编译原理DO-WHILE循环语句的中间代码生成

实验报告

学院：计算机科学与工程

班级：200402

姓名：夏新宇

学号：20203644

实验三：中间代码生成

1. 实验题目

DO-WHILE循环语句的中间代码生成

二． 实验目的

通过设计、编制、调试一个 DO-WHILE 循环语句的语法及语义分析程序，加深对语法及语义分析原理的理解，并实现词法分析程序对单词序列的词法检查和分析。  
三． 实验内容及结果

1. 根据给出的样例程序，补充填写部分并运行;

(1)程序清单，填写部分用红色标注

#include<iostream>

#include<string>

#include <fstream>

#include<cctype>

using namespace std;

int const ACC = 1<<(31-1);

string filename; //用来输入文件名用

string symbol[200];

string blankCharacter = " ";

int num = 0 ;

//int start; //用在输出输入串时确定开始的位置

//------------------------------------------------------

struct stateStack { //定义状态栈

int top;

int m[100];

};

void InitStateStack(stateStack& s) { //建立状态栈

s.top = 0;

}

int PushState(stateStack& s, int i) { //把元素压入栈

if (s.top == 100) return -1;

else {

s.m[s.top] = i;

s.top++;

}

return 0;

}

int GetStateTop(stateStack& s) { //取栈顶符号

int i;

if (s.top == 0) return -1;

else {

i = s.m[ s.top - 1];

}

return i;

}

void PopStateTop(stateStack& s) { //删除栈顶元素

if (s.top == 0) cout << "wrong!";

else

--s.top;

}

//-----------------------------------------------------------------

//定义一个用于四元式输出的栈

struct quaternionStack {

int top;

string s[5];

};

//-----------------------------------------------------------------

//

struct symbolStack { //定义符号栈

int top;

string st[100];

};

void InitSymbolStack(symbolStack& s) { //建立符号栈

s.top = 0;

}

int PushSymbol(symbolStack& s, string str) { //把元素压入栈

if (s.top == 100) return -1;

else {

s.st[s.top] = str;

s.top++;

}

return 0;

}

string GetSymbolTop(symbolStack& s) { //取栈顶符号

string str;

if (s.top == 0) cout << "wrong!";

else {

str = s.st[ s.top - 1];

}

return str;

}

void PopSymbolTop(symbolStack& s) { //删除栈顶元素

if (s.top == 0) cout << "wrong!";

else

--s.top;

}

//----------------------------------------------------------

//词法分析部分

const string keyWord[] = { //定义关键字表

"int","double","float","void","long","for","if","else","while","include",

"return","break","continue","do","true","false","case","switch"

};

void showLex(string& s) {

symbol[num] = s;

++num;

ofstream fout("词法分析.txt", ios::out | ios::app);

if (isdigit(s[0])) {

fout << "< " << s << ",常数 >" << endl;

s = "";

} else

for (int i = 0; i < 18; ++i) {

if (s == keyWord[i]) {

fout << "< " << s << "，关键字 | 指向 " << s << " 的关键字表项的指针 >" << endl;

break;

}

if (i == 17) {

if (ispunct(s[0]) && s[0] != '\_') {

fout << "< " << s << ",运算符 | 指向 " << s << " 的运算符表项的指针 >" << endl;

} else {

fout << "< id,标识符 | 指向 " << s << " 的符号表项的指针 >" <<endl;

}

}

}

s = "";

}

bool isDelimiters(char ch) { //判断是否为界限符

bool tag = 0;

string s = "{[()]},;\'\"";

for (int i = 0; i < s.size(); i++) {

if (ch == s[i]) {

tag = 1;

break;

}

}

return tag;

}

bool isBlank(char ch) { //判断是否为空

bool tag = 0;

if (ch == ' ')

tag = 1;

return tag;

}

bool isDecimalPoint(char ch) { //判断是否为小数点

bool tag = 0;

if (ch== '.' )

tag = 1;

return tag;

}

void lexAnalysis(string s) {

string yunsf, biaosf;

for (int i = 0; i < s.size(); i++) {

if (isBlank(s[i])) { //判断是否为空

if (biaosf != "")

showLex(biaosf);

} else if (ispunct(s[i])) {

//C 库函数 int isgraph(int c) 检查所传的字符是否有图形表示法。带有图形表示法的字符是除了空白字符（比如 ' '）以外的所有可打印的字符。

if (isDelimiters(s[i])) {

if (yunsf != "")

showLex(yunsf);

if (biaosf != "")

showLex(biaosf);

ofstream fout("词法分析.txt", ios::out | ios::app);

fout << "< " << s[i] << ",界限符 | 指向 " << s[i] << " 的界限符的指针 >" << endl;

symbol[num] = s[i];

num++;

} else if (isDecimalPoint(s[i]) || s[i] == '\_') {

biaosf += s[i];

} else {

if (biaosf != "")

showLex(biaosf);

yunsf += s[i];

}

}//end ispunct( )

else {

if (yunsf != "")

showLex(yunsf);

biaosf += s[i];

}

}

symbol[num] = "#";

}//词法分析结束

//-------------------------------------------------------------------

//语法分析部分

//推导式部分

string production[18] = {

"S'-->S",

"S-->doAwhileB",

"A-->{F}",

"F-->M" ,

"F-->FM",

"M-->i=E;",

"E-->E+i",

"E-->E-i",

"E-->E\*i",

"E-->E/i",

"E-->i",

"E-->(E)",

"B-->(iOi)",

"O--><",

"O--><=",

"O-->>",

"O-->>=",

"O-->!=",

};

//-----------------------------------------------------------------

//GOTO控制部分

void GOTO(int i, char b, stateStack& state\_stack, symbolStack& symbol\_stack) {

// cout<<"GOTOzhong fterminals wei: "<<b<<endl;

if (i == 0 && b == 'S') {

PushState(state\_stack, 1);

} else if (i == 2 && b == 'A') {

PushState(state\_stack, 3);

} else if (i == 4 && b == 'B') {

PushState(state\_stack, 29);

} else if (i == 5 && b == 'F') {

PushState(state\_stack, 6);

} else if (i == 5 && b == 'M') {

PushState(state\_stack, 9);

} else if (i == 6 && b == 'M') {

PushState(state\_stack, 8);

} else if (i == 11 && b == 'E') {

PushState(state\_stack, 13);

} else if (i == 24 && b == 'O') {

PushState(state\_stack, 26);

} else if (i == 34 && b == 'E') {

PushState(state\_stack, 35);

}

string B = "";

B = B + b;

PushSymbol(symbol\_stack, B);

}

//--------------------------------------------------------------------

//Action控制表

//n大于0表示移进

//n小于0则达标要进行规约，-n就代表用哪个推导式进行规约

//ACC代表规约成功 0表示出错

const int Action[38][19] = {

{2}, //状态0 入栈do

{0,0,0,0,0,0,0,0,0,0,0,0,0,ACC}, //状态1 # ACC

{0,0,0,0,0,0,0,0,0,5}, //状态2 入栈{

{0,4}, //状态3 入栈while

{0,0,0,0,0,0,0,0,0,0,0,23 }, //状态4 入栈 (

{0,0,10}, //状态5 入栈i

{0,0,10,0,0,0,0,0,0,0,7,0}, //状态6 入栈i }

{0,-2}, //状态7 规约 2. A→{F} //while

{0,0,-4,0,0,0,0,0,0,0,-4}, //状态8 规约4. F→FM //i }

{0,0,-3,0,0,0,0,0,0,0,-3}, //状态9 规约3. F→M //i }

{0,0,0,11}, //状态10 入栈=

{0,0,12,0,0,0,0,0,0,0,0,34},//状态11 入栈 i (

{0,0,0,-10,-10,-10,-10,-10,-10,0,0,0,-10},

//状态12规约10.E→i= + - \* / ;

{0,0,0,0,15,16,17,18,14}, //状态13 入栈 + - \* / ;

{0,0,-5,0,0,0,0,0,0,0,-5}, //状态14 //规约 5. M→i=E;//i }

{0,0,19}, //状态15 入栈i

{0,0,20}, //状态16 入栈i

{0,0,21}, //状态17 入栈i

{0,0,22}, //状态18 入栈i

{0,0,0,0,-6,-6,-6,-6,-6,0,0,0,-6},

//状态19 规约 6.E→E+i // + - \* / ;

{0,0,0,0,-7,-7,-7,-7,-7,0,0,0,-7},

//状态20规约 7. E→E-i // + - \* / ;

{0,0,0,0,-8,-8,-8,-8,-8,0,0,0,-8},

//状态21规约 8. E→E\*i // + - \* / ;

{0,0,0,0,-9,-9,-9,-9,-9,0,0,0,-9},

//状态22 规约 9.E→E/i // + - \* / ;

{0,0,24}, //状态23 入栈i

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,30,31,32,33},//入栈 < <= > >= !=

{0,0,-13}, //状态25 规约 13. O→<

{0,0,27}, //状态26 入栈 i

{0,0,0,0,0,0,0,0,0,0,0,0,28}, //状态27 入栈 )

{0,0,0,0,0,0,0,0,-12,0,0,0,0,0},//状态28 规约12. B→(iOi) //;

{0,0,0,0,0,0,0,0,37}, //状态29 入栈 ；

{0,0,-14}, //状态30 规约14. O→<= //i

{0,0,-15}, //状态31 规约15. O→> //i

{0,0,-16}, //状态32 规约16. O→>= //i

{0,0,-17}, //状态33 规约17. O→!= //i

{0,0,12,0,0,0,0,0,0,0,0,34}, //状态34 入栈 i (

{0,0,0,0,15,16,17,18,0,0,0,0,36},//状态35 入栈 + - \* / )

{0,0,0,0,-11,-11,-11,-11,-11,0,0,0,-11},//状态36 规约

//11.E→(E)//+ - \* / ;

{0,0,0,0,0,0,0,0,0,0,0,0,0,-1}//状态37 规约 1.S→doAwhileB; //#

};

//--------------------------------------------------------------

//终结符表

string terminals[19] = {

"do","while","i","=","+","-","\*","/",";","{","}","(",")","#","<","<=",">",">=","!="

};

//--------------------------------------------------------------------

//输出每步的分析过程 存储在文件“语法分析.txt”中

void showResult(stateStack& state\_stack, symbolStack& symbol\_stack, int start) {

//每次归约完毕后查看当前符号栈栈顶状态

cout<<GetSymbolTop(symbol\_stack)<<" "<<GetStateTop(state\_stack)<<endl;

ofstream yffout("语法分析.txt", ios::out | ios::app);

for (int i = 0; i < state\_stack.top; i++) {

if (state\_stack.m[i] >= 10) {

yffout << "(" << state\_stack.m[i] << ")";

} else

yffout << state\_stack.m[i];

}

yffout << " ";

for (int j = 0; j < symbol\_stack.top; j++)

yffout << symbol\_stack.st[j];

yffout << " ";

for (int k = start; k <= num + 1; k++)

yffout << symbol[k];

yffout << endl;

}

//-----------------------------------

//表达式规约部分

void Reduce() {

ofstream syfout("四元式.txt", ios::out | ios::app);

int start = 0, now = 0;

quaternionStack quaternion\_stack;

quaternion\_stack.top = 0;

int order = 0;

int m = 0, n, j;

stateStack state\_stack; //建状态栈

InitStateStack(state\_stack);

PushState(state\_stack, m);

symbolStack symbol\_stack;//建符号栈

InitSymbolStack(symbol\_stack);

PushSymbol(symbol\_stack, "#");

//对栈进行初始化

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

{

showResult(state\_stack, symbol\_stack,i); //i+1代表已归约现在可查看新的状态

string temp = symbol[i];

j = 0;

while (terminals[ j ] != temp) { //查找终结符对应的下标

++j;

if ( j >= 19) {

j = 2; //如果是变量或者常量，令：j = 2，即对应终结符 i

break;

}

}

m = GetStateTop(state\_stack); //查看当前状态栈的栈顶

n = Action[m][ j];

int rightLength;

while ( n <0 ) {

n=-n;

if (n == 14||n==16||n==17)

        //请填写程序语句；

                        rightLength = production[n ].size() - 5;

else if(n==1)

    //请填写程序语句；

                        rightLength = production[n ].size() - 8;

else

rightLength = production[n ].size() - 4;

                    //cout<<"表达式"<<(n-40)<<"的右部长度为"<<rightLength<<endl;

                    //----------------------------------------------------------

                    //四元式输出控制部分

string ss0, ss1, ss2;

string t[] = { "t0","t1","t2","t3","t4","t5","t6","t7","t8","t9","t10" };

if (n == 10) { //当用 10.E→i 式归约时,开始建立栈

quaternion\_stack.s[quaternion\_stack.top] = symbol[i - 1];

++quaternion\_stack.top;

} else if (n >= 6 && n <= 9) { //6. E→E+i 7.E→E-i 8.E→E\*I 9.E→E/i

--quaternion\_stack.top;

ss0 = quaternion\_stack.s[quaternion\_stack.top];

--quaternion\_stack.top;

ss1 = quaternion\_stack.s[quaternion\_stack.top];

--quaternion\_stack.top; //请填写程序语句；

ss2 = quaternion\_stack.s[quaternion\_stack.top];//请填写程序语句；

syfout << now << " (" << ss1 << "," << ss2 << "," << ss0 << "," << t[order]

      << ")" << endl;//运算的四元式

++now;

quaternion\_stack.s[quaternion\_stack.top] = t[order];

++quaternion\_stack.top;

++order;

} else if (n == 5) { //赋值语句的四元式；5. M→i=E;

int l = symbol\_stack.top - 4;

quaternion\_stack.top = 0;

syfout << now << " (=:," << quaternion\_stack.s[quaternion\_stack.top]<< ",-," << symbol\_stack.st[l] << ")" << endl;

++now;

} else if (n == 13 || n == 14 || n == 15 || n == 16) //判断语句的四元式

syfout << now << " if " << symbol[i - 2] << symbol[i - 1] << symbol[i]<< " goto " << start << endl;

//------------------------------------------------------

for (int k = 1; k <= rightLength; k++) {

PopSymbolTop(symbol\_stack);

PopStateTop(state\_stack);

}

// showLex(state\_stack,symbol\_stack,i);

GOTO(GetStateTop(state\_stack), production[n][0], state\_stack,symbol\_stack);//此处用到了状态栈和符号栈的引用

showResult(state\_stack, symbol\_stack, i);

m = GetStateTop(state\_stack); //while xunhuanblankCharacterzhibufen

n = Action[m][j];

}

if (n == 0)

{

cout << "归约出错";

break;

}

if (n == ACC)

cout << "规约成功" << endl;

else

{

PushState(state\_stack, n);

// cout<<n<<endl;

PushSymbol(symbol\_stack, temp);

//每次移进后都重新输出状态栈，符号栈和符号串

if (quaternion\_stack.top > 0) {

quaternion\_stack.s[quaternion\_stack.top] = temp;

// cout<<quaternion\_stack.s[quaternion\_stack.top];

++quaternion\_stack.top;

}

}

}

}

//------------------------------------------------------------------

//主函数

int main() {

//cout << "输入需要进行分析的程序的名称:\n";

//cin >> filename;

filename = "test.cpp";

ifstream fin(filename.c\_str()); //待添加当文件不存在 的处理

string line, s;

fin >> line; //直接读取到空格，并且省略回车,最后至少要有空格或省略号

while (!fin.eof()) {

s = s + line + blankCharacter;

fin >> line;

}

fin.close();

ofstream fout("词法分析.txt");

fout << "";

cout << s <<"--------" << endl; //在字符界面查看需要分析的字符串，可以删除

lexAnalysis( s ); //词法分析

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

cout<<i<<": "<<symbol[i]<<endl;

ofstream yffout("语法分析.txt");

yffout << "状态栈" << "\t\t符号栈" << "\t\t\t\t输入串" << endl;

ofstream syfout("四元式.txt");

syfout << "";

Reduce(); //语法分析，进行规约

return 0;

}

（2）程序测试用例

int a = 1, b = 2, c = 3;

do

{

a = a \* b + b - c;

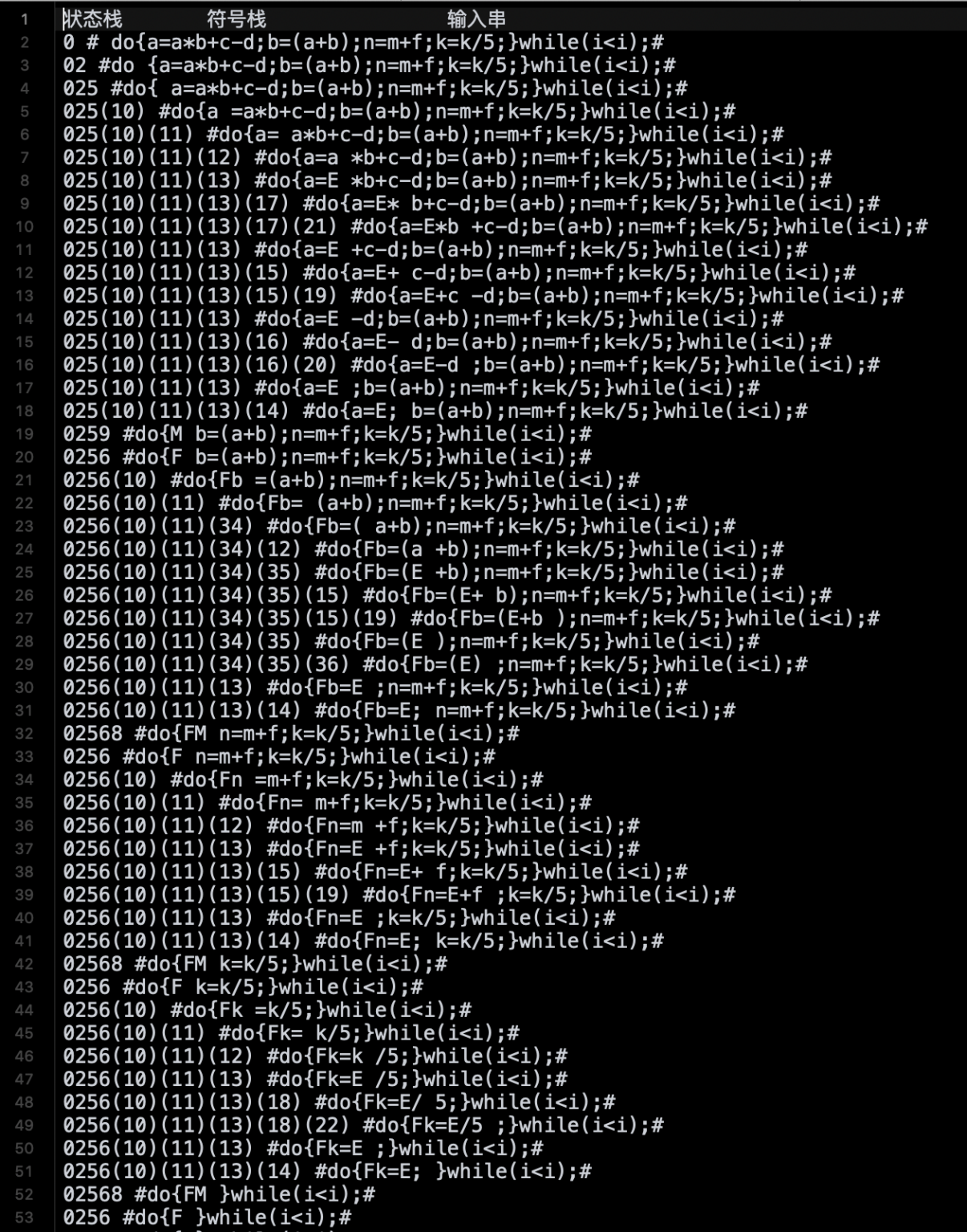
i=i+100;

} while ( i < i ) ; #

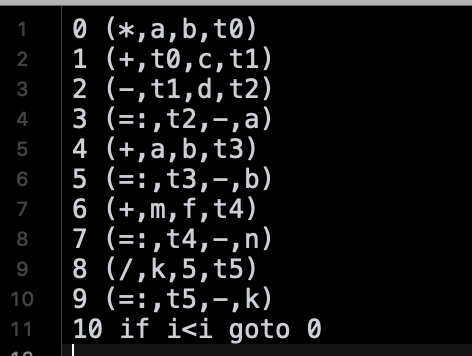
1. 词法分析生成文件内容截图



1. 语法分析生成文件内容截图



（5）中间代码生成文件内容截图



1. 根据所设计的分析程序和所给定的测试用例，自行构造识别活前缀的DFA、ACTION表和GOTO表，上机测试并通过所设计的分析程序。

（1）自行构造的识别活前缀的DFA图：

I0:S'->.S

1. >.do{E;}while(B)

I1:S'->S.

I2:S->do.{E;}while(B)

I3:S->do{.E;}while(B)

E->.=I op I

I->.id

14:S->do{E.;}while(B)

I5:E->l.=I op I

I6:I->id.

17:S->do{E;.}while(B)

I8:E->I=.I op I

I->.id

I9:S->do{E;}.while(B)

I10:E->l =I. op I

I11:S->do{E;}while.(B)

I12:E->I=I op .I

I=.id

I13:S->do{E;}while(.B)

B->.I rop I

I->.id

I14:E->I=I op I.

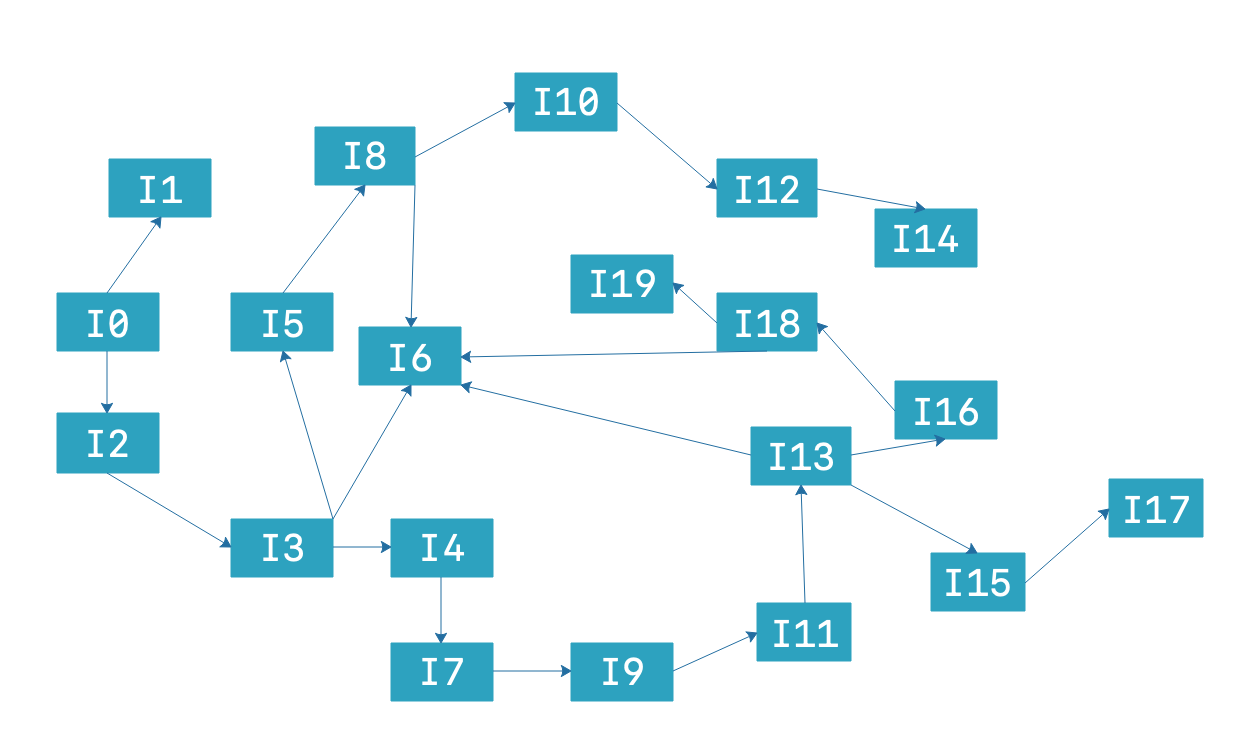
I15:S->do{E;}while(B.)

116:B->I .rop I

I17:S->do{E;}while(B).

118:B->I rop .I

I19:B->I rop I.



(2)自行设计的ACTION表和ACTION表所对应的C语言源代码：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态 | ACTION | | | | | | | | | | | |
|  | Do | { | = | ***；*** | } | While | ( | ) | Rop | Op | Id | # |
| 0 | S2 |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  | acc |
| 2 |  | S3 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  | S6 |  |
| 4 |  |  |  | S7 |  |  |  |  |  |  |  |  |
| 5 |  |  | S8 |  |  |  |  |  |  |  |  |  |
| 6 | R4 | R4 | R4 | R4 | R4 | R4 | R4 | R4 | R4 | R4 | R4 | R4 |
| 7 |  |  |  |  | S9 |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  | S6 |  |
| 9 |  |  |  |  |  | S11 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  | S12 |  |  |
| 11 |  |  |  |  |  |  | S13 |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |  |  |  | S6 |  |
| 14 | R3 | R3 | R3 | R3 | R3 | R3 | R3 | R3 | R3 | R3 | R3 | R3 |
| 15 |  |  |  |  |  |  |  | S17 |  |  |  |  |
| 16 |  |  |  |  |  |  |  |  | S18 |  |  |  |
| 17 | R1 | R1 | R1 | R1 | R1 | R1 | R1 | R1 | R1 | R1 | R1 | R1 |
| 18 |  |  |  |  |  |  |  |  |  |  | S6 |  |
| 19 | R2 | R2 | R2 | R2 | R2 | R2 | R2 | R2 | R2 | R2 | R2 | R2 |

源代码：

int Action(State &state,int actionnum,ofstream &outfile){

int row=0,col=0,numchange=0;

int choice=O;

int ct=100;int m=0;

if(actionnum>=1&&actionnum<=18)

choice=1;

else

choice =actionnum;

switch(choice)

{

case 0:{

isillegal=1;

cout<<"isillegal="<<isillegal<<endl;

break;

}

case 1:{//移进项目{

cout<<setw(4)<<step+l<<" ";state.showState();

cout<<setw(8)<<actionnum<<endl;

state.CurState++;state.stkState[state.CurState]-actionnum;state.CurSymbol+t;

strcpy(state.stkSymbol[state.CurSymbol],state.InStr[state.CurInstr]);

state.CurInstr++;

strcpy(next,state.InStr[state.CurInstr]);ropOrOp(next);row=state.stkState[state.CurState];

col=Index(next);

numchange-table[row][col];

break;

}

case 20://接收i

cout<<setw(4)<<step+f<<" ";state.showState();

cout<<setw(8)<<"acc"<<endl;break;

case 21://r1 S-->while(B){E;}

{

cout<<setw(4)<<step+f<<"";state.showState();

cout<<setw(8)<<actionnum;

for(int i=9;i>0;i--)

{state.stkState[state.CurState--]-0;}for(i=9-1;i>=0;i--)

{

strcpy(B[i],state.stkSymbol[state.CurSymbol]);

strcpy(state.stkSymbol[state.CurSymbol--],"");

outfile<<"B.false: "<<'\n':

state.CurSymbol++;

strcpy(state.stkSymbol[state.CurSymbol],"S");

l/B-->Iroplstrcpy(next,state.stkSymbol[state.CurSymbol]);

ropOrOp(next);

row=state.stkState[state.CurState];

col=Index(next);

numchange=table[row][col];

numchange=Goto(state,numchange);

break;

}

}

case 22://r2 B-->Iropl

{

cnt=0;

cout<<setw(4)<<step++<<"";

state.showState();

cout<<setw(8)<<actionnum;for( int i=3;i>0;i--)

{state.stkState[state.CurState--]=0;}

for(i=3-1;i>O;i--)

{

strcpy(B[i],state.stkSymbol[state.CurSymbol]);

strcpy(state.stkSymbol[state.CurSymbol--],"");

}

outfile<<"102 t1:="<<I[0]<<B[1]<<I[1]<<endl;

outfile<<"103 if t1.val=true"<<" goto 100"<<endl;

outfile<<"104 goto 105"<<endl;

state.CurSymbol++;

strcpy(state.stkSymbol[state.CurSymbol],"B");//归约B-->Iropl

strcpy(next,state.stkSymbol[state.CurSymbol]);

ropOrOp(next);row=state.stkState[state.CurState];

col=Index(next);

numchange-table[row][col];

numchange=Goto(state,numchange);

break;

}

case 23://r3E-->I-lopl

{

cnt=0;

cout<<setw(4)<<step++<<" ";

state.showState();

cout<<setw(8)<<actionnum;for(int i=5;i>0;i--)

{state.stkState[state.CurState--]=0;}

for(i=5-1;i>=O;i--)

{

strcpy(E[i],state.stkSymbol[state.CurSymbol]);

strcpy(state.stkSymbol[state.CurSymbol--],"");

}

outfile<<"100"<<"t1:="<<][1]<<E[3]<<I[2]<<endl;

outfile<<"101"<<][0]<<":=t1"<<endl;

state.CurSymbol+t;

strcpy(state.stkSymbol[state.CurSymbol],"E");

strcpy(next,state.stkSymbol[state.CurSymbol]);

ropOrOp(next);

row=state.stkState[state.CurState];

col=Index(next);

numchange-table[row][col];

numchange=Goto(state,numchange);

break;

}

case 24://r4归约L-->id

{

cout<<setw(4)<<step+t<<" ";state.showState();

cout<<setw( 8)<<actionnum;

state.stkState[state.CurState--]=0;

strcpy(I[cnt++],state.stkSymbol[state.CurSymbol]);//{L.value=id.value}

strcpy(state.stkSymbol[state.CurSymbol],"T");//归约I-->id

strcpy(next,state.stkSymbol[state.CurSymbol]);

ropOrOp(next);

row=state.stkState[state.CurState];

col-Index(next);

numchange-table[row][col];

numchange-Goto(state,numchange);

break;

}

}

return numchange;

}

1. 自行设计的GOTO表和GOTO表所对应的C语言源代码：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 状态 | GOTO | | | |
|  | S | B | E | I |
| 0 | 1 |  |  |  |
| 2 |  |  | 4 | 5 |
| 8 |  |  |  | 10 |
| 12 |  |  |  |  |
| 13 |  | 15 |  | 16 |
| 18 |  |  |  | 19 |

源代码：

int GOTO(int gotonum,stateStack& state\_stack, symbolStack& symbol\_stack) {

int row=0,col=0,numchange=0;

cout<<s(6)<<gotonum<<endl;

state.Curstate++;

state.stkState[state.Curstate]=gotonum;

strcpy(next,state,Instr[state.CurInstr]);

ropOrOp(next);

row=state.stkState[state.CurState];

col=Index(next);

return table[row][col];

}

四． 实验心得

通过四节实验课的学习，在老师的带领下，我对编译原理的所学知识有了更深刻的认识和理解，每次在编写实验报告的同时都会有一些新的心得体会，编码的能力更加扎实了，以后会不断学习，提高自己。