基于c0文法编译器V1.0

10011603 2016302731 冯宇

**一.主要功能**

在词法分析器的基础上添加语法分析器，将读取到的程序转换成四元式

语法分析器实现的主要语法有 ：

＜程序＞ ::= ［＜常量说明＞]［＜变量说明＞]{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞ 函数部分省略处理

＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}

＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}| char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

＜主函数＞ ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

＜语句列＞ ::=｛＜语句＞｝

<语句> ::= ＜条件语句＞｜＜循环语句＞| ‘{’＜语句列＞‘}’|＜赋值语句＞;

＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞

＜条件语句＞ ::= if ‘(’＜条件＞‘)’＜语句＞［else＜语句＞］

＜循环语句＞ ::= for‘(’＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞‘)’＜语句＞

＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}

＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

＜因子＞ ::= ＜标识符＞｜＜标识符＞‘[’＜表达式＞‘]’｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞|‘(’＜表达式＞‘)’

＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞

**二.实现代码**

**重要变量：**

skipFlag表示是否能够跳过语句，一般为FALSE，但是在发生错误的时候会置为TRUE，跳过一些语句后恢复为FALSE

程序中出现大量这样的结构：

if (sy == identi) {//如果sy符合要求

进行相关处理

insymbol();//读取下一个词

}

else if (!skipFlag) {//不符合要求的时候

error(IdLost); // Identifier lost

symbol nexts[] = { comma, semicolon, rbrace };//一直跳过直到遇到next[]中的符号

int length = sizeof(nexts) / sizeof(symbol);

skipUntil(nexts,length);//跳过语句

}

**关键函数：**

/\*

读取下一个数，并且判断是正数还是负数，更新sy

get the next integer and update sy and infixstring = (str)inum

输入：infixString

输出: infixString（=（str）inum）(sy)

\*/

void integer(string &infixString)

/\* ＜程序＞ ::= ［＜常量说明＞]［＜变量说明＞]＜主函数＞

\*/

void program()

// ＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}

void constState()

/\* ＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}

| char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞} \*/

void constDef()

// ＜主函数＞ ::= int main‘(’‘)’ ‘{’＜复合语句＞‘}’

void mainDef()

// ＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

void complexState()

// ＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

void varState()

// ＜变量定义＞ ::= ＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’}

//一行最多定义两个变量（没用循环）

void varDef(){

if (sy == intsy || sy == charsy)定义vartype

if (sy == identi)定义IDName

if (sy == lbrack){读到左中括号进行数组的读取

insertTable(vars, varType, idName, inum, level, 0);//记录到idtable包括变量地址，长 度，等级

char CH = inum;

insertInfix("VAR", charToString(CH), varType == ints ? "int" : "char", idName);

//记录到infixtable中，以四元式的情况存在

if (sy == rbrack)

}

if (isDefinable(idName)) {

insertTable(vars, varType, idName, 0, level, 0);

insertInfix("VAR", " ", varType == ints ? "int" : "char", idName);

}

while(sy==comma)//如果是逗号循环读取

}

/\*

<语句> :: = ＜条件语句＞｜＜循环语句＞ | ‘{ ’＜语句列＞‘ }’ | ＜赋值语句＞;＜返回语句＞; ）｜＜空＞;

\*/

void statement()

利用switch语句对读取到的的语句进行选择，进入不同的处理函数入口

// ＜语句列＞ ::=｛＜语句＞｝

void stateList()

// ＜条件语句＞ ::= if ‘(’＜条件＞‘)’＜语句＞［else＜语句＞］

void ifState(){

if (sy == ifsy)继续读取

if (sy == lparent) 继续读取

judgement(judgeResult) <条件>

firstLabel = "LABEL" + intToString(labelIndex);

labelIndex++;

insertInfix("BEQ", judgeResult, "0", firstLabel); // If judgement = 0 then jump firstLabel

if (sy == rparent)继续读取

statement();//<语句>

secondLabel = "LABEL" + intToString(labelIndex);

labelIndex++;

insertInfix("JMP", " ", " ", secondLabel); // Jump the else (If not, jump the next label)

insertInfix("LABEL", " ", " ", firstLabel); // Where if is false go

labelIndex++;

if (sy == elsesy)

insertInfix("LABEL", " ", " ", secondLabel); // Where if is true go

}

// ＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞

/\*

输入infixString

输出infixString（创建临时变量保存BOOL值）

\*/

void judgement(string &infixString)

// ＜循环语句＞ ::= for‘(’＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞‘)’＜语句＞

void loopState(){

string judgeLabel = "";//判断条件LABEL

string increaseLabel = "";//步长LABEL

string bodyLabel = "";//循环体LABEL

string endLabel = "";

if (sy == forsy)

if (sy == lparent)

if (sy == identi)判断变量是否已声明

if (sy == becomes)

expression(expressionString)//表达式

insertInfix("ASSIGN", " ", expressionString, idNameString);

bodyLabel = labelStr + intToString(labelIndex);//循环体的LABEL bodyLabel

labelIndex++;

}

if (sy == semicolon)

//<条件>

judgeLabel = labelStr + intToString(labelIndex);

labelIndex++;

insertInfix("LABEL", " ", " ", judgeLabel);//条件语句的（i<2）LABEL

insertInfix("BNE", judgeResult, "0", bodyLabel); // Jump to body If judgeResult != 0(为真) then jump bodyLabel

endLabel = labelStr + intToString(labelIndex);

labelIndex++;

insertInfix("BEQ", judgeResult, "0", endLabel); // Jump to end If judgeResult = 0(为假) then jump endLabel

//;

if (sy == semicolon)

//<标志符>

increaseLabel = labelStr + intToString(labelIndex);

++labelIndex;

insertInfix("LABEL", " ", " ", increaseLabel);//步长部分的label（i=i+1）

insertInfix("LABEL", " ", " ", bodyLabel);

**三.测试结果**

|  |  |
| --- | --- |
| Input code（测试if语句） | 四元式 |
| int main(){  int x,y,z;  x=1+6;  y=2;  z=3;  if(x>0)x=y+1;  else x=x\*2;  return;  } | int main()  int x  int y  int z  z = 3  y = 2  x = 1 + 6  #t1 = x > 0  goto LABEL0, if #t1 == 0  x = y + 1  jmp LABEL1  LABEL0  x0 = x  x = x0 \* 2  LABEL1  return |
| Input code（测试for语句） | 四元式 |
| int main(){  int x,y,z;  x=1+6;  y=2;  z=3;  for(x=0;y<4;y=y+1)  z=z\*2;  return;  } | int main()  int x  int y  int z  x = 0  z = 3  y = 2  #t0 = 1 + 6  LABEL1  #t1 = y < 4  goto LABEL0, if #t1 != 0  goto LABEL2, if #t1 == 0  LABEL3  y0 = y  y = y0 + 1  jmp LABEL1  LABEL0  z0 = z  z = z0 \* 2  jmp LABEL3  LABEL2  return |

**四.Debug以及C++知识点**

**1.全局变量声明**

<https://blog.csdn.net/ouyangxin95/article/details/51273460>

一般的全局变量两种定义方式，一种是在头文件\*.h文件中直接定义，在包含该头文件的.cpp文件中可直接使用，第二种是使用extern关键字，extern用在变量声明中常常有这样一个作用，你在\*.c文件中声明了一个全局的变量，这个全局的变量如果要被引用，就放在\*.h中并用extern来声明。需要注意的是这种方法定义的量在头文件中只是声明，在.cpp文件中才是定义。

在实验中我便犯了这样的额错误，在\*.cpp文件中没有定义变量，只是在头文件中声明，导致程序编译一直没有通过

**2.重编译错误**

在包含多个头文件的程序中最重要的便是避免重编译错误

假设你的工程里面有4个文件，分别是a.cpp,b.h,c.h,d.h。

a.cpp的头部是：

#include "b.h "

#include "c.h "

b.h和c.h的头部都是:

#include "d.h "

而d.h里面有class D的定义。

这样一来，编译器编译a.cpp的时候，先根据#include "b.h "去编译b.h这个问题，再根据b.h里面的#include "d.h "，去编译d.h的这个文件，这样就把d.h里面的class D编译了；

然后再根据a.cpp的第二句#include "c.h "，去编译c.h，最终还是会找到的d.h里面的class D，但是class D之前已经编译过了，所以就会报重定义错误。

加上ifndef/define/endif，就可以防止这种重定义错误。

例如：

A.h 里

#ifndef A\_H

#define A\_H

...

#endif

B.h 和 C.h都include "A.h "

D.h里

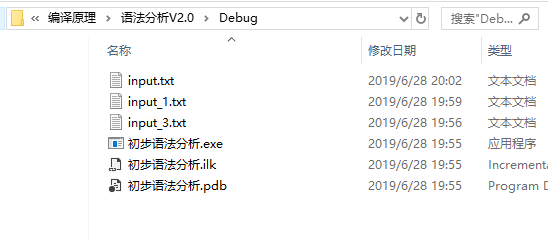
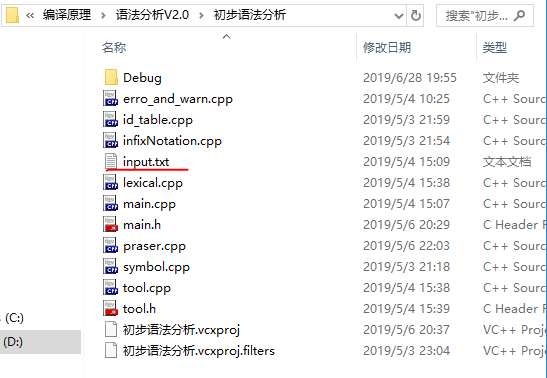
include "B.h "

include "C.h "

在预编译的过程中，执行到include "C.h "时会因为在上一句的时候已经定义了A\_H这个宏，所以此时的ifndef条件不满足，也就不会再一次包含A.h，起到了防止重复引用头文件的效果。

**3.文件调试**

由于我使用文本指针打开文本文件的时候，并没有添加路径，所以程序默认打开的是当前文件夹下的文件。而在VS2017中，生成的程序和调试部分的程序并不是在一个位置，所以在调试代码的时候也需要添加一个相同的文本文件

否则，在DEBUG的时候程序会一直提示未找到文本文件，文本指针打开失败。

**4.全局变量初始化**

全局初始化变量不能放在局部函数里，多处定义同一全局变量可能会导致引用不同导致出错。

**5.函数重定义**

不要在头文件里面定义函数，会出现重定义的情况