

**课 程 实验 报 告**

**题目： C语言编译器的设计与实现**

**课程名称： 编译技术**

**专业班级： 软件工程1504班**

**学 号： U201517070**

**姓 名： 刘博谦**

**指导教师： 徐丽萍**

**报告日期： 2017.01.24**

**软件学院**

目录

[1选题背景 1](#_Toc504675302)

[2系统关键定义 2](#_Toc504675303)

[2.1单词文法描述 2](#_Toc504675304)

[2.2语句文法描述 3](#_Toc504675305)

[2.3 符号表结构描述 3](#_Toc504675306)

[2.4 错误类型码描述 4](#_Toc504675307)

[2.5中间代码描述 4](#_Toc504675308)

[2.6．汇编代码的描述 6](#_Toc504675309)

[3系统设计与实现 10](#_Toc504675310)

[3.1编译程序符号表结构 10](#_Toc504675311)

[3.2编译程序报错功能 10](#_Toc504675312)

[3.3词法分析器设计与实现 13](#_Toc504675313)

[3.4语法语义分析器设计与实现 17](#_Toc504675314)

[3.5中间代码生成功能 19](#_Toc504675315)

[3.6汇编代码生成功能 20](#_Toc504675316)

[4系统测试与评价 23](#_Toc504675317)

[4.1测试用例 23](#_Toc504675318)

[4.2正确性测试 24](#_Toc504675319)

[4.3报错功能测试 25](#_Toc504675320)

[4.4系统的优点 25](#_Toc504675321)

[4.5系统的缺点 26](#_Toc504675322)

[5实验小结或体会 27](#_Toc504675323)

[**参考文献** 28](#_Toc504675324)

[**附件：源代码** 29](#_Toc504675325)

# 1选题背景

本次课程设计是构造一个高级语言的子集的编译器，目标代码可以是汇编语言业也可以是其他形式的机器语言。按照任务书，实现的方案可以有很多种选择。

可以根据自己对编程语言的定义选择实现语言的特定功能。

编译器的语法和词法分析采用教材相关章节的内容进行相应修改，重点在语义分析、符号表结构设计、中间代码、目标代码存储结构设计等阶段的实现。

实验的任务主要是通过对简单编译器的完整实现，加深课程中关键算法的理解，提高自己对系统软件编写的兴趣。

# 2系统关键定义

## 2.1单词文法描述

组织输入、扫描、分析、输出；

接收字符串形式的源程序，按照源程序输入的次序依次扫描源程序，在扫描的同时根据语言的词法规则识别出具有独立意义的单词，并产生与源程序等价的属性字（Token）流 .

|  |  |  |
| --- | --- | --- |
| 单词符号 | 种别编码 | 单词值 |
| main | 1 |  |
| int | 2 |  |
| float | 3 |  |
| double | 4 |  |
| char | 5 |  |
| if | 6 |  |
| else | 7 |  |
| do | 8 |  |
| while | 9 |  |
| l(l|d)\* | 10 | 内部字符串 |
| ( +|-|ε ) dd\*(.dd\* | ε)( e ( +|-|ε ) dd\*|ε) | 20 | 二进制数值表示 |
| = | 21 |  |
| + | 22 |  |
| - | 23 |  |
| \* | 24 |  |
| / | 25 |  |
| ( | 26 |  |
| ) | 27 |  |
| { | 28 |  |
| } | 29 |  |
| , | 30 |  |
| ; | 31 |  |
| > | 32 |  |
| >= | 33 |  |
| < | 34 |  |
| <= | 35 |  |
| == | 36 |  |
| != | 37 |  |
| # | 0 |  |

## 2.2语句文法描述

在词法分析的基础上将单词序列组合成各类语法短语，如“程序”，“语句”，“[表达式](https://baike.baidu.com/item/%E8%A1%A8%E8%BE%BE%E5%BC%8F)”等等.语法分析程序判断源程序在结构上是否正确.源程序的结构由上下文无关文法描述.

<程序> ::= main()<语句块>

<语句块> ::= ‘{‘<语句串>’}’

<语句串>::=<语句>{;<语句>};

<语句>::=<赋值语句>|<条件语句>|<循环语句>

<赋值语句>::=ID=<表达式>

<条件语句>::=if<条件><语句块>[else <语句块>]

<循环语句>::=do <语句块>while <条件>

<条件>::=<表达式><关系运算符><表达式>

<表达式> ::= <项>{ +<项>|-<项>}

<项> ::= <因子>{\*<因子>|/<因子>}

<因子> ::=ID|num|(<表达式>)

num::= ( +|-|ε ) 数字数字\*(.数字数字\* | ε)( e ( +|-|ε ) 数字数字\*|ε)

ID::=字母(字母|数字)\*

字母::=a|b|c…|z|A|B|C…|Z

数字::=0|1|2…|9

<关系运算符> ::= <|<=|>|>=|==|!=

## 2.3 符号表结构描述

符号表结构采用结构体的方式。

typedef struct Variable

{

string var; //变量

string value; //初始化值

}Variable;

typedef struct IDwords

{

int id; //标志

string word; //单词

}IDwords;

vector<Variable> var\_table;

struct Variable中，变量名和初始化的值，struct IDwords包含符号表的标记，和标识符，两个结构体在实现过程连接。并在语法分析过程完成符号表的初始化。

## 2.4 错误类型码描述

由于时间比较仓促，没有具体的错误类型码和错误位置描述。当编译文件遇到错误时，编译工作将会停止，然后根据扫描到的C语言语句，输出具体的错误信息，主要有以下几种错误信息的输出：

"源文件名不能为空"

"error:错误的字符串数据："

"error:不合法的标识符："

"error:无法打开源文件"

"error:错误的if语句：缺少'('"

"error:错误的if条件语句：缺少')'"

"error:缺少main"

"error:main函数缺少'{'"

"无法创建汇编文件"

"else 没有找到匹配的 if"

## 2.5中间代码描述

四元式用如下结构体保存：

typedef struct Target

{

string dsf; //结果

string op; //操作

string dst; //目的操作数

string dsc; //源操作数

string mark; //标志

string step; //跳转位置

}Target;

四元式的添加函数：

void add\_target\_code(string dsf, string op, string dst, string dsc, string mark, string step)

{

Target tmp;

tmp.dsf = dsf;

tmp.op = op;

tmp.dst = dst;

tmp.dsc = dsc;

tmp.mark = mark;

tmp.step = step;

target\_code.push\_back(tmp);//加入末尾

}

语义分析的调用过程，完成语法树的生成和分析，得到四元式。

//语义分析

void syntax\_analysis(vector<IDwords> &AnalysisResults)

{

vector<IDwords>::iterator it = AnalysisResults.begin();

if (it->word != "main")

{

cout << "error:缺少main" << endl;

exit(-1);

}

it = it + 3; //跳过“（）”

if (it->word != "{")

{

cout << "error:main函数缺少'{'" << endl;

exit(-1);

}

it++;

//获取变量声明

add\_var\_table(it);

//获取代码段的操作

while (it != AnalysisResults.end())

{

//遇到printf

if (it->word == "printf")

{

printf\_analysis(it);//添加到代码

}

// if 语句

else if (it->word == "if")

{

if\_analysis(it);

}

else if (it->word == "}")

break;

//表达式分析

else

{

expression(it); //表达式分析

}

it++;

}

}

## 2.6．汇编代码的描述

目前编译器的功能还比较简单，只能够编译简单的加减乘除、赋值、printf语句、if语句，各自对应的汇编代码转换如下。

加减运算：

//加减法转换

void addsub\_asm(ofstream &out,string dsf,string op,string dst,string dsc)

{

out<<" mov BL,"<<dst<<endl;

if(op == "+")

out<<" add BL,"<<dsc<<endl;

else

out<<" sub BL,"<<dsc<<endl;

out<<" mov "<<dsf<<",BL"<<endl;

}

//乘除运算

//乘法

void mul\_asm(ofstream &out,string dsf,string dst,string dsc)

{

out<<" mov AL,"<<dst<<endl;

out<<" mov BH,"<<dsc<<endl;

out<<" mul BH"<<endl;

out<<" mov BL,1"<<endl;

out<<" div BL"<<endl;

out<<" mov "<<dsf<<",AL"<<endl;

}

//除法

void div\_asm(ofstream &out,string dsf,string dst,string dsc)

{

out<<" mov AL,"<<dst<<endl;

out<<" CBW"<<endl;

out<<" mov BL,"<<dsc<<endl;

out<<" div BL"<<endl;

out<<" mov "<<dsf<<",AL"<<endl;

}

赋值运算：

//赋值运算

void sign\_asm(ofstream &out,string dsf,string dst)

{

out<<" mov BL,"<<dst<<endl;

out<<" mov "<<dsf<<",BL"<<endl;

}

Printf语句识别：

//输出转换

void print\_asm(ofstream &out,string dsf,string mark)

{

//以字符格式输出

if(mark=="%c")

{

out<<" mov DL,"<<dsf<<endl;

out<<" mov AH,02H"<<endl;

out<<" int 21H"<<endl;

}

//以整数格式输出

else if(mark=="%d")

{

out<<" mov AL,"<<dsf<<endl;

out<<" CBW"<<endl;

out<<" mov BL,10"<<endl;

out<<" DIV BL"<<endl;

out<<" mov BH,AH"<<endl;

out<<" add BH,30H"<<endl;

out<<" add AL,30H"<<endl;

out<<" CMP AL,48"<<endl;

//确定十位是否是0

lab = lab + 2;

string step2 = "step" + char\_to\_str(lab);

out<<" JE "<<step2<<endl;

string step1 = "step" + char\_to\_str(lab-1);

out<<" "<<step1<<":"<<endl;

out<<" mov DL,AL"<<endl;

out<<" mov AH,2"<<endl;

out<<" int 21H"<<endl;

//输出个位

out<<" "<<step2<<":"<<endl;

out<<" mov DL,BH"<<endl;

out<<" mov AH,2"<<endl;

out<<" int 21H"<<endl;

}

//字符串输出

else

{

out<<" LEA DX,"<<mark<<endl;

out<<" mov AH,09"<<endl;

out<<" int 21H"<<endl;

}

}

If语句：

//if语句转换

void if\_asm(ofstream &out,string dst,string dsc,string mark,string step)

{

out<<" mov AL,"<<dst<<endl;

out<<" CMP AL,"<<dsc<<endl;

if(mark == ">")

out<<" JG "<<step<<endl;

else if(mark == "<")

out<<" JL "<<step<<endl;

else

{

cout<<"暂不支持其他条件判断"<<endl;

exit(-1);

}

}

# 3系统设计与实现

## 3.1编译程序符号表结构

符号表结构采用结构体的方式。

typedef struct Variable

{

string var; //变量

string value; //初始化值

}Variable;

typedef struct IDwords

{

int id; //标志

string word; //单词

}IDwords;

vector<Variable> var\_table;

struct Variable中，变量名和初始化的值，struct IDwords包含符号表的标记，和标识符，两个结构体在实现过程连接。并在语法分析过程完成符号表的初始化。

## 3.2编译程序报错功能

//单词判断类型报错

//判断单词类型

int word\_token(string s)

{

int size = s.size();

//字符数据

if(s[0]=='\'')

{

if(s[size-1] == '\'')

return K\_CHAR;

else

{

cout<<"error:错误的字符串数据："<<s<<endl;

exit(-1);

}

}

//字符串数据

else if(s[0]=='\"')

{

if(s[size-1]=='\"')

return K\_STRING;

else

{

cout<<"error:错误的字符串数据："<<s<<endl;

exit(-1);

}

}

//整数

else if(isdigit(s[0]))

{

for(int i=1;i<size;i++)

{

if(!isdigit(s[i]))

{

cout<<"error:不合法的标识符："<<s<<endl;

exit(-1);

}

}

return K\_DIGIT;

}

else

{

for(int i=0;i<size;i++)

{

if(!isalnum(s[i]) && s[i]!='\_')

{

cout<<"error:不合法的标识符："<<s<<endl;

exit(-1);

}

}

//数据类型

if(s=="int" || s=="char")

return K\_TYPE;

//关键字

else if(s=="if" || s=="else" || s=="printf" || s=="main")

return K\_KEYWORDS;

//自定义标识符

else

return K\_IDENTIFIER;

}

}

//词法分析部分报错

char ch;

ifstream rfile(source.c\_str());

if(!rfile.is\_open())

{

cout<<"error:无法打开源文件"<<endl;

exit(-1);

}

if(ch == '\"')

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

break;

}

else

{

cout<<"error:不合法的标识符："+temp<<endl;

exit(-1);

}

//if语句分析报错

if(it->word != "(")

{

cout<<"error:错误的if语句：缺少'('"<<endl;

exit(-1);

}

it++;

dst = it->word;

it++;

mark = it->word;

it++;

dsc = it->word;

op = "if";

add\_target\_code(" ",op,dst,dsc,mark,"step"+char\_to\_str(lab+1));

it++;

if(it->word != ")")

{

cout<<"error:错误的if条件语句：缺少')'"<<endl;

exit(-1);

}

//语法分析部分报错

if(it->word != "main")

{

cout<<"error:缺少main"<<endl;

exit(-1);

}

it = it+3; //跳过“（）”

if(it->word != "{")

{

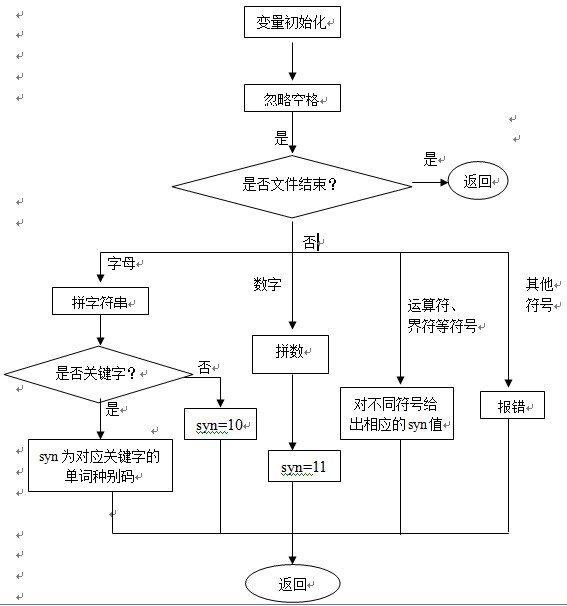
cout<<"error:main函数缺少'{'"<<endl;

exit(-1);

}

it++;

## 3.3词法分析器设计与实现



//词法分析

void cifa\_analysis(string source,vector<IDwords> &AnalysisResults)

{

char ch;

ifstream rfile(source.c\_str());

if(!rfile.is\_open())

{

cout<<"error:无法打开源文件"<<endl;

exit(-1);

}

rfile>>noskipws; //不过滤空格

while(rfile>>ch)

{

int state=0; //判断状态

string temp(""); //字符串缓存

char try\_ch; //探测前面的字符

switch(state)

{

case 0:

if(ch=='/') //可能是注释

{

rfile>>try\_ch;

if(try\_ch=='/')

{

while(rfile>>try\_ch)

{

if(try\_ch=='\n')

break; //这是一行注释

}

break;

}

else if(try\_ch=='\*')

{

while(rfile>>try\_ch)

{

if(try\_ch=='\*')

{

rfile>>try\_ch;

if(try\_ch=='/')

break; //这是多行注释

}

}

break;

}

else

{

add\_keywords(AnalysisResults,K\_OPERATOR,char\_to\_str(ch));

ch = try\_ch; //继续状态1

}

}

case 1:

if(is\_operator(ch)) //判断操作符

{

add\_keywords(AnalysisResults,K\_OPERATOR,char\_to\_str(ch));

break;

}

case 2:

if(is\_bracket(ch)) //大括号、小括号

{

add\_keywords(AnalysisResults,K\_BRACKET,char\_to\_str(ch));

break;

}

case 3:

if(is\_blank(ch)) //空白符

break;

case 4:

if(ch=='#') //跳过预处理

{

while(rfile>>ch)

{

if(is\_blank(ch))

{

break;

}

}

break;

}

default://判断单词类型

temp = temp + char\_to\_str(ch);

while(rfile>>try\_ch)

{

if(try\_ch == '\"')

{

temp = temp + char\_to\_str(try\_ch);

if(ch == '\"')

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

break;

}

else

{

cout<<"error:不合法的标识符："+temp<<endl;

exit(-1);

}

}

else if(is\_blank(try\_ch) )

{

if(ch != '\'' && ch != '\"')

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

break;

}

else

temp = temp + char\_to\_str(try\_ch);

}

else if(is\_operator(try\_ch) )

{

if(ch !='\'' && ch != '\"' )

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

add\_keywords(AnalysisResults,K\_OPERATOR,char\_to\_str(try\_ch));

break;

}

else

temp = temp + char\_to\_str(try\_ch);

}

else if(is\_bracket(try\_ch))

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

add\_keywords(AnalysisResults,K\_BRACKET,char\_to\_str(try\_ch));

break;

}

else

temp = temp + char\_to\_str(try\_ch);

}

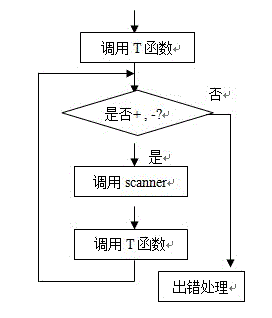
}

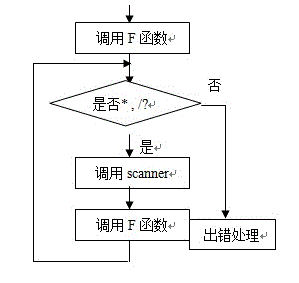
}

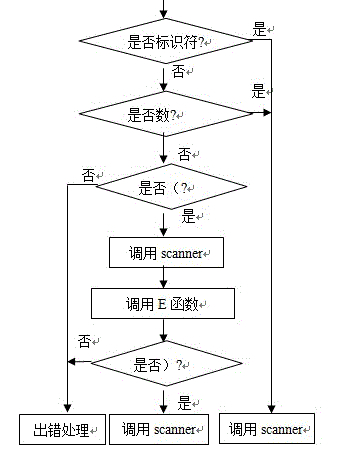
rfile.close();

}

## 3.4语法语义分析器设计与实现







//语法分析

void yufa\_analysis(vector<IDwords> &AnalysisResults)

{

vector<IDwords>::iterator it=AnalysisResults.begin();

if(it->word != "main")

{

cout<<"error:缺少main"<<endl;

exit(-1);

}

it = it+3; //跳过“（）”

if(it->word != "{")

{

cout<<"error:main函数缺少'{'"<<endl;

exit(-1);

}

it++;

//获取变量声明

add\_var\_table(it);

//获取代码段的操作

while(it != AnalysisResults.end())

{

//遇到printf

if(it->word == "printf")

{

printf\_analysis(it);

}

// if 语句

else if(it->word == "if")

{

if\_analysis(it);

}

else if(it->word == "}")

break;

//表达式分析

else

{

expression(it); //表达式分析

}

it++;

}

}

## 3.5中间代码生成功能

相关核心函数如下：

四元式用如下结构体保存：

typedef struct Target

{

string dsf; //结果

string op; //操作

string dst; //目的操作数

string dsc; //源操作数

string mark; //标志

string step; //跳转位置

}Target;

四元式的添加函数：

void add\_target\_code(string dsf, string op, string dst, string dsc, string mark, string step)

{

Target tmp;

tmp.dsf = dsf;

tmp.op = op;

tmp.dst = dst;

tmp.dsc = dsc;

tmp.mark = mark;

tmp.step = step;

target\_code.push\_back(tmp);//加入末尾

}

//输出语法分析结果

void print\_syntax()

{

vector<Variable>::iterator it;

cout<<"变量声明及初始化"<<endl;

for(it = var\_table.begin();it != var\_table.end();it++)

{

cout<<it->var<<" "<<it->value<<endl;

}

vector<Target>::iterator it2;

cout<<"中间代码"<<endl;

for(it2 = target\_code.begin();it2 != target\_code.end();it2++)

{

cout<<it2->dsf<<" "<<it2->op<<" "<<it2->dst<<" "<<it2->dsc<<" "<<it2->mark<<" "<<it2->step<<endl;

}

}

## 3.6汇编代码生成功能

//生成汇编文件

void create\_asm(string file)

{

//变量声明

ofstream wfile(file.c\_str());

if(!wfile.is\_open())

cout<<"无法创建汇编文件"<<endl;

vector<Variable>::iterator it\_var;

wfile<<"ASSUME CS:codesg,DS:datasg"<<endl;

//数据段

wfile<<"datasg segment"<<endl;

for(it\_var=var\_table.begin();it\_var!=var\_table.end();it\_var++)

{

wfile<<" "<<it\_var->var<<" DB ";

if(it\_var->value != "")

wfile<<it\_var->value<<endl;

else

wfile<<"\'?\'"<<endl;

}

wfile<<"datasg ends"<<endl;

//代码段

wfile<<"codesg segment"<<endl;

wfile<<" start:"<<endl;

wfile<<" mov AX,datasg"<<endl;

wfile<<" mov DS,AX"<<endl;

vector<Target>::iterator it;

Target tmp;

for(it = target\_code.begin();it != target\_code.end();it++)

{

//加减法转化

if(it->op == "+" || it->op=="-")

addsub\_asm(wfile,it->dsf,it->op,it->dst,it->dsc);

//乘法转换

else if(it->op == "\*")

mul\_asm(wfile,it->dsf,it->dst,it->dsc);

//除法转换

else if(it->op == "/")

div\_asm(wfile,it->dsf,it->dst,it->dsc);

//赋值运算

else if(it->op == "=")

sign\_asm(wfile,it->dsf,it->dst);

//输出操作

else if(it->op == "p")

print\_asm(wfile,it->dsf,it->mark);

//if语法分析

else if(it->op == "if")

{

if\_asm(wfile,it->dst,it->dsc,it->mark,it->step);

}

else if(it->op == "else")

{

cout<<"else 没有找到匹配的 if"<<endl;

exit(-1);

}

//跳转语句

else if(it->op == "jmp")

{

wfile<<" JMP "<<it->step<<endl;

}

//跳转语句段标识

else if(it->op == "pstep")

{

wfile<<" "<<it->step<<":"<<endl;

}

//其他

else

{

cout<<"编译器暂不支持该语法操作"<<endl;

exit(-1);

}

}

//代码段结束

wfile<<" mov ax,4C00H"<<endl;

wfile<<" int 21H"<<endl;

wfile<<"codesg ends"<<endl;

wfile<<" end start"<<endl;

wfile.close();

}

# 4系统测试与评价

**4.1测试用例**

测试用例1：

main(){

int x,y,z;

x=2;

y=5;

z=6;

y=x\*y;

z=z+y;

z=z-2;

z=z/2;

printf("The result of is %d",z);

/\*if(2<1){

printf("2 is smaller than 1");

}

else{

printf("1 is smaller than 2");

}\*/

//printf("Hello World!");

//我是注释

//我想试试编译器能不能忽略掉我

//如果你看到我，说明编译器的编写失败了

return;

}

测试用例2：

main(){

int b;

b=2;

if(b<1)

{

printf("2 is smaller than 1");

}

printf("2 is smaller than 1");

else

{

printf("1 is smaller than 2");

}

//printf("Hello World!");

//我是注释

//我想试试编译器能不能忽略掉我

//如果你看到我，说明编译器的编写失败了

return;

}

测试用例3：

main({

int b;

b=2;

printf("2 is smaller than 1");

else

{

printf("1 is smaller than 2");

}

//printf("Hello World!");

//我是注释

//我想试试编译器能不能忽略掉我

//如果你看到我，说明编译器的编写失败了

return;

}

**4.2正确性测试**

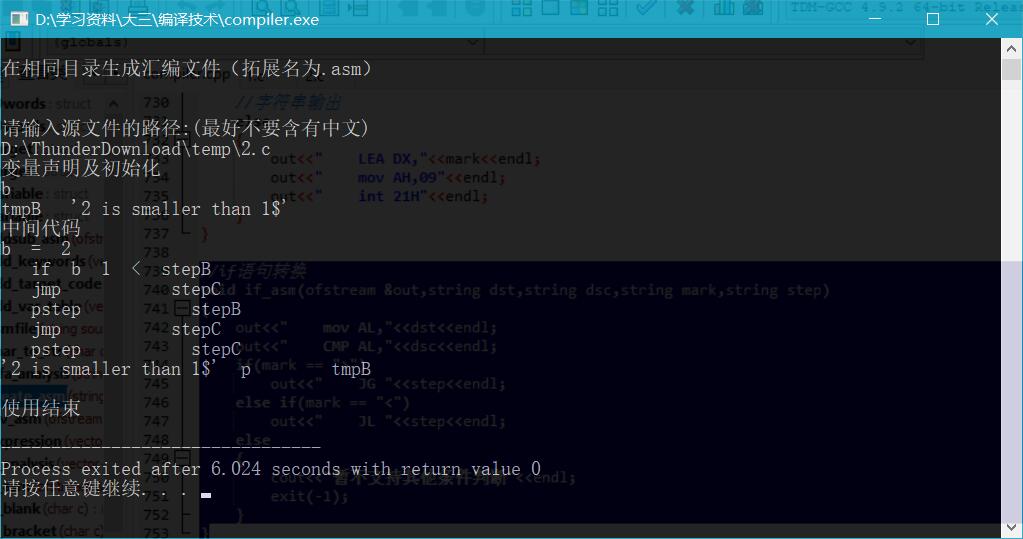
测试用例1运行结果：



结果分析：

加减乘除功能正常，printf语句功能正常，忽略注释功能正常。

测试用例2运行结果：

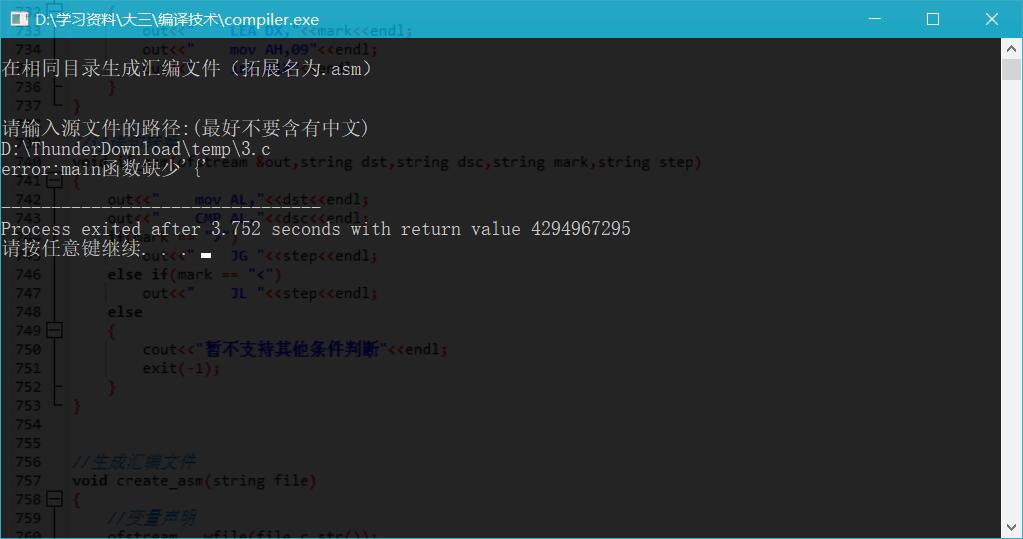


结果分析：

大部分结果同测试用例1，if/else/then语句功能正常，但是这次printf语句嵌套在if语句中，没能成功执行。

**4.3报错功能测试**

测试用例3运行结果：



**4.4系统的优点**

识别用c语言编写的高级语言文件，并能够进行简单的加减乘除和赋值操作，语句输出，if、then、else条件语句执行的功能，还有一点小功能就是可以加注释，这个编译器会自动忽略c文件里的一行和多行注释。依次对c文件进行词法分析、语法分析、语义分析、中间代码生成后，此编译器能够在c文件的相同目录，生成一个asm汇编文件，经过验证，只要运行时不报错，生成的asm一般可以执行。

**4.5系统的缺点**

这个编译器还存在着一些问题，虽然不会莫名其妙的停止运行，但是当printf语句嵌套在if/then/else语句中时，printf会被忽略分析，即无法被翻译成汇编语句，但是printf语句在其他地方是没有问题的。我一直在尝试解决这个问题，但是很遗憾没能成功解决。还有就是只能生成asm文件，无法直接生成可执行程序，还需要用masm进行进一步加工。

# 5实验小结或体会

刚开始做词法分析和语法分析的时候，因为有书本和老师的指导，其实感觉都还好，不是特别难，但是当做到符号表真的是挺懵逼的，终于查找了许多网络资料终于给搞出来了，但是后面的语义分析和中间代码是真的要命。。生成中间代码和汇编代码更是让人头大。最后用了很暴力的手段实现汇编代码的转换，Yacc那个东西实在是用不来。

这次课设，让我对编译器的实现原理和主要原理步骤，有了进一步的认识，巩固了自身的知识，对算法和编译的知识认识更加深刻，感谢老师辛勤的指导！

**参考文献**

[1] 吕映芝等. 编译原理(第二版). 北京：清华大学出版社，2005

[2] 胡伦俊等. 编译原理(第二版). 北京：电子工业出版社，2005

[3] 王元珍等. 80X86汇编语言程序设计. 武汉：华中科技大学出版社,2005

[4] 王雷等. 编译原理课程设计. 北京：机械工业出版社，2005

[5] 曹计昌等. C语言程序设计. 北京：科学出版社，2008

**附件：源代码**

#include<iostream>

#include<fstream>

#include<string>

#include<cctype>

#include<vector>

#include<stack>

#include<stdlib.h>

#define K\_DIGIT 3 //整数

#define K\_CHAR 4 //字符

#define K\_STRING 5 //字符串

#define K\_TYPE 6 //数据类型

#define K\_KEYWORDS 7 //关键字

#define K\_OPERATOR 8 //运算符

#define K\_IDENTIFIER 9 //标识符

#define K\_BRACKET 10 //括号

using namespace std;

//存储分词类型

typedef struct IDwords

{

int id; //标志

string word; //单词

}IDwords;

typedef struct Variable

{

string var; //变量

string value; //初始化值

}Variable;

//目标代码元素

typedef struct Target

{

string dsf; //结果

string op; //操作

string dst; //目的操作数

string dsc; //源操作数

string mark; //标志

string step; //跳转位置

}Target;

//保存声明变量

vector<Variable> var\_table;

//保存中间代码

vector<Target> target\_code;

char lab='A'; //记录跳转标志

char vab='A'; //记录中间变量

//生成的汇编文件名称

string asmfile(string source)

{

if(source.size()==0)

{

cout<<"源文件名不能为空"<<endl;

exit(-1);

}

string temp="";

int i,j;

j = source.size();

for(i = j-1;i>=0;i--)

{

// if(source[i] == '\\' || source[i]== '/')

// break;

if(source[i] == '.')

{

j = i;

break;

}

}

temp = source.substr(0,j) + ".asm";

return temp;

}

//运算符优先级

int level(string s)

{

if(s=="#")

return 1;

else if(s=="=")

return 2;

else if(s=="+" || s=="-")

return 3;

else if(s=="\*" || s=="/")

return 4;

else

return -1;

}

//保存到目标代码

void add\_target\_code(string dsf,string op,string dst,string dsc,string mark,string step)

{

Target tmp;

tmp.dsf = dsf;

tmp.op = op;

tmp.dst = dst;

tmp.dsc = dsc;

tmp.mark = mark;

tmp.step = step;

target\_code.push\_back(tmp);

}

//字符转字符串

string char\_to\_str(char c)

{

char s[2] = " ";

s[0] = c;

return string(s);

}

//是否为运算操作符

int is\_operator(char c)

{

if(c == '+' || c=='-'||c=='\*'||c=='/'||c==','||c=='=' ||c=='>' || c=='<')

return 1;

else

return 0;

}

//是否为大括号、小括号、分号

int is\_bracket(char c)

{

if(c=='{' || c=='}' || c=='(' || c==')' ||c==';')

return 1;

else

return 0;

}

//是否为空白

int is\_blank(char c)

{

if(c=='\n' || c=='\t' || c==' ' || c=='\r')

return 1;

else

return 0;

}

//判断单词类型

int word\_token(string s)

{

int size = s.size();

//字符数据

if(s[0]=='\'')

{

if(s[size-1] == '\'')

return K\_CHAR;

else

{

cout<<"error:错误的字符串数据："<<s<<endl;

exit(-1);

}

}

//字符串数据

else if(s[0]=='\"')

{

if(s[size-1]=='\"')

return K\_STRING;

else

{

cout<<"error:错误的字符串数据："<<s<<endl;

exit(-1);

}

}

//整数

else if(isdigit(s[0]))

{

for(int i=1;i<size;i++)

{

if(!isdigit(s[i]))

{

cout<<"error:不合法的标识符："<<s<<endl;

exit(-1);

}

}

return K\_DIGIT;

}

else

{

for(int i=0;i<size;i++)

{

if(!isalnum(s[i]) && s[i]!='\_')

{

cout<<"error:不合法的标识符："<<s<<endl;

exit(-1);

}

}

//数据类型

if(s=="int" || s=="char")

return K\_TYPE;

//关键字

else if(s=="if" || s=="else" || s=="printf" || s=="main")

return K\_KEYWORDS;

//自定义标识符

else

return K\_IDENTIFIER;

}

}

//添加分词结果

void add\_keywords(vector<IDwords> &v,int id,string word)

{

IDwords temp;

temp.id = id;

temp.word = word;

v.push\_back(temp);

}

//词法分析

void cifa\_analysis(string source,vector<IDwords> &AnalysisResults)

{

char ch;

ifstream rfile(source.c\_str());

if(!rfile.is\_open())

{

cout<<"error:无法打开源文件"<<endl;

exit(-1);

}

rfile>>noskipws; //不过滤空格

while(rfile>>ch)

{

int state=0; //判断状态

string temp(""); //字符串缓存

char try\_ch; //探测前面的字符

switch(state)

{

case 0:

if(ch=='/') //可能是注释

{

rfile>>try\_ch;

if(try\_ch=='/')

{

while(rfile>>try\_ch)

{

if(try\_ch=='\n')

break; //这是一行注释

}

break;

}

else if(try\_ch=='\*')

{

while(rfile>>try\_ch)

{

if(try\_ch=='\*')

{

rfile>>try\_ch;

if(try\_ch=='/')

break; //这是多行注释

}

}

break;

}

else

{

add\_keywords(AnalysisResults,K\_OPERATOR,char\_to\_str(ch));

ch = try\_ch; //继续状态1

}

}

case 1:

if(is\_operator(ch)) //判断操作符

{

add\_keywords(AnalysisResults,K\_OPERATOR,char\_to\_str(ch));

break;

}

case 2:

if(is\_bracket(ch)) //大括号、小括号

{

add\_keywords(AnalysisResults,K\_BRACKET,char\_to\_str(ch));

break;

}

case 3:

if(is\_blank(ch)) //空白符

break;

case 4:

if(ch=='#') //跳过预处理

{

while(rfile>>ch)

{

if(is\_blank(ch))

{

break;

}

}

break;

}

default://判断单词类型

temp = temp + char\_to\_str(ch);

while(rfile>>try\_ch)

{

if(try\_ch == '\"')

{

temp = temp + char\_to\_str(try\_ch);

if(ch == '\"')

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

break;

}

else

{

cout<<"error:不合法的标识符："+temp<<endl;

exit(-1);

}

}

else if(is\_blank(try\_ch) )

{

if(ch != '\'' && ch != '\"')

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

break;

}

else

temp = temp + char\_to\_str(try\_ch);

}

else if(is\_operator(try\_ch) )

{

if(ch !='\'' && ch != '\"' )

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

add\_keywords(AnalysisResults,K\_OPERATOR,char\_to\_str(try\_ch));

break;

}

else

temp = temp + char\_to\_str(try\_ch);

}

else if(is\_bracket(try\_ch))

{

add\_keywords(AnalysisResults,word\_token(temp),temp);

add\_keywords(AnalysisResults,K\_BRACKET,char\_to\_str(try\_ch));

break;

}

else

temp = temp + char\_to\_str(try\_ch);

}

}

}

rfile.close();

}

//输出词法分析结果

void print\_lexical(vector<IDwords> &v)

{

vector<IDwords>::iterator it;

for(it = v.begin();it != v.end();it++)

cout<<it->id<<" "<<it->word<<endl;

}

//获取变量声明

void add\_var\_table(vector<IDwords>::iterator &it)

{

while(it->id == K\_TYPE)

{

it++;

while(it->word != ";")

{

if(it->id == K\_IDENTIFIER)

{

Variable tmp;

tmp.var = it->word;

string tmp\_var = it->word;

if((it+1)->word=="=") //判断变量有没有初始化

{

it = it+2;

tmp.value = it->word;

add\_target\_code(tmp\_var,"=",tmp.value," "," "," ");

}

var\_table.push\_back(tmp);

}

it++;

}

it++;

}

}

//表达式分析

void expression(vector<IDwords>::iterator &it)

{

string dsf,op,dst,dsc;

//保存非操作符栈

stack<string> word\_stack;

//操作符栈

stack<string> oper\_stack;

oper\_stack.push("#");

while(it->word != ";") //遇到';'一条语句结束

{

if(it->word == "(")

oper\_stack.push(it->word);

else if(it->word == ")")

{

while(oper\_stack.top() != "(")

{

op = oper\_stack.top();

oper\_stack.pop();

// oper\_stack.push(it->word);

dsc = word\_stack.top();

word\_stack.pop();

dst = word\_stack.top();

word\_stack.pop();

vab = vab+1;

if(vab == 91)

vab = '0';

dsf = "tmp" + char\_to\_str(vab);

Variable tmp;

tmp.var = dsf;

var\_table.push\_back(tmp);

word\_stack.push(dsf);

add\_target\_code(dsf,op,dst,dsc," "," ");

}

oper\_stack.pop();

}

else if(it->id != K\_OPERATOR)

word\_stack.push(it->word);

else if(oper\_stack.top() == "(")

{

oper\_stack.push(it->word);

}

else if(level(it->word) < level(oper\_stack.top())) //优先级低

{

op = oper\_stack.top();

oper\_stack.pop();

oper\_stack.push(it->word);

dsc = word\_stack.top();

word\_stack.pop();

dst = word\_stack.top();

word\_stack.pop();

vab = vab+1;

if(vab == 91)

vab = '0';

dsf = "tmp" + char\_to\_str(vab);

Variable tmp;

tmp.var = dsf;

var\_table.push\_back(tmp);

word\_stack.push(dsf);

add\_target\_code(dsf,op,dst,dsc," "," ");

}

else //优先级高

oper\_stack.push(it->word);

it++;

}

//弹出剩下的

while(oper\_stack.top() != "#")

{

op = oper\_stack.top();

oper\_stack.pop();

dsc = word\_stack.top();

word\_stack.pop();

dst = word\_stack.top();

word\_stack.pop();

if(op=="=")//赋值运算

{

add\_target\_code(dst,op,dsc," "," "," ");

}

else

{

vab = vab+1;

if(vab == 91)

vab = '0';

dsf = "tmp" + char\_to\_str(vab);

Variable tmp;

tmp.var = dsf;

var\_table.push\_back(tmp);

word\_stack.push(dsf);

add\_target\_code(dsf,op,dst,dsc," "," ");

}

}

}

//分析printf输出

void printf\_analysis(vector<IDwords>::iterator &it)

{

int j,i=1;

it = it+2;

string str = it->word; //获取输出内容

string strvar; //获取输出变量

Variable tmp;

//分析输出内容及格式

for(j=1;j<str.size()-1;)

{

if(str[j]=='%')

{

if(i != j)

{

vab = vab + 1;

if(vab == 91)

vab = '0';

add\_target\_code("\'"+str.substr(i,j-i)+"$\'","p"," "," ","tmp"+char\_to\_str(vab)," ");

tmp.var = "tmp"+char\_to\_str(vab);

tmp.value = "\'"+str.substr(i,j-i)+"$\'";

var\_table.push\_back(tmp);

}

i = j+2;

it = it+2; //获取对应变量

strvar = it->word;

add\_target\_code(strvar,"p"," "," ",str.substr(j,2)," ");

j = i;

continue;

}

j++;

}

if(i!=j)

{

vab = vab+1;

if(vab == 91)

vab = '0';

add\_target\_code("\'"+str.substr(i,j-i)+"$\'","p"," "," ","tmp"+char\_to\_str(vab)," ");

tmp.var = "tmp"+char\_to\_str(vab);

tmp.value = "\'"+str.substr(i,j-i)+"$\'";

var\_table.push\_back(tmp);

}

it = it+2; //略过“)}”

}

//分析if语句

void if\_analysis(vector<IDwords>::iterator &it)

{

string op,mark,dst,dsc;

it++;

if(it->word != "(")

{

cout<<"error:错误的if语句：缺少'('"<<endl;

exit(-1);

}

it++;

dst = it->word;

it++;

mark = it->word;

it++;

dsc = it->word;

op = "if";

add\_target\_code(" ",op,dst,dsc,mark,"step"+char\_to\_str(lab+1));

it++;

if(it->word != ")")

{

cout<<"error:错误的if条件语句：缺少')'"<<endl;

exit(-1);

}

it++; //略过‘{’

it++;

//分析else

vector<IDwords>::iterator it2 = it;

while(it2->word != "}")

{

it2++;

}

it2++;

//判断有没有else

if(it2->word == "else")

{

it2++; //略过‘{’

while(it2->word != "}")

{

expression(it2);

it2++;

}

}//else分析完成

else

it2--;

lab = lab + 2;

add\_target\_code(" ","jmp"," "," "," ","step"+char\_to\_str(lab));

add\_target\_code(" ","pstep"," "," "," ","step"+char\_to\_str(lab-1));

while(it->word != "}")

{

expression(it); //表达式分析

it++;

}

add\_target\_code(" ","jmp"," "," "," ","step"+char\_to\_str(lab));

add\_target\_code(" ","pstep"," "," "," ","step"+char\_to\_str(lab));

it = it2;

}

//语法分析

void yufa\_analysis(vector<IDwords> &AnalysisResults)

{

vector<IDwords>::iterator it=AnalysisResults.begin();

if(it->word != "main")

{

cout<<"error:缺少main"<<endl;

exit(-1);

}

it = it+3; //跳过“（）”

if(it->word != "{")

{

cout<<"error:main函数缺少'{'"<<endl;

exit(-1);

}

it++;

//获取变量声明

add\_var\_table(it);

//获取代码段的操作

while(it != AnalysisResults.end())

{

//遇到printf

if(it->word == "printf")

{

printf\_analysis(it);

}

// if 语句

else if(it->word == "if")

{

if\_analysis(it);

}

else if(it->word == "}")

break;

//表达式分析

else

{

expression(it); //表达式分析

}

it++;

}

}

//输出语法分析结果

void print\_syntax()

{

vector<Variable>::iterator it;

cout<<"变量声明及初始化"<<endl;

for(it = var\_table.begin();it != var\_table.end();it++)

{

cout<<it->var<<" "<<it->value<<endl;

}

vector<Target>::iterator it2;

cout<<"中间代码"<<endl;

for(it2 = target\_code.begin();it2 != target\_code.end();it2++)

{

cout<<it2->dsf<<" "<<it2->op<<" "<<it2->dst<<" "<<it2->dsc<<" "<<it2->mark<<" "<<it2->step<<endl;

}

}

//加减法转换

void addsub\_asm(ofstream &out,string dsf,string op,string dst,string dsc)

{

out<<" mov BL,"<<dst<<endl;

if(op == "+")

out<<" add BL,"<<dsc<<endl;

else

out<<" sub BL,"<<dsc<<endl;

out<<" mov "<<dsf<<",BL"<<endl;

}

//乘法

void mul\_asm(ofstream &out,string dsf,string dst,string dsc)

{

out<<" mov AL,"<<dst<<endl;

out<<" mov BH,"<<dsc<<endl;

out<<" mul BH"<<endl;

out<<" mov BL,1"<<endl;

out<<" div BL"<<endl;

out<<" mov "<<dsf<<",AL"<<endl;

}

//除法

void div\_asm(ofstream &out,string dsf,string dst,string dsc)

{

out<<" mov AL,"<<dst<<endl;

out<<" CBW"<<endl;

out<<" mov BL,"<<dsc<<endl;

out<<" div BL"<<endl;

out<<" mov "<<dsf<<",AL"<<endl;

}

//赋值运算

void sign\_asm(ofstream &out,string dsf,string dst)

{

out<<" mov BL,"<<dst<<endl;

out<<" mov "<<dsf<<",BL"<<endl;

}

//输出转换

void print\_asm(ofstream &out,string dsf,string mark)

{

//以字符格式输出

if(mark=="%c")

{

out<<" mov DL,"<<dsf<<endl;

out<<" mov AH,02H"<<endl;

out<<" int 21H"<<endl;

}

//以整数格式输出

else if(mark=="%d")

{

out<<" mov AL,"<<dsf<<endl;

out<<" CBW"<<endl;

out<<" mov BL,10"<<endl;

out<<" DIV BL"<<endl;

out<<" mov BH,AH"<<endl;

out<<" add BH,30H"<<endl;

out<<" add AL,30H"<<endl;

out<<" CMP AL,48"<<endl;

//确定十位是否是0

lab = lab + 2;

string step2 = "step" + char\_to\_str(lab);

out<<" JE "<<step2<<endl;

string step1 = "step" + char\_to\_str(lab-1);

out<<" "<<step1<<":"<<endl;

out<<" mov DL,AL"<<endl;

out<<" mov AH,2"<<endl;

out<<" int 21H"<<endl;

//输出个位

out<<" "<<step2<<":"<<endl;

out<<" mov DL,BH"<<endl;

out<<" mov AH,2"<<endl;

out<<" int 21H"<<endl;

}

//字符串输出

else

{

out<<" LEA DX,"<<mark<<endl;

out<<" mov AH,09"<<endl;

out<<" int 21H"<<endl;

}

}

//if语句转换

void if\_asm(ofstream &out,string dst,string dsc,string mark,string step)

{

out<<" mov AL,"<<dst<<endl;

out<<" CMP AL,"<<dsc<<endl;

if(mark == ">")

out<<" JG "<<step<<endl;

else if(mark == "<")

out<<" JL "<<step<<endl;

else

{

cout<<"暂不支持其他条件判断"<<endl;

exit(-1);

}

}

//生成汇编文件

void create\_asm(string file)

{

//变量声明

ofstream wfile(file.c\_str());

if(!wfile.is\_open())

cout<<"无法创建汇编文件"<<endl;

vector<Variable>::iterator it\_var;

wfile<<"ASSUME CS:codesg,DS:datasg"<<endl;

//数据段

wfile<<"datasg segment"<<endl;

for(it\_var=var\_table.begin();it\_var!=var\_table.end();it\_var++)

{

wfile<<" "<<it\_var->var<<" DB ";

if(it\_var->value != "")

wfile<<it\_var->value<<endl;

else

wfile<<"\'?\'"<<endl;

}

wfile<<"datasg ends"<<endl;

//代码段

wfile<<"codesg segment"<<endl;

wfile<<" start:"<<endl;

wfile<<" mov AX,datasg"<<endl;

wfile<<" mov DS,AX"<<endl;

vector<Target>::iterator it;

Target tmp;

for(it = target\_code.begin();it != target\_code.end();it++)

{

//加减法转化

if(it->op == "+" || it->op=="-")

addsub\_asm(wfile,it->dsf,it->op,it->dst,it->dsc);

//乘法转换

else if(it->op == "\*")

mul\_asm(wfile,it->dsf,it->dst,it->dsc);

//除法转换

else if(it->op == "/")

div\_asm(wfile,it->dsf,it->dst,it->dsc);

//赋值运算

else if(it->op == "=")

sign\_asm(wfile,it->dsf,it->dst);

//输出操作

else if(it->op == "p")

print\_asm(wfile,it->dsf,it->mark);

//if语法分析

else if(it->op == "if")

{

if\_asm(wfile,it->dst,it->dsc,it->mark,it->step);

}

else if(it->op == "else")

{

cout<<"else 没有找到匹配的 if"<<endl;

exit(-1);

}

//跳转语句

else if(it->op == "jmp")

{

wfile<<" JMP "<<it->step<<endl;

}

//跳转语句段标识

else if(it->op == "pstep")

{

wfile<<" "<<it->step<<":"<<endl;

}

//其他

else

{

cout<<"编译器暂不支持该语法操作"<<endl;

exit(-1);

}

}

//代码段结束

wfile<<" mov ax,4C00H"<<endl;

wfile<<" int 21H"<<endl;

wfile<<"codesg ends"<<endl;

wfile<<" end start"<<endl;

wfile.close();

}

int main(int argc,char\* argv[])

{

vector<IDwords> AnalysisResults;

string source;

cout<<"\n在相同目录生成汇编文件（拓展名为.asm）\n"<<endl;

cout<<"\n请输入源文件的路径:(最好不要含有中文)\n";

cin>>source;

cifa\_analysis(source,AnalysisResults);

yufa\_analysis(AnalysisResults);

print\_syntax();

create\_asm(asmfile(source));

cout<<"\n使用结束"<<endl;

return 0;

}