福州大学

《编译系统设计实践》

实验项目三：语法制导翻译与生成中间代码

学号：041701320

姓名：杨鑫杰

年级：2017 级

学院：数计学院

专业：软件工程

本组其它成员：学号 221701114 姓名 张玉麟

学号 221701117 姓名 余嘉宸

学号 221701121 姓名 沈明炜

学号 221701131 姓名 郑志成

实验时间：2019－2020 学年第二学期

任课教师：陈晖

目录

1 报告概要

【摘要】

本报告将具体描述本小组在编译原理实践课程中第三次实验《语法制导翻译与生成中间代码》的完成情况，介绍实验的目的，以及开发环境，还有具体的实验内容，包括实验过程中涉及的文法，实验的设计思路，还有具体的实验要求和实验代码分析过程和对应运行文件的运行结果的展示。

同时该报告还将介绍代码部分和文档部分等的团队分工情况，以及最后附录将提供学习过程中参考书目。

关键词：语法制导翻译、中间代码生成、四元式

2 实验目的

通过语法制导或翻译模式生成中间代码。

3 实验准备

本实验的运行环境为 dev-c++，采用 c++11 新标准，将文件分为多个头文件和一个主函数 cpp。

参考书籍的语义规则描述：

产生式	语义规则
$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$
$B \rightarrow B_1 \,\&\&\, B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$
$B \rightarrow ! B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \mid\mid E_2.code$ $\mid\mid gen(\text{'if'}\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ \text{'goto'}\ B.true)$ $\mid\mid gen(\text{'goto'}\ B.false)$
$B \rightarrow$ **true**	$B.code = gen(\text{'goto'}\ B.true)$
$B \rightarrow$ **false**	$B.code = gen(\text{'goto'}\ B.false)$

图 6-37　为布尔表达式生成三地址代码

产生式	语义规则
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('goto'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('goto'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

图 6-36　控制流语句的语法制导定义

4 实验内容

4.1 设计思路

在实验二的基础上，实验二完成的是语法分析实验（根据给出的文法编制 LR（1）分析程序，以便对任意输入的符号串进行分析），在第二实验的代码基础上，增加语义规则代码，实验 2 规约时执行语义规则，构造语法树，根据遍历语法树，根据不同的语音规则，输出三地址码在文件里。

首先在实验二的基础上，在规约过程中构造语法树。

相应代码：

```
/*若不是 acc 也不是 err，说明是 s 移进或者 r 规约*/
if (tableValue[0] == 's') {
    ……..忽略前面代码……
    symbol.push_back(words[0].first); //push 进栈 symbol
    attribution.push_back(trans); //push 进栈 attribution
     ……..忽略后面代码……
}
else if(tableValue[0] == 'r'){
    ……..忽略前面代码……
    symbolStack.pop_back();
    DyingAttr.push(S.top());//DyingAttr 先记录下 S 需要 pop 的节点
    S.pop();//pop 栈 S
    ……..忽略中间代码……
    /*入栈*/
    symbolStack.push_back(make_pair(production.first, trans));
    cout<<"符号栈入栈："<<symbolStack[0].first<<endl;
    symbol.push_back(production.first); //push 栈 symbol
    cout<<"production.first:"<<production.first<<endl;
    attribution.push_back(trans); //push 栈 attribution
    S.push(cnt++); //push 栈 S
```

```
    while (!DyingAttr.empty()) {
        GraphOfAdjacencyList[cnt - 1].push_back(DyingAttr.front());
    //构造 Tree
        DyingAttr.pop();
    }
    ……..忽略后面代码……
}
```

在实验二的规约过程中，就是在实验二的基础上，具体的操作是增加一个 GraphOfAdjacencyList 的 vector 数组来存储语法树的节点，同时维护节点的结构数组来存储具体节点信息，包括地址、未赋值的四元式字符串等。

然后，根据上述构造的语法树，通过深搜算法进行遍历，遍历过程同时将对应语法的四元式进行构造，同时存储在最后需要输出的结果数组中。

```
void dfs(int u) {
    Trans &tran = attribution[u];
    /*如果为空，跳出当前循环 */
    if (GraphOfAdjacencyList[u].empty()) return; /*{，代码块的开始，则新建符号表数组*/
    if (symbol[GraphOfAdjacencyList[u].front()] == "{")
    {
        symbolTable.push_back(vector<string>());
```

```
    }

    /*遍历*/
     for(int i=0;i<GraphOfAdjacencyList[u].size();i++){
        dfs(GraphOfAdjacencyList[u][i]);
    }
    /*清空四元式内容*/
    attribution[u].code.clear();                    tran    =
attribution[GraphOfAdjacencyList[u][0]];
    /*添加儿子节点的四元式*/
    for(int i=0;i<GraphOfAdjacencyList[u].size();i++){
        for(auto                quaternion                :
attribution[GraphOfAdjacencyList[u][i]].code){
            tran.code.push_back(quaternion);
        }
    }
     if          (symbol[u]          ==          "assignment"          &&
symbol[GraphOfAdjacencyList[u][0]] == "ID") {//赋值语句
     assignment_id_function();
     }
     else          if          (symbol[u]          ==          "primary"          &&
symbol[GraphOfAdjacencyList[u][0]] == "NUM") {//规约 NUM
```

```
            primary_num_function(tran,u);
        }
        else if (symbol[u] == "primary" &&
symbol[GraphOfAdjacencyList[u][0]] == "ID") {      //规约 ID
            primary_id_function();
        }
       else if (symbol[u] == "term" && GraphOfAdjacencyList[u].size() >
1 && symbol[GraphOfAdjacencyList[u][1]] == "*") {//乘法表达式
            term_mul_function(tran,u);
        }
        else if (symbol[u] == "expr" && GraphOfAdjacencyList[u].size() >
1 && symbol[GraphOfAdjacencyList[u][1]] == "+") {//加法表达式
            expr_add_function(tran,u);

        else if (symbol[u] == "rel" && GraphOfAdjacencyList[u].size() > 1)
{//关系表达式
            rel_function(tran,u);
        }
        else if (symbol[GraphOfAdjacencyList[u][0]] == "WHILE")
{//WHILE 语句
            while_function(tran,u);
        }
```

```
    else if (symbol[GraphOfAdjacencyList[u][0]] == "IF" &&
GraphOfAdjacencyList[u].size() == 5) {//IF 语句
        if_function(tran,u);
    }
    else if (symbol[GraphOfAdjacencyList[u][0]] == "IF" &&
GraphOfAdjacencyList[u].size() == 7) {//IF ELSE 语句
        if_else_function(tran,u);
    }
    if (symbol[GraphOfAdjacencyList[u].back()] == "}")/*}，代码块的
结束，则删除符号表数组*/
    {
       symbolTable.pop_back();
    }
}
```

在遍历过程中，遍历每个节点，获取四元式中每个变量的对应值得内容，根据对应的文法类型，构造不同的四元式代码。

将变量值传递给四元式代码的构造函数中，在该函数内部将其构造成一个 pair 键值对，将地址，和值内容存储到 root 中的结构数组对应的四元式字符串区域，然后通过遍历的方式将其输出。

```
/*构造四元式*/
pair<int, vector<string>> genarateQuaternion(string a, string b, string
c, string d) {
```

```
    vector<string> vec{ a,b,c,d };
    pair<int, vector<string>> p = make_pair(address++, vec);
    return p;
}
```

构造四元式完成后，在主函数中将其输出到文件中。

```
for (pair<int, vector<string>> it : attribution[root].code){
    fout << "地址" << it.first << "对应的四元式:" << " (" <<
it.second[0] << ", " << it.second[1]<< ", " << it.second[2] << ", " <<
it.second[3] << ")" << endl;
}
```

4.2 文法说明

本次实验使用之前的文法，来进行实验内容相应的操作。

program → block

block→{ decls stmts}

decls → decls decl | ε

decl → type id;

type → type[num] | basic

stmts → stmts stmt | ε

stmt → loc=bool;

| if(bool)stmt

| if(bool)stmt else stmt

| *while(bool)stmt*

| *do stmt while(bool);*

| *break;*

| *block*

Loc → loc[bool] | *id*

bool →bool || *join* | *join*

join → join && *equality* | *equality*

equality → equality==rel | *equality* ！= *rel* | *rel*

rel → expr<expr |expr<=expr|expr>=expr|expr>expr|expr

expr → expr+term |expr-term |term

*term → term*unary|term/unary|unary*

unary→！unary | -unary | factor

factor→ (bool) | loc | num | real | true |false

4.3 实验要求和实现

在实验中，在以下的运行结果中可以看到，测试了 if *语句，*if-else *语句，*while *语句，并生成了四元式代码，在条件语句中也有类型的比较，同时使用* symbol *表存储变量地址。*

5 代码分析和运行结果

5.1 示例 **1**

因为实验一二三是紧密结合的，实验二中使用实验一产生的文件，

同时实验三也是在实验二的基础上进行进一步完成内容的结果。

代码块使用左右括号将代码区域包围起来（{代码块}），第一个测试示例为以下 txt 文件中的代码：

```
{
    int a;int b ;int c;
    a = 100;
    b = 108;
    c = 99;
    while(a < 110) {
        a = a + 1;
        if(a > 105){a = 107;}

    }
}

```

（输入文件）

根据实验一的词法分析的结果后，实验二读取实验一产生的 txt 结果文件来构建 LR（1）分析表，同时进行语法分析的过程同时生成语法树，最后实验三的实验结果如下：

```
1  100 (=, 100, -, T1)
2  101 (=, T1, -, a)
3  102 (=, 108, -, T2)
4  103 (=, T2, -, b)
5  104 (=, 99, -, T3)
6  105 (=, T3, -, c)
7  106 (=, 110, -, T4)
8  107 (j<, a, T4, 109)
9  108 (j, -, -, 118)
10 109 (=, 1, -, T5)
11 110 (+, a, T5, T6)
12 111 (=, T6, -, a)
13 112 (=, 105, -, T7)
14 113 (j>, a, T7, 115)
15 114 (j, -, -, 117)
16 115 (=, 107, -, T8)
17 116 (=, T8, -, a)
18 117 (j, -, -, 106)
```

（四元式输出文件）

首先定义了三个变量，输出四元式中对应将三个变量进行赋值语句的输出。然后将数值 100 作为值 T4，在 while 语句块中进行循环操作，如果不符合则跳转到地址 118，也就是跳出整个程序部分，满足则跳转到 109 处进行加法和赋值操作，进入 if 语句块，同样，若满足则跳转 115，不满足则进入 116，重新判断 while 语句是否满足条件，满足则继续，不满足则跳出循环。

5.2 示例 2

错误示例，在示例 1 的前提下，将 int c;语句删去，但未删去第 5 行 c 的 赋值语句，发现语句报错。

```
{
    int a;int b ;
    a = 100;
    b = 108;
    c = 99;
    while(a < 110) {
        a = a + 1;
        if(a > 105){a = 107;}

    }
}

```

```
变量c未声明

--------------------------------
Process exited after 1.023 seconds with return value 0
请按任意键继续. . .
```

（控制台输出结果）

5.3 示例 3

由于代码是在实验二基础上做修改的，所以在语法分析过程如果失败则不会进行翻译。

```
{
    int a;int b ;int c;
    a = 100;
    b = 108;
    c = 99;
    while(a  110) {
        a = a + 1;
        if(a > 105){a = 107;}

    }
}
```

（输入文件）

```
error

--------------------------------
Process exited after 1.91 seconds with return value 0
请按任意键继续. . .
```

（控制台输出结果）

5.4 示例 **4**

```
{
    int a;int b ;int c;
    a = 100;
    b = 108;
    c = 99;
    a = a + a;
    a = a * a + a;
}

```

（输入文件）

```
100 (=, 100, -, T1)
101 (=, T1, -, a)
102 (=, 108, -, T2)
103 (=, T2, -, b)
104 (=, 99, -, T3)
105 (=, T3, -, c)
106 (+, a, a, T4)
107 (=, T4, -, a)
108 (*, a, a, T5)
109 (+, T5, a, T6)
110 (=, T6, -, a)
```

（四元式输出文件）

5.5 示例 5

if-else 语句生成四元式的测试。

```
{
    int a;int b ;int c;
    a = 100;
    b = 108;
    c = 99;
    while(a < 110) {
        a = a + 1;
        if(a > 105){a = 107;}
        else {a = a + 1;}

    }
}

```

（输入文件）

```
100 (=, 100, -, T1)
101 (=, T1, -, a)
102 (=, 108, -, T2)
103 (=, T2, -, b)
104 (=, 99, -, T3)
105 (=, T3, -, c)
106 (=, 110, -, T4)
107 (j<, a, T4, 109)
108 (j, -, -, 122)
109 (=, 1, -, T5)
110 (+, a, T5, T6)
111 (=, T6, -, a)
112 (=, 105, -, T7)
113 (j>, a, T7, 115)
114 (j, -, -, 118)
115 (=, 107, -, T8)
116 (=, T8, -, a)
117 (j, -, -, 121)
118 (=, 1, -, T9)
119 (+, a, T9, T10)
120 (=, T10, -, a)
121 (j, -, -, 106)
```

（四元式输出文件）

6 团队分工

实验三的实验内容是通过语法制导或翻译模式生成中间代码。其中在实验二的基础上，根据生成语法树，同时保存对应节点信息，遍历过程时生成四元式，然后输出。整个过程分工过程具体如下：

整个过程分为实验前准备工作、代码编写工作、代码整合、代码测试工作、文档编写这五个方面的内容，前期准备工作是全组五人共同协商完成的，然后共同研讨相应的知识点部分，然后进行下一部分的分工。其中余嘉宸、张玉麟负责根据实验的代码在移进规约部分同时生成语法树，节点具体信息的数组构造和文档的编写，郑志成、杨

鑫杰分别负责翻译过程语法树的遍历，沈明炜负责对应的四元式的具体构造。然后各自代码完成后进行集体会议来进行代码整合，同时在代码编写过程中也会互相讨论数据结构的命名等等，代码测试由余嘉宸、张玉麟两人负责。

最后，大家在完成自己分工任务的过程中，遇到问题也会互相讨论，互相促进，共同进步。

7 实验总结

在本次实验过程中，秉持了小组内互相帮助的原则，首先根据复习知识进行实验前的知识点复习，然后根据不同的分工进行各自代码任务的编写工作。

在编写代码的过程中遇到了很多意想不到的各种问题，比如语法树构造过程中 if 语句和 if-else 语句的语义规则混杂，没有处理好，导致后来检查了很久也不知道什么原因导致代码的错误运行，最后通过集体代码讨论解决了；再者就是遍历语法树过程的遍历算法 return 语句放置的位置不对，导致遍历结果是错的，后来重新温习了算法内容，解决了这个问题。

通过实验三的学习，我们更清晰了翻译的任务，在经过语义分析和正确性检查后，翻译成中间代码，语法制导翻译的基本思想其实就是基于属性文法的处理过程，对单词符号串进行语法分析，构造语法分析树，然后根据需要构造属性依赖图，遍历语法树并在语法树的各结点处按语义规则进行计算。

在学习的过程中遇到的困难我们也请教了同学，也上网查询相关的知识，也希望我们小组以后在工作上、学业上也能秉持这种求学态度，积极地解决问题，积极完成各自的分工。

8 附录

8.1 参考书目

1.Alfred V.Aho等著，赵建华译《编译原理》，机械工业出版社，2009年

2.Andrew W.Appe1著，赵克佳等译《现代编译原理C语言描述》人民邮电出版社，2006年