中国地质大学（北京）

编译原理课程设计

实验报告

学　　号：1004161221

姓　　名：白云鹏

目录

编译程序概述 **1**

功能 1
结构 1

词法分析器 **2**

功能 2
实现 3
测试 5

语法分析器 **6**

功能 6
实现 6
测试 11

语义分析和中间代码生成器 **13**

功能 13
实现 13
测试 18

总结 **19**

出现的问题 19

经验与展望 19

附录 **20**

语法分析器中的测试结果 20

代码 21

编译程序概述

功能

经过几周的编译原理课程设计，完成了一个简单的编译程序。编译过程分为词法分析、语法分析、语义分析和中间代码生成等，在所完成的编译程序中，完成了词法分析器，语法分析器以及语义和中间代码生成器。在词法分析器中，读入源程序，识别单词符号及其值。而在语法分析阶段，则穿插有语义分析以及中间代码生成，按照语法规则对源程序进行了产生式的推导，最后将源程序转化为四元式作为中间代码进行输出。

结构

整个编译程序则按词法分析、语法分析、语义分析和中间代码生成分为了三个阶段，但是三个阶段是相互穿插进行的。从逻辑的角度而言，其结构划分为：

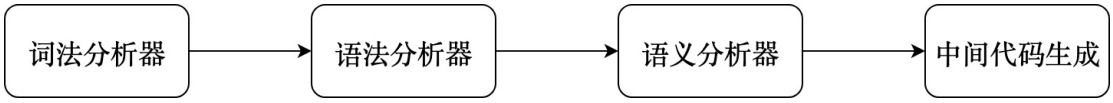

图 1 编译程序逻辑结构图

而在编写代码或者说是程序运行的角度而言，其结构则划分为：

从文件组织的角度而言，因为语法分析，语义分析以及中间代码生成是穿插进行的，因此整个结构只划分成了三个文件：

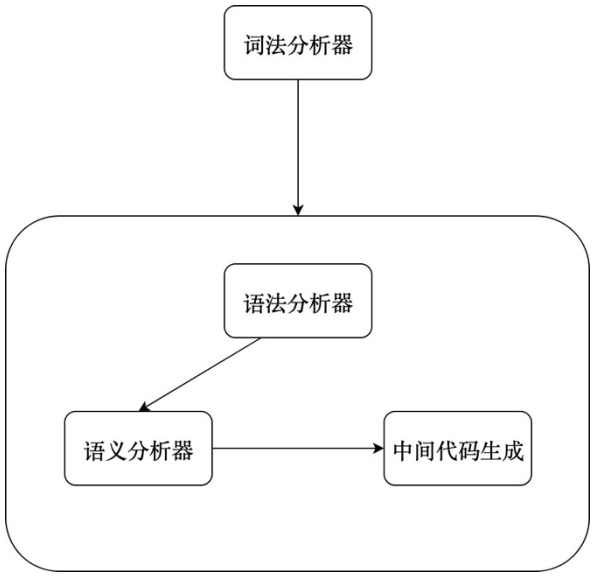

图 2 运行结构图

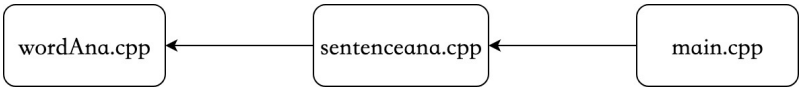

图 3 文件组织结构

main.cpp 中包含了 main 函数，他通过调用语法分析器中的接口进行语义分析等，而在语法分析器中，则是在需要的时候调用词法分析器进行单词识别。

词法分析器

功能

词法分析器是在整个 wordAna. cpp 中进行实现的。在实现的词法分析器中，通过文件输入源程序，不断识别其中的单词符号，将单词种别代码以及单词字面值存储到 wordList 类型的 word 中来进行间接返回。

代码段 1:

wordList结构体定义

```
struct wordList {
    int code = -1;
    string str="";
};
```

在词法分析器中，能够完成的任务主要包括：识别运算符，识别关键字，识别部分保留字，识别界符以及识别标识符、常量。

表格 1 识别的运算符

signs.insert('!');	signs.insert('+');	signs.insert('[');
signs.insert('&');	signs.insert('-');	signs.insert(']');
signs.insert('(');	signs.insert('*');	signs.insert('}');
signs.insert(')');	signs.insert('/');	signs.insert(',');
signs.insert(';');	signs.insert('%');	signs.insert('<');
signs.insert('{');	signs.insert('=');	signs.insert('|');
signs.insert('}');	signs.insert('>');	signs.insert(',');

表格 2 识别的关键字

rs.insert("int");	rs.insert("else");	rs.insert("if");	rs.insert("while");	rs.insert("void");

对于上面的每一个元素，其都有一个特定的单词种别值与之对应：

表格 3 单词种别值对应关系

mm["int"]	5;	mm["=="]	51;
mm["else"]	15;	mm["!="]	52;
mm["if"]	17;	mm["&&"]	53;
mm["while"]	20;	mm["||"]	54;
mm["+"]	41;	mm["++"]	56;
mm["-"]	42;	mm["!"]	55;
mm["*"]	43;	mm["--"]	57;
mm["/"]	44;	mm[")"]	82;
mm["%"]	45;	mm["("]	81;
mm["="]	46;	mm[";"]	84;
mm[">"]	47;	mm["{"]	86;
mm[">="]	48;	mm["}"]	87;
mm["<"]	49;	mm["["]	88;
mm["<="]	50;	mm[","]	90;
mm["]"]	89;		

实现

全局辅助函数说明

表格 4 辅助函数

GetChar()	缓冲区指针后移	Concat()	将缓冲区中的一个字符连接到当前单词	IsLetter(char letter)	判断当前符号是否是字母
GetBC()	读空格	IsDigit(char letter)	判断是否是数字	Reserve(string strtk)	判断是否是保留字
Retract()	回退	IsSign(char letter)	判断是否是符号		

标识符以及保留字的识别

代码段 2:

标识符以及保留字的识别

```
                    while (IsLetter(ch) || IsDigit(ch)) {
```

```
        Concat();
        GetChar();
    }
    if (ch != '#') Retract();
    int code = Reserve(strToken);
    // cout << strToken << endl;
    if (code == 0) {
        tem = InsertID(111, strToken);
    } else {
        tem = InsertID(code, strToken);
    }
    strToken.clear();
    return tem;
```

在实现的过程中，首先判断是否是字母或者是数字，如果是的话，将当前的 ch 连接到当前符号的后面。在当前符号读入完毕后，判断是否是保留字，如果是，那么设置其种别值以及字面值，否则就该符号就作为标识符进行插入。

常量的判断

代码段 3:

常量的识别

```
    while (IsDigit(ch)) {
        Concat();
        GetChar();
    }
    if (ch != '#') Retract();
    tem = InsertConst(strToken);
    strToken.clear();
    return tem;
```

常量的判断逻辑较为简单，不断读入新的字符，如果它是数字，那么就连接到当前的 z 符号，否则就停止。

符号的判断

代码段 4:

符号的识别

```
while (IsSign(ch)) {
    Concat();
    GetChar();
}
if (ch != '#') Retract();
int code = 0;
while (code == 0) {
    code = mm[strToken];
    if (code != 0) {
        tem = InsertID(code, strToken);
        strToken.clear();
        return tem;
    } else {
        Retract();
        strToken.pop_back();
    }
}
```

符号的判断则稍显复杂，因为有的符号可以连接从而构成新的操作符号，因此在判断的时候，采取了最长符号最为符号，比如&&，那么就应该识别为一个&& 而非两个&，这里在实现的时候，首先是一直读所有连续的符号，然后不断的回退缓冲区和弹出当前符号的最后一个字符，判断是否这个符号是否是合法的符号。

测试

- 测试样例

```
if(a && b) {
    while(d){
        x=j+k;
    }
} else {
    ++p;
    o%=9;
}
while(o<=90){
    o += 2;
```

图 4 测试样例

- 测试结果

```
< 17, if >
< 81, ( >
< 111, a >
< 53, && >
< 111, b >
< 82, ) >
< 86, { >
< 20, while >
< 81, ( >
< 111, d >
< 82, ) >
< 86, { >
< 111, x >
< 46, = >
< 111, j >
< 41, + >
< 111, k >
< 84, ; >
< 87, } >
< 87, } >
< 15, else >
< 86, { >
< 56, ++ >
< 111, p >
< 84, ; >
< 111, o >
< 45, % >
< 46, = >
< 100, 9 >
< 84, ; >
< 87, } >
< 20, while >
< 81, ( >
< 111, o >
< 50, <= >
< 100, 90 >
< 82, ) >
< 86, { >
< 111, o >
< 41, + >
< 46, = >
< 100, 2 >
< 84, ; >
< 87, } >
```

图 5 测试结果

语法分析器

功能

在语法分析器中，穿插着进行了语义分析以及中间代码生成。它为每一个非终结符设置了一个递归函数。主要完成的是根据词法分析器的输出，对整个程序的语法进行分析，分析其语句是如何组成的，程序中的单词符号是否按照给定的语法规则出现的。在实现的语法分析器中，能够识别出 while、if 即控制语句，赋值语句及算术运算，布尔表达式。*在语法分析器中，已经删除了需要输出的信息，完整的代码请参阅附录中的链接。*

实现

while语句的分析

如果程序读到了 while 符号，那么就会认为接下来的语句为 while 控制语句，即跳转进入 while 语句的分析。

代码段 5:

while 语句的分析

```
node stmtt;
node boolt;
word = WordAnlay();
if (word.code == -1) {
```

```
            cout << "error while";
            exit(0);
        }
        if (word.str == "(") {
            word = WordAnlay();
            if (word.code == -1) {
                cout << "error (";
                exit(0);
            }
            boolt = Bool();
            if (word.str == ")") {
                word = WordAnlay();
                if (word.code == -1) {
                    cout << "error )";
                    exit(0);
                }
                stmtt = stmt();
                node ret;
                return ret;
            } else {
                cout << "error
missing )";
                exit(0);
            }
```

在 while 语句判断的过程中，首先读入 while，表明当前语句应进行 while 语句的分析，接下来奖进行一次语法分析表示将单词指针移。按照产生式，接下来应读入的为“（”，如果该符号不为“（”，表示程序有误，否则就应进行 bool 语句的识别，在 bool 语句识别结束后，期望的应为一个“）”，在“）”读入结束后，调用 stmt 对后续进行分析。

if 语句的分析

当程序读到 if 符号的时候，会认为所分析的语句是 if-else 语句。

代码段 6:

if 语句的分析

```
        node stmt1, stmt2;
        node boolt;
        word = WordAnlay();
        if (word.code == -1) {
            cout << "error stmt";
            exit(0);
        }
        if (word.str == "(") {
            word = WordAnlay();
            if (word.code == -1)
exit(0);
            boolt = Bool();
        } else {
            cout << "error stmt";
            exit(0);
        }
        if (word.str == ")") {
            word = WordAnlay();
            if (word.code == -1)
exit(0);
            backpatch(boolt.truelist,
m1);
            stmt1 = stmt();
        } else {
            cout << "error stmt";
            exit(0);
        }
        if (word.str == "else") {
            word = WordAnlay();
```

```
        if (word.code == -1)
exit(0);
        stmt2 = stmt();
        int ret;
        node d;
            return d;
        } else {
            cout << "error";
            exit(0);
        }
```

同样的，程序首先会读到一个 if，然后期望读到“（”，然后进入布尔表达式的判定中，在结束布尔表达式判定后，也就是 if 语句中的条件已经分析结束，程序接下来会期望读到一个“）”，并继续进行剩余语句的分析。当读到 else 时，布尔表达式中的 rest4 等会直接推空，然后进行语句分析。

赋值语句的分析

如果一个语句，不是控制语句，也不是布尔表达式，那么程序会默认这个语句为赋值语句，这里认为赋值语句为含有“=”的语句。

代码段 7:

赋值语句的分析

```
node b = loc();
if (word.str == "=") {
    word = WordAnlay();
    node tt = expr();
} else {
    cout << "error missing =";
    exit(0);
}
```

这里，loc 是用来分析变量是数组的形式还是以单个变量形式出现的，在等号的右边，则应是一个算术表达式，陷入 expr 进行分析。

布尔表达式的分析

布尔表达式的形式一般为含有“<”，“<=”“>”“>=”，“==”，“!=”或者不含运算符只有变量的形式。

“==”，“！=”形式的布尔表达式分析

代码段 8:

“ == ”，“! = ”形式的布尔表达式分析

```
node rest4(node &rest4in) {
    if (word.str == "==" || word.str == "!=") {
        word = WordAnlay();
        if (word.code == -1) {
            cout << "error rest4";
            exit(0);
        }
        auto d = rel();
        auto ret = rest4(d);
        return ret;
    } else {
        auto d = rest4in;
        return d;
    }
}
```

这里在读到分析到“==”，“！=”的时候，按照文法来调用 rel，rel 可以推出一个算术表达式，而在结束后再去调用 rest4 来继续判定是否仍有布尔表达式没有被分析，如果没有，那么 rest4 将会推空。

“<”，“<=”，“>”，“>=”形式的布尔表达式分析

代码段 9:

“ < ”，“ <= ”，“ > ”，“ >= ”形式的布尔表达式分析

```
node rop_expr(node exprin) {
    if (word.str == "<" || word.str == "<=" || word.str == ">" ||
        word.str == ">=") {
        word = WordAnlay();
        if (word.code == -1) {
            cout << "error";
            exit(0);
        }
        node d = expr();
        return exprin;
    } else {
        return exprin;
    }
}
```

程序在读入符号之后，首先将字符指针移动，指向下一个符号，这里我们假定表达式的右部只能为表达式形式，调用 expr 进行接下来的分析。

算术表达式的分析

在算术表达式中，主要由 rest5, rest6, factor, term, unary 等函数组成。

代码段 10:

算术表达式分析

```
node rest5(node rest5in) {
       word = WordAnlay();
       if (word.code == -1) {
          cout << "error rest5";
          exit(0);
       }
       node termvalue = term();
       if (op == "+") {
       } else {
       }
       node rest5val =
rest5(rest5in);
       return rest5val;
   } else {
       return rest5in;
   }
}
node term() {
   node ret;
   node unaryvalue = unary();
   ret = rest6(unaryvalue);
   return ret;
}
node expr() {
   node b = term();
   node ret;
   ret = rest5(b);
   return ret;
}
```

```
node factor() {
   if (word.code == 100) {
       node t;
       t.name = word.str;
       t.result = toint(word);
       word = WordAnlay();
       return t;
   } else if (word.str == "(") {
       word = WordAnlay();
       if (word.code == -1) {
          exit(0);
       }
       node val = expr();
       if (word.str == ")") word =
WordAnlay();
       return val;
   } else if (word.code == -1 ||
word.code == 0) {
       cout << "error factor";
       exit(0);
   } else {
       node val = loc();
       if (val.offset.size() == 0) {
       } else {
          val.name = d;
       }
       return val;
   }
}
```

```
node unary() {
    node t = factor();
    return t;
}
node rest6(node rest6in) {
    if (word.str == "*" || word.str
== "/") {
        node rest6innode;
        string op = word.str;
        word = WordAnlay();
        if (word.code == -1) {
            cout << "error";
            exit(0);
        }
        if (op == "*") {
            node unaryvalue = unary();
        } else {
            node unaryvalue = unary();
            if (unaryvalue.result ==
0) {
                exit(0);
            }
        }
        node ret = rest6(rest6in);
        return ret;
    } else {
        return rest6in;
    }
}
```

其中，factor 用来分析右值持有部分，可能是一个常量，一个变量或者是一个表达式，rest6 用来处理乘法以及除法，rest5 则用来处理加法以及减法。rest5 以及 rest6 的逻辑都很相似，首先读入符号，如果是相应的运算符，那么取下一个单词之后，进行 term 操作，而 term 操作识别操作数之后首先进行 rest6 的分析，因为 rest6 分析的为乘法或除法，而它们的优先级是高于 rest5 的，因此应先计算 rest6，在 rest5 中调用的 term 过程结束后，再进行加减法的求值。

测试

- 测试数据 1

```
1  if(a <= b)
2      while(c[i,j])
3          x=j+k;
4  else
5      o= o*9;
```

- 测试 1 结果

按使用产生式结果:

```
stmts -> stmt rest0
stmt -> if(bool) stmt else
stmt
bool -> equality
equality -> rel rest4
rel -> expr rop_exp
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
relop -> <=expr
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
```

```
rest6 -> {}
rest5 -> {}
rest4 -> {}
stmt ->  while(bool) stmt
bool -> equality
equality -> rel rest4
rel -> expr rop_exp
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> [elist]
elist -> expr rest1
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
rest1 -> ,expr rest1
```

```
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
rest6 -> {}
rest5 -> {}
rop_exp -> {}rest4 -> {}
stmt -> loc = expr;
loc -> id resta
resta -> {}
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
rest6 -> {}
term -> unary rest6
unary -> factor
```

```
factor -> loc
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
stmt -> loc = expr;
loc -> id resta
resta -> {}
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
rest6 ->*unary rest6
unary -> factor
factor -> num
rest6 -> {}
rest5 -> {}
rest0 -> {}
```

按推导过程结果

由于结果过多，已移至附录。

- 测试样例 2

```
if(a <= b)
    x = b;
```

- 测试 2 结果

在该测试中，由于 if 缺少 else，因此会报错。

```
stmts -> stmt rest0
stmt ->  if(bool) stmt else
stmt
bool -> equality
equality -> rel rest4
rel -> expr rop_exp
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
```

```
rest6 -> {}
rest5 -> {}
relop -> <=expr
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
rest4 -> {}
```

```
stmt -> loc = expr;
loc -> id resta
resta -> {}
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
error
```

- 测试样例 3

```
if(a <= b)
    x
else
    b=5;
```

- 测试 3 结果

在该测试中，由于程序期望第二行应为赋值语句，但是缺少“=”，因此报错。

```
stmts -> stmt rest0
```

```
stmt -> if(bool) stmt else
stmt
bool -> equality
equality -> rel rest4
rel -> expr rop_exp
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
```

```
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
relop -> <=expr
expr -> term rest5
term -> unary rest6
unary -> factor
factor -> loc
```

```
loc -> id resta
resta -> {}
rest6 -> {}
rest5 -> {}
rest4 -> {}
stmt -> loc = expr;
loc -> id resta
resta -> {}
error missing =
```

语义分析和中间代码生成器

功能

- 选做部分

赋值表达式	数组	布尔表达式	控制语句
✓	✓	✓	✓

在进行数值演算时，需在主函数中使用 expr 函数作为开始，其余均以 stmts 函数作为开始。在此部分中，是在语法分析的基础上，穿插进行数值的计算以及中间代码的生成。与语法分析相同的部分不再赘述。

实现

返回类型定义

代码段 11:
语义分析返回类型定义

```
struct node {
    string name; //中间变量名或者变量名
    int result = 1;//数值计算结果
    int inndim;//数组维度
    string offset;//在数组中的偏移量
    string array;
    int truelist = -1;//条件为真的下一代码序号
    int falselist = -1; //条件为假的下一代码序号
    string inarray;
    int quad = -1;//代码标号
    int op4 = -1;//四元式的第四字段
    int nextlist = -1;//下一代码序号
};
```

中间代码类型定义

代码段 12:

语义分析返回类型定义

```
struct emits {
    int code;
    string str_code;
    string op1;//操作符
    string op2;//第一个操作数
    string op3;//第二个操作数
    string op4;//中间变量或者下一代码序号
    int next = -1;

    emits(string ce, string str1, string str2, string str3, string str4) {
        str_code = ce;
        op1 = str1;
        op2 = str2;
        op3 = str3;
        op4 = str4;
        code = atoi(ce.c_str());
    }//构造函数
};
```

部分辅助函数

代码段 13:

中间变量名生成函数

```
string newtemp() {
    return "T" + to_string(temp++);
}
```

代码段 14:

中间代码生成函数

```
void emit(string ce, string op1, string op2, string op3, string op4) {
    emits tem(ce, op1, op2, op3, op4);
    outstring.push_back(tem);
    ++ncode;
    ++nextquad;
}
```

代码段 14:

连接回填及归并函数

```
void backpatch(int nc, int n) {
    while (nc != -1) {
        for (auto &c : outstring) {
            if (c.code == nc) {
                c.op4 = to_string(n);
                nc = c.next;
                break;
            }
        }
    }
}
int Merge(int a2, int a1) {
    int back = a2;
    emits *pre = nullptr;
    while (a2 != -1) {
        bool flag = 0;
        for (auto &c : outstring) {
            if (c.code == a2) {
                flag = 1;
                c.op4 = to_string(a1);
                pre = &c;
                a2 = c.next;
            }
        }
        if (flag == 0) break;
    }
    a2 = back;
    int head = a1;
    // pre = nullptr;
    // pre->next = a2;
    int lc = 0;
    for (; lc < outstring.size();
++lc) {
        if (outstring[lc].code == a1)
break;
    }
    if (lc != outstring.size()) {
        while (outstring[lc].next !=
-1) { lc = outstring[lc].next; }
        outstring[lc].next = a2;
    }
    return head;
}
```

函数改写

在计算算术表达式的过程中，需要完成的有数字值的传递，这是通过综合属性来进行传递的，也就是在回溯的时候才可以获得，另一个则是数值的计算，在获得值之后进行加减乘除运算，这一部分增加的属性以及其中用到的内容将以红色注释的形式出现在代码中。

代码段 15:

expr 函数

```
node expr() {
    //expr 函数中，以 term 的返回值作为其中间
    //变量并且作为综合属性传递给 rest5，最后
    //rest5 返回的中间变量名将作为 expr 的返回值
    node ret;
    ret = rest5(term());
    return ret;
}

node term() {
    //term 函数中，首先进行 unary 保留其右值作为
    //中间变量并传递给 rest6 进行乘法/除法运算
    node ret;
    ret = rest6(unary());
    return ret;
}
```

代码段 16:

rest5 用来处理+/-的逻辑

```
node rest5(node rest5in) {
      //在 rest5 中，接受的参数为+/-左边的中间变量
       if (word.str == "+" || word.str == "-") {
            ...
       //调用 term 识别+/-右边的变量以及进行优先级高的*
或/操作
       node termvalue = term();
       //进行+/-运算，产生新的中间变量并且计算新的值
       if (op == "+") {
          node rest5innode;
          //生成新的中间变量
          rest5innode.name = newtemp();
          //记录中间代码
```

```
            emit(to_string(ncode), "+",
rest5in.name, termvalue.name,
                rest5innode.name);
            //计算值
            rest5in.result = rest5in.result +
termvalue.result;
            rest5in.name = rest5innode.name;
        } else {
            //减法操作同上
            node rest5innode;
            rest5innode.name = newtemp();
            emit(to_string(ncode), "-",
rest5in.name, termvalue.name,
                rest5innode.name);
            rest5in.result = rest5in.result -
termvalue.result;
            rest5in.name = rest5innode.name;
        }
        node rest5val = rest5(rest5in);
        return rest5val;
    } else {
        //如果 rest5 推空，说明只有+/-左边的部分，直接返
回
        return rest5in;
    }
}
```

代码段 17:

rest6 用来处理*或/的逻辑

```
node rest6(node rest6in) {
      //rest6 中的操作与 rest5 中的类似，只是操作符变为*
或/
        if (word.str == "*" || word.str == "/") {
            ...
        if (op == "*") {
           //产生新的中间变量
            rest6innode.name = newtemp();
            //识别操作符右边的变量
            node unaryvalue = unary();
            //记录中间代码
            emit(to_string(ncode), "*",
rest6in.name, unaryvalue.name,
                rest6innode.name);
            //计算值
            rest6in.result = rest6in.result *
unaryvalue.result;
            rest6in.name = rest6innode.name;
        } else {
           //除法操作同上
            node unaryvalue = unary();
            rest6innode.name = newtemp();
            emit(to_string(ncode), "/",
rest6in.name, unaryvalue.name,
                rest6innode.name);
            rest6in.result = rest6in.result /
unaryvalue.result;
            rest6in.name = rest6innode.name;
        }
        node ret = rest6(rest6in);
        return ret;
    } else {
        return rest6in;
    }
}
```

代码段 18:

处理右值

```
node factor() {
    if (word.code == 100) {
      //右值是一个常量
       ...
    } else if (word.str == "(") {
      //如果右值持有者是一个表达式，那么就调用 expr 函数
进行分析，得到的 expr 的值也保留在中间变量中
        word = WordAnlay();
        node val = expr();
        return val;
    } else if (word.code == -1 || word.code == 0)
{
        cout << "error factor";
        exit(0);
    } else {
      //右值是一个变量或者是数组中的元素
        node val = loc();
        if (val.offset.size() == 0) {
        } else {
          //产生新的中间变量
           auto d = newtemp();
           //记录中间代码
           emit(to_string(ncode), "=[]", val.name
+ "[" + val.offset + "]",
                "-", d);
           val.name = d;
        }
        return val;
    }
}
```

代码段 19:

部分逻辑判断

```
node rop_expr(node exprin) {
    if (word.str == "<" || word.str == "<=" ||
word.str == ">" || word.str == ">=") {
      //当进行到这一步时，说明已经调用过布尔表达式，那么
就需要设置其真出口以及假出口
        string sign = word.str;
      //初始设置真出口为当前值，假出口为下一值
        exprin.truelist = makelist(ncode);
        exprin.falselist = makelist(ncode + 1);
        node d = expr();
        //产生中间代码
        emit(to_string(ncode), "j" + sign,
exprin.name, d.name, "-");
```

```
        emit(to_string(ncode), "j", "-", "-", "-
");
        return exprin;
    } else {
        //执行到这里时，说明此布尔表达式只是一个变量，判断
其是否为 0 即可
        exprin.truelist = makelist(ncode);
        exprin.falselist = makelist(ncode + 1);
        emit(to_string(ncode), "jnz", exprin.name,
"-", "-");
        emit(to_string(ncode), "j", "-", "-", "-
");
        return exprin;
    }
}
```

代码段 20:

rest5 处理控制语句以及赋值语句

```
node stmt() {
    if (word.str == "if") {
        if (word.str == "(") {
            word = WordAnlay();
            if (word.code == -1) exit(0);
            boolt = Bool();
        }
        if (word.str == ")") {
            word = WordAnlay();
            if (word.code == -1) exit(0);
            m1 = M();
            //回填布尔表达式所连接的真出口为当前的代码序号
            backpatch(boolt.truelist, m1);
            stmt1 = stmt();
            nn = N();
        }
        if (word.str == "else") {
            m2 = M();
            //回填布尔表达式所连接的假出口为当前的代码序号
            backpatch(boolt.falselist, m2);
            word = WordAnlay();
            if (word.code == -1) exit(0);
            stmt2 = stmt();
            int ret;
            ret = Merge(stmt2.nextlist, nn);
            node d
            //回填布尔表达式所连接的真出口为当前的代码序号
            d.nextlist = ret;
            return d;
        }
    } else if (word.str == "while") {
        int m1quad, m2quad;
        node stmtt;
        node boolt;
        word = WordAnlay();
        if (word.str == "(") {
            word = WordAnlay();
            m1quad = M();
            //括号内应为布尔表达式
            boolt = Bool();
            if (word.str == ")") {
                word = WordAnlay();
                m2quad = M();
                stmtt = stmt();
            //回填布尔表达式所连接的真出口为当前的代码序号
                backpatch(boolt.truelist, m2quad);
            //记录中间代码
                emit(to_string(ncode), "j", "-", "-
", to_string(m1quad));
            //回填布尔表达式所连接的假出口为当前的代码序号
                backpatch(stmtt.nextlist, m1quad);
                node ret;
                ret.nextlist = boolt.falselist;
                return ret;
            }
        }
    } else {
        node ret;
        node b = loc();
        if (word.str == "=") {
            word = WordAnlay();
           //等号左边为算术表达式
            node tt = expr();
            if (b.offset.size() == 0) {
              //记录中间代码
                emit(to_string(ncode), "=",
tt.name, "-", b.name);
            } else {
              //记录中间代码
                emit(to_string(ncode), "=[],",
tt.name, "-", b.name + "[" + b.offset + "])");
            }
            ret.nextlist = makelist(nextquad);
        }
        return ret;
    }
}
```

代码段 21:

loc 处理是否是一个变量或数组

```
node loc() {
    if (word.code == 111) {
      //如果初始是一个变量，说明其为一个变量或者是数组名
       t.inarray = t.name;
       word = WordAnlay();
      //分析后续是否是数组
       node restat = resta(t);
       node ret;
       ret.name = restat.name;
       ret.offset = restat.offset;
       return ret;
    }
}
```

代码段 22:

resta 数组的后半部分([]中的部分)

```
node resta(node &b) {
    //分析是否为数组
    if (word.str == "[") {
        node elistin;
        elistin.inarray = b.inarray;
        node t = elist(elistin);
        b.name = newtemp();
       //如果是数组，需要记录计算偏移量
        emit(to_string(ncode), "-", t.array, "C",
b.name);
        b.offset = newtemp();
        emit(to_string(ncode), "*", t.offset, "w",
b.offset);
        word = WordAnlay();
        return b;
    } else {
        //如果不是数组，直接进行返回
        b.name = b.inarray;
        b.offset.clear();
        return b;
    }
}
```

测试

- 测试样例 1

  数值计算

```
1    6/2+5*8-6
```

- 测试结果

```
~ ./a.out
sc
37
~
```

- 测试样例 2

  中间代码生成

```
while(a<b)
    if(c)
        x=y+z;
    else
        x=y-z;
a=y;
o=u;
j=i;

while(a)
    while(b)
while(c)
     if(a<d)
          b=3;
     else
          d=5;
if(a[j])
    q=6;
else
    while(d)
        q=6;
a=6/b+5*c-d;
```

- 测试结果

```
0:  j<,a,b,2
1:  j,-,-,10
2:  jnz,c,-,4
3:  j,-,-,7
4:  +,y,z,T1
5:  =,T1,-,x
6:  j,-,-,0
7:  -,y,z,T2
8:  =,T2,-,x
9:  j,-,-,0
10: =,y,-,a
11: =,u,-,o
12: =,i,-,j
13: jnz,a,-,15
14: j,-,-,27
15: jnz,b,-,17
16: j,-,-,13
17: jnz,c,-,19
18: j,-,-,15
19: j<,a,d,21
20: j,-,-,23
21: =,3,-,b
22: j,-,-,17
23: =,5,-,d
24: j,-,-,17
25: j,-,-,15
26: j,-,-,13
27: -,a,C,T3
28: *,j,w,T4
29: =[],T3[T4],-,T5
30: jnz,T5,-,32
31: j,-,-,34
32: =,6,-,q
33: j,-,-,38
34: jnz,d,-,36
35: j,-,-,33
36: =,6,-,q
37: j,-,-,34
38: /,6,b,T6
39: *,5,c,T7
40: +,T6,T7,T8
41: -,T8,d,T9
42: =,T9,-,a
```

总结

出现的问题

经过几周的练习,完成了一个简单的编译器，能够做到从源程序识别单词符号，然后经过递归下降分析器进行语法分析，并且在语法分析的过程中穿插语义分析以及中间代码生成来计算表达式的值以及生成四元式形式的中间代码。

首先，编写的第一阶段是词法分析器，在刚开始实现的过程中，首先是遍历整个源程序获得全部的单词符号来供语法分析器来使用，但是在检查之后才明白这样已经是二遍扫描，在改正之后，将词法分析器作为一个过程函数，当语法分析器需要使用的时候再去调用。由此才减少了程序运行所需的时间和空间。

而语法分析器我相信是最复杂的一个子程序了，期间每一个函数都涉及相互调用，稍有不慎就可能会陷入死循环或者分析错误。在过程中只能不断的根据输出的产生式不断的去寻找函数的错误，而如果在开始阶段没有设计好函数，在后面修改时则会显得牵一发而动全身，庞大的文法往往导致难以定位错误，这给编写带来了不便。

在语义分析阶段，如果语法分析器实现的比较完善，那么在语义分析阶段只是将属性计算规则放置到合适的位置上。实现起来也比较简单。

而在四元式形式的中间代码生成上，虽然在实验四中已经完成了三元式形式的分析器，但是在四元式上却增加了代码间的跳转，如果发生错误则难以定位，因为中间所调用的任何一个过程都可能对中间代码产生修改，因此只能根据中间代码的前后代码，首先大致确定是哪一个过程然后再进行精调。

经验与展望

在大学期间，相比而言并没有很多课程想编译原理需要大量的代码，因此在编写代码过程中，经常会出现因为因为很多细小的错误导致整个代码陷入崩溃的情况，这一方面说明自己在编写的过程中仍不够细心，另一方面也说明自己在调试代码时不能快速定位，这些都是自己值得改进的地方。而代码必然会跟随自己的人生，因此在以后编写代码时，应稳扎稳打，尽量不返工，如果确实需要返工，也应根据出现的情况快速定位。

最后，通过编译原理的实验，首先是对常用的编辑器有了一个初步的了解，不再将其作为一个黑箱来使用，但是现在所用的编译器仍有很多复杂的理论没有参透，但是我认为，自己已经懂得了自上而下推导法的原理。同时，在编译器如何生成代码上也有所收获，这也对如何写出高效的代码有所启迪。由此，我相信自己在以后的学习工作过程中能够编写出更优美的代码。

附录

语法分析器中的测试结果

```
stmt rest0
if(bool) stmt else stmt rest0
if(equality) stmt else stmt rest0
if(equality) stmt else stmt rest0
if(rel rest4) stmt else stmt rest0
if(expr rop_exp rest4) stmt else stmt rest0
if(term rest5 rop_exp rest4) stmt else stmt rest0
if(term rest5 rop_exp rest4) stmt else stmt rest0
if(unary rest6 rest5 rop_exp rest4) stmt else stmt rest0
if(factor rest6 rest5 rop_exp rest4) stmt else stmt rest0
if(loc rest6 rest5 rop_exp rest4) stmt else stmt rest0
if(id resta rest6 rest5 rop_exp rest4) stmt else stmt rest0
if(idrest6 rest5 rop_exp rest4) stmt else stmt rest0
if(id rest5 rop_exp rest4) stmt else stmt rest0
if(idrop_exp rest4) stmt else stmt rest0
if(id<= expr rest4) stmt else stmt rest0
if(id<= term rest5 rest4) stmt else stmt rest0
if(id<= term rest5 rest4) stmt else stmt rest0
if(id<= unary rest6 rest5 rest4) stmt else stmt rest0
if(id<= factor rest6 rest5 rest4) stmt else stmt rest0
if(id<= loc rest6 rest5 rest4) stmt else stmt rest0
if(id<= id resta rest6 rest5 rest4) stmt else stmt rest0
if(id<= idrest6 rest5 rest4) stmt else stmt rest0
if(id<= id rest5 rest4) stmt else stmt rest0
if(id<= idrest4) stmt else stmt rest0
if(id<= id) stmt else stmt rest0
if(id<= id) while(bool) stmtelse stmt rest0
if(id<= id) while(equality) stmtelse stmt rest0
if(id<= id) while(equality) stmtelse stmt rest0
if(id<= id) while(rel rest4) stmtelse stmt rest0
if(id<= id) while(expr rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(term rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(term rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(unary rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(factor rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(loc rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id resta rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [elist] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [expr rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [term rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [term rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [unary rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [factor rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [loc rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id resta rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [idrest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [idrest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,expr rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
```

```
if(id<= id) while(id [id,term rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,term rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,unary rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,factor rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,loc rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,id resta rest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,idrest6 rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,id rest5 rest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,idrest1] rest6 rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,idrest1]rest5 rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,idrest1] rop_exp rest4) stmtelse stmt rest0
if(id<= id) while(id [id,idrest1]rest4) stmtelse stmt rest0
if(id<= id) while(id [id,idrest1]) stmtelse stmt rest0
if(id<= id) while(id [id,idrest1]) loc = expr;else stmt rest0
if(id<= id) while(id [id,idrest1]) id resta = expr;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= expr;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= term rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= term rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= unary rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= factor rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= loc rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id resta rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= idrest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + term rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + unary rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + factor rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + loc rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + id resta rest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + idrest6 rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + id rest5;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else stmt rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else loc = expr; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id resta = expr; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= expr; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= term rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= term rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= unary rest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= factor rest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= loc rest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= id resta rest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= idrest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= id* unary rest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= id* factor rest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= id* num rest6 rest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= id* numrest5; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= id* num; rest0
if(id<= id) while(id [id,idrest1]) id= id + id ;else id= id* num;
```

代码

sentenceana.cpp: https://paste.ubuntu.com/p/GyyMKH67cj/

wordAna.cpp: https://paste.ubuntu.com/p/73B5fjYrxy/

main.cpp: https://paste.ubuntu.com/p/yzrfRf5NKN/