编译原理实验报告

类 C 编译器设计与实现

学　　号　1750844

姓　　名　周展田

专　　业　计算机科学与技术

授课老师　高秀芬

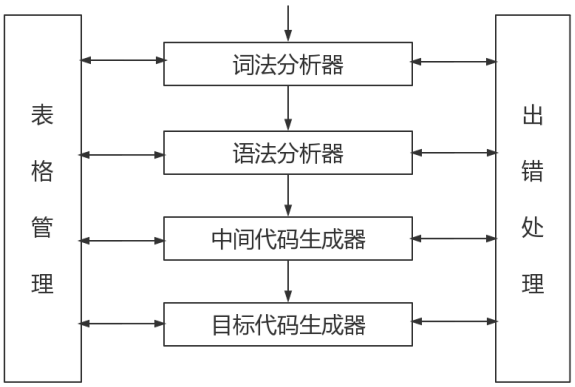
1.需求分析

使用高级程序语言实现一个类 C 语言的编译器，可以提供词法分析、语法分析、符号表管理、中间代码生成以及目标代码生成等功能。具体要求如下：
（1）使用高级程序语言作为实现语言，实现一个类 C 语言的编译器。编程序实现编译器的各组成部分。
（2）要求的类 C 编译器是个一遍的编译程序，词法分析程序作为子程序，需要的时候被语法分析程序调用。
（3）使用语法制导的翻译技术，在语法分析的同时生成中间代码，并保存到文件中。
（4）要求输入类 C 语言源程序，输出中间代码表示的程序;
（5）要求输入类 C 语言源程序，输出目标代码（可汇编执行）的程序。
（6）实现过程、函数调用的代码编译
其中（1)、(2)（3)（4）是必做内容，(5)（6）是选作内容。

2.系统设计

本次实验使用 Qt 完成，在上个学期的词法、语法分析器的基础上进行修改与添加，最终完成编译器的设计。

编译器的总体设计如下：



2.1 词法分析

2.1.1 操作步骤

将字符逐个读入，使用数组模拟自动机，以判断属于哪种类型。
a) ‘ ’，\t，\r，直接跳过
b) \n，记录下来，用于后面的行数计算，记为 NL
c) 数字，扫描至数字的结束符(+ = >=等)，记为 NUM
d) 操作符，+ -记为 OP1， * /记为 OP2， ， 记为 SEP，;记为 DEL，(记为 LP，)记为 RP，{记为 LB，}记为 RB，> < = >= <= == !=记为 RELOP（在语法和词法的角度，暂且忽略各个符号的真实含义，方便化简）
e) 保留字(int void while if else return)，按照不同的保留字分别记为 INT

VOID WHILE IF ELSE RETURN
f） 标志符(自定义的变量名称、main)，记为 ID
g） 注释信息，记为 COM，不在最终的文件里显示

2.1.2 设计思路

词法分析的过程其实就是一个模拟自动机运行的过程。并根据输入的情况决定下一步跳转到哪一状态或直接输出结果。

每次开始分析的过程是，首先读入一个字符，根据符号的类型选择进入调用不同的函数作为自动机的入口。每个函数会返回一个值，0 表示成功识别，正数表示出错行数。在函数内部不断读取字母，直到遇到不符合的字母，会返回结果，并让读取字母的指针回退（因为多读一个字母）。

根据当前所处状态以及输入的情况，作为下一步跳转到另一状态或是跳出此次测试的依据。跳出时在前后两指针之间的内容，也做为下一步的输入。调用不同的输出函数，并把结果输出在文件中。

2.2 语法分析

2.2.1 化简文法，使其符合 LL(1)规则

Program ::= <类型> < ID>’(‘ ‘)’<语句块>
<类型>::=int | void
<语句块> ::= ‘{‘<内部声明> <语句串>’}’
<内部声明> ::= 空 |<内部变量声明><内部声明>
<内部变量声明>::=int <ID><next>；
<next> ::= ,<ID><next> | 空
<语句串> ::= <语句> <语句串> | 空
<语句> ::= <if 语句> |< while 语句> | <return 语句>; | <赋值语句>；
<赋值语句> ::= <ID> =<表达式>
<return 语句> ::= return retBlock
retBlock ::= 空 | 表达式
<while 语句> ::= while ‘(‘<表达式> ‘)’ <语句块>
<if 语句> ::= if ‘(‘<表达式>’)’ <语句块> elseBlock
elseBlock ::= else <语句块> | 空
<表达式>::=<加法表达式> <comp>
<comp> ::= relop <加法表达式><comp> | 空
<加法表达式> ::= <项> <op1>
<op1> ::= +<项><op1> | 空
<项> ::= <因子> <op2>
<op2> ::= * <因子> <op2> | 空
<因子> ::=num | ‘(‘<表达式>’)’ |<ID>

2.2.2 用定义好的枚举类型表示产生式

typedef enum { INT, VOID, ID, LP, RP, LB, RB, WHILE, IF, ELSE, RETURN, ASSIGN, OP1, OP2, RELOP, DEL, SEP, NUM, NL, PROGRAM, TYPE, SENBLOCK, INNERDEF, INNERVARIDEF, NEXT, SENSEQ, SENTENCE, ASSIGNMENT,

RETURNSEN, RETBLOCK, WHILESEN, IFSEN, ELSEBOCLK, EXPRESSION, COMP, PLUSEX, OPPLUSDEC, TERM, OPMULDIV, FACTOR, EPSILON, END, ERROR}tokenType;

2.2.3 符号表

符号表	Enum
int	INT
void	VOID
ID	ID
(	LP
)	RP
{	LB
}	RB
while	WHILE
if	IF
else	ELSE
return	RETURN
=	ASSIGN
+ -	OP1
* /	OP2
> < >= <= == !=	RELOP
;	DEL
,	SEP
num	NUM
\n	NL
<Program>	PROGRAM
<类型>	TYPE
<语句块>	SENBLOCK
<内部声明>	INNEREDEF
<内部变量声明>	INNERVARIDEF
<next>	NEXT
<语句串>	SENSEQ
<语句>	SENTENCE
<赋值语句>	ASSIGNMENT
<return 语句>	RETURNSEN
<retBlock>	RETBLOCK
<while 语句>	WHILESEN
<if 语句>	IFSEN
<elseblock>	ELSEBLOCK
<表达式>	EXPRESSION
<comp>	COMP
<加法表达式>	PLUSEX

<op1>	OPPLUSEDEC
<项>	TERM
<op2>	OPMULDIV
<因子>	FACTOR
空	EPSILON
#	END

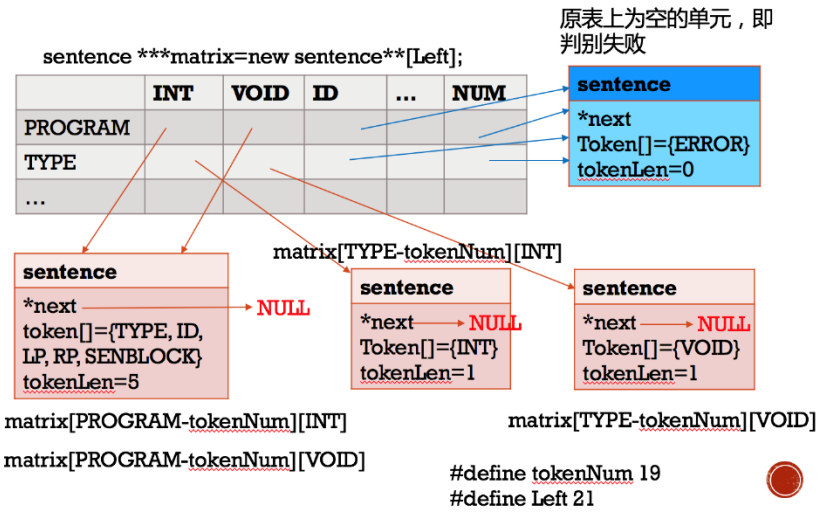
2.2.4 构建 FIRST 集合和 FOLLOW 集合

		frist	follow
2	program	int-void	#
3	类型	int-void	ID
4	语句块	{	else-E-#
5	内部声明	E-int	if-while-return-ID-E-}
6	语句串	if-while-return-ID-E	}
7	内部变量声明	int	if-while-return-ID-E-int
8	语句	if-while-return-ID	if-while-return-ID-}
9	if语句	if	if-while-return-ID-}
10	while语句	while	if-while-return-ID-}
11	return语句	return	if-while--ID-;
12	赋值语句	ID	if-while-return-ID-;
13	表达式	num-ID-(	)-;
14	加法表达式	num-ID-(	)-;
15	项	num-ID-(	)-;
16	因子	num-ID-(	)-;
17	next	,-E	;
18	comp	relop-E	)-;
19	retBlock	num-ID-(-E	;
20	op1	E-+	)-;
21	op2	E-*	)-;
22	elseBlock	else-E	if-while-return-ID-}

PROGRAM=: INT:TYPE ID LP RP SENBLOCK | VOID:TYPE ID LP RP SENBLOCK
TYPE=: INT:INT | VOID:VOID
SENBLOCK=: LB:LB INNERDEF SENSEQ RB
INNERDEF=: INT:INNERVARIDEF INNERDEF | ID:EPSILON | RB:EPSILON | IF:EPSILON | ELSE:EPSILON | RETURN:EPSILON
INNERVARIDEF=: INT:INT ID NEXT DEL
NEXT=: DEL:EPSILON | SEP:SEP ID NEXT
SENSEQ=: ID:SENTENCE SENSEQ | RB:EPSILON | WHILE:SENTENCE SENSEQ | IF:SENTENCE SENSEQ | RETURN:SENTENCE SENSEQ
SENTENCE=: ID:ASSIGNMENT DEL | WHILE:WHILESEN | IF:IFSEN | RETURN:RETURNSEN DEL
ASSIGNMENT=: ID:ID ASSIGN EXPRESSION
RETURNSEN=: RETURN:RETURN RETBLOCK
RETBLOCK=: ID:EXPRESSION | LP:EXPRESSION | DEL:EPSILON | NUM:EXPRESSION
WHILESEN=: WHILE:WHILE LP EXPRESSION RP SENBLOCK
IFSEN=: IF:IF LP EXPRESSION RP SENBLOCK ELSEBOCLK
ELSEBOCLK=: ID:EPSILON | RB:EPSILON | WHILE:EPSILON |

IF:EPSILON | ELSE:ELSE SENBLOCK | RETURN:EPSILON
EXPRESSION=: ID:PLUSEX COMP | LP:PLUSEX COMP | NUM:PLUSEX COMP
COMP=: RP:EPSILON | RELOP:RELOP PLUSEX COMP | DEL:EPSILON
PLUSEX=: ID:TERM OPPLUSDEC | LP:TERM OPPLUSDEC | NUM:TERM OPPLUSDEC
OPPLUSDEC=: RP:EPSILON | RETURN:EPSILON | OP1:OP1 TERM OPPLUSDEC | RELOP:EPSILON | DEL:EPSILON
TERM=: ID:FACTOR OPMULDIV | LP:FACTOR OPMULDIV | NUM:FACTOR OPMULDIV
OPMULDIV=: RP:EPSILON | RETURN:EPSILON | OP1:EPSILON | OP2:OP2 FACTOR OPMULDIV | RELOP:EPSILON | DEL:EPSILON
FACTOR=: ID:ID | LP:LP EXPRESSION RP | NUM:NUM

2.2.5 LL1 分析表



2.3 中间代码生成

语法分析后产生语法树，进行一定的调整生成抽象语法树。表达式采用逆波兰式的方式组织，使操作简便。根据抽象语法树生成三地址码的过程需要维护变量的作用域信息，符号表 gtable 以及跳转指令的标号管理 Qtable。采用递归的方式表示每一个语句块。最后将三地址代码生成四元式，更符合目标代码的结构。

2.4 目标代码生成

选择 mips 汇编作为目标代码。
根据中间代码阶段生成的四元式，初始化四元式表 formatlist，每条四元式由 op, arg1, arg2, result 及行号组成。

初始化寄存器，符号表，根据四元式的内容得到标号表 laberlist，并初始化 32 个寄存器为可用状态。

在进行寄存器分配时，遵循尽可能留，尽可能用，及时腾空的原则。

3.程序具体实现

3.1 词法分析，扫描器

```
int Lex::startScanner()
{
    char ch = fgetc(source);
    int states = 0;
    while (!feof(source) && states == 0)
    {
        if (ch == ' ' || ch == '\t' || ch == '\r' || ch == '\l')
        {
            ;//nothing jump it!
        }
        else if (ch == '\n') {
            addNewLine();
            clearState();
        }
        else if (ch == '/')
        {
            comment(ch, 0);
            clearState(); //清除状态信息
        }
        else if (ch >= '0'&&ch <= '9')
        {
            states = number(ch, 1); //处理数字
            clearState();
        }
        else if (isOperator(ch))
        {
            states = myOperator(ch, 2); //处理操作符
            clearState();
        }
        else if (isLiter(ch))
        {
            states = identifier(ch, 3); //处理标志符
            clearState();
        }
        else
        {
            return lineNum;
        }
```

```
        ch = fgetc(source);
    }
    return states;
}
```

3.2 语法分析

```
int Grammar::analyze(string filename)
{
    vector<int> levelCount = vector<int>(128, 0);
    vector<int> level = vector<int>(1024, -1);
    int level2[1024];
    treeNode **stack2 = new treeNode*[1024];
    ifstream fin(filename, std::ios::in);
    if (!fin) {        return -1;    }
    ofstream fout(OFileName, std::ios::out);
    ofstream fout2("tree.txt", std::ios::out);
    if (!fout) {        return -1;    }
    string line;
    int lineNum = 1;
    int point = 0;                          //始终指向栈顶元素
    int point2 = 0;
    localStack[point] = END;
    grammarTree = new treeNode();
    grammarTree->type = PROGRAM;
    localStack[++point] = PROGRAM;
    stack2[point] = grammarTree;
    level2[0] = 0;
    level[point] = 0;
    bool flag = true;
    treeNode *nodeTemp;
    getline(fin, line);
    string v;
    int attribute;
    SplitString(line, v, attribute);
    int count = getToken(v);
    while (flag) {
        while (count == NL) {                   //忽略换行符
            lineNum++;
            getline(fin, line);
            SplitString(line, v, attribute);
            count = getToken(v);
        }
```

```
        if (localStack[point] < tokenNum) {
            if (localStack[point] == count) {       //即栈顶符号 x 与当前输入符 a 匹配，则将 x 从栈顶弹出，输入串指针后移，读入下一个符号存入 a，继续对下一个字符进行分析。
                for (int i = 1; i < level[point]; i++)
                    if (levelCount[i] != 0) {
                        fout << special[0];
                        fout2 << " ";
                    }
                    else {
                        fout << special[2];
                        fout2 << " ";
                    }
                    fout << special[1];
                    fout2 << " ";
                    levelCount[level[point]]--;
                    fout << strTokens[localStack[point]];
                    fout2 << localStack[point];

                    switch (localStack[point])
                    {
                    case ID:
                    case NUM:
                    case OP1:
                    case OP2:
                    case RELOP:
                        fout << " - " << attribute << ' ';
                        fout2 << " - " << attribute << ' ';
                        break;
                    default:
                        break;
                    }
                    fout << endl;
                    fout2 << " " << lineNum << endl;
                    if (getline(fin, line)) {//读取下一个字符，若已经读至文件末尾，返回-1，证明语法正确，检验成功
                        SplitString(line, v, attribute);
                        //cout << v[1] << endl;
                        count = getToken(v);
                        point--;
                        continue;
                    }
```

```
                    else {
                        fin.close();
                        fout.close();
                        fout2.close();
                        generateTree();
                        for (int i = 0; i < 20; i++)
                            varTable.newTemp(i);
                        cout << localId << endl;
                        return 0; //正确判断的输出结果为0
                    }            }
            else {                flag = false;            }        }
        else if (localStack[point] == END) {
            if (localStack[point] == count) {
                levelCount[level[point]]--;
                point--;
                continue;
            }
            else {                flag = false;            }        }
        else if (localStack[point] >= PROGRAM && localStack[point] <= F
ACTOR) {//栈顶为非终结符
            for (int i = 1; i < level[point]; i++)
                if (levelCount[i] != 0) {
                    fout << special[0];
                    fout2 << " ";                }
                else {
                    fout << special[2];
                    fout2 << " ";                }
                if (localStack[point] != PROGRAM) {
                    fout << special[1];
                    fout2 << " ";                }
                fout << strTokens[localStack[point]];
                fout2 << localStack[point];
                switch (localStack[point])
                {
                case ID:
                case NUM:
                case OP1:
                case OP2:
                case RELOP:
                    fout << " - " << attribute << ' ';
                    fout2 << " - " << attribute << ' ';
                default:
```

```
                    break;                }
                sentence *s = new sentence;
                s = matrix[localStack[point] - tokenNum][count];
                if (s->tokens[0] == ERROR) {//为空，则发现语法错误，调用出
错处理程序进行处理
                    flag = false;
                }
                else if (s->tokens[0] == EPSILON) {//A→ε ，则只将 A 自栈顶
弹出。
                    fout << " - @";
                    fout2 << " - @";
                    levelCount[level[point]]--;
                    point--;
                }
                else {
                    int j = level[point];
                    levelCount[j]--;
                    /*          nodeTemp = stack2[point];*/
                    int ktemp = point;
                    for (int i = s->tokenLen - 1; i >= 0; i--) {
                        localStack[point] = s->tokens[i];
                        level[point] = j + 1;
                        levelCount[j + 1]++;
                        point++;
                    }
                    point--;
                }
                fout << endl;
                fout2 << " " << lineNum << endl;
        }
    }
    return lineNum;
}
```

3.3 中间代码生成

```
  //生成四元式
    void fourformat(vector<string>& v, int addr, string &code)
    {        int i, jaddr;
        if(v[0][0]=='L')  //去掉标号带来的影响
            v.erase(v.begin());
        stringstream ss;
```

```
        if(v[0]=="goto"){ //goto 语句，直接 jump，生成 addr (j,-,-,jaddr)的
格式
            qDebug()<<QString::fromStdString(v[0])<<QString::fromStdStr
ing(v[1]);
            string temp=v[1]+":";
            for(i=0;i<=cnt;i++){
                if(temp==labellist[i].Lname){
                    jaddr = labellist[i].Laddr;
                    ss << addr << " (j,-,-," << jaddr <<")" << endl;
                    code=ss.str();
                }
            }
        }
        else if(v[0]=="if"){ //if 语句，条件转移，生成
addr (jrelop,x,y,jaddr)，relop 为< > <= >= == !=
            string temp=v[v.size()-1]+":";
            for(i=0;i<=cnt;i++){
                if(temp==labellist[i].Lname){
                    jaddr = labellist[i].Laddr;
                    ss << addr << " (j" << v[v.size()-
4] << "," << v[v.size()-5] << "," << v[v.size()-
3] << "," << jaddr << ")" << endl;
                    code=ss.str();
                }
            }
        }
        else if(v[1]==":="){ //赋值语句
            if(v.size()==3){ //直接赋值，三个参数
                ss << addr << " (" << v[1] << "," << v[2] << ",-," << v
[0] << ")" << endl;
                code=ss.str();
            }
            else{ //计算赋值，四个参数
                ss << addr << " (" << v[3] << "," << v[2] << "," << v[4
] << "," << v[0] << ")"<< endl;
                code=ss.str();
            }        }    } };
```

3.4 目标代码生成

```
int object::object_code()
{
    QFile target("target.asm");
```

```
    if(!target.open(QFile::WriteOnly|QFile::Text))
           qDebug()<<"error";
    QTextStream out_t(&target);
    format temp;
    for(int i=0;i<format_list.size();i++)
    {   temp=format_list[i];
        for(int i=0;i<laberlist.size();i++)
        {
            if(temp.line==laberlist[i])
                out_t<<"L"<<temp.line<<":"<<endl;
        }
        if(temp.op.toStdString()==":=")
        {
            if(temp.op1[0]>='0'&&temp.op1[0]<='9')
            {
                int reg_num=is_alloc(temp.op3);
                if(reg_num<0)//该变量未在寄存器当中,那么就加进去
                {
                    reg new_reg;
                    new_reg.name_id=namelist.indexOf(temp.op3);
                    new_reg.value=temp.op1.toInt();
                    if(next_reg>=31)
                        next_reg=1;
                    reg_list[next_reg]=new_reg;
                    out_t<<"addi $"<<next_reg++<<" $0 "<<temp.op1.toStd
String().c_str()<<endl;                }
                else//已经在寄存器当中
                { out_t<<"addi $"<<reg_num<<" $0 "<<temp.op1.toStdStrin
g().c_str()<<endl;                }
            }
            else//寄存器之间的赋值
            {   int reg_s=is_alloc(temp.op1);
                int reg_d=is_alloc(temp.op3);
                out_t<<"add $"<<reg_d<<" $"<<reg_s<<" $0"<<endl;      }
        }
        else if (temp.op.toStdString()=="+")
        {
            int reg_num=is_alloc(temp.op3);
            int reg_s1=is_alloc(temp.op1);
            int reg_s2=is_alloc(temp.op2);
            if(reg_num<0)//该变量未在寄存器当中,那么就加进去
            {
```

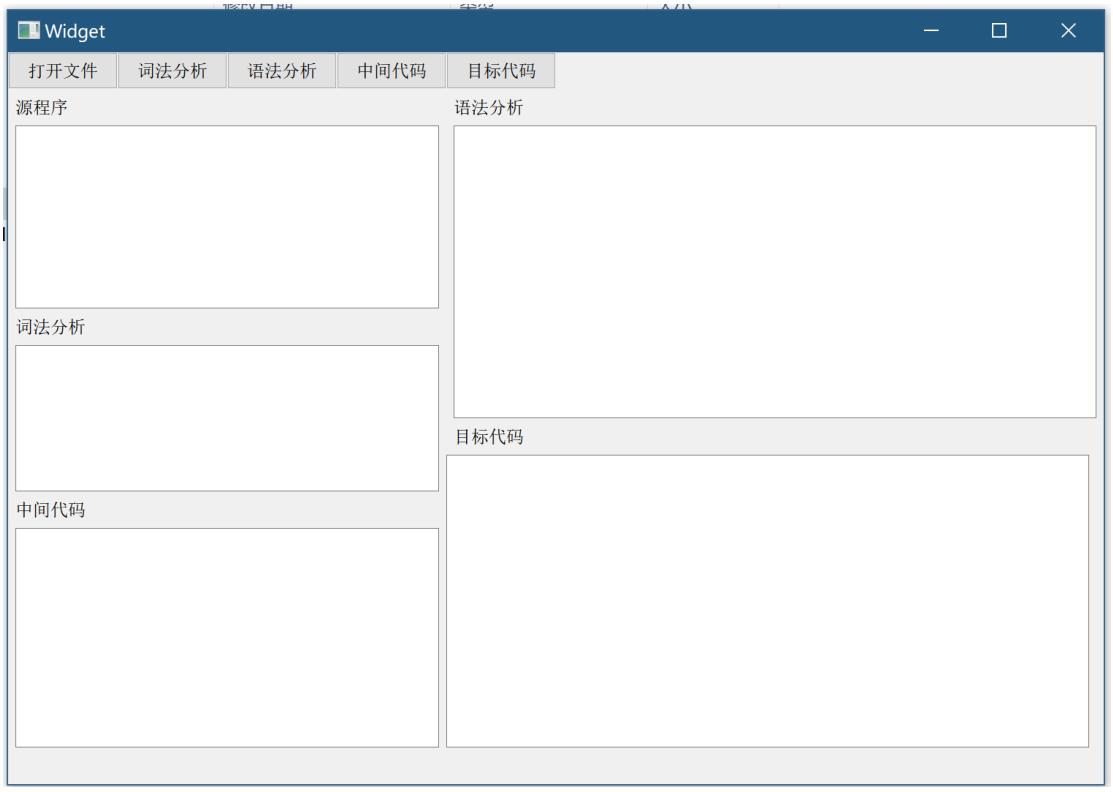
```
                reg new_reg;
                new_reg.name_id=namelist.indexOf(temp.op3);
                new_reg.value=temp.op1.toInt();
                //cout<<new_reg.name_id<<" "<<new_reg.value<<endl;
                if(next_reg>=31)
                    next_reg=1;
                reg_list[next_reg]=new_reg;
                out_t<<"add $"<<next_reg++<<" $"<<reg_s1<<" $"<<reg_s2<
<endl;            }
            else
            {out_t<<"add $"<<reg_num<<" $"<<reg_s1<<" $"<<reg_s2<<endl}
        }
        else if (temp.op.toStdString()=="-")
        {
            int reg_num=is_alloc(temp.op3);
            int reg_s1=is_alloc(temp.op1);
            int reg_s2=is_alloc(temp.op2);
            if(reg_num<0)//该变量未在寄存器当中,那么就加进去
            {
                reg new_reg;
                new_reg.name_id=namelist.indexOf(temp.op3);
                new_reg.value=temp.op1.toInt();
                if(next_reg>=31)
                    next_reg=1;
                reg_list[next_reg]=new_reg;
                out_t<<"sub $"<<next_reg++<<" $"<<reg_s1<<" $"<<reg_s2<
<endl;            }
            else
            {out_t<<"sub $"<<reg_num<<" $"<<reg_s1<<" $"<<reg_s2<<endl;
        }
        else if (temp.op.toStdString()=="*")
        {
            int reg_num=is_alloc(temp.op3);
            int reg_s1=is_alloc(temp.op1);
            int reg_s2=is_alloc(temp.op2);
            if(reg_num<0)//该变量未在寄存器当中,那么就加进去
            {
                reg new_reg;
                new_reg.name_id=namelist.indexOf(temp.op3);
                new_reg.value=temp.op1.toInt();
                if(next_reg>=31)
                    next_reg=1;
```

```
                reg_list[next_reg]=new_reg;
         out_t<<"mul $"<<next_reg++<<" $"<<reg_s1<<" $"<<reg_s2<<endl;}
            else
            {out_t<<"mul $"<<reg_num<<" $"<<reg_s1<<" $"<<reg_s2<<endl;
            }        }
        else if (temp.op.toStdString()=="/")
        {
            int reg_num=is_alloc(temp.op3);
            int reg_s1=is_alloc(temp.op1);
            int reg_s2=is_alloc(temp.op2);
            if(reg_num<0)//该变量未在寄存器当中,那么就加进去
            {
                reg new_reg;
                new_reg.name_id=namelist.indexOf(temp.op3);
                new_reg.value=temp.op1.toInt();
                if(next_reg>=31)
                    next_reg=1;
                reg_list[next_reg]=new_reg;
                out_t<<"div $"<<next_reg++<<" $"<<reg_s1<<" $"<<reg_s2<
<endl;
            }
            else
            {
                out_t<<"div $"<<reg_num<<" $"<<reg_s1<<" $"<<reg_s2<<en
dl;
            }
        }
        else if (temp.op[0]=='j')//跳转类型
        {
            laberlist.push_back(temp.op3.toInt());
            if(temp.op=="j")
            {
                out_t<<"j L"<<temp.op3.toStdString().c_str()<<endl;
            }
            else if(temp.op=="j=")
            {
                int reg_s1=is_alloc(temp.op1);
                int reg_s2=is_alloc(temp.op2);
                out_t<<"beq $"<<reg_s1<<" $"<<reg_s2<<" L"<<temp.op3.to
StdString().c_str()<<endl;
            }
            else if(temp.op=="j!=")
```

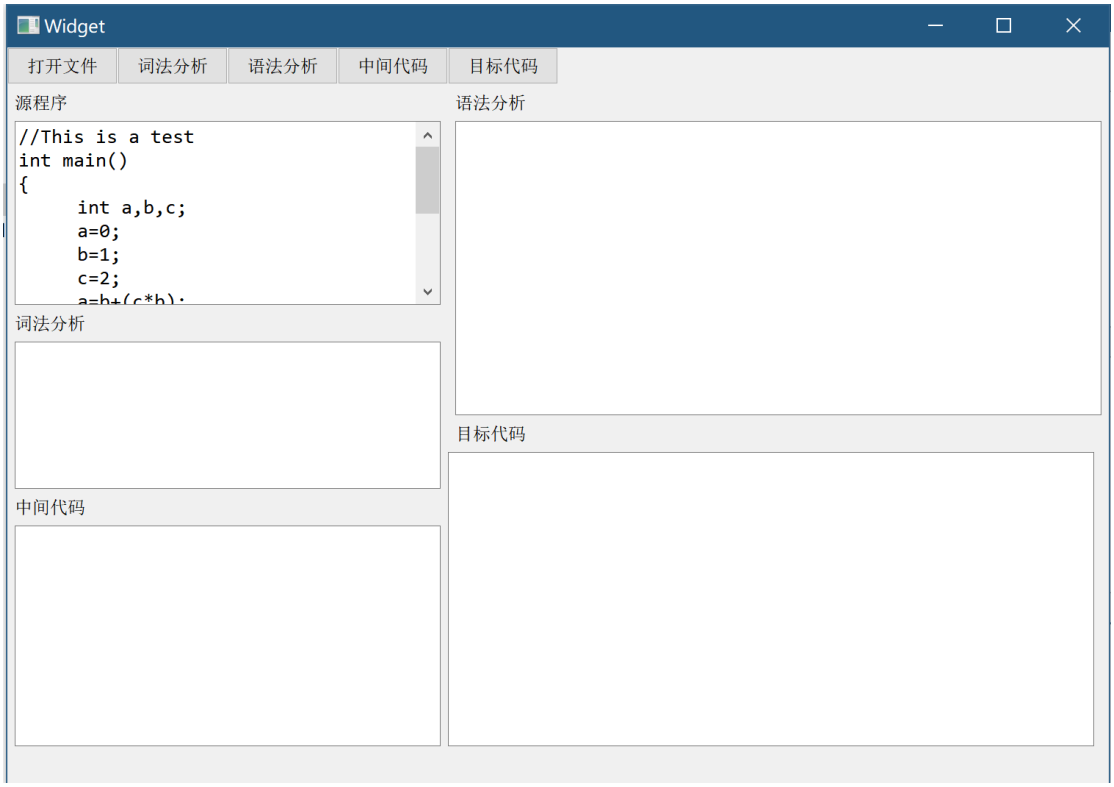
```
            {
                int reg_s1=is_alloc(temp.op1);
                int reg_s2=is_alloc(temp.op2);
                out_t<<"bne $"<<reg_s1<<" $"<<reg_s2<<" L"<<temp.op3.to
StdString().c_str()<<endl;
            }
            else if(temp.op==">")
            {
                int reg_s1=is_alloc(temp.op1);
                int reg_s2=is_alloc(temp.op2);
                out_t<<"bgt $"<<reg_s1<<" $"<<reg_s2<<" L"<<temp.op3.to
StdString().c_str()<<endl;
            }
            else if(temp.op==">=")
            {
                int reg_s1=is_alloc(temp.op1);
                int reg_s2=is_alloc(temp.op2);
                out_t<<"bge $"<<reg_s1<<" $"<<reg_s2<<" L"<<temp.op3.to
StdString().c_str()<<endl;
            }
            else if(temp.op=="<")
            {
                int reg_s1=is_alloc(temp.op1);
                int reg_s2=is_alloc(temp.op2);
                out_t<<"blt $"<<reg_s1<<" $"<<reg_s2<<" L"<<temp.op3.to
StdString().c_str()<<endl;
            }
            else if(temp.op=="<=")
            {
                int reg_s1=is_alloc(temp.op1);
                int reg_s2=is_alloc(temp.op2);
                out_t<<"ble $"<<reg_s1<<" $"<<reg_s2<<" L"<<temp.op3.to
StdString().c_str()<<endl;
            }        }    }
    target.close();
    return 0;
}
```

4.执行界面与运行结果

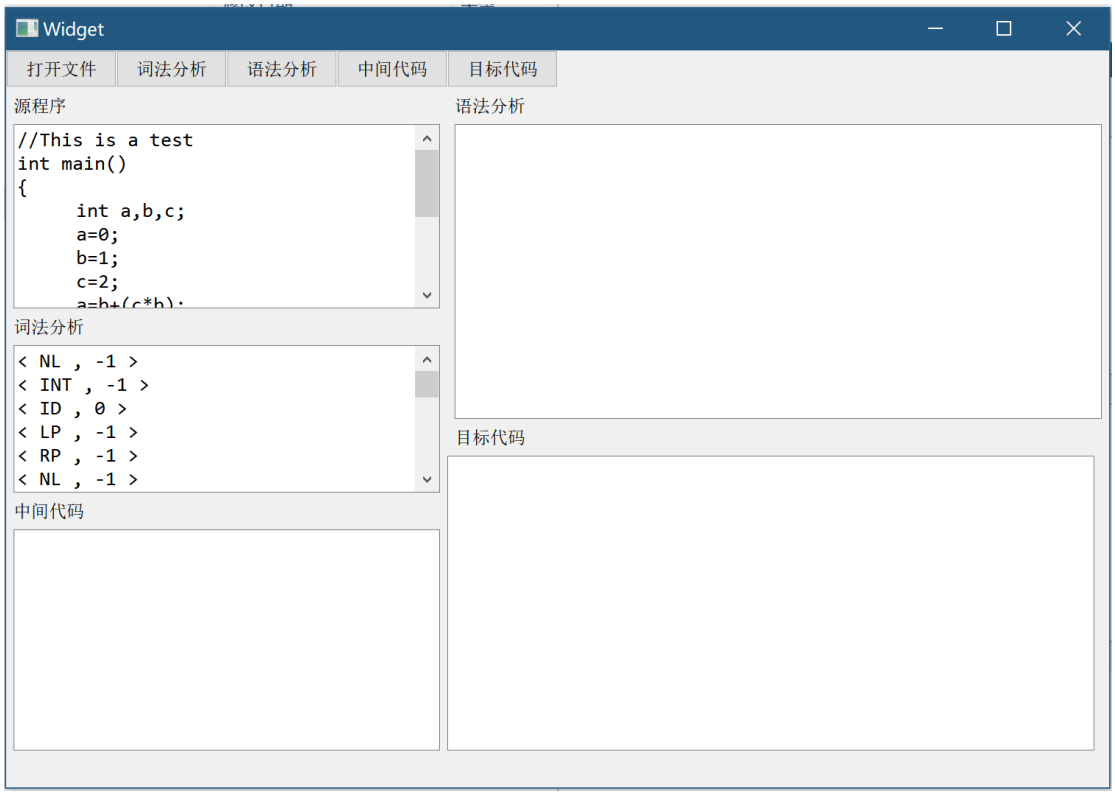
打开程序：


Widget
打开文件
词法分析
语法分析
中间代码
目标代码
源程序
语法分析
词法分析
目标代码
中间代码

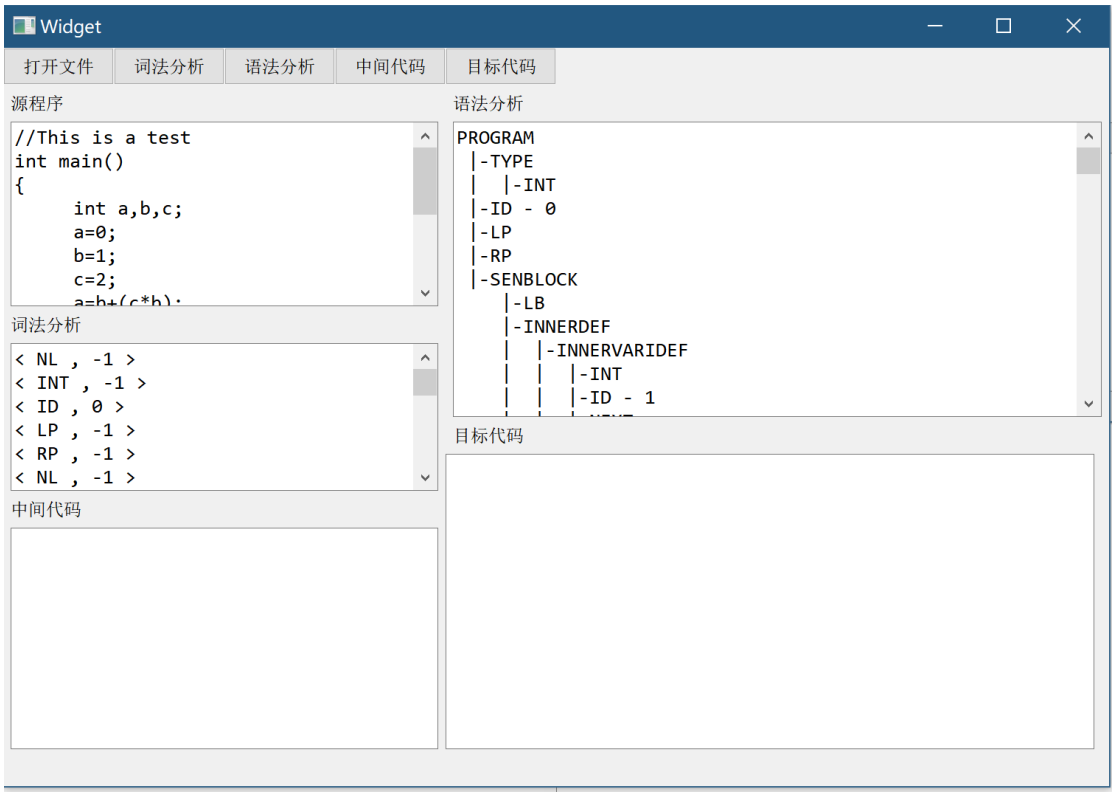
打开文件：


Widget
打开文件
词法分析
语法分析
中间代码
目标代码
源程序
//This is a test
int main()
{
int a,b,c;
a=0;
b=1;
c=2;
语法分析
词法分析
目标代码
中间代码

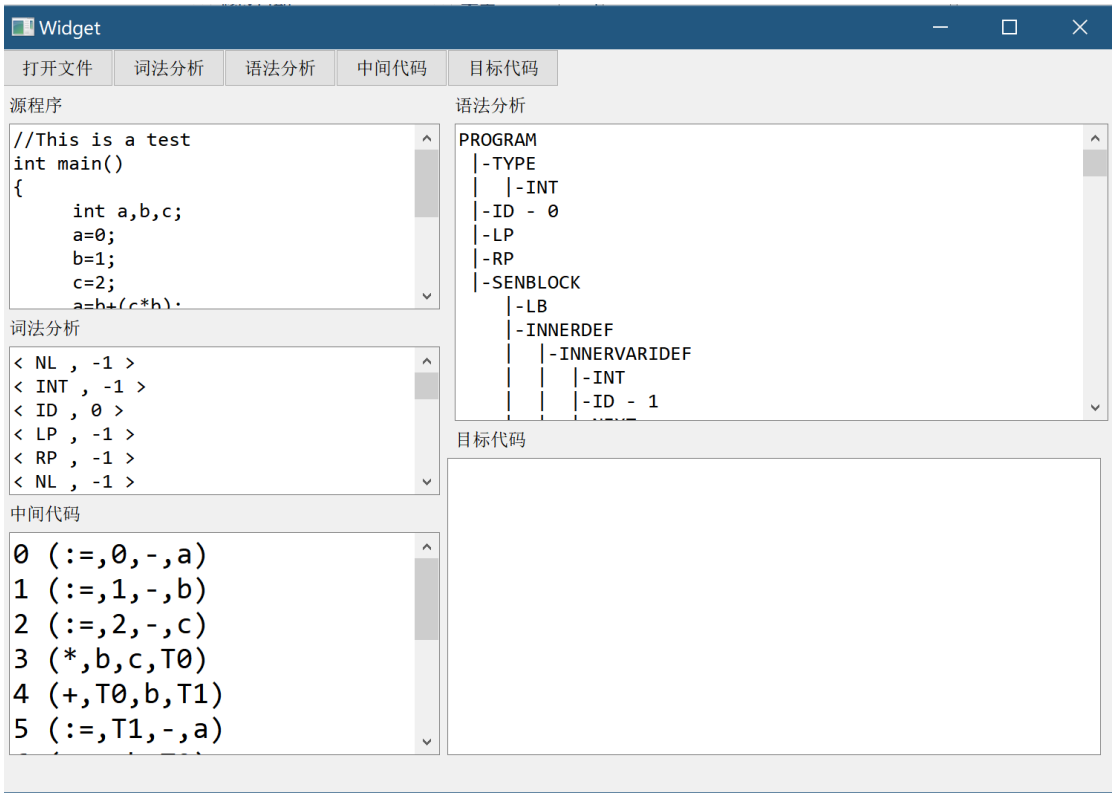
词法分析：


Widget
打开文件
词法分析
语法分析
中间代码
目标代码
源程序
//This is a test
int main()
{
int a,b,c;
a=0;
b=1;
c=2;
词法分析
< NL , -1 >
< INT , -1 >
< ID , 0 >
< LP , -1 >
< RP , -1 >
< NL , -1 >
中间代码
语法分析
目标代码

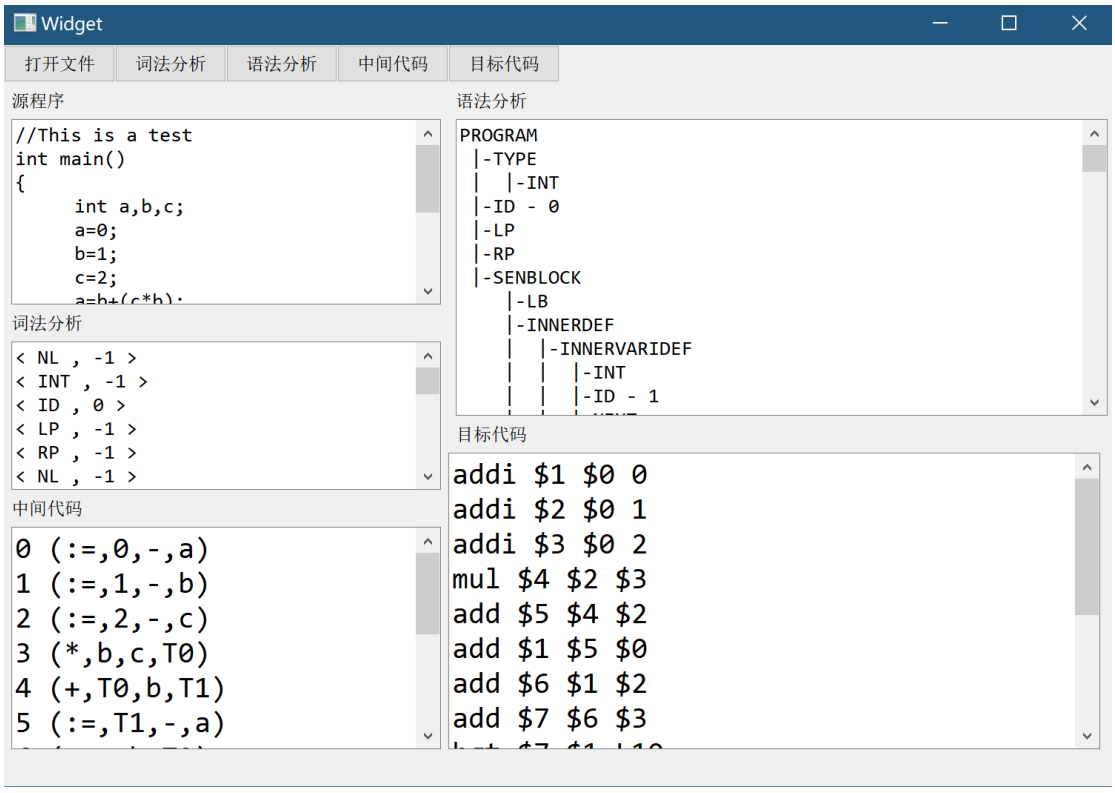
语法分析：


Widget
打开文件
词法分析
语法分析
中间代码
目标代码
源程序
//This is a test
int main()
{
int a,b,c;
a=0;
b=1;
c=2;
词法分析
< NL , -1 >
< INT , -1 >
< ID , 0 >
< LP , -1 >
< RP , -1 >
< NL , -1 >
中间代码
语法分析
PROGRAM
|-TYPE
| |-INT
|-ID - 0
|-LP
|-RP
|-SENBLOCK
|-LB
|-INNERDEF
| |-INNERVARIDEF
| | |-INT
| | |-ID - 1
目标代码

中间代码生成：



目标代码生成：



同时各种分析结果均以文件的形式输出。

5.设计体会

本次课程设计在上个学期的词法和语法分析器的基础上完成，相应的工作有

了一定程度的减少，词法分析器与语法分析器的部分基本没有太大的改动，对原本的 ui 布局进行了一些修改，新增了中间代码与目标代码部分。

我认为实验的一大难点是语法分析，在上个学期的调试中也遇到了很多问题。剩余的两部分相对语法分析来说稍简单一些，只需要明白原理，再加以代码实现即可，对应的代码也并不是很多，调试的过程也较为顺利。

在对程序进行发布时遇到了一些问题，在网上查找了对 QT 程序的发布方法后进行了几次尝试，最后得以顺利发布。