ε

Init1 -> Exp | { ElemList }

ElemList -> elem Elem // elem: const 常量，character 字符

Elem -> , elem Elem | ε // 连续初始化

Va -> identifier Va1 // identifier 标识符

Va1 -> [ 整型常量 ] | ε // 数组

StmtList -> Stmt StmtList | ε

Stmt -> AssignList | if (Exp) Stmt Other | CompSt | return Exp(or return S); | while (Exp) Stmt | break ; | continue ;

Other -> else Stmt | ε

AssignList -> Assign AssignList | ε

Assign -> Ass Assign1

Assign1 -> , Ass Assign1 | ; // , 允许一条赋值语句里出现多个赋值表达式

Ass -> Va sign1 Ass | Exp // sign1 : =, +=, -=, \*=, / =, %=

Exp -> C; // 表达式

C -> D C1

C1 -> || D C1 | ε

D -> E D1

D1 -> && E D1 |ε

E -> ! E | - E | F

F -> G F1

F1 -> sign2 G F1 | ε // sign2 : >, <, >=, <=, ==, !=

G -> H G1

G1 -> sign3 H G1 | ε // sign3: + , -

H -> I H1

H1 -> sign4 I H1 | ε // sign4: \*, /

I -> (Exp) | Va J | ++ Va | -- Va | const | char // const: 实型或整型常量

J -> ++ | -- | ε

可实现：

1. (外部)变量声明、定义及初始化

其中可定义的变量类型：  
基本类型: int float char double short long signed unsigned

复杂类型: 数组

1. 主函数的定义和使用
2. 赋值语句，可实现变量连续赋值，赋值符号包括：=, +=, -=, \*=, / =, %=
3. if...else 条件语句，条件语句可嵌套
4. while 循环语句，有break，continue 语句，循环可嵌套
5. 表达式，包括算数表达式和逻辑表达式。  
    其中算数表达式可实现的功能有：+, -, \*, /, ++, --, -(取负)，

逻辑表达式可实现的功能有：&&, || , !, >, <, >=, <=, ==, !=

**3.3.2 数据结构**

extern int Flag; // 主函数中定义的全局变量，初始化为1，语法分析时有语法错误置0；

extern vector<token>::iterator word; // 词法分析后得到的token向量，语法分析中分析的每一个单词

int ifLoop = 0; // 语法分析中用来判断源程序中是否有循环

int tip = 0; //++,-- 作为后缀生成四元式时的辅助判断

**3.3.3 算法**

⑴ 扩展文法：

增设一个产生式，作为主程序：Z`->Z ，

⑵ 入出口约定：

子程序入口时，其首符号已经读来！

子程序出口时，其后继符应该读来！

⑶ 子程序内容设计：

遇终结符，判定之 ，确认后读下一单词；

遇非终结符,调用之,返回后不读下一单词；

遇空串( α ) ，直接出口；

以下列出所有语法分析中用到的递归子程序：

void Program();

void ExtDefList();

void FunDef();

void Statement();

void CompSt();

void StmtList();

void DefList();

void Stmt();

void S();

void Other();

void Def();

void DecList(); // 定义的变量列表

void Dec(); //定义的变量

void Dec1();

void Init(); //变量定义时初始化 Init -> = Init1 | ε

void Init1(); // 文法变换中间符号 Init1 -> Exp | { ElemList }

void Va(); // Va -> identifier { PUSH(identifier) } identifier 标识符

void Va1();

void ElemList();

void Elem();

void AssignList(); // 赋值语句列表 AssignList -> Assign AssignList | ε

void Assign(); //赋值语句 Assign -> Ass Assign1

void Assign1(); // Assign1 -> , Ass Assign1 | ; // , 允许一条赋值语句里出现多个赋值表达式

void Ass(); // 文法变换中间符号 Ass -> Va sign1 Ass { ASSI() }| Exp // sign1 : =, +=, -=, \*=, / =, %=

// 表达式文法相关产生式

void Exp(); // Exp -> C; // 表达式

void C(); // C -> D C1

void C1(); // C1 -> || D { OR() } C1 | ε

void D(); // D -> E D1

void D1(); // D1 -> && { AND () } E D1 |ε

void E(); // E -> ! E { NOT () } | - E { MINUS () }| F

void F(); // F -> G F1

void F1(); // F1 -> sign2 G { sign2()} F1 | ε

void G(); // G -> H G1

void G1(); // G1 -> sign3 H { GEQ(sign3) } G1 | ε

void H(); // H->I H1

void H1(); // H1->sign4 I{ GEQ(sign4) } H1 | ε // sign4: \*, /

void I(); // I -> (Exp) | Va J | ++ Va { GEQ(++) } | -- Va { GEQ(--) } | const { PUSH(const) } | char // const: 实型或整型常量

void J(); // J -> ++ | -- | ε

**注：具体的流程框图见第四部分程序设计与实现**

**3.4 语义分析模块**

**3.4.1 功能**

语义分析的功能和任务是按照语法分析器识别的语法范畴进行语义检查和处理，产生相应的中间代码。比如语义分析的一个工作是进行类型审查，审查每个算符是否具有语言规范允许的运算对象，当不符合语言规范时，编译程序应报告错误。

中间代码是高级程序语言中，各种语法成分的语义结构表示；它介于源语言和目标语言之间。

在原文法的基础上构造的翻译文法如下：

Program -> ExtDefList

ExtDefList -> DefList FunDef CompSt

FunDef -> int main ()

CompSt -> { DefList Statement }

Statement -> StmtList Statement | DefList Statement | ε

DefList -> Def DefList | ε

Def ->specifier DecList; // specifier: int float char double short long signed unsigned

DecList -> Dec Dec1

Dec1 -> , Dec Dec1| ε // , 连续定义多个变量

Dec -> Va Init // Init 进行初始化

Init -> = Init1 | ε

Init1 -> Exp | { ElemList }

Va -> identifier { PUSH(identifier) } Va1 // identifier 标识符

Va1 -> [ {PUSH( [ ) } 整型常量 { PUSH(整型常量) } ] { PUSH( ] ) }

StmtList -> Stmt StmtList | ε

Stmt -> AssignList | if (Exp) { IF () } Stmt Other { IE () } | CompSt | return Exp(or return S) ; { RETURN() }

| while { WH() } (Exp) { DO() } Stmt | break ; { BREAK() } | continue ; { CONTINUE() }

Other -> else { EL() } Stmt | ε

AssignList -> Assign AssignList | ε

Assign -> Ass Assign1

Assign1 -> , Ass Assign1 | ; // , 允许一条赋值语句里出现多个赋值表达式

Ass -> Va sign1 Ass { ASSI() }| Exp // sign1 : =, +=, -=, \*=, / =, %=

Exp -> C; // 表达式

C -> D C1

C1 -> || D { OR() } C1 | ε

D -> E D1

D1 -> && { AND () } E D1 |ε

E -> ! E { NOT () } | - E { MINUS () }| F

F -> G F1

F1 -> sign2 G { sign2()} F1 | ε // sign2 : >, <, >=, <=, ==, !=

G -> H G1

G1 -> sign3 H { GEQ(sign3) } G1 | ε // sign3: + , -

H -> I H1

H1 -> sign4 I { GEQ(sign4) } H1 | ε // sign4: \*, /

I -> (Exp) | Va J | ++ Va { GEQ(++) } | -- Va { GEQ(--) } | const { PUSH(const) } | char // const: 实型或整型常量

J -> ++ | -- | ε

**3.4.2 数据结构**

extern stack <token> SEM; // C++ STL中的栈，语义栈

extern string QUAT[4]; // 四元式区，存放四元式中的每一元，四元式中的元素是每一个token结构体中的单词的字符串

/\* 进行语义分析时一些辅助变量 \*/

int countVa = 0; // 对赋值语句进行语义分析时，若出现连续赋值，记录标识符个数

extern token curToken; // 保存从语义栈中弹出的token

extern int num; // 四元式临时变量 t 的下标

extern int tip; // 自增、自减 后缀时的判断

extern int countVa; // 连续赋值操作时记录被赋值变量的个数

extern SymbolTable Stb; // 符号表，主要检查变量是否未定义就使用

string tempVa; // 临时变量，保存四元式每一元

token temp\_token; // 临时变量

**3.4.3 算法**

翻译文法的构造见语义分析模块中的功能模块，下面列出所有语义动作的声明：

// 赋值语句四元式函数

void ASSI(string sign1);

// if 语句四元式相关函数

void IF();

void EL();

void IE();

// while 语句四元式相关函数

void WH();

void DO();

void WE();

void BREAK();

void CONTINUE();

void RETURN(); // return 四元式函数

// 表达式四元式相关函数

void GEQ(string Operator); // +, -, \*, /, ++, --

void OR();

void AND();

void NOT();

void MINUS();

void GT();

void LT();

void GE();

void LE();

void EQ();

void NE();

**注：具体的流程框图见第四部分程序设计与实现**

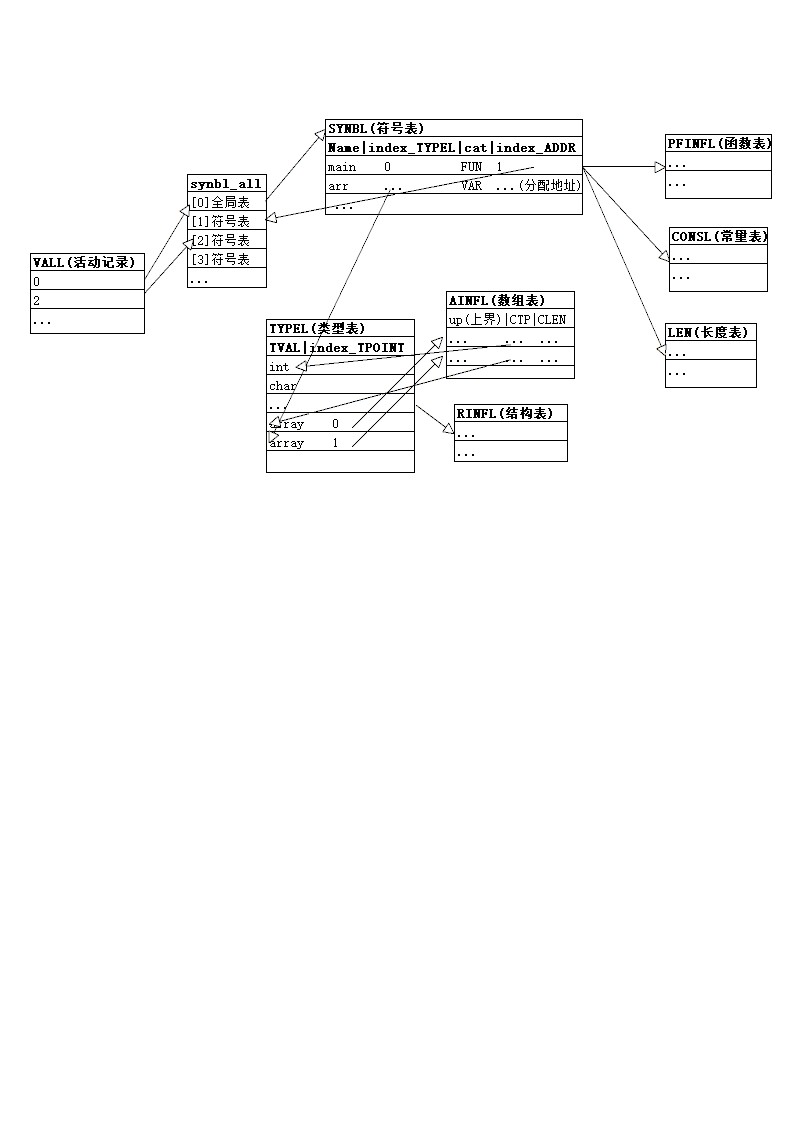
**3.5符号表模块**

**3.5.1功能：**

符号表是标识符的动态语义词典，属于编译中语义分析的知识库。主要完成定义和重定义检查、类型匹配校验、数据的越界和溢出检查、值单元存储分配信息等等。

具体实现了简单数据类型，函数，多维数组的定义与重定义检查（区分不同作用域）；返回标识符类型以供类型匹配校验；返回数据类型的大小以供数据的越界和溢出检查；根据数据类型大小分配地址空间等等。

**3.5.2数据结构：**



自顶向下：

**class SymbolTable**

//符号表主体，包含符号表所有的对外接口，符号表内各数据单位以私有成员变量身份存入其中。

**数据：**

vector<map <string, SYNBL>> synbl\_all; //以不同程序块区分的所有符号表所S

vector<TYPEL> typel; //类型表

vector<AINFL> ainfl; //数组表

vector<RINFL> rinfl; //结构表

vector<CONSL> consl; //常量表

vector<LENL> lenl; //长度表

vector <int> vall; //活动记录,存储 synbl\_all 的index

int var\_addr = 0; //地址分配记录

**接口**：

bool beginDefFunction(token& tk); //开始定义一个名为tk的函数

bool endDefFunction(token& tk); //结束定义函数tk

bool beginNewProcess(); //开始一个新的{过程}

bool endLastProcess(); //结束上一个{过程}

bool AddToken(token& tk); //新增标识符tk

bool locateToken(token& tk); //查找tk是否曾定义

bool setCAT(token& tk, emCAT cat); //设置tk的cat值（FUN\_, VAR\_, CONST\_）

bool setTYPEL(token& tk, emTVAL tval); //设置tk的tval值（int，char…）

int getTYPEL(token& tk); //获得tk的tval值

int getTvalSize(emTVAL tval); //获得tval的大小

int getTvalSize(int index\_TYPEL); //获得类型typel[index\_TYPEL]的大小

bool addAINFL\_d(…); //设置 tk为多维数组

void show(); //展示符号表

其他：

enum emTVAL//TVAL常量定义（int\_, char\_,…）

enumemCAT //CAT常量定义(FUN\_, VAR\_, CONST\_,…)

structCONSL//常数表

struct LENL//长度表

structTYPEL//类型表表项

struct AINFL//数组表表项

structRINFL//结构表表项

structSYNBL//符号表表项

**3.5.3算法**

符号表作为标识符的动态语义词典，语义分析的知识库，不仅需要完成各种标识符的分类组织管理，而且应当尽可能的方便其他模块对符号表的读写以及查找操作

**总体思路：**

****

**标识符的重定义与作用域：**

本设计采用一个代码块（“{}”）一张符号表的原则，在同一层代码块中所有标识符处于同一符号表内，不允许重名。

在不同层次中处于不同符号表内，允许重名。

查找标识符时，依次从本层向外层查找，找到立即返回查找结果。

活动记录VALL承担主要层次控制作用。

此处的VALL不同于课件上的设计，对VALL进行了改造，以类似栈的形式，只存储符号表在synbl\_all中的标号。

VALL栈顶元素指出的符号表为当前活跃的符号表，任何对符号表的操作都指向活跃符号表。

当出现新的代码块{}时，意味着需要新建一个符号表以容纳新代码块中的标识符，将此新符号表的标号压入VALL栈中，实现活跃符号表的更新。

当代码块结束时（遇到“}”）,将VALL栈顶弹出，恢复之前的活跃符号表。

新增标识符时，只要当前活跃符号表中无同名标识符，即可插入当前活跃符号表。

查找标识符时，从VALL栈顶符号表依次往下层符号表查询，找到即停，返回查找结果。以此实现标识符作用域内的识别。

**标识符的定义：**

****

**标识符的查找：**

****

**3.5 目标代码生成器模块**

**3.5.1 功能**

生成的目标代码选择上，我选择了课堂上讲过的虚拟机及其指令系统；

代码生成器的输入：中间代码四元式的数组，中间代码中值的名字可以提供目标代码中直接操作的量。

**3.5.2 数据结构**

typedef struct

{

int flag=1; //变量的活跃信息

string word; //变量的名字

}operatorN; //通过结构体将变量的两种特征结合起来

目标代码形式：J1: LR A,B

typedef struct

{

string operator0; //目标代码操作指令（对应例子中的LR）

string operator1; //目标代码操作数1（对应例子中的A）

string operator2; //目标代码操作数2（对应例子中的B）

string operatorex; //目标代码跳转前缀（对应例子中的J1)

void showASS() {

cout << setw(8) <<operatorex<<setw(4)<<operator0 << setw(4) << operator1 << "," << operator2 << endl; //目标代码输出函数

}

}assembly; //目标代码结构体

typedef operatorN q\_operator[4]; //中间代码变量的四元数组

class FinalCode //定义目标代码生成器的类

{

public:

FinalCode();

~FinalCode();

int i = 0;//目标代码数组索引

int j = 0;//数据段变量数组索引

void Code(q\_operator q);//汇编代码生成

void nToq(string\* n);//四元式数组向含活跃状态的数组转换

string expToass(string s);//运算符号与汇编指令的对应

csdata csdatas[100];//数据段变量

assembly assum[100];//目标代码数组

private:

int Jt = 1;//跳转指令的编号

stringstream ss;//跳转编号的前缀

stack<int> sw;//有关跳转的栈

operatorN RDL = { 0,"0" };//寄存器状态描述

};

**3.5.3 算法**

FinalCode()是构造函数，里面为空，采用默认构造方式；

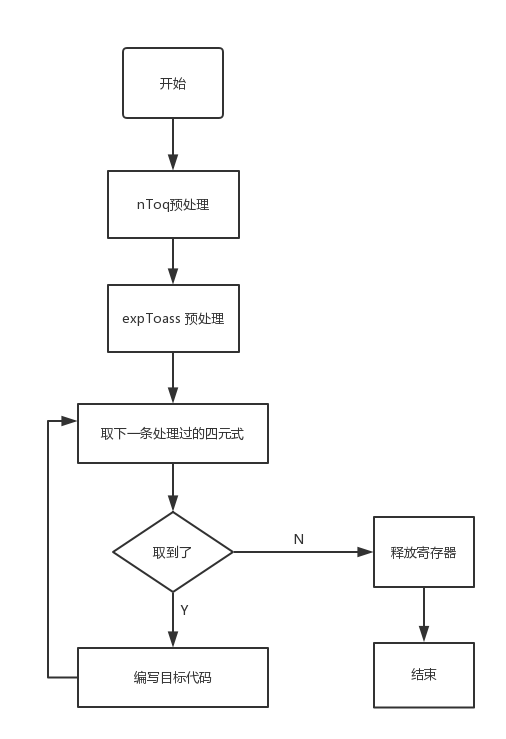
~FinalCode()是析构函数，里面为空，采用默认析构模式；

void FinalCode::nToq(string\* n)负责将中间代码那种只有变量名字的四元式转换成生成目标代码所需要的operatorN四元式，即变量中包含活跃信息，函数参数为中间代码四元式数组；

string FinalCode::expToass (string s) 负责将扫描中间代码时遇到的操作符（“+”、“-”、“\*”等）转变成目标代码操作数（ADD、SUB、MUL等），函数参数为中间代码字符串；

void FinalCode::Code(q\_operator q) 是目标代码生成的主要函数，负责中间代码向目标代码的转换，函数参数为将经过nToq（）函数转换后的中间代码四元式数组，函数的主要实现方式是通过判断语句，逐个将中间代码四元式转化成目标指令代码并储存到一个新的数组中。

目标代码生成器算法流程图：

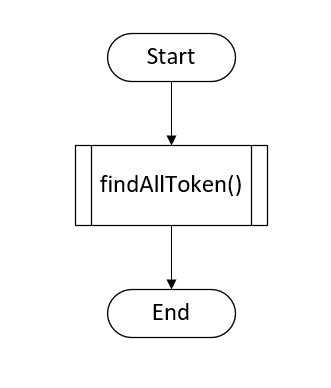


**4.程序设计与实现**

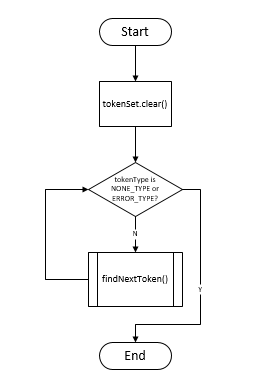
**4.1 程序流程图**

**词法分析模块**

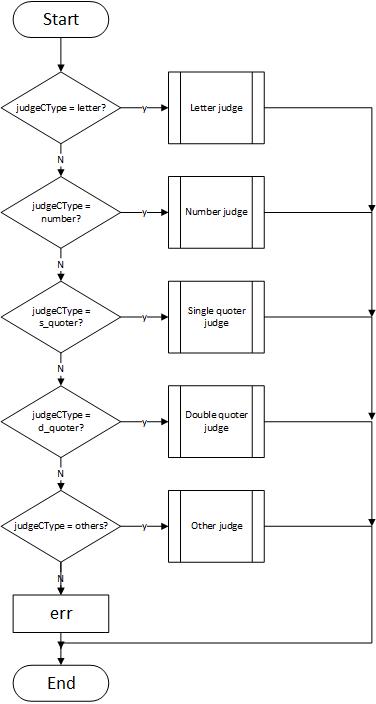
主进程，调用findAllToken()函数



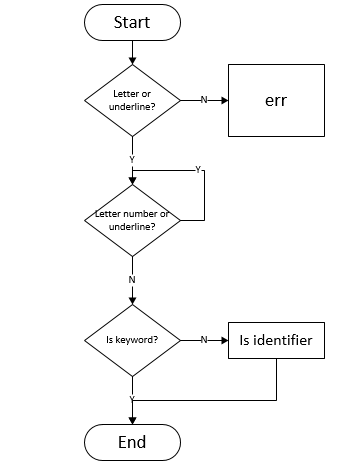
findAllToken进程，如果token类型正常则调用findNextToken()



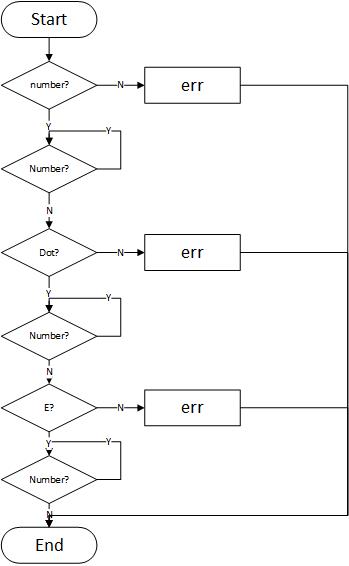
findNextToken 进程，判断第一个字符为何种类型，分别进入对应的子程序



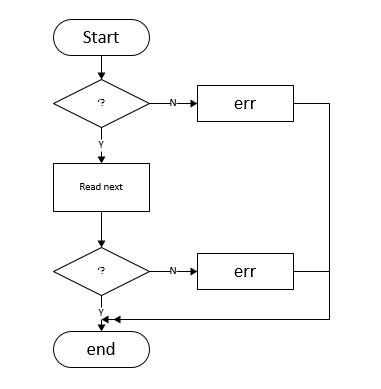
字符子程序，第一字符为字母或下划线，后续字母/数字/下划线



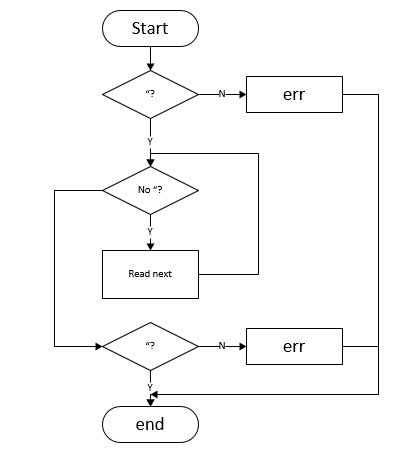
数字类型，格式为num.num e num



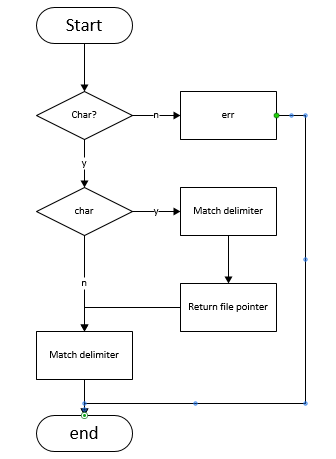
第一字符为单引号，则判断是不是字符类型



第一字符为双引号，则判断是不是字符串类型



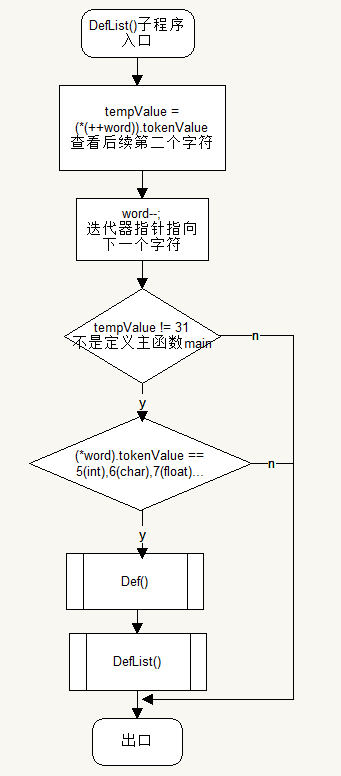
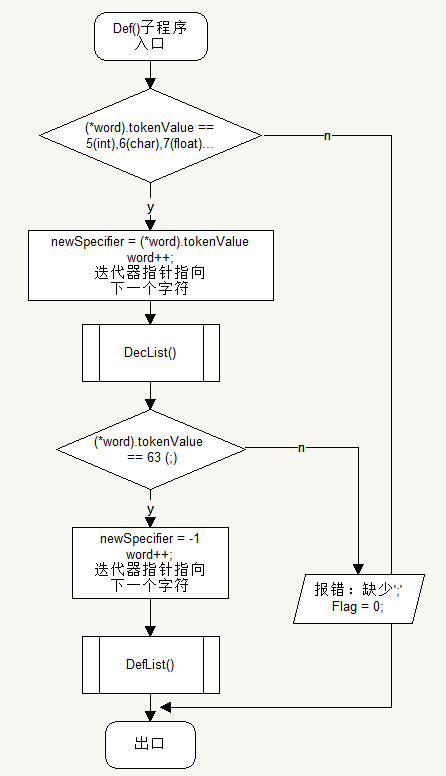
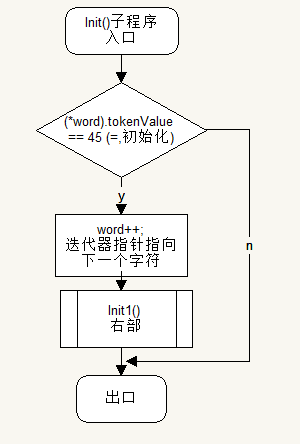
如果都没有匹配上，则要么是界符类型，要么err 输出

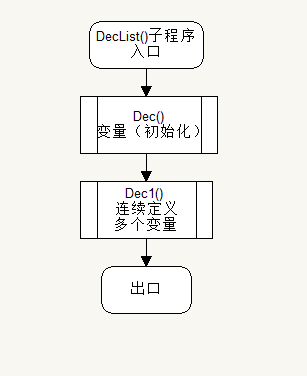


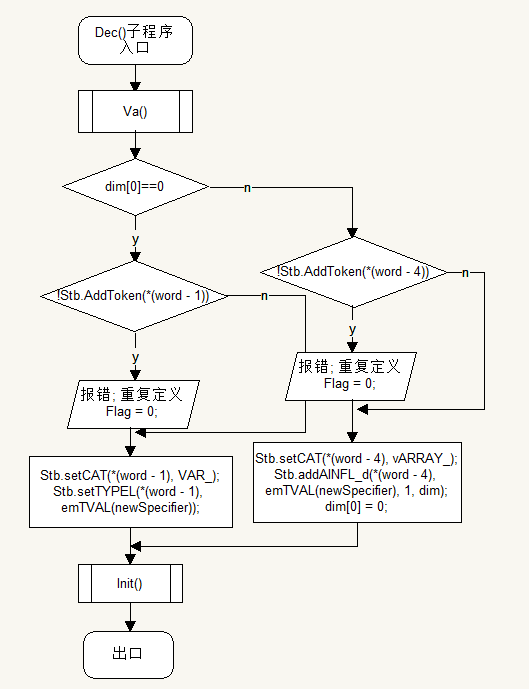
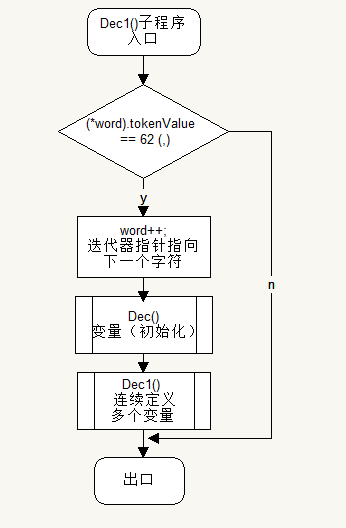
**语法分析模块**

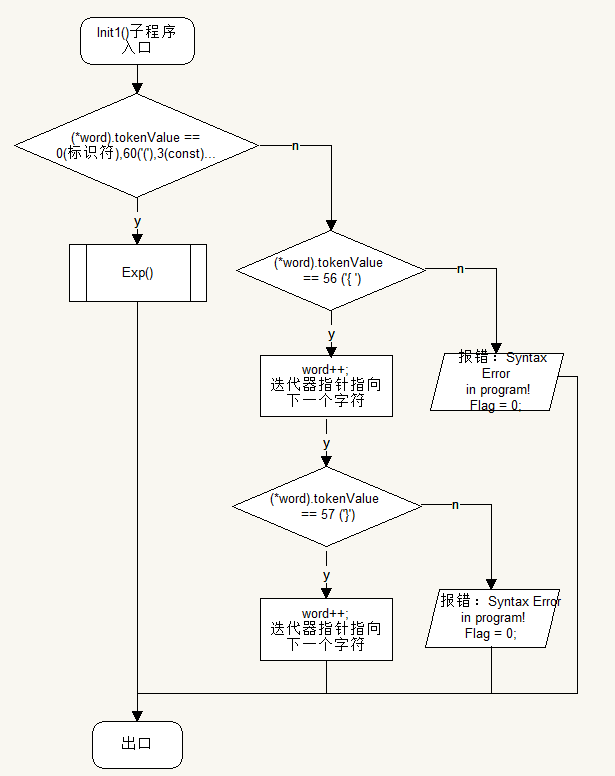
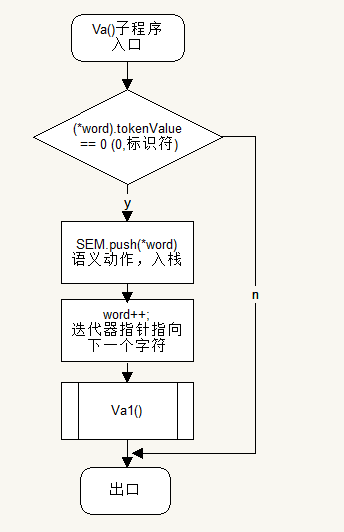
以变量定义(及初始化)模块为例进行说明：

如 int a, b = 5; 此时语法分析器要分析的下一个单词为 int，后续第二个单词为标识符 a, 不是主函数 main，则进入子程序Def(), 用newSpecifier记录变量的数据类型，进入子程序DecList()，之后进入Dec(), 在Dec() 中进入Va(), 当前单词 a 为标识符，a 入栈，取下一个单词，进入Va1(), 由于不是数组定义，Va1() 遇空直接出口，返回至 Va() , 再至 Dec(), 此时查符号表检查标识符a是否已被定义过，若是则报出错误，之后进入Init(), 由于变量a没有初始化，直接出口，子程序Dec() 返回至DecList()处理后续“, b = 5”token串，对“, b = 5”的分析与对变量a的分析类似，唯一不同的是变量b有初始化，会调用Init() 和 Init1()子程序，这里不再赘述。

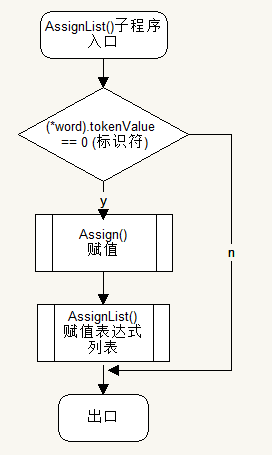
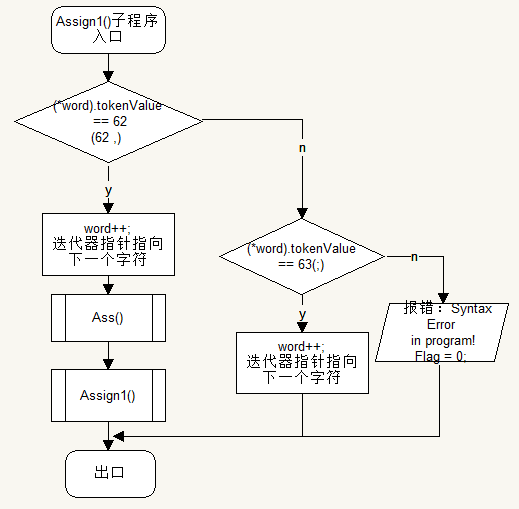
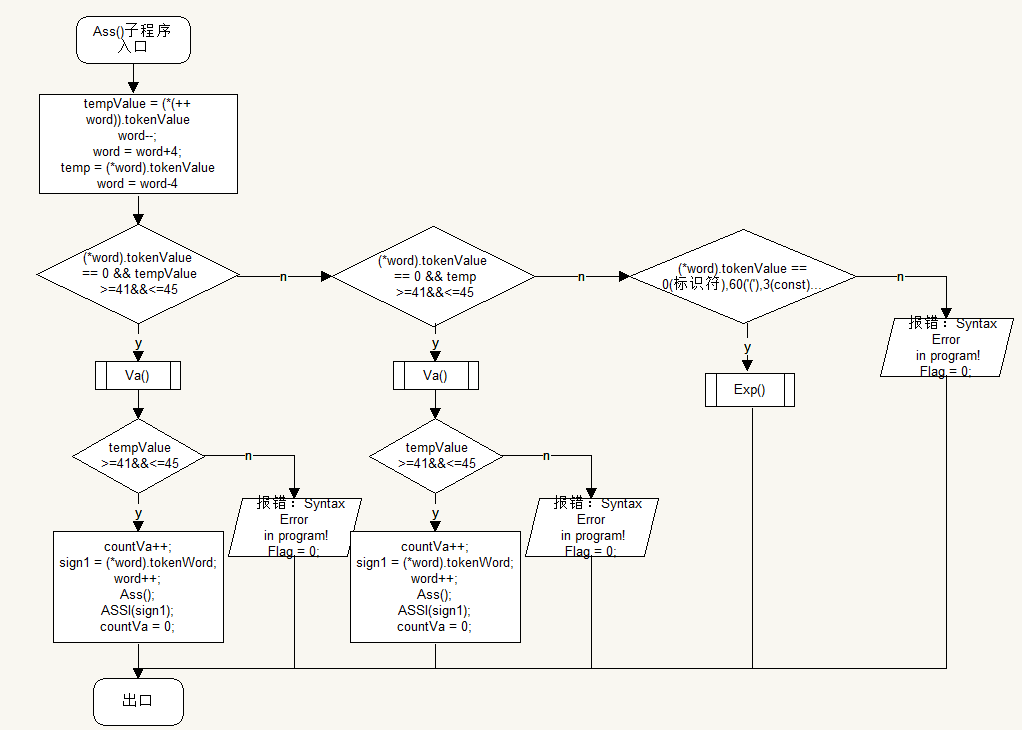
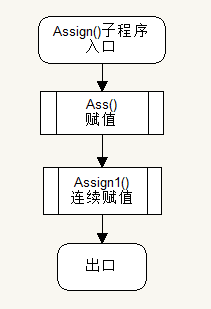






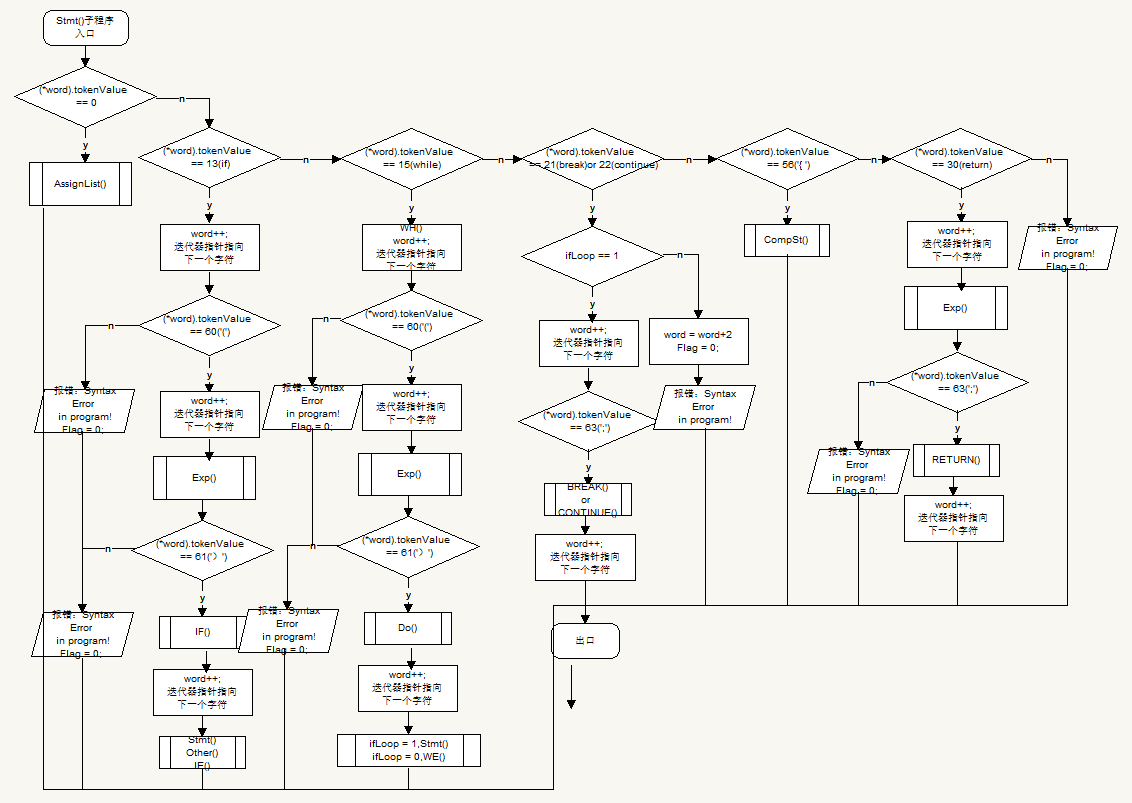


赋值语句模块子程序框图：

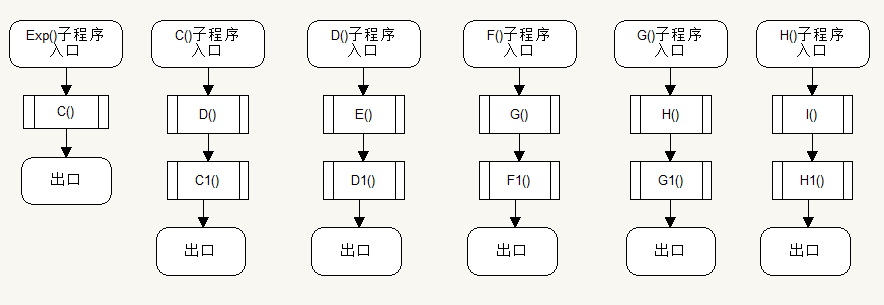
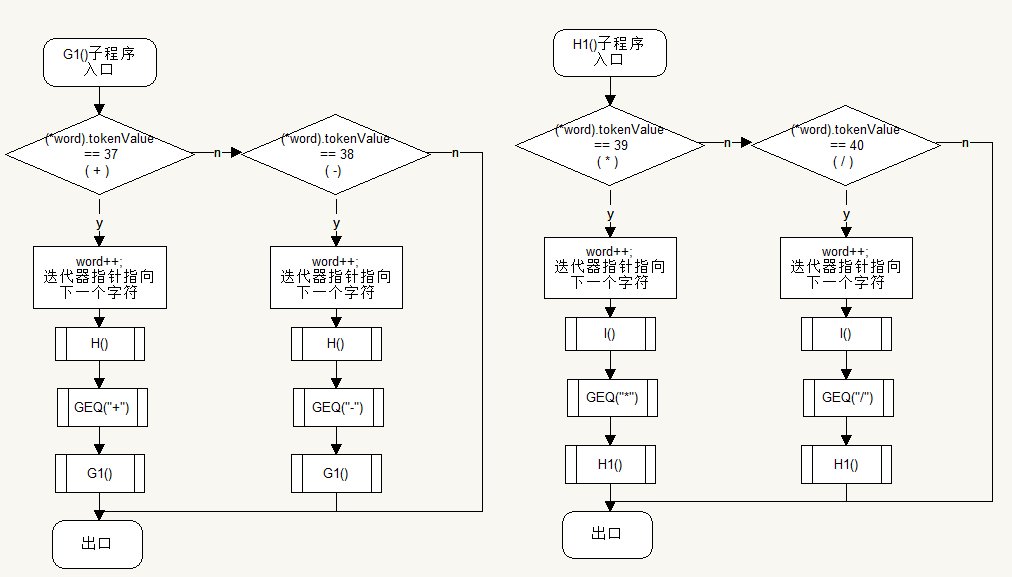


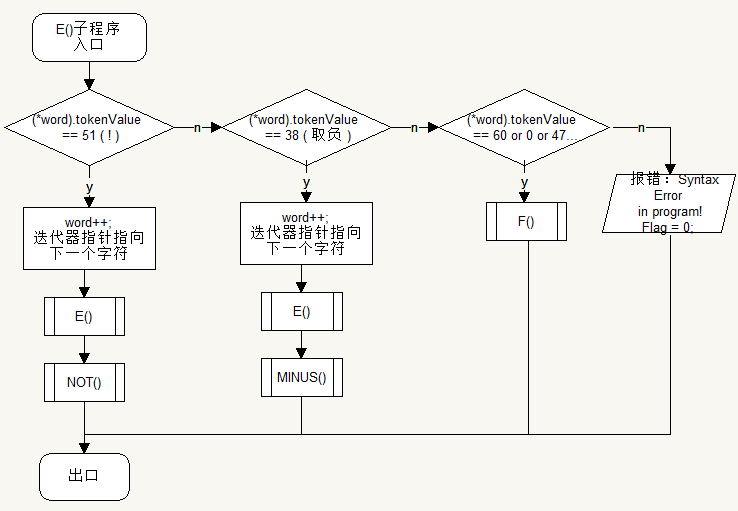
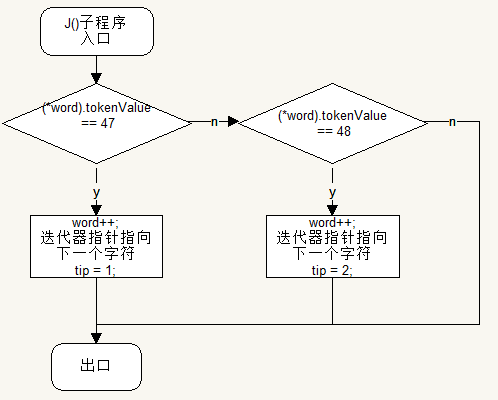
语句框图

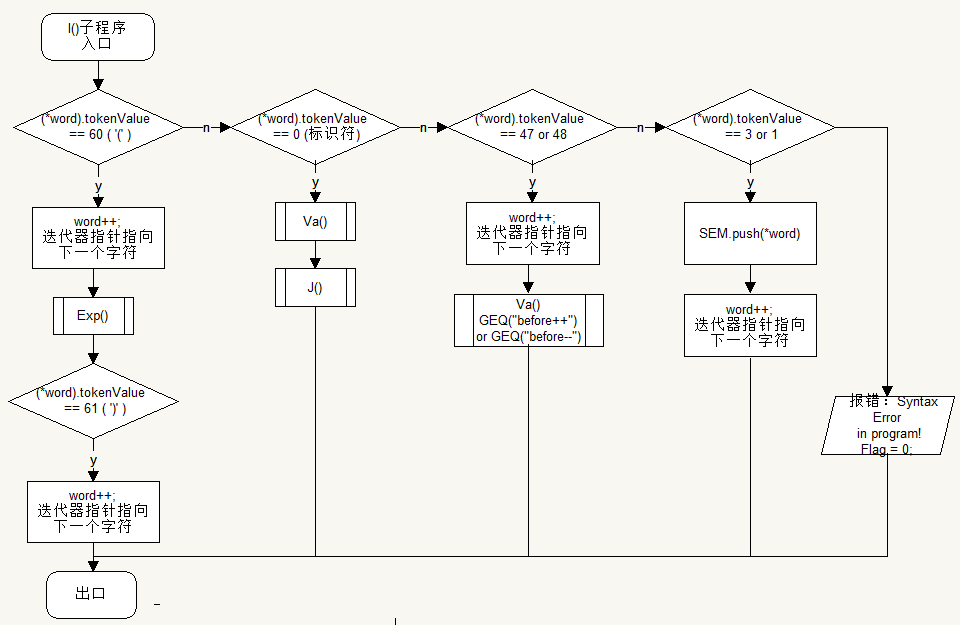
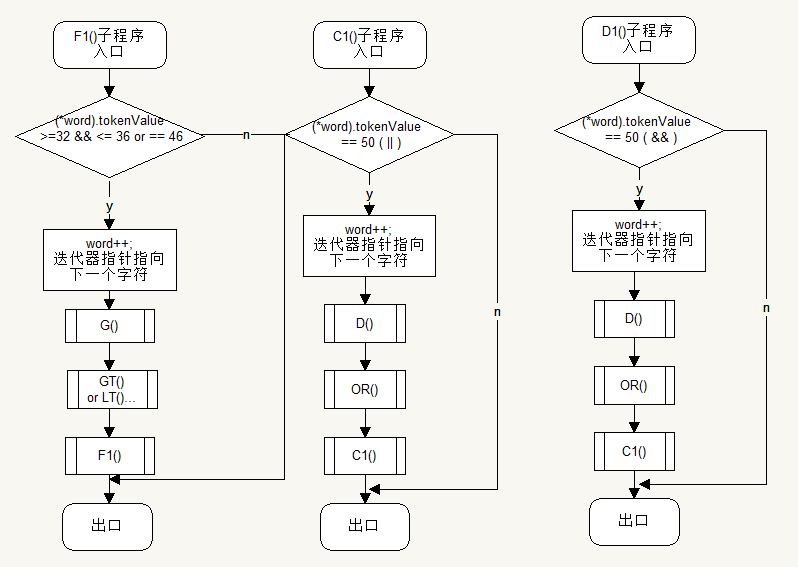
当前分析的单词token类码值为0时，证明要分析的语句是赋值语句，进入AssignList() 子程序；若类码值为13，则当前单词是 if, 进入对if语句的分析；若类码值是15，则当前单词为 while, 进入对while循环语句的分析，若类码值是50，则当前单词是‘{’，进入CompSt()子程序，return, break, continue语句类似。



表达式模块子程序框图

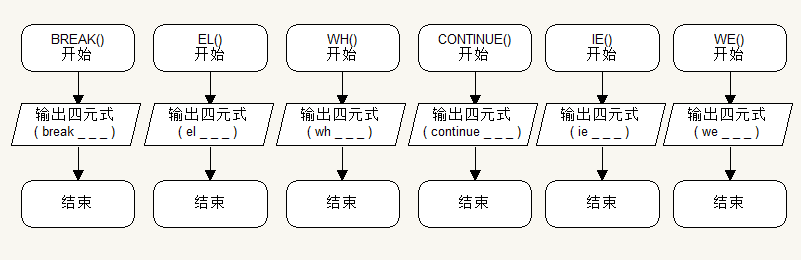




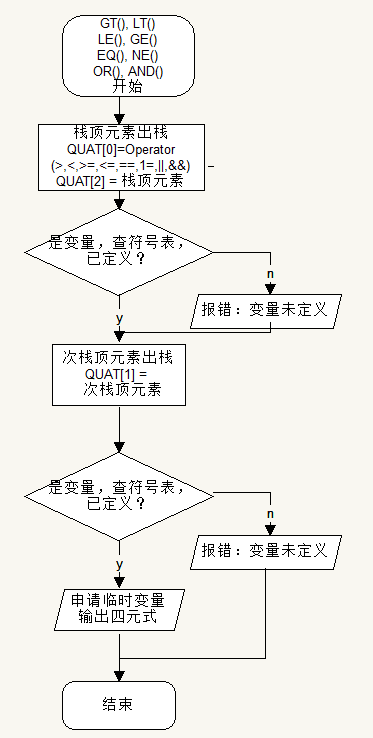
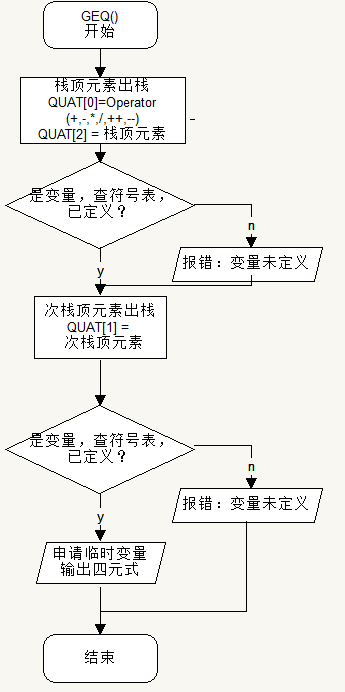
****

**语义分析模块**

if条件语句的转向四元式，结束四元式函数即 EL(), IE(), while循环语句的开始四元式，循环四元式函数即WH(), WE(), 以及break语句和continue语句四元式函数均是直接输出，流程图如下：



逻辑运算即大于、小于、大于等于、小于等于、等于、不等于、与、或的四元式函数与算数运算的四元式函数类似，四元式第一元为相应运算，栈顶元素出栈作为四元式的第三元，次栈顶元素出栈作为四元式的第一元，同时元素若是变量在函数体里调用符号表相关函数，查符号表若变量未定义则报错，申请临时变量作为四元式第四元，最后输出四元式。



**符号表模块**

SymbolTable::SymbolTable() //构造函数



bool SymbolTable::beginDefFunction(token& tk)//开始定义一个名为tk的函数

****

bool SymbolTable::endDefFunction(token & tk) //结束定义函数tk

****

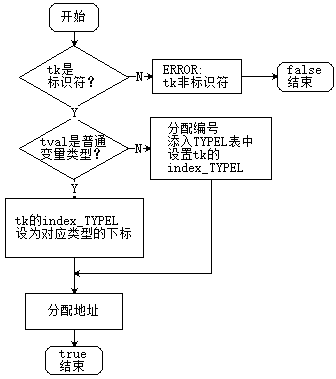
bool SymbolTable::AddToken(token & tk) //新增标识符tk

****

bool locateToken(token& tk);//查找tk是否曾定义

****

bool setTYPEL(token& tk, emTVAL tval);//设置tk的tval值（int，char…）

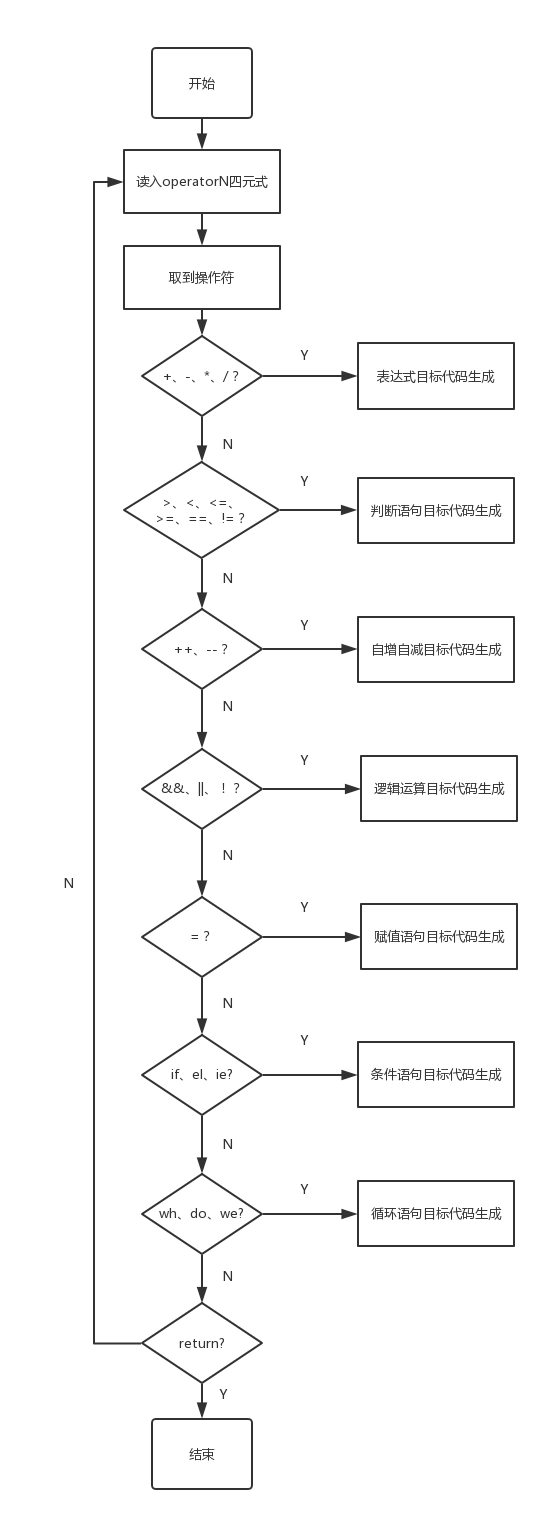


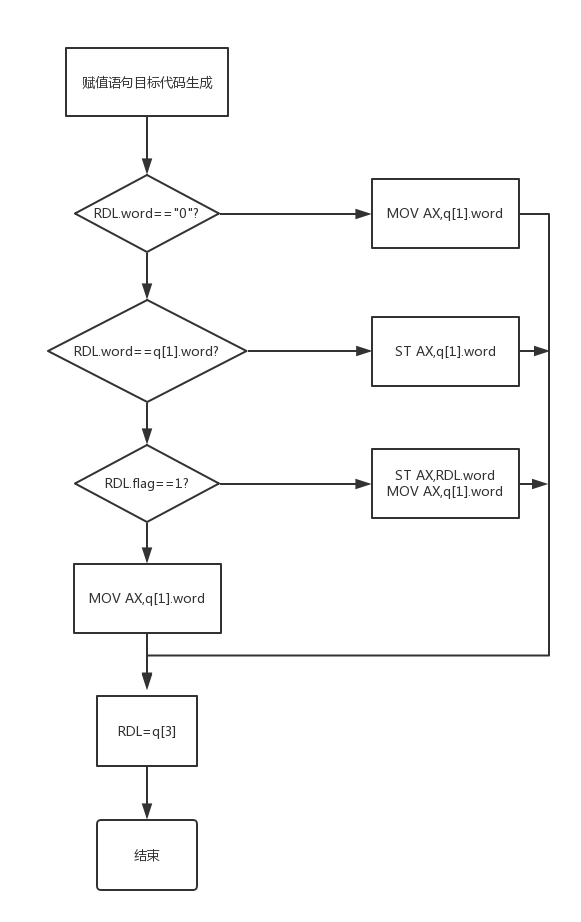
bool SymbolTable::addAINFL\_d(token & tk, emTVAL baseTVAL, int num\_dim, int \* up\_dim)

//设置 tk为多维数组

****

**目标代码模块**





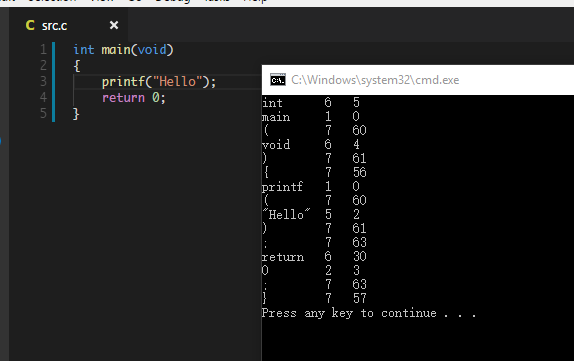
**4.2 程序说明**

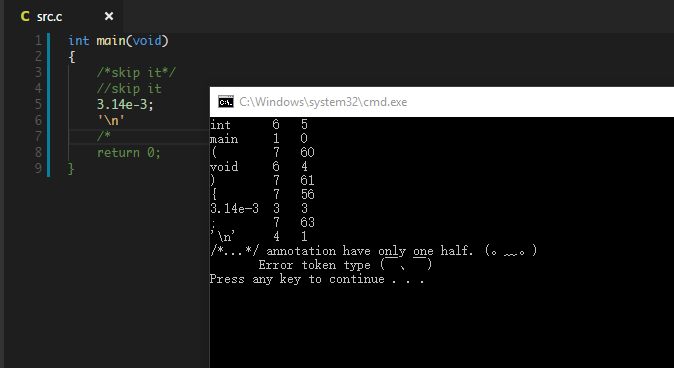
各子函数执行过程具体说明见程序4.1中的文字描述。程序执行时，从主函数开始，调用词法分析相关函数，完成对源程序的词法分析，得到源程序中所有单词的token序列并输出。语法分析时从左至右扫描分析token序列，同时建立完善符号表，插入语义动作，生成四元式，并输出相关结果。在生成的四元式的基础上，结合目标代码生成算法，生成汇编语言程序并输出。

**4.3 实验结果**

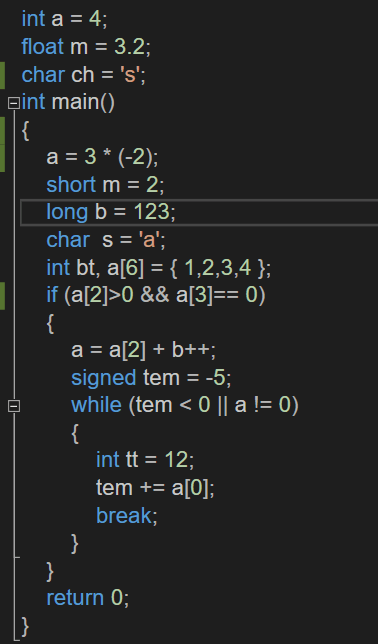
**词法分析部分的结果如下图所示：**

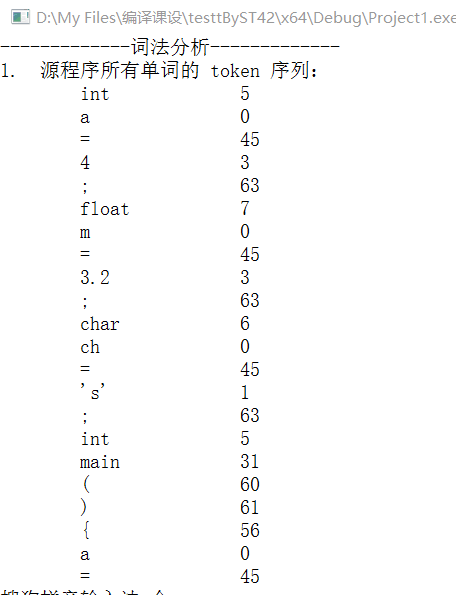
（单独测试）

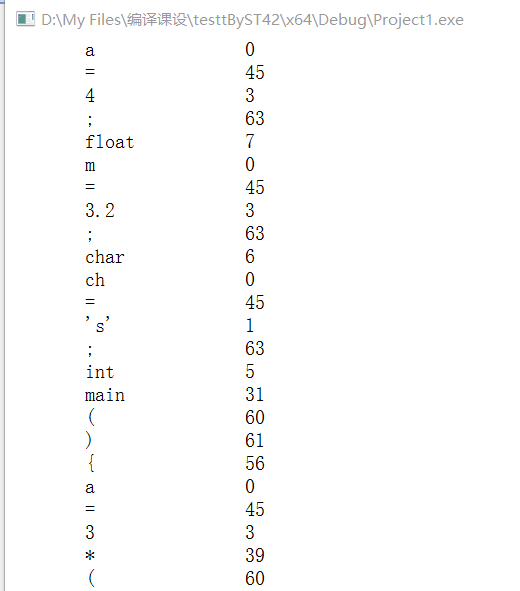


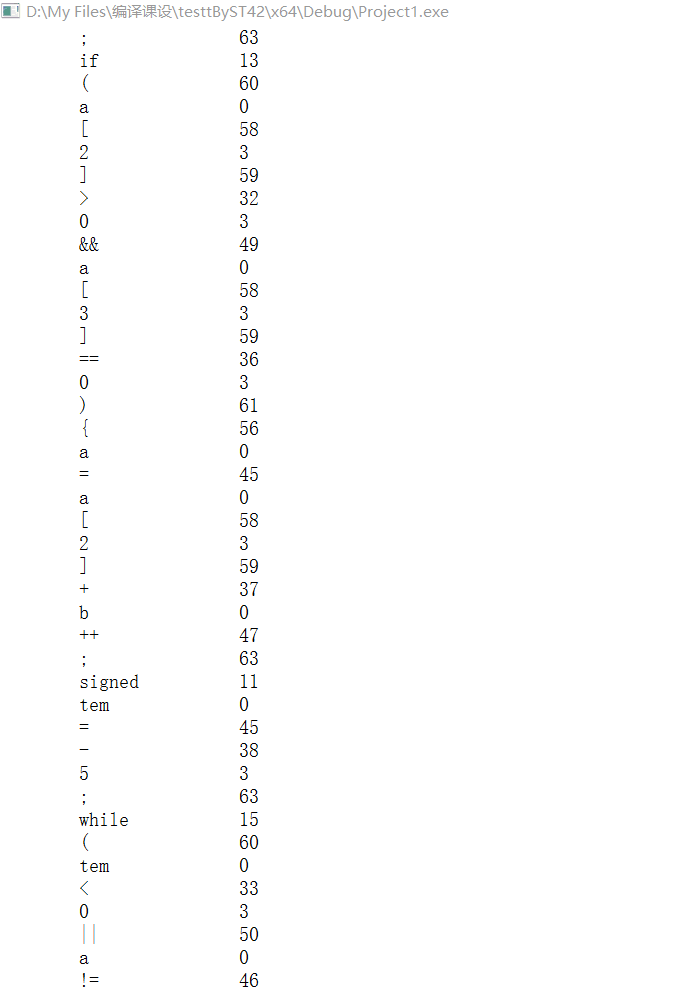


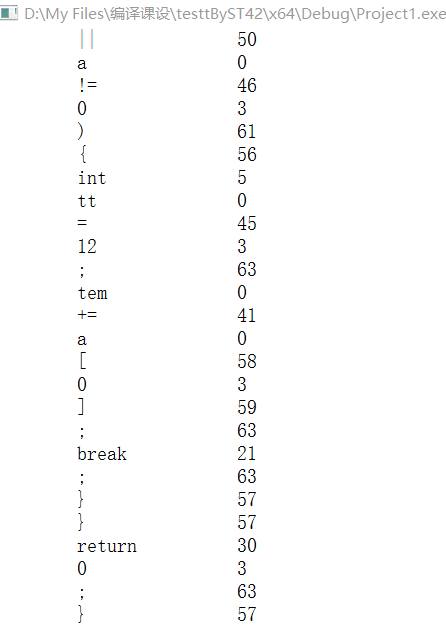
在下面截图中的源码下测试词法分析：(注：token序列的输出项有修改)

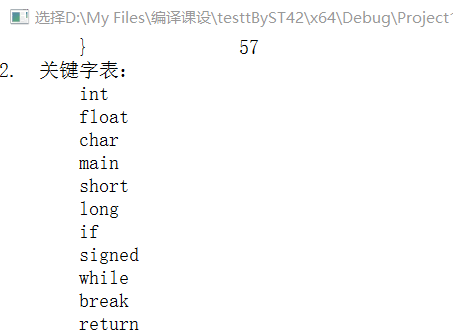


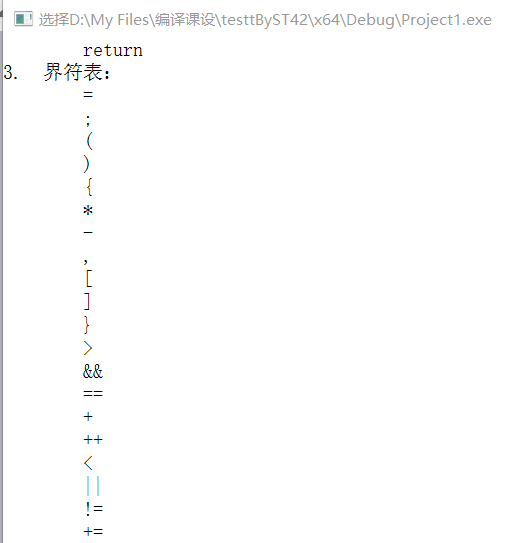


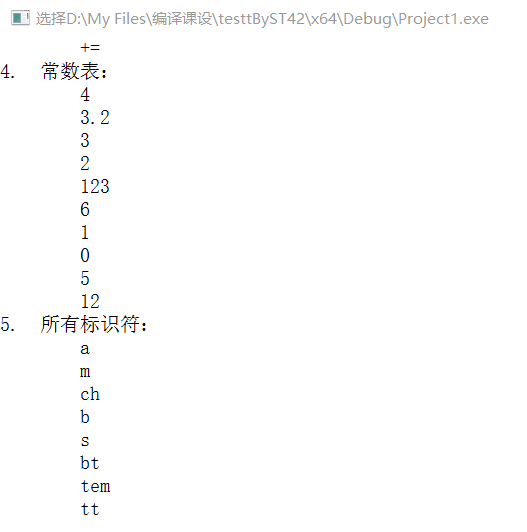






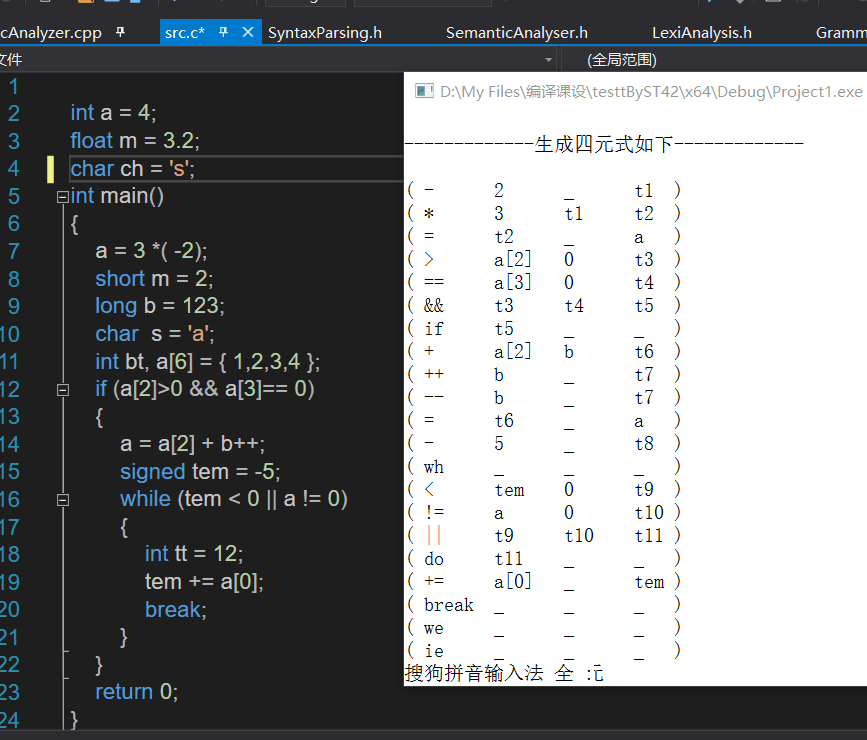






**语法分析及语义分析部分的结果如下图所示：**

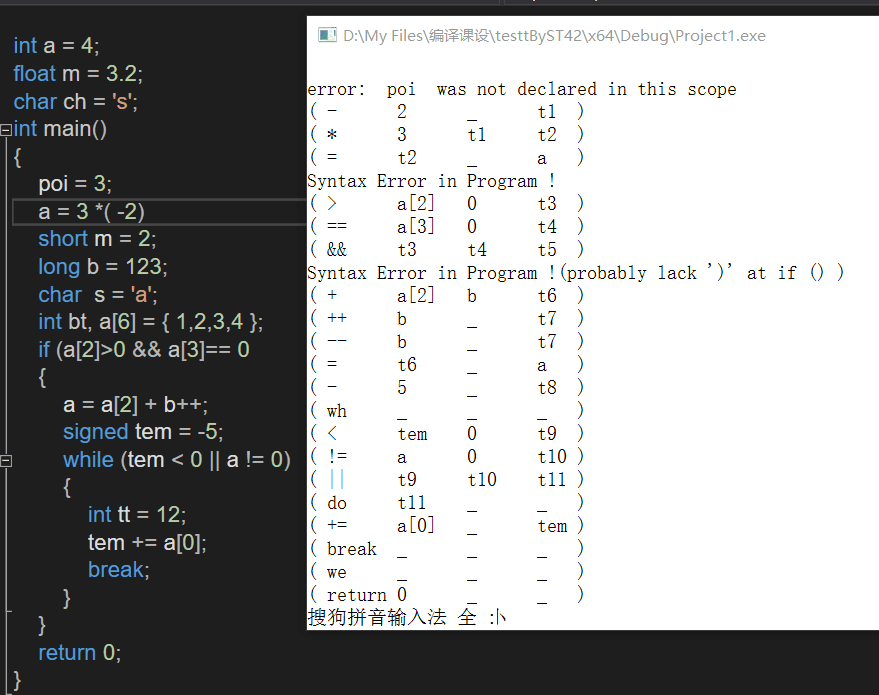
**语法正确时：**



对以上源程序做以下错误改动：

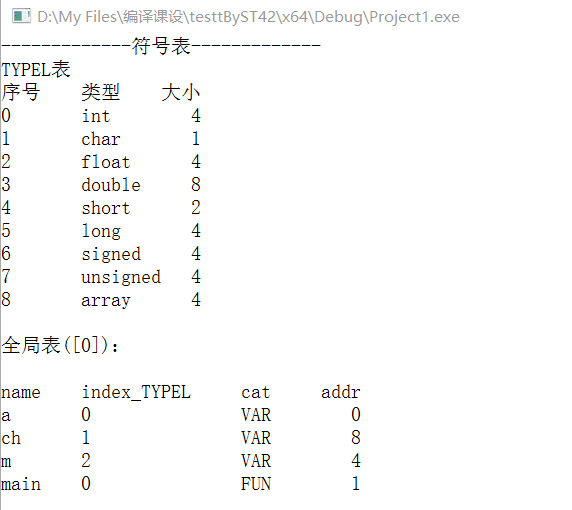
1. 增加赋值语句 poi = 3; 其中poi是未定义的标识符
2. 赋值语句a = 3 \*( -2)；去掉分号
3. if (a[2]>0 && a[3]== 0) 语句去掉 右括号

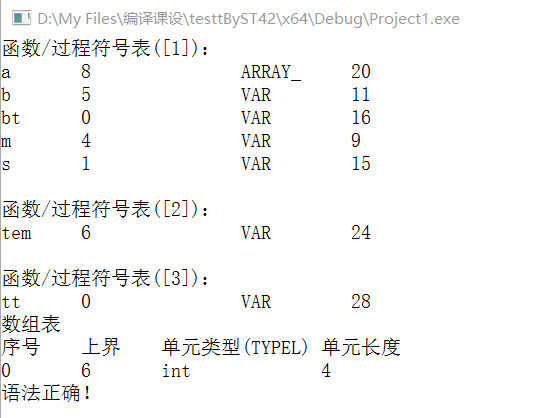
实验结果如下图：



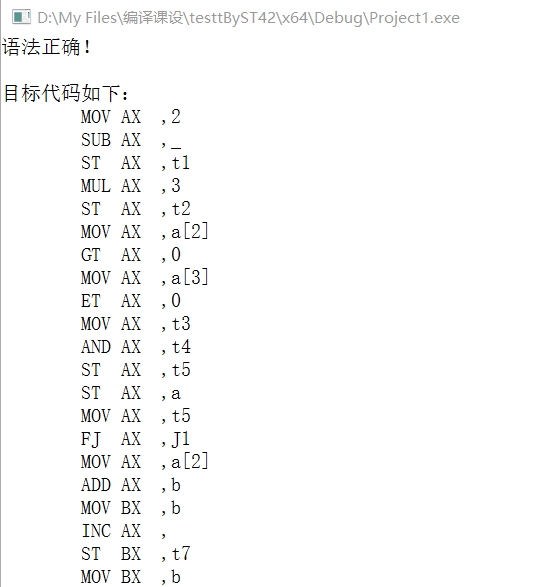
**符号表部分的结果如下图所示：**

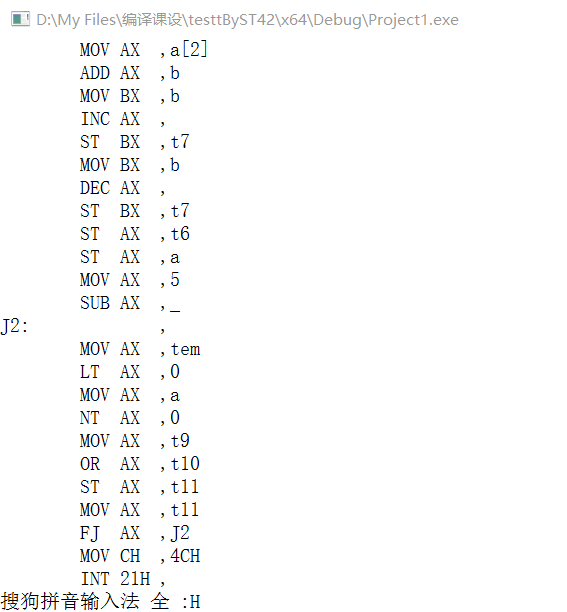
（源程序为上图修改前的正确的源程序）





**目标代码生成部分的结果如下图所示：**





1. **结论**

对输入的正确的源程序，经过词法分析器的词法分析，语法分析部分的语法分析，以及语义分析，最终可以生成全部正确的四元式，相应的符号表系统和目标代码部分的汇编程序。

对输入有误的源程序，若源程序中的单词不符合规则，词法分析部分会报出相应的错误；若源程序中存在像缺少’ ; ’之类的语法错误，语法分析的结果将是存在语法错误，并会在相应的位置提示；若源程序中变量未定义就被使用，则通过查符号表可报出变量未定义的错误。

通过小组四人两周坚持不懈的努力与合作，此次课程设计的编译程序设计合理，功能较全，完成了编译器前端的所有要求。

**6. 参考文献**

1、陈火旺.*《程序设计语言编译原理》（第3版）*. 北京：国防工业出版社.2000.

2、美 Alfred V.Aho Ravi Sethi Jeffrey D. Ullman著.李建中，姜守旭译.《*编译原理*》.北京：机械工业出版社.2003.

3、美 Kenneth C.Louden著.冯博琴等译.《*编译原理及实践*》.北京：机械工业出版社.2002.

4、金成植著.《*编译程序构造原理和实现技术*》. 北京：高等教育出版社. 2002.

**7. 收获、体会和建议**

**史英慧：**在这次课程设计中，我担任组长，负责了整个编译程序的文法和翻译文法设计，以及语法分析和语义分析。我很清楚文法是编译程序的基石，首要的任务便是设计一个合理清晰而不太复杂的文法，由于起初对设计文法不是很自信，况且本身难度也不小，希望能通过查找相关资料找到一个强大的文法。后来发现令我满意的逻辑清楚合理的几乎没有，便只能结合已有知识，自己由小到大，由简单到复杂一步步地设计与尝试。对于语法分析，我采用的是递归下降子程序分析方法，一是因为它相对简单明了，二是因为起初想用LR(0)分析法，构造出可归约前缀图后发现工程过于庞大将会导致程序出Bug时很难调试，便只好放弃。我当时已经知道现在编译技术领域主流的语法分析方法是自底向上的LR()分析法，递归子程序分析方法要求文法必须是LL(1)文法，其实有不小的局限性，我在之后的编程中便感受到要实现功能稍微复杂一点的文法只查看一个字符是做不到的，必须结合后续单词进行选择判断，这便暴露出了LL(1)文法的不足之处。所以觉得没有用主流的分析方法是这次课程设计中的一个小遗憾。此外，由于自己考虑问题时倾向于完美主义，比如说我在设计变量定义的文法时，因为平时编程习惯地对变量进行初始化和在一条语句中连续定义多个变量，便要考虑到文法要有这个功能，又比如我在设计算数运算表达式的文法时，考虑到自增、自减、取负等功能，在设计逻辑运算表达式的文法时想着要尽量包含所有常用逻辑，就投入了更多的时间，也因此导致到最后没有充足的时间去实现子程序，这也是一个遗憾。总而言之，从这次为期两周的课程设计中，我真的觉得自己学到了很多，从考虑问题方面，从团队协作方面，从程序调试培养耐心等各个方面，当然也认识到了自己的很多不足之处，仍然要不断地不懈地努力追赶。非常感谢我的几个组员对我的工作非常地配合，最后的建议就是希望能增加一些课时，希望任课老师在课堂上有时间可以讲一些PPT中较难的算法的实现。

**梁晨：**在这次课程设计中，我负责的模块是词法分析器Tokenizer。在词法分析器的具体实现上，我大量的运用了工程化的思想。比如大量运用宏来提高程序的精简程度以及可扩展性，有效的封装为后续的操作提供接口，方便后续对词法分析的使用等等。在这次课程设计中，我在队伍中引入了一些开发工具，比如Git、Github以及cmake，大大提高了队伍的开发效率以及协作程度。这次课程设计再一次让我意识到了工程能力的宝贵，一个好的设计对整个代码工程而言都是有极大益处的。建议在课程设计的过程中引入一些工程化思想，对学生以后的工作与研究都会有很大帮助。

**张宇航：**

收获主要有三，其一本次实践综合运用的一个学期以来所学习的编译原理知识，深刻感受到了编译技术在处理文字信息时强大的生命力。其二再次感受到在团队协作中，队员之间充分交流沟通的重要性。其三在编程实践中学习采用至上而下的程序设计方法，使得程序设计稳步有序进行。

我负责的符号表部分，与其他队员的工作有着密切的联系，这就需要与他们进行详细的沟通交流，比如某些接口的参数如何设定？哪些参数不容易取得？何时调用比较方便？在沟通之后才能产生最有效的代码。这是本次实践最深刻的感悟，为以后的团队合作积累了宝贵的经验。

**武博：**我本人负责的是目标代码生成的部分，之前对这一部分的知识总是心有敬畏，但是真地动起手来做这方面工作的时候，反倒能沉下心来好好理解这方面的知识并加以运用，而且在课程设计期间，我们的团队经常在一起讨论，甚至会一整天待在一起共同攻坚克难，让我体会到了团队的意义，所以本次课程设计，可以说收获颇丰；不过因为时间不足以及能力不足，这份工作并没有做完整，最后只是生成了形式上的目标代码，并不能直接执行，因为变量的地址信息没有加入其中，主要是因为和符号表的结合还没有完成，算是一大遗憾。