实验三、中间代码生成

姓名：瞿久尧　　学号：120L022314

1　功能介绍及实现

1.1 Compile

编译语法树，对一般语法树生成中间代码，其中包含了处理函数定义和处理各类产生式的函数入口（包含了判断是否有声明语句）

在将产生式转换成中间代码的部分，设置了对一般的计算赋值类型的产生式的函数接口，重点写的是条件和循环语句的跳转逻辑

```
Node* fundec=tree->child->next;
/*
    VarDec → ID
    FunDec → ID LP VarList RP | ID LP RP
    VarList → ParamDec COMMA VarList | ParamDec
    ParamDec → Specifier VarDec
*/
Node* varlist=fundec->child->next->next;
fprintf(foutput,"FUNCTION %s :\n",fundec->child->str_val);
if(!varlist->isTerminal){   //varlist不为空
    while(1){
        Node* paramdec=varlist->child;
        int arg=gid++;
        fprintf(foutput,"PARAM t%d\n",arg);
        Node* vardec=paramdec->child->next;
        varnames[varptr]=vardec->child->str_val;
        vars[varptr++]=arg; //连同名字存入变量表
        if(varlist->subtype == 1)   //VarList->ParamDec
            break;
        varlist=varlist->child->next->next; //VarList → ParamDec
    }
}
Node* compst=tree->child->next->next;
Compile(compst);
return;
```

```
Node* deflist=tree->child->next;
Node* stmtlist=tree->child->next;
if(!deflist->isTerminal && deflist->type == Unterm_DefList){
    stmtlist=stmtlist->next;
    //处理deflist中的变量定义
    while(deflist != NULL){     //枚举DefList中的每一行Def
        Node* def=deflist->child;
        Node* declist=def->child->next;
        while(1){               //枚举DecList中的每一个Dec（定义的单个变量）
            Node* dec=declist->child;
            Node* vardec=dec->child;
            int var=gid++;      //新分配一个临时变量
            if(vardec->subtype == 0){   //VarDec->ID
                varnames[varptr]=vardec->child->str_val;
                vars[varptr++]=var;
                if(dec->subtype == 1){  //Dec → VarDec ASSIGNOP Exp
                    //处理赋初值表达式
                    int re=CalcExp(dec->child->next->next,0);
                    fprintf(foutput,"t%d := %s\n",var,Trans(re,0));
                }
            }
            if(declist->subtype == 0)   //DecList->Dec
                break;
            declist=declist->child->next->next; //DecList → Dec COMMA De
        }
        deflist=deflist->child->next;
    }
}
if(!stmtlist->isTerminal && stmtlist->type == Unterm_StmtList)
    Compile(stmtlist);      //递归调用处理stmtlist
```

```
case 4:{//IF LP Exp RP Stmt ELSE Stmt
    int true_label=glabel++;    //True语句体入口
    int end_label=glabel++;     //if后续语句
    Node* condexp=tree->child->next->next;
    CondExp(condexp,true_label);
    if(tree->subtype == 4)  //IF LP Exp RP Stmt ELSE Stmt
        Compile(tree->child->next->next->next->next->next->next);
    fprintf(foutput,"GOTO label%d\n",end_label);
    fprintf(foutput,"LABEL label%d :\n",true_label);
    Compile(tree->child->next->next->next->next);
    fprintf(foutput,"LABEL label%d :\n",end_label);
    break;
}
case 5:{    //WHILE LP Exp RP Stmt
    int begin_label=glabel++;   //循环入口
    int true_label=glabel++;    //循环体入口
    int end_label=glabel++;     //循环后续语句
    fprintf(foutput,"LABEL label%d :\n",begin_label);
    Node* condexp=tree->child->next->next;
    CondExp(condexp,true_label);
    fprintf(foutput,"GOTO label%d\n",end_label);    //结束循环
    fprintf(foutput,"LABEL label%d :\n",true_label);
    Compile(tree->child->next->next->next->next);
    fprintf(foutput,"GOTO label%d\n",begin_label);
    fprintf(foutput,"LABEL label%d :\n",end_label);
    break;
}
```

1.2 CalcExp

对于非条件的产生式生成计算代码并返回临时变量，其实主要处理的就是函数调用，变量赋值之类的都是简单的查找对应变量的属性值。

函数调用这里会分为自定义函数调用和 read()、write()调用

```
//特殊处理输入输出函数
if(strcmp(funcname,"read") == 0){
    int ret=gid++;
    fprintf(foutput,"READ t%d\n",ret);
    return ret;
}else if(strcmp(funcname,"write") == 0){
    fprintf(foutput,"WRITE %s\n",Trans(arglist[0],0));
    return 0;
}
for(int i=ptr - 1; i >= 0; i--)
    fprintf(foutput,"ARG %s\n",Trans(arglist[i],0));
int ret=gid++;
fprintf(foutput,"t%d := CALL %s\n",ret,funcname);
return ret;
```

```
case 5:{    //Exp->ID LP Args RP
    //Args -> Exp COMMA Args | Exp
    char* funcname=tree->child->str_val;
    Node* args=tree->child->next->next;
    static int arglist[30];
    int ptr=0;
    if(!args->isTerminal){
        while(1){
            Node* exp=args->child;
            arglist[ptr++]=CalcExp(exp,0);
            if(args->subtype == 0)  //Args->Exp
                break;
            args=args->child->next->next;
        }
    }
```

1.3 CondExp

处理 if 和 while，如果为真将跳转到指定标签

实现方式

由于在 Compile 中已经定义了 if 和 while 的逻辑结构，即跳转到那个 label，因此在这里额外写一个函数是为了处理 if 和 while 中的条件。首先会检查条件中的逻辑运算符，然后再在最后选择跳转到哪个标签

```
//处理if/while使用的条件表达式，若真则跳转到指定标签
//保证条件表达式一定为Exp RELOP Exp或者普通Exp（判断非0）
if(condexp->subtype == 14){ //Exp->Exp RELOP Exp
    int a=CalcExp(condexp->child,0);
    int b=CalcExp(condexp->child->next->next,0);
    char* rel;
    int relop=condexp->child->next->relop;
    switch(relop){
    case RELOP_EQU: rel="=="; break;
    case RELOP_NEQ: rel="!="; break;
    case RELOP_GE:  rel=">="; break;
    case RELOP_LE:  rel="<="; break;
    case RELOP_GT:  rel=">";  break;
    case RELOP_LT:  rel="<";  break;
    }
    fprintf(foutput,"IF %s %s %s GOTO label%d\n",Trans(a,0),rel,
}else{  //普通Exp，先计算然后判断是否非0
    int cond=CalcExp(condexp,0);
    fprintf(foutput,"IF %s != #0 GOTO label%d\n",Trans(cond,0),
```

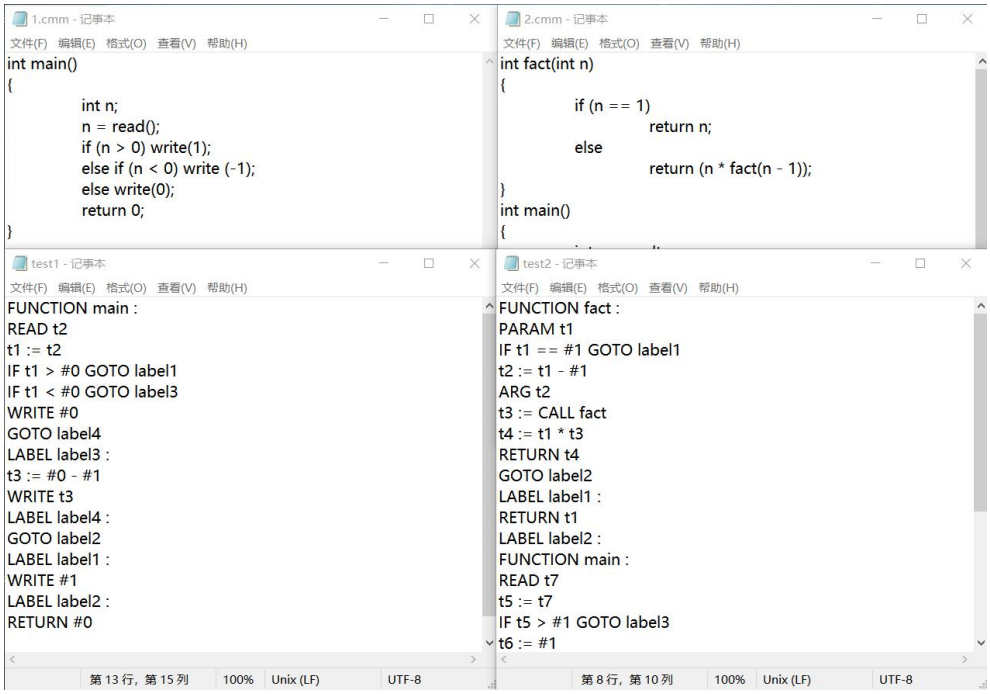
2 编译过程

在文件夹中打开终端，输出./make.sh 得到文件 cmm

然后对于要生成中间代码的文件 1.cmm,假设中间代码的结果要放在文件 test1 中，则输入 ./cmm 1.cmm test1

3 测试结果

```
/code$ ./cmm 1.cmm test1
/code$ ./cmm 2.cmm test2
/code$
```




```
FUNCTION main :
READ t2
t1 := t2
IF t1 > #0 GOTO label1
IF t1 < #0 GOTO label3
WRITE #0
GOTO label4
LABEL label3 :
t3 := #0 - #1
WRITE t3
LABEL label4 :
GOTO label2
LABEL label1 :
WRITE #1
LABEL label2 :
RETURN #0
```

运行 单步 重置 清屏

```
>请输入t2的值：-6
-1
程序执行结束，返回值为0。总执行步数：8
>
```

```
FUNCTION fact :
PARAM t1
IF t1 == #1 GOTO label1
t2 := t1 - #1
ARG t2
t3 := CALL fact
t4 := t1 * t3
RETURN t4
GOTO label2
LABEL label1 :
RETURN t1
LABEL label2 :
FUNCTION main :
READ t7
t5 := t7
IF t5 > #1 GOTO label3
t6 := #1
GOTO label4
LABEL label3 :
ARG t5
t8 := CALL fact
t6 := t8
LABEL label4 :
WRITE t6
RETURN #0
```

运行 单步 重置 清屏

```
>请输入t7的值：5
120
程序执行结束，返回值为0。总执行步数：39
>
```