# 

# 《编译原理》实验报告

**本文档集合了编译原理大作业的实验报告加代码**

**实验主要内容为用C++实现了词法分析程序；语法语义以及四元式生成程序**

**代码见附录，复制进VS后程序绝对可编译执行。**

文档代码为原创，谨慎使用（姚砺的大作业）

# 实 验 设 计 一

## [一、实验名称]

词 法 分 析 程 序

## [二、实验目的]

（1）设计一个词法分析程序，每调用一次就从源程序文件中顺序识别出一个单词符号，并返回该单词符号的内部编码、单词符号自身、行列位置信息。

（2）要能处理单行注释。

## [三、实验内容及要求]

单词种类与识别规则

1. 标识符：首字符为字母或下划线，其后由字母、数字或下划线组成、

长度不超过255个字符；

1. 整数：由1到8个数字组成。
2. 小数：数字串1 **.** 数字串2，其中：数字串1由1-8个数字符组成；

数字串2由0-8个数字符组成，即：数字串2可以为空。

1. 字符串：由一对“”括起来的符号串，长度不超过255个字符；
2. 保留字：if、else、while、do、integer、float、string、input、output、

and、or、function、end、def、as、begin

1. 数学运算符： **＋**、**－**、**\***、**/**、**=**
2. 比较运算符： **<**、**<=**、**>**、**>=**、**<>**、**==**
3. 逻辑运算符： and、or
4. 分隔符： **{**、**}**、**(**、**)**、；、**，**

## [四、实验环境]

操作系统：Win7/其他

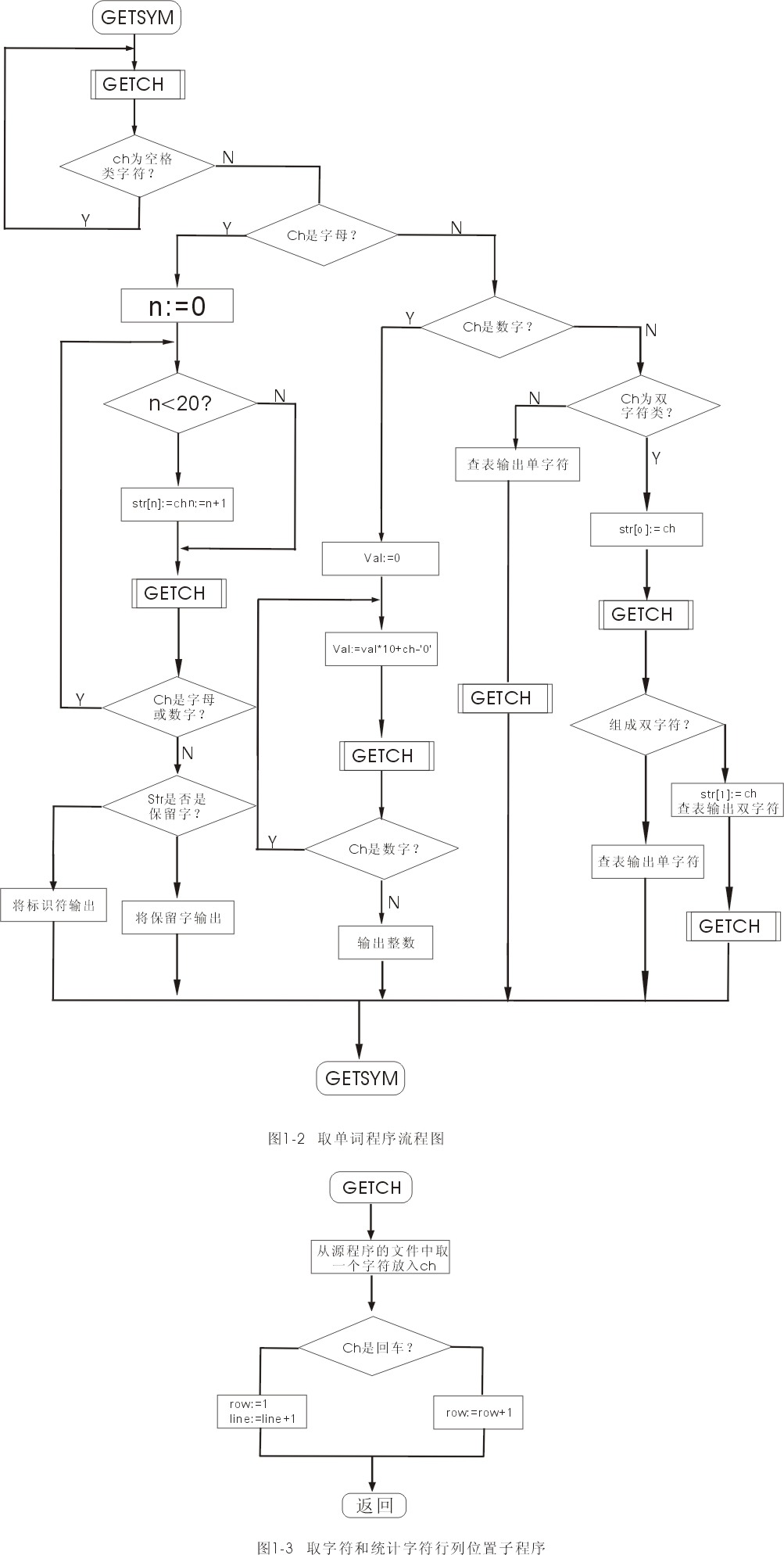
编译工具：VC++6.0 / CFree / VS2012

## [五、设计 ]

### 1设计大体思路

### 将读取的文件采用一遍扫描的方法，即从左到右只扫描一次源程序，将读取的数据存放在一个二维数组里。然后通过扫描函数scan，再从数组中一行一行的读取数据，每调用其依次返回一个单词的类型，同时单词本身以及行列号存放在全局变量中。而说词法分析作为语法分析的一个子程序，故在编写词法分析程序时，将会反复调用scan函数来获取一个个单词信息。

### 3设计流程图



### 4函数设计

**/\*词法分析函数\*/**

**int scan( string s ,int line )**

**框架：**

{

**初始化工作**

**是空格直接跳过，知直到读取到一个字符**

if( 是字母 )

{

**查表判断是否为关键字**

**判断是否为逻辑运算符and或or**

Else

**则为标识符**

}

else if( 是否为数字 )

{

**判断是整数**

**Else**

**是小数**

}

Else

{

**其余情况判断是否为运算符，字符串等**

}

else if( getchar==’/’ )

{

**if( content[line][i+1]=='/') //向前看一个,确定是否为行注释；**

**如果是，则游标指向行末，跳过行注释**

**if( content[line][i+1]=='/\*')**

**如果向前看一个发现时块注释**

**则一直向前扫描直到出现“\*/”停止，略过块注释**

**如果都不是则**

**Else**

**判断为除号，返回运算符类型**

}

}

**2 对其中部分实现的说明**

**(1) 数字识别**

while( content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字

{

text += content[line][i];i++;flag=1;

}

if( flag==1 )

{

if( content[line][i]=='.' )

{

text += content[line][i];i++;

while( content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字

{

text += content[line][i];i++;

}

return 2; //整数

}

return 3; //小数

}

每读入一个字符，判断是否数字，然后找小数点，找到即为小数

1. **标识符处理**

while( (content[line][i]>=65 && content[line][i]<=90) || (content[line][i]>=91 && content[line][i]<=122) || content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字或者字母或者下划线

{

text += content[line][i];i++;

}

for( j=0; j<=13 ; j++ )

if( text==key[j] ) //查表判断是否为保留字

return 5;

检查到读取的字符为字母时，进行查表判断，找到即说明为关键字

1. **空格，注释，行号的处理**

if( mode==0 )

{if( i<content[k].length() ) i++;} //如果为行注释 直接跳到下一行

else

if( mode==-1 ) //如果为块注释 找回行号

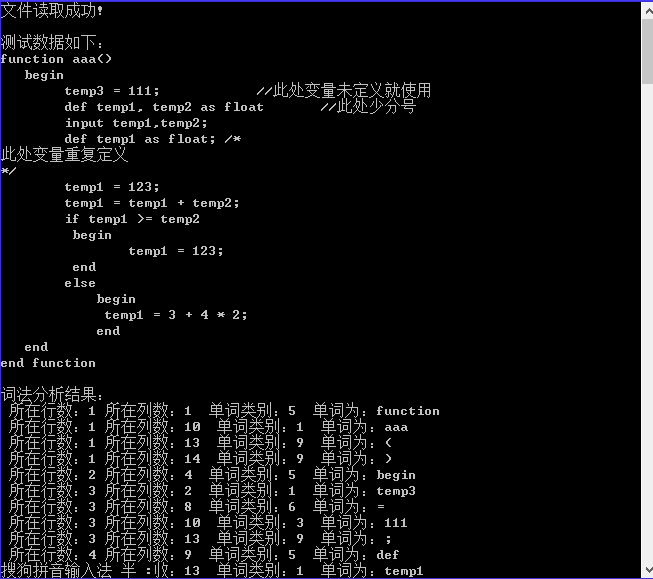
k=templine;

## [六、程序源代码]

见源代码文件。

## [七、实验数据、结果]

测试数据见文件



## [八、总结]

之前的每周作业曾经写过一个词法分析程序，所以这次写词法分析程序没什么太多困难。

在读取源文件时，最初的想法是一行一行的读取，读一次就存一行，这样可以节省空间资源，提高效率。但是仔细一想考虑到块注释分布在不同的行中，而且这样对行列的操作十分麻烦，而且许多地方涉及到要向前看，（比如判断注释与/除号运算符时）,最后只好一次全部读完，存在一个二维数组中。再用scan函数来处理。这样一来行列的处理就自由多了。

第二点，在写scan函数的时候，最初的想法是每调用一次，返回一个单词，然后在语法分析时不断调用它。但是仅仅返回单词是不够的，于是打算返回一个结构体，把单词信息行列号什么的全部存进去，这个在后来的语法语义分析中可以看到，这里任然采用的是返回一个单词类型。单词本身，和行列号作为全局变量保存。

然后只要按照流程图一步一步判断，慢慢写就行了，没有太多难度，唯一要注意的是单词位置行列的计算一定要细心。这个是要穿插在整个词法分析的全部地方的。所有涉及到读入一个字符的地方都需要考虑到如何修改行列变量。尤其是本程序拓展了块注释，要注意行列变化。

# 实 验 设 计 二

**注：由于语义分析和四元式生成都是在语法分析的基础上拓展得到的，这一章实验报告囊括了语法，语义和四元式。**

## [一、实验名称]

语 法 语 义 以 及 四 元 式 分 析 程 序

## [二、实验目的]

设计一个语 法 语 义 以 及 生 成 四 元 式 的 分 析 程 序

## [三、实验内容及要求]

**设计要求**：构造相应文法的语法分析程序，应能指出源程序中出现的错误。

为语法分析程序中添加类型检查功能，包括：

① 变量重复定义；

② 变量未定义就使用；

③ 变量未赋值就引用；

将语法正确的源程序翻译成四元式。

**文法见附录**

## [四、实验环境]

操作系统：Win7/其他

编译工具：VC++6.0 / CFree / VS2012

## [五、设计 ]

**1.语法分析设计**

1. **相关数据机构**

struct Token

{

string value; //值

int mode; //类型

int row; //行

int col; //列

}token;

说明：这本应是词法分析时的结构，但是scan函数返回的是但单一值，于是又设计了Token函数，对scan函数进行扩充，每个单词的信息均保存一次在结构体中，方便错误报告函数进行处理。

**（2）各个非终结符函数形式**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*语法语义 各产生式 函数\*/

void chengxu();

void hanshukuai();

void hanshu();

void yujukuai();

void yuju();

void bianliangdingyiyuju();

void shujuleixing();

void shuruyuju();

void shuchuyuju();

void fuzhiyuju();

void fenzhiyuju();

void xunhuanyuju();

void biaodashi();

void xiang();

void yinzi();

void buerbiaodashi();

void guanxibiaodashi();

void guanxi();

**其他函数：**

void error();

void warning();

每个函数具体作用在代码中都有注释说明

**（3）语法分析整体设计思路**

**采用递归下降的方法，为每个非终结符设计一个函数，在函数中对此处可能出现的语法错误进行处理。**

**处理也语法错误的思路主要分两种：**

1. **查找关键字**

**由于进行语法分析时检查到错误不可能就此停下，要一次性尽量找到所有的错误，而且测试数据的语法错误形式千奇百怪，这就要求发现错误之后如何找到继续进行分析的位置。这里对于一些不常见的错误采取查找关键字的方法，一旦出现错误在相应函数中查找下一个出现的关键字的位置，然后继续分析。**

1. **跳过错误单词，直接分析下一个单词**

**由于查找关键字方法会跳过许多代码，从而有可能造成一些信息不必要的遗漏。尤其是对于漏分号这种错误，与是这里采取，直接当前错误，默认后续单词可能就是正确单词，继续分析。但是这个方法在一些地方并不适用，可能会导致一连串的错误出现，于是在使用时要考虑好取舍。**

**编译错误处理函数：**

void error( Token token,char \*msg) //编译错误处理函数

{

cout << "编译错误："; printf( "%s",msg); cout << endl << "错误位置：第" << token.row << "行，第" << token.col << "列 -> " <<token.value;

cout << endl;cout << endl;

errornum++;

}

**每出现错误时，通过传递错误信息参数以及单词信息，调用编译错误处理函数进行实时报错。**

**（3）测试数据和结果**

**见后文语义分析部分**

1. **语法分析部分的小总结**

**原本打算试试自底向上归约方法设计的，结果想了想发现实在麻烦，老师给的文法又挺庞杂的，创建分析表什么的就要好久。后来一想用递归下降方法好处很多，做到之后的语义分析和四元式生成的时候在语法分析的基础上拓展很方便，整体框架非常清晰。在错误处理方面考虑的还是挺合理的，对于一些常见错误都能有效指出。并且程序只有一个出口，任何代码输入都会从那一个出口结束，不会引起程序崩溃。**

**2、语义分析设计**

**语义分析主要实现3个功能**

① 变量重复定义；

② 变量未定义就使用；

③ 变量未赋值就引用；

**(1)相关数据结构**

struct Symtable //符号表结构

{

string name;

int type;

bool value;

}S;

struct compare: binary\_function<Symtable, string,bool> //仿函数和绑定器

{

bool operator()( Symtable s, string str) const

{

if (s.name== str)

return true;

else

return false;

}

};

**(2)相关函数**

void push\_in\_token( Token token ) //进表操作

{

S.name = token.value;

S.type = -1;

S.value = false;

ST.push\_back( S );

}

vector<Symtable> :: iterator find\_token( Token token ) //查表操作

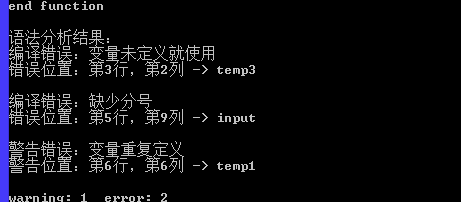
{

it = find\_if( ST.begin(), ST.end(), bind2nd( compare(),token.value ));

return it;

}

1. **测试数据和结果**



**（4）设计思路**

**语义部分主要采用符号表结构，符号表中记录了每个非终结符的信息。（名称，类型，是否赋值）。然后再程序中每次调用Token函数发现读取到标识符时都进行查表操作。**

**根据查表结果进行相关分析。**

**在赋值函数中，如果查表找到其信息就说明重复定义。**

**再其余函数中**

**1.没有找到说明为声明就使用。 2.找到发现未赋值说明未赋值就引用。**

**（5）语义部分小总结**

**假如只实现以上3个功能语义部分并不难，关键是符号表的构建以及进表查表等操作。**

**在这一部分我使用了仿函数和绑定器进行查表。由于我把相应表结构都存在vector内，导致查找结构内成员信息的不便，但是自己写一套查找操作代码估计效率不高，索性使用仿函数进行查找，泛型操作一般效率较高而且代码较少也很清晰。假如时间更充足一点其实还可以把数据类型匹配这一部分拓展一下，毕竟也是查表比较，但是时间有限只好先完成要求的3个任务。**

**3、四元式分析设计**

**（1）相关数据结构**

struct Quaternary //四元式结构

{

int serial;

string op;

string v1;

string v2;

string result;

}Q;

stack<string> operator\_stack; //操作数堆栈

stack<string> operand\_stack; //操作符堆栈

**（2）相关函数**

Quaternary Quaternary\_generater( int serial, string op, string v1, string v2, string result ) //四元式生成函数

{

Q.serial = serial;

Q.op = op;

Q.v1 = v1;

Q.v2 = v2;

Q.result = result;

return Q;

}

void Quaternary\_maker()

{

while( !operand\_stack.empty() && !operator\_stack.empty() )

{

op = operator\_stack.top();

operator\_stack.pop();

v1 = operand\_stack.top();

operand\_stack.pop();

v2 = operand\_stack.top();

operand\_stack.pop();

res = result[r];r++;

Q = Quaternary\_generater( serial, op, v1, v2, res );serial++;

QV.push\_back( Q );

////////////////

for( it2=QV.begin(); it2!=QV.end(); it2++ ) //找回真出口

if( (\*it2).result=="null" )

{

stringstream ss;

ss<<QV.back().serial;

string s=ss.str();

(\*it2).result=s;

break;

}

///////////////

operand\_stack.push( res );

}

}

void Quaternary\_output()

{

while( !QV.empty() )

{

cout << QV.front().serial << " ( " << QV.front().op << " , " << QV.front().v2 << " , " << QV.front().v1 << " , " << QV.front().result << " ) " << endl;

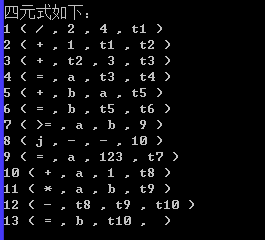
it2 = QV.begin();

QV.erase( it2 );

}

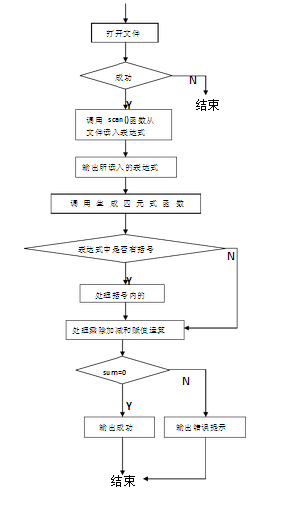
}

1. **测试数据和结果**



**（4）设计思路**

**赋值运算部分整体结构流程如下：**



**说明：**

**首先建立操作符栈和运算符栈。每次扫描遇到操作符遍进栈，遇到运算符如果当前栈空则进栈，否则和栈顶操作符的优先级进行比较，如果小于等于其优先级，则pop栈顶符号，进行四元式生成，再将四元式压入四元式队列，然后继续与栈顶元素比较直至其优先级大于其优先级或栈空则进栈。**

**If语句部分**

**在对布尔表达式进行判断时，先让操作符和运算符进栈，转移序号先赋为null**

**等到扫描到if then之后的部分通过对其四元式生成确定了序号，这时候再查找队列进行回填序号。**

**函数具体结构：**

void biaodashi()

{

xiang();

while( token.value=="+" || token.value=="-" )

{

///////////////////////////////////////////////四元式

while( !operator\_stack.empty() && operator\_stack.top()!="=" ) //此处要注意用while循环，if会出错，因为需要不断判断直至栈顶的符号优先级小于需要压栈的符号

{

此处生成四元式，四元式进队列

for( it2=QV.begin(); it2!=QV.end(); it2++ ) //找回真出口

if( (\*it2).result=="null" )

{

stringstream ss;

ss<<QV.back().serial;

string s=ss.str();

(\*it2).result=s;

break;

}

}

operator\_stack.push( token.value ); //如果栈空或者准备进栈的符号的优先级大于栈顶的优先级则进栈

token = getToken();

xiang();

}

}

**（5）四元式的总结：**

**这部分是我觉得此次试验最难的一部分。而且也是我花费时间最多的一部分。**

**在写赋值运算的那部分时，把一个while循环写成了if（我一开始以为运算符进栈只需判断一次）导致测试数据结果出错找了好久错误才找到。If语句那部分的四元式的编写也十分困难，尤其是找回真假出口让我考虑了很久。而且测试数据整个代码有许多这种语句，应当在每个语句结束时就输出一次四元式，不然会导致次序的混乱。（一开始我没发现，是把所有四元式进队列到最后程序结束时一起输出的），最后调试了许久，才发现问题，于是在语句结果判断分号时，就调用四元式输出函数进行输出。**

## [六、整体总结 ]

**这次编译原理大作业让真的我受益良多，整个近千行代码基本都是我一点一点慢慢敲出来的，过程固然辛苦但是学到了不少。尤其是在做语法分析和四元式生成的部分，代码量较大，一开始思路也不太清晰，但是还是一点一点硬着头皮往下写，结果在写的过程中一些思路就出来了，看来有时候还是要多动手，光苦思冥想并不行。对于这次作业我自己是比较满意的但是还是有一些不足：语法部分本来想对文法进行拓展把布尔表达式那部分的附加题解决掉，无奈时间有限，实在来不及写，可能留到暑假自己再完善吧。四元式部分其实做得并不完完善，有些测试数据会出现错误结果，主要是还是因为布尔表达式那部分文法没有拓展，在进行复杂布尔表达式判断时会出错，而且if语句真假口如果遇到复杂的算数运算可能会出口回填错误。总体来说，这次作业基本任务还是都完成了，通过这次大作业也对整个编译器的设计有了比以前更深的了解和体会，看来还是实践才能出真知。**

**附录**

**文法如下：**

1. <程序>—> <函数块>
2. <函数块>—> <函数> [<函数>]
3. <函数>—> **function id ( )** <语句块> **end function**
4. <语句块>—> **begin** 语句 [语句] **end**
5. <语句>—><分支语句>|<赋值语句>|<循环语句>|

<输入语句>|<输出语句>|<变量定义语句>

1. <变量定义语句>—> **def** **id** [ **, id** ] **as** <数据类型> **；**
2. <数据类型>—> **integer** | **float** | **string**
3. <输入语句>—> **input** **id** [ **, id** ] **；**
4. <输出语句>—> **output** <表达式>[ **,** <表达式> ] **；**
5. <赋值语句>—> **id =** <表达式> **；**
6. <分支语句>—> **if** <布尔表达式> <语句块> { **else** <语句块> }
7. <循环语句>—> **while** <布尔表达式> **do** <语句块>
8. <表达式>—> <项> [ **+**|**－** <项> ]
9. <项>—> <因子> [ **\***|**/** <因子> ]
10. <因子>—> **id** | **con** | **deci** | **(** <表达式> **)**

14. <布尔表达式>—> <关系表达式> [ **and** | **or** <布尔表达式> ]

15. <关系表达式>—> <表达式> <关系> <表达式>

16. <关系>—> **<** | **<=** | **>** | **>=** | **==** | **<>**

**附录： 词法分析完整代码（复制可执行）**

**/\***

**词法分析程序**

**BY 谢卓函**

**ON June 20th**

**\*/**

**#include <iostream>**

**#include <algorithm>**

**#include <cstring>**

**#include <fstream>**

**#include <string>**

**#define MAXLINE 30**

**using namespace std;**

**//////////////////////////**

**//标识符1，整数2，小数3，字符串4，保留字5，数学运算符6，比较运算符7，逻辑运算符8，分隔符9**

**string key[14]={"if","else","while","do","float","string","begin","end","def","integer","input","output","as","function"}; //保留字**

**string border[5]={ ";" , "{" , "}" , "(" , ")" }; //分隔符 5**

**string arithmetic[9]={"+" , "-" , "\*" , "/" ,"<" , "<=" , "=" , ">" , ">=" }; //运算符 6**

**string text; //记录标识符 1**

**string content[MAXLINE]; //保存从文件读取的内容**

**int i=0; //i为字符游标**

**int templine = 0;**

**int scan( string s ,int line )**

**{**

**int j, flag =0;**

**text = ""; //赋空text**

**while( content[line][i]==' ' ) //判断空格**

**i++; //是空格跳过**

**if( (content[line][i]>=65 && content[line][i]<=90) || (content[line][i]>=91 && content[line][i]<=122) ) //判断是否为字母或者下划线**

**{**

**text += content[line][i];i++;**

**while( (content[line][i]>=65 && content[line][i]<=90) || (content[line][i]>=91 && content[line][i]<=122) || content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字或者字母或者下划线**

**{**

**text += content[line][i];i++;**

**}**

**for( j=0; j<=13 ; j++ )**

**if( text==key[j] ) //查表判断是否为保留字**

**return 5;**

**if( text=="and" || text=="or" ) //判断是否为逻辑运算符**

**return 8;**

**if( j==14 )**

**return 1; //若查表失败说明为标识符**

**}**

**else**

**{**

**while( content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字**

**{**

**text += content[line][i];i++;flag=1;**

**}**

**if( flag==1 )**

**{**

**if( content[line][i]=='.' )**

**{**

**text += content[line][i];i++;**

**while( content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字**

**{**

**text += content[line][i];i++;**

**}**

**return 2; //整数**

**}**

**return 3; //小数**

**}**

**}**

**if( content[line][i]== ';' || content[line][i]=='{' || content[line][i]=='}' || content[line][i]=='(' || content[line][i]==')' || content[line][i]==',' ) //判断是否为分隔符**

**{ text = content[line][i];i++; return 9;}**

**if( content[line][i]== '+' || content[line][i]=='-' || content[line][i]=='\*' || content[line][i]=='=' ) //判断是否为运算符**

**{ text = content[line][i]; i++;return 6;}**

**if( content[line][i]=='<' || content[line][i]=='>' )**

**if( content[line][i+1]=='=' )**

**{text = content[line][i];text += content[line][i+1];i=i+2;return 7;}**

**else**

**{ text = content[line][i];i++; return 7;}**

**if( content[line][i]=='“' ) //判断是否为字符串**

**{**

**text += content[line][i];i++;**

**while( content[line][i]!='”' )**

**text += content[line][i];i++;**

**text += content[line][i];i++;**

**return 4;**

**}**

**if( content[line][i]=='/' ) //判断是否为注释**

**if( content[line][i+1]=='/')**

**{i=content[line].length();return 0;} //如果为行注释那么游标指向行末**

**else**

**if( content[line][i+1]=='\*' )**

**{**

**i=i+2;**

**while( content[line][i]!='\*' || content[line][i+1]!='/' )**

**if( i==content[line].length() )**

**{line++;i=0;}**

**else**

**i++;**

**i=i+2;templine = line;return -1; //-1代表块注释**

**}**

**else**

**{ text = content[line][i];i++; return 0;}**

**return 0;**

**}**

**int main()**

**{**

**ifstream infile("ceshi1.txt",ios::in);**

**if( infile )**

**cout << "文件读取成功! "<< endl;**

**else**

**cout << "文件读取失败!" << endl;**

**int hang, mode,k=1,numline; //k用于定位行数**

**cout << endl;**

**cout << "测试数据如下：" << endl;**

**while ( getline(infile,content[k]) )**

**{**

**cout << content[k] << endl;**

**k++;**

**}**

**numline = k--; //numline记录了文件总行数**

**cout << endl;**

**cout << "词法分析结果：" << endl;**

**for( k=1; k<=numline; k++ )**

**{**

**while( i!=content[k].length() )**

**{**

**mode = scan( content[k], k );**

**if( mode!=0 && mode!= -1 )**

**cout << " 所在行数：" << k << " 所在列数：" << i-text.length()+1 << " 单词类别："<< mode << " 单词为：" << text <<endl ;**

**else**

**if( mode==0 )**

**{if( i<content[k].length() ) i++;} //如果为行注释 直接跳到下一行**

**else**

**if( mode==-1 ) //如果为块注释 找回行号**

**k=templine;**

**}**

**i = 0; //一行结束定位到下一行行头**

**}**

**getchar();**

**return 0;**

**}**

**附录二 语法语义四元式生成代码（复制可执行）**

**/\***

**语法+语义+四元式分析程序**

**BY 谢卓函**

**ON June 25th**

**\*/**

**#include <iostream>**

**#include <algorithm>**

**#include <cstring>**

**#include <fstream>**

**#include <string>**

**#include <cstdio>**

**#include <vector>**

**#include <iterator>**

**#include <algorithm>**

**#include <functional>**

**#include <stack>**

**#include <queue>**

**#include <cstdlib>**

**#include <sstream>**

**#define MAXLINE 30**

**using namespace std;**

**//////////////////////////**

**//标识符1，整数2，小数3，字符串4，保留字5，数学运算符6，比较运算符7，逻辑运算符8，分隔符9**

**string key[14]={"if","else","while","do","float","string","begin","end","def","integer","input","output","as","function"}; //保留字**

**string border[5]={ ";" , "{" , "}" , "(" , ")" }; //分隔符 5**

**string arithmetic[9]={"+" , "-" , "\*" , "/" ,"<" , "<=" , "=" , ">" , ">=" }; //运算符 6**

**string text; //记录标识符 1**

**string content[MAXLINE]; //保存从文件读取的内容**

**string result[MAXLINE] = { "t1","t2","t3","t4","t5","t6","t7","t8","t9","t10"};**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**/\*语法语义 各产生式 函数\*/**

**void chengxu();**

**void hanshukuai();**

**void hanshu();**

**void yujukuai();**

**void yuju();**

**void bianliangdingyiyuju();**

**void shujuleixing();**

**void shuruyuju();**

**void shuchuyuju();**

**void fuzhiyuju();**

**void fenzhiyuju();**

**void xunhuanyuju();**

**void biaodashi();**

**void xiang();**

**void yinzi();**

**void buerbiaodashi();**

**void guanxibiaodashi();**

**void guanxi();**

**void error();**

**void Quaternary\_output();**

**void Quaternary\_maker();**

**int i=0,k=1,numline,mode; //i为字符游标,k为行号,numline记录了文件总行数**

**int templine = 0;**

**int errornum = 0;**

**int warningnum = 0;**

**int r = 0, serial = 1;**

**string op,v1,v2,res;**

**struct Token**

**{**

**string value; //值**

**int mode; //类型**

**int row; //行**

**int col; //列**

**}token;**

**struct Symtable //符号表结构**

**{**

**string name;**

**int type;**

**bool value;**

**}S;**

**struct Quaternary //四元式结构**

**{**

**int serial;**

**string op;**

**string v1;**

**string v2;**

**string result;**

**}Q;**

**Quaternary Quaternary\_generater( int serial, string op, string v1, string v2, string result ); //函数声明**

**vector<Symtable> ST;**

**vector<Symtable> :: iterator it;**

**vector<Quaternary> QV;**

**vector<Quaternary> :: iterator it2;**

**stack<string> operator\_stack; //操作数堆栈**

**stack<string> operand\_stack; //操作符堆栈**

**struct compare: binary\_function<Symtable, string,bool> //仿函数和绑定器**

**{**

**bool operator()( Symtable s, string str) const**

**{**

**if (s.name== str)**

**return true;**

**else**

**return false;**

**}**

**};**

**void push\_in\_token( Token token ) //进表操作**

**{**

**S.name = token.value;**

**S.type = -1;**

**S.value = false;**

**ST.push\_back( S );**

**}**

**vector<Symtable> :: iterator find\_token( Token token ) //查表操作**

**{**

**it = find\_if( ST.begin(), ST.end(), bind2nd( compare(),token.value ));**

**return it;**

**}**

**int scan( string s ,int line )**

**{**

**int j, flag =0;**

**text = ""; //赋空text**

**while( content[line][i]==' ' ) //判断空格**

**i++; //是空格跳过**

**if( (content[line][i]>=65 && content[line][i]<=90) || (content[line][i]>=91 && content[line][i]<=122) ) //判断是否为字母或者下划线**

**{**

**text += content[line][i];i++;**

**while( (content[line][i]>=65 && content[line][i]<=90) || (content[line][i]>=91 && content[line][i]<=122) || content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字或者字母或者下划线**

**{**

**text += content[line][i];i++;**

**}**

**for( j=0; j<=13 ; j++ )**

**if( text==key[j] ) //查表判断是否为保留字**

**return 5;**

**if( text=="and" || text=="or" ) //判断是否为逻辑运算符**

**return 8;**

**if( j==14 )**

**return 1; //若查表失败说明为标识符**

**}**

**else**

**{**

**while( content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字**

**{**

**text += content[line][i];i++;flag=1;**

**}**

**if( flag==1 )**

**{**

**if( content[line][i]=='.' )**

**{**

**text += content[line][i];i++;**

**while( content[line][i]>='0' && content[line][i]<='9' ) //判断是否为数字**

**{**

**text += content[line][i];i++;**

**}**

**return 2; //整数**

**}**

**return 3; //小数**

**}**

**}**

**if( content[line][i]== ';' || content[line][i]=='{' || content[line][i]=='}' || content[line][i]=='(' || content[line][i]==')' || content[line][i]==',' ) //判断是否为分隔符**

**{ text = content[line][i];i++; return 9;}**

**if( content[line][i]== '+' || content[line][i]=='-' || content[line][i]=='\*' || content[line][i]=='=' ) //判断是否为运算符**

**{ text = content[line][i]; i++;return 6;}**

**if( content[line][i]=='<' || content[line][i]=='>' )**

**if( content[line][i+1]=='=' )**

**{text = content[line][i];text += content[line][i+1];i=i+2;return 7;}**

**else**

**{ text = content[line][i];i++; return 7;}**

**if( content[line][i]=='“' ) //判断是否为字符串**

**{**

**text += content[line][i];i++;**

**while( content[line][i]!='”' )**

**text += content[line][i];i++;**

**text += content[line][i];i++;**

**return 4;**

**}**

**if( content[line][i]=='/' ) //判断是否为注释**

**if( content[line][i+1]=='/')**

**{i=content[line].length();return 0;} //如果为行注释那么游标指向行末**

**else**

**if( content[line][i+1]=='\*' )**

**{**

**i=i+2;**

**while( content[line][i]!='\*' || content[line][i+1]!='/' )**

**if( i==content[line].length() )**

**{line++;i=0;}**

**else**

**i++;**

**i=i+2;templine = line;return -1; //-1代表块注释**

**}**

**else**

**{ text = content[line][i]; i++;return 6;} //说明是除号**

**//{ text = content[line][i];i++; return 0;}**

**return 0;**

**}**

**Token getToken() //该函数用于取字符**

**{**

**for( ; k<=numline; k++ )**

**{**

**while( i!=content[k].length() )**

**{**

**mode = scan( content[k], k );**

**if( mode!=0 && mode!= -1 )**

**{**

**token.value = text;**

**token.mode = mode;**

**token.row = k;**

**token.col = i-text.length()+1;**

**return token;//cout << " 所在行数：" << k << " 所在列数：" << i-text.length()+1 << " 单词类别："<< mode << " 单词为：" << text <<endl ;**

**}**

**else**

**if( mode==0 )**

**{if( i<content[k].length() ) i++;} //如果为行注释 直接跳到下一行**

**else**

**if( mode==-1 ) //如果为块注释 找回行号**

**k=templine;**

**}**

**i = 0; //一行结束定位到下一行行头**

**}**

**token.value = "EOF";**

**return token;**

**}**

**int main()**

**{**

**cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*编译原理大作业 语法语义四元式分析程序 by 谢卓函\*\*\*\*\*\*\*\*\*\*\*\* "<< endl;**

**string filename;**

**cout << "请输入测试文件名: "<< endl;**

**cin >> filename;**

**ifstream infile( filename,ios::in);**

**//此处在VC上可能编译不过 可改为ifstream infile( "ceshi1.txt",ios::in); 手动输入文件名**

**if( infile )**

**cout << "文件读取成功! "<< endl;**

**else**

**cout << "文件读取失败!" << endl;**

**int hang, mode,k=1; //k用于定位行数**

**cout << endl;**

**cout << "测试数据如下：" << endl;**

**while ( getline(infile,content[k]) )**

**{**

**cout << content[k] << endl;**

**k++;**

**}**

**numline = k--; //numline记录了文件总行数**

**cout << endl;**

**cout << "语法分析结果：" << endl;**

**chengxu();**

**if( warningnum )**

**cout << "warning: " << warningnum;**

**if( errornum )**

**cout << " error: " << errornum << endl;**

**else**

**cout << "编译成功！" << endl;**

**cout << endl;**

**if( !errornum )**

**{**

**cout << "四元式如下：" << endl;**

**Quaternary\_maker();**

**Quaternary\_output();**

**}**

**getchar();**

**return 0;**

**}**

**void error( Token token,char \*msg) //编译错误处理函数**

**{**

**cout << "编译错误："; printf( "%s",msg); cout << endl << "错误位置：第" << token.row << "行，第" << token.col << "列 -> " <<token.value;**

**cout << endl;cout << endl;**

**errornum++;**

**}**

**void warning( Token token,char \*msg) //警告错误处理函数**

**{**

**cout << "警告错误："; printf( "%s",msg); cout << endl << "警告位置：第" << token.row << "行，第" << token.col << "列 -> " <<token.value;**

**cout << endl;cout << endl;**

**warningnum++;**

**}**

**void chengxu()**

**{**

**token = getToken();**

**hanshukuai();**

**}**

**void hanshukuai()**

**{**

**hanshu();**

**while( token.value=="function" )**

**{**

**hanshu();**

**}**

**}**

**void hanshu()**

**{**

**if( token.value=="function" )**

**token = getToken();**

**else**

**error( token, "缺少函数声明标志或函数声明标志不正确" );**

**if( token.mode==1 )**

**token = getToken();**

**else**

**error( token, "函数名不正确" );**

**if( token.value=="(" )**

**token = getToken();**

**else**

**error( token, "函数名后应出现左括号" );**

**if( token.value==")" )**

**token = getToken();**

**else**

**error( token, "函数名后应出现右括号" );**

**yujukuai();**

**if( token.value=="end" )**

**token = getToken();**

**else**

**error( token, "缺少函数结束符end" );**

**if( token.value=="function" )**

**token = getToken();**

**else**

**error( token, "缺少关键字function" );**

**}**

**void yujukuai()**

**{**

**if( token.value=="begin" )**

**token = getToken();**

**else**

**error( token, "缺少关键字begin" );**

**yuju();**

**while( token.value=="def" || token.value=="while" || token.value=="input" || token.value=="output" || token.value=="if" || token.mode ==1 )**

**yuju();**

**if( token.value=="end" )**

**token = getToken();**

**else**

**error( token, "缺少语句块结束符end" );**

**}**

**void yuju()**

**{**

**if( token.value=="def" )**

**bianliangdingyiyuju();**

**if( token.value=="while" )**

**xunhuanyuju();**

**if( token.value=="input" )**

**shuruyuju();**

**if( token.value=="output" )**

**shuchuyuju();**

**if( token.mode==1 )**

**fuzhiyuju();**

**if( token.value=="if" )**

**fenzhiyuju();**

**}**

**void bianliangdingyiyuju()**

**{**

**if( token.value=="def" )**

**token = getToken();**

**else**

**error( token, "缺少关键字def" );**

**if( token.mode==1 )**

**{**

**/////////////////////////////////////////以下是语义分析**

**it = find\_token( token );**

**if( it==ST.end() ) //没有找到**

**push\_in\_token( token ); //进表**

**else**

**warning( token, "变量重复定义" );**

**token = getToken();**

**}**

**else**

**error( token, "标识符错误" );**

**while( token.value=="," )**

**{**

**token = getToken();**

**if( token.mode==1 )**

**{**

**/////////////////////////////////////////以下是语义分析**

**it = find\_token( token );**

**if( it==ST.end() ) //没有找到**

**push\_in\_token( token ); //进表**

**else**

**warning( token, "变量重复定义" );**

**token = getToken();**

**}**

**else**

**error( token, "标识符错误" );**

**}**

**if( token.value=="as" )**

**token = getToken();**

**else**

**error( token, "缺少关键字as" );**

**shujuleixing();**

**if( token.value==";" )**

**token = getToken();**

**else**

**error( token, "缺少分号" );**

**}**

**void shujuleixing()**

**{**

**if( token.value=="integer" || token.value=="float" || token.value=="string" )**

**token = getToken();**

**else**

**warning( token, "数据类型定义错误" );**

**for( it=ST.begin(); it!=ST.end(); it++ )**

**if( (\*it).type==-1 ) //若变量未定义**

**{**

**if( token.value=="integer" )**

**(\*it).type=1; //1为整数，2为小数，3为字符串**

**else**

**if( token.value=="float" )**

**(\*it).type=2;**

**else**

**(\*it).type=3;**

**(\*it).value=true;**

**}**

**}**

**void shuruyuju()**

**{**

**if( token.value=="input" )**

**token = getToken();**

**else**

**error( token, "缺少input关键字" );**

**if( token.mode==1 )**

**{**

**it = find\_token( token ); //语义分析**

**if( it== ST.end() ) //没找到**

**error( token, "变量未定义就使用" );**

**token = getToken();**

**}**

**else**

**error( token, "标识符错误" );**

**while( token.value=="," )**

**{**

**token = getToken();**

**if( token.mode==1 )**

**token = getToken();**

**else**

**error( token, "标识符错误" );**

**}**

**if( token.value==";" )**

**token = getToken();**

**else**

**error( token, "缺少分号" );**

**}**

**void shuchuyuju()**

**{**

**if( token.value == "output" )**

**token = getToken();**

**else**

**error( token, "缺少output关键字" );**

**biaodashi();**

**while( token.value=="," )**

**{**

**token = getToken();**

**biaodashi();**

**}**

**if( token.value==";" )**

**token = getToken();**

**else**

**error( token, "缺少分号" );**

**}**

**void fuzhiyuju()**

**{**

**if( token.mode==1 )**

**{**

**it = find\_token( token ); //语义分析**

**if( it== ST.end() ) //没找到**

**error( token, "变量未定义就使用" );**

**else**

**if( (\*it).value==false )**

**warning( token, "变量未赋值就引用" );**

**operand\_stack.push( token.value );**

**token = getToken();**

**}**

**else**

**error( token, "标识符错误" );**

**if( token.value=="=" )**

**{**

**operator\_stack.push( token.value );**

**token = getToken();**

**}**

**else**

**error( token, "缺少等于号" );**

**biaodashi();**

**if( token.value==";" )**

**{token = getToken();Quaternary\_maker();}**

**else**

**error( token, "缺少分号" );**

**}**

**void fenzhiyuju()**

**{**

**if( token.value=="if" )**

**{**

**token = getToken();**

**//operand\_stack.push( token.value );**

**}**

**else**

**error( token, "缺少if关键字" );**

**buerbiaodashi();**

**yujukuai();**

**if( token.value == "else" )**

**{**

**token = getToken();**

**yujukuai();**

**}**

**}**

**void xunhuanyuju()**

**{**

**if( token.value=="while" )**

**token = getToken();**

**else**

**error( token, "缺少while关键字" );**

**buerbiaodashi();**

**if( token.value=="do" )**

**token = getToken();**

**else**

**error( token, "缺少do关键字" );**

**yujukuai();**

**}**

**////////////////////////////////////////////////////**

**//算数符号优先级：\*/ > +- > =**

**void biaodashi()**

**{**

**xiang();**

**while( token.value=="+" || token.value=="-" )**

**{**

**///////////////////////////////////////////////四元式**

**while( !operator\_stack.empty() && operator\_stack.top()!="=" ) //此处要注意用while循环，if会出错，因为需要不断判断直至栈顶的符号优先级小于需要压栈的符号**

**{**

**op = operator\_stack.top();**

**operator\_stack.pop();**

**v1 = operand\_stack.top();**

**operand\_stack.pop();**

**v2 = operand\_stack.top();**

**operand\_stack.pop();**

**res = result[r];r++;**

**Q = Quaternary\_generater( serial, op, v1, v2, res );serial++;**

**QV.push\_back( Q );**

**operand\_stack.push( res );**

**for( it2=QV.begin(); it2!=QV.end(); it2++ ) //找回真出口**

**if( (\*it2).result=="null" )**

**{**

**stringstream ss;**

**ss<<QV.back().serial;**

**string s=ss.str();**

**(\*it2).result=s;**

**break;**

**}**

**}**

**operator\_stack.push( token.value ); //如果栈空或者准备进栈的符号的优先级大于栈顶的优先级则进栈**

**token = getToken();**

**xiang();**

**}**

**}**

**void xiang()**

**{**

**yinzi();**

**while( token.value=="\*" || token.value=="/" )**

**{**

**operator\_stack.push( token.value );**

**token = getToken();**

**yinzi();**

**}**

**}**

**void yinzi()**

**{**

**if( token.mode==1 || token.mode==2 || token.mode==3 )**

**{**

**if( token.mode==1 )**

**{**

**it = find\_token( token ); //语义分析**

**if( it== ST.end() ) //没找到**

**error( token, "变量未定义就使用" );**

**}**

**operand\_stack.push( token.value );**

**token = getToken();**

**}**

**else**

**{**

**if( token.value == "(" )**

**token = getToken();**

**else**

**error( token, "缺少左括号" );**

**biaodashi();**

**if( token.value == ")" )**

**token = getToken();**

**else**

**error( token, "缺少右括号" );**

**}**

**}**

**void buerbiaodashi()**

**{**

**guanxibiaodashi();**

**while( token.value=="and" || token.value=="or" )**

**{**

**token = getToken();**

**buerbiaodashi();**

**}**

**}**

**void guanxibiaodashi()**

**{**

**biaodashi();**

**guanxi();**

**biaodashi();**

**////////////////////////////////////////**

**//布尔表达式四元式生成**

**op = operator\_stack.top();**

**operator\_stack.pop();**

**v1 = operand\_stack.top();**

**operand\_stack.pop();**

**v2 = operand\_stack.top();**

**operand\_stack.pop();**

**res= "null";**

**Q = Quaternary\_generater( serial, op, v1, v2, res );serial++; //res现在还未知,出口为真**

**QV.push\_back( Q );**

**Q = Quaternary\_generater( serial, "j", "-", "-", res );serial++; //出口为假**

**QV.push\_back( Q );**

**}**

**void guanxi()**

**{**

**if( token.mode==7 )**

**{**

**operator\_stack.push( token.value ); //关系判断符号进栈**

**token = getToken();**

**}**

**else**

**{///////////////////////////////////////////////////关键字规则**

**error( token, "关系运算符错误" );**

**for( i=0; i<=13 ; i++ )**

**if( text==key[i] ) //查表判断是否为保留字**

**break;**

**}**

**}**

**Quaternary Quaternary\_generater( int serial, string op, string v1, string v2, string result ) //四元式生成函数**

**{**

**Q.serial = serial;**

**Q.op = op;**

**Q.v1 = v1;**

**Q.v2 = v2;**

**Q.result = result;**

**return Q;**

**}**

**void Quaternary\_maker()**

**{**

**while( !operand\_stack.empty() && !operator\_stack.empty() )**

**{**

**op = operator\_stack.top();**

**operator\_stack.pop();**

**v1 = operand\_stack.top();**

**operand\_stack.pop();**

**v2 = operand\_stack.top();**

**operand\_stack.pop();**

**res = result[r];r++;**

**Q = Quaternary\_generater( serial, op, v1, v2, res );serial++;**

**QV.push\_back( Q );**

**////////////////**

**for( it2=QV.begin(); it2!=QV.end(); it2++ ) //找回真出口**

**if( (\*it2).result=="null" )**

**{**

**stringstream ss;**

**ss<<QV.back().serial;**

**string s=ss.str();**

**(\*it2).result=s;**

**break;**

**}**

**///////////////**

**operand\_stack.push( res );**

**}**

**}**

**void Quaternary\_output()**

**{**

**while( !QV.empty() )**

**{**

**cout << QV.front().serial << " ( " << QV.front().op << " , " << QV.front().v2 << " , " << QV.front().v1 << " , " << QV.front().result << " ) " << endl;**

**it2 = QV.begin();**

**QV.erase( it2 );**

**}**

**}**