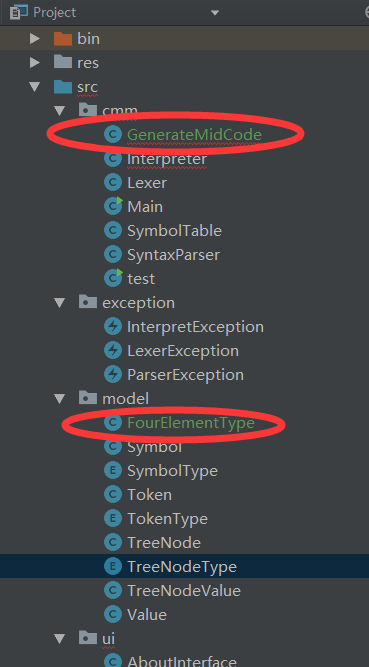
**中间代码生成--说明文档**

2015302580086 谢添

1. **任务**

完成IR 特色，如 SSA（静态单赋值）形式，也可以编译为 java 字节码、CIL（都有 现成的字节码生成库，可以直接使用）；

1. **代码结构**



1. **功能**

中间代码（Intermediate Representation或者IR）:复杂性介于源程序语言和机器语言的一种表示形式。

编译程序锁使用的中间代码有多种形式。常见的有逆波兰记号，三元式，四元式，和树形表示。

通过在词法分析，语法分析和语义分析程序的基础上，将C—源代码翻译成中间代码，认识中间代码的表示形式和生成中间代码的原理和技巧，掌握对简单赋值语句的翻译过程，从而达到对编译器的编译原理有更深的理解，提高代码能力和代码修养。

而我们这里会选择四元式——一种普遍采用的中间代码形式作为我们的中间代码，其中每条”指令“包含操作符和三个地址，两个是为运算对象的，一个是为结果的。

1. **主要设计思路**
2. 明确类型

public static final String *JUMP* = "JUMP";  
public static final String *READ* = "READ";//读取用户输入  
public static final String *WRITE* = "WRITE";//输出变量值  
public static final String *IN* = "IN";//进入大括号，level+1  
public static final String *OUT* = "OUT";//反上  
public static final String *INT* = "INT";  
public static final String *REAL* = "REAL";  
public static final String *ASSIGN* = "ASSIGN";//赋值  
public static final String *PLUS* = "PLUS";  
public static final String *MINUS* = "MINUS";  
public static final String *MUL* = "MUL";  
public static final String *DIV* = "DIV";  
public static final String *GT* = ">";  
public static final String *LT* = "<";  
public static final String *GET* = ">=";  
public static final String *LET* = "<=";  
public static final String *EQ* = "==";  
public static final String *NEQ* = "!=";//not equal

1. 构造四元式函数

（1） 创建四元式的构造函数

public FourElementType(String first, String second, String third, String forth) {  
 super();  
 this.first = first;  
 this.second = second;  
 this.third = third;  
 this.forth = forth;  
}

1. 测试

四元式形式：

public String toString() {  
 return String.*format*("%s %s %s %s", first, second, third, forth);

}

测试主函数：

package cmm;  
  
import exception.InterpretException;  
import exception.LexerException;  
import exception.ParserException;  
  
public class test {  
 public static void main(String[] args) throws ParserException, InterpretException, LexerException {  
 String file = "test/test1\_变量声明.cmm";  
 GenerateMidCode.generateCode(file);  
 }  
}

3.实现遍历：搞清楚TreeNode中存储的数据的意义, 这里按照需要生成中间代码的情况进行处理.

为了遍历树结构生成四元式，我们必须把每个子结点的方向弄清楚，主要是三个：

public TreeNode getLeft() {  
public TreeNode getMiddle() {  
public TreeNode getRight() {

我画了一个草图（不太完整，但是思路是清晰的，如下面所示：）



那么interpret(TreeNode node)里面就需要多种类型的四元式模板：switch case来实现，类型如下：

1. If语句：

case IF\_STMT:  
 interpretIfStmt(node);  
 break;

1. While语句：

case WHILE\_STMT:  
{  
 int jmpline = mLine + 1;  
 FourElementType falsejmp = new FourElementType(FourElementType.JUMP, interpretExp(node.getLeft()), null, null);  
 codes.add(falsejmp);  
 mLine++;  
 codes.add(new FourElementType(FourElementType.IN, null, null, null));  
 mLine++;  
 mLevel++;  
 interpret(node.getMiddle());  
 SymbolTable.getSymbolTable().deregister(mLevel);  
 mLevel--;  
 codes.add(new FourElementType(FourElementType.OUT, null, null, null));  
 mLine++;  
 codes.add(new FourElementType(FourElementType.JUMP, null, null, jmpline + ""));  
 mLine++;  
 falsejmp.setForth(String.valueOf(mLine + 1));  
 break;  
}

1. Read语句：

case READ\_STMT:  
{  
 String varname = node.getMiddle().getLeft().getValue();  
 SymbolType type = symbolTable.getSymbolType(varname);  
 switch (type) {  
 case SINGLE\_INT:  
 case SINGLE\_DOUBLE:  
 codes.add(new FourElementType(FourElementType.READ, null, null, varname));  
 mLine++;  
 break;  
 case ARRAY\_INT:  
 case ARRAY\_DOUBLE:  
 codes.add(new FourElementType(FourElementType.READ, null, null, varname + "[" + interpretExp(node.getMiddle().getLeft().getMiddle()) + "]"));  
 mLine++;  
 break;  
 case TEMP:  
 default:  
 throw new InterpretException("输入语句有误");  
 }  
 break;  
}

1. Write语句：

case WRITE\_STMT:  
 codes.add(new FourElementType(FourElementType.WRITE, null, null, interpretExp(node.getMiddle())));  
 mLine++;  
 break;

1. 声明语句：

case DECLARE\_STMT:  
{  
 SymbolTable table = SymbolTable.getSymbolTable();  
 TreeNode var = node.getMiddle();  
 if (node.getLeft().getMiddle() == null) {//单值  
 while(var != null) {  
 String value = null;  
 if (var.getMiddle() != null) {  
 value = interpretExp(var.getMiddle());  
 }  
 if (node.getLeft().getLeft().getType() == TreeNodeType.INT) {  
 codes.add(new FourElementType(FourElementType.INT, value, null, var.getLeft().getValue()));  
 mLine++;  
 Symbol symbol = new Symbol(var.getLeft().getValue(), SymbolType.SINGLE\_INT, mLevel);  
 table.register(symbol, var.getLineNo());  
 } else if (node.getLeft().getLeft().getType() == TreeNodeType.DOUBLE) {  
 codes.add(new FourElementType(FourElementType.REAL, value, null, var.getLeft().getValue()));  
 mLine++;  
 Symbol symbol = new Symbol(var.getLeft().getValue(), SymbolType.SINGLE\_DOUBLE, mLevel);  
 table.register(symbol, var.getLineNo());  
 }  
 var = var.getRight();  
 }  
 } else {  
 String len = interpretExp(node.getLeft().getMiddle().getMiddle());  
 while(var != null) {  
 if (node.getLeft().getLeft().getType() == TreeNodeType.INT) {  
 codes.add(new FourElementType(FourElementType.INT, null, len, var.getLeft().getValue()));  
 mLine++;  
 Symbol symbol = new Symbol(var.getLeft().getValue(), SymbolType.ARRAY\_INT, mLevel);  
 table.register(symbol, var.getLineNo());  
 } else {  
 codes.add(new FourElementType(FourElementType.REAL, null, len, var.getLeft().getValue()));  
 mLine++;  
 Symbol symbol = new Symbol(var.getLeft().getValue(), SymbolType.ARRAY\_DOUBLE, mLevel);  
 table.register(symbol, var.getLineNo());  
 }  
 var = var.getRight();  
 }  
 }  
  
 break;  
}

1. 赋值语句：

case ASSIGN\_STMT:  
{  
 String value = interpretExp(node.getLeft());  
  
 TreeNode var = node.getLeft();  
 if (var.getLeft().getMiddle() == null) {//单值  
 codes.add(new FourElementType(FourElementType.ASSIGN, value, null, var.getLeft().getValue()));  
 mLine++;  
 } else {  
 String index = interpretExp(var.getLeft().getMiddle());  
  
 codes.add(new FourElementType(FourElementType.ASSIGN, value, null, var.getLeft().getValue() + "[" + index + "]"));  
 mLine++;  
 }  
 break;  
}

遍历存储结构实现如下：

1. 条件跳转语句：

private void interpretIfStmt(TreeNode node) throws InterpretException {  
 if (node.getType() == TreeNodeType.IF\_STMT) {  
 //条件跳转 jmp 条件 null 目标 条件为假时跳转  
 FourElementType falsejmp = new FourElementType(FourElementType.JUMP, interpretExp(node.getLeft()), null, null);  
 codes.add(falsejmp);  
 mLine++;  
 codes.add(new FourElementType(FourElementType.IN, null, null, null));  
 mLine++;  
 mLevel++;  
 interpret(node.getMiddle());  
 SymbolTable.getSymbolTable().deregister(mLevel);  
 mLevel--;  
 codes.add(new FourElementType(FourElementType.OUT, null, null, null));  
 mLine++;  
 if (node.getRight() != null) {  
 FourElementType outjump = new FourElementType(FourElementType.JUMP, null, null, null);  
 codes.add(outjump);  
 mLine++;  
 falsejmp.setForth(String.valueOf(mLine + 1));  
 codes.add(new FourElementType(FourElementType.IN, null, null, null));  
 mLine++;  
 mLevel++;  
 interpret(node.getRight());  
 codes.add(new FourElementType(FourElementType.OUT, null, null, null));  
 mLine++;  
 SymbolTable.getSymbolTable().deregister(mLevel);  
 mLevel--;  
 outjump.setForth(String.valueOf(mLine + 1));  
 } else {  
 falsejmp.setForth(String.valueOf(mLine + 1));  
 }  
 }  
}

1. 表达式：

private String interpretExp(TreeNode node) throws InterpretException {  
 if (node.getType() == TreeNodeType.EXP) {  
 if(node.getMiddle() != null){  
 return interpretLogicExp(node);  
 }  
 node = node.getLeft();  
 if(node.getMiddle() != null){  
 return interpretAddtiveExp(node);  
 }  
 node = node.getLeft();  
 if(node.getMiddle() != null){  
 return interpretTermExp(node);  
 }  
 throw new InterpretException("复合表达式非法");  
 } else if (node.getType() == TreeNodeType.FACTOR) {  
 if (node.getDataType() == TokenType.MINUS) {  
 String temp = symbolTable.getTempSymbol().getName();  
 codes.add(new FourElementType(FourElementType.MINUS, interpretExp(node.getLeft()), null, temp));  
 mLine++;  
 return temp;  
 } else {  
 return interpretExp(node.getLeft());  
 }  
 } else if (node.getType() == TreeNodeType.VAR) {  
 if (node.getLeft() == null) {//单值  
 if (symbolTable.getSymbolType(node.getValue()) == SymbolType.SINGLE\_INT || symbolTable.getSymbolType(node.getValue()) == SymbolType.SINGLE\_DOUBLE) {  
 return node.getValue();  
 }  
 } else {  
 if (symbolTable.getSymbolType(node.getValue()) == SymbolType.ARRAY\_INT || symbolTable.getSymbolType(node.getValue()) == SymbolType.ARRAY\_DOUBLE) {  
 String temp = symbolTable.getTempSymbol().getName();  
 String index = interpretExp(node.getLeft());  
 codes.add(new FourElementType(FourElementType.ASSIGN, node.getValue() + "[" + index + "]", null, temp));  
 mLine++;  
 return temp;  
 }  
 }  
 } else if (node.getType() == TreeNodeType.LITREAL) {  
 return node.getValue();  
 }  
 throw new InterpretException("表达式非法");  
}

1. 逻辑比较：

private String interpretLogicExp(TreeNode node) throws InterpretException {  
 String temp = symbolTable.getTempSymbol().getName();  
 switch (node.getMiddle().getDataType()) {  
 case GT:  
 codes.add(new FourElementType(FourElementType.GT, interpretAddtiveExp(node.getLeft()), interpretAddtiveExp(node.getRight()), temp));  
 break;  
 case GET:  
 codes.add(new FourElementType(FourElementType.GET, interpretAddtiveExp(node.getLeft()), interpretAddtiveExp(node.getRight()), temp));  
 break;  
 case LT:  
 codes.add(new FourElementType(FourElementType.LT, interpretAddtiveExp(node.getLeft()), interpretAddtiveExp(node.getRight()), temp));  
 break;  
 case LET:  
 codes.add(new FourElementType(FourElementType.LET, interpretAddtiveExp(node.getLeft()), interpretAddtiveExp(node.getRight()), temp));  
 break;  
 case EQ:  
 codes.add(new FourElementType(FourElementType.EQ, interpretAddtiveExp(node.getLeft()), interpretAddtiveExp(node.getRight()), temp));  
 break;  
 case NEQ:  
 codes.add(new FourElementType(FourElementType.NEQ, interpretAddtiveExp(node.getLeft()), interpretAddtiveExp(node.getRight()), temp));  
 break;  
 default:  
 throw new InterpretException("逻辑比较非法");  
 }  
 mLine++;  
 return temp;  
}

1. 算术运算：

private String interpretAddtiveExp(TreeNode node) throws InterpretException {  
 String temp = symbolTable.getTempSymbol().getName();  
 switch (node.getMiddle().getDataType()) {  
 case PLUS:  
 codes.add(new FourElementType(FourElementType.PLUS, interpretTermExp(node.getLeft()), interpretTermExp(node.getRight()), temp));  
  
 break;  
 case MINUS:  
 codes.add(new FourElementType(FourElementType.MINUS, interpretTermExp(node.getLeft()), interpretTermExp(node.getRight()), temp));  
  
 break;  
 default:  
 throw new InterpretException("算数运算非法");  
 }  
 mLine++;  
 return temp;  
}

1. Term：注意term的ParserTree left一定是factor。

private String interpretTermExp(TreeNode node) throws InterpretException {  
 String opcode = getOpcode(node.getMiddle().getDataType());  
 String temp1 = symbolTable.getTempSymbol().getName();  
 if (node.getRight().getType() == TreeNodeType.FACTOR) {  
 codes.add(new FourElementType(opcode, interpretExp(node.getLeft()), interpretExp(node.getRight()), temp1));  
 mLine++;  
 } else {  
 codes.add(new FourElementType(opcode, interpretExp(node.getLeft()), interpretExp(node.getRight().getLeft()), temp1));  
 mLine++;  
 node = node.getRight();  
 String temp2 = null;  
 while (node.getRight() != null && node.getRight().getType() != TreeNodeType.FACTOR) {  
 opcode = getOpcode(node.getMiddle().getDataType());  
 temp2 = symbolTable.getTempSymbol().getName();  
 codes.add(new FourElementType(opcode, temp1, interpretExp(node.getRight().getLeft()), temp2));  
 mLine++;  
 node = node.getRight();  
 temp1 = temp2;  
 }  
 opcode = getOpcode(node.getMiddle().getDataType());  
 temp2 = symbolTable.getTempSymbol().getName();  
 codes.add(new FourElementType(opcode, temp1, interpretExp(node.getRight()), temp2));  
 mLine++;  
 temp1 = temp2;  
 }  
 return temp1;  
}

……

**（五） 运行分析**

首先，中间代码形式如下：

比如JMP指令:

(jmp, 条件, null, 目标)

当条件为假的时候, 跳转到目标.

又如ASSIGN指令:

(assign, 变量, null, 值)

将值赋给变量.

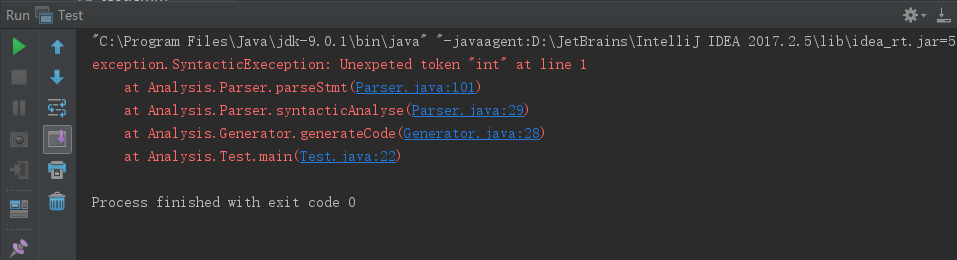
INT/REAL指令:

(int/real, null, 值/null, 变量名)

声明某个名称为变量, 如果第三个元素不为null, 代表是声明该长度的数组. 如果第二个元素不为null, 说明在声明的同时给变量赋值, 此时声明的一定是单个变量.

以下为测试用例的运行结果：

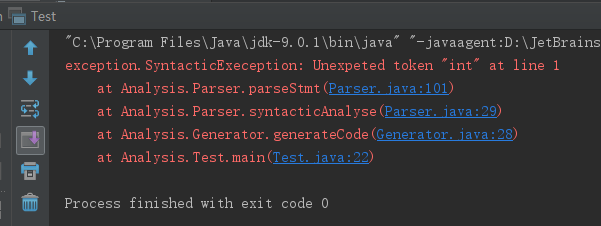
（1）error1\_ID.cmm



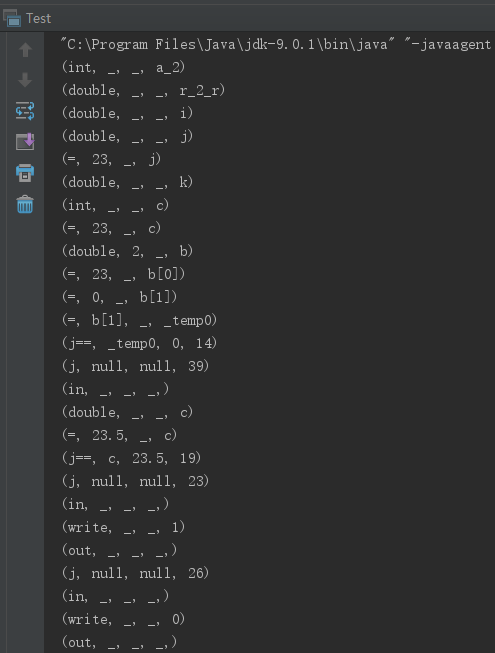
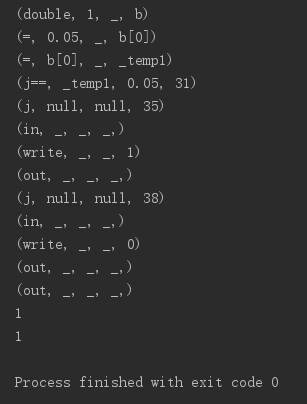
1. error2\_array.cmm



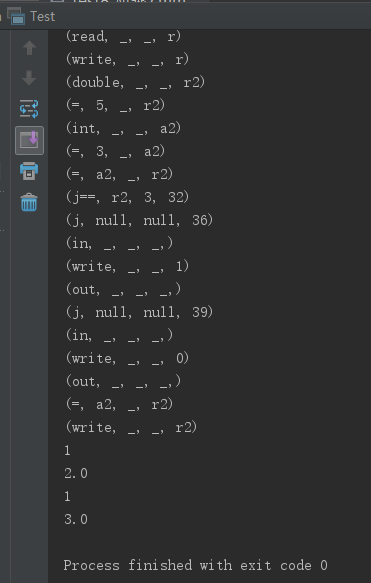
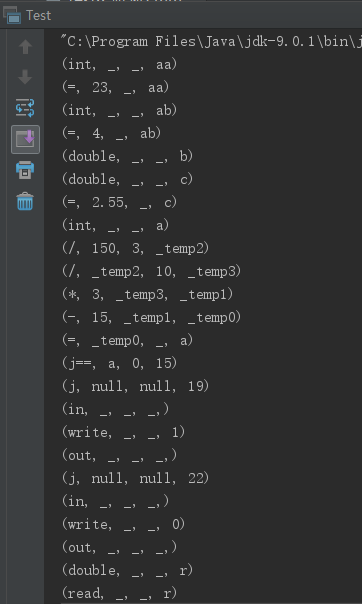
(3) error3\_comment.cmm



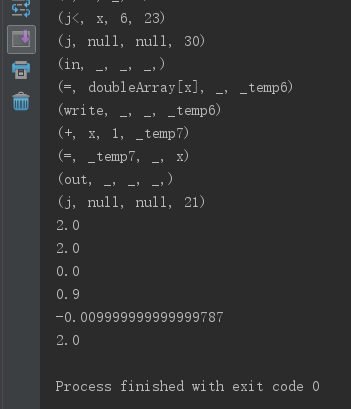
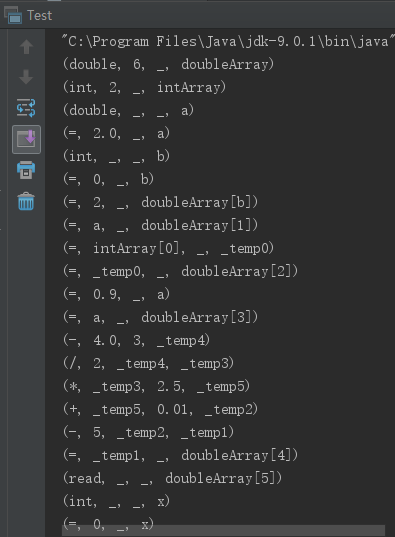
1. test1\_变量声明.cmm（太长，截取成两部分）

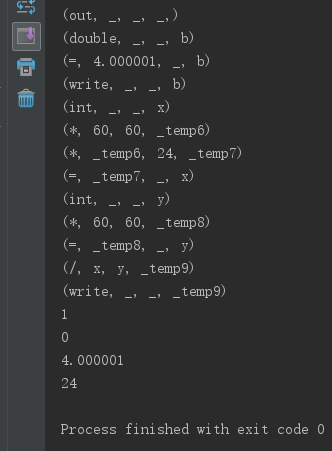
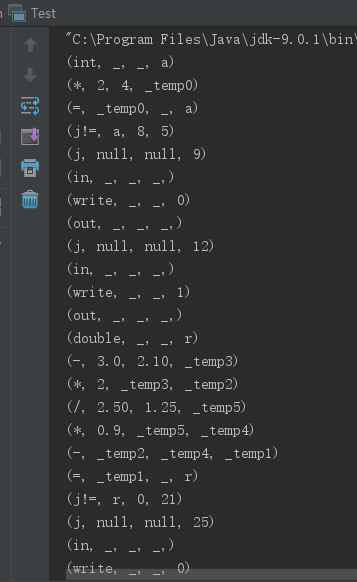
 

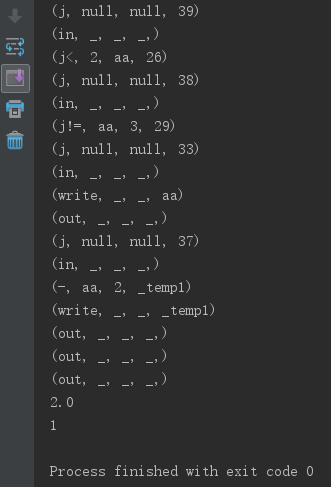
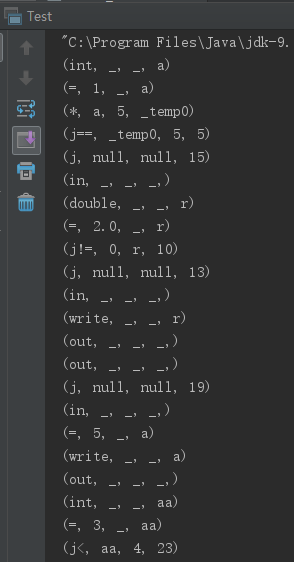
1. Test2\_一般变量赋值.cmm



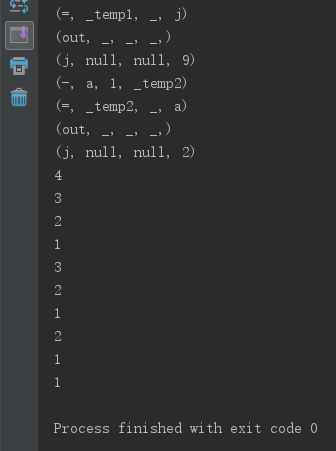
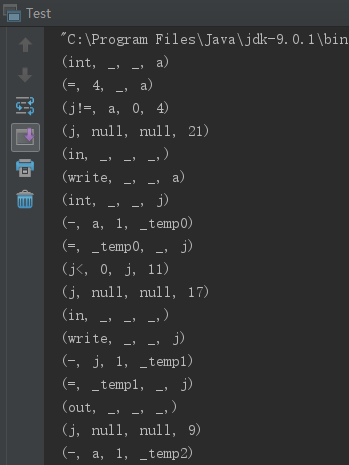
1. Test3\_数组.cmm（太长，截取成两部分）



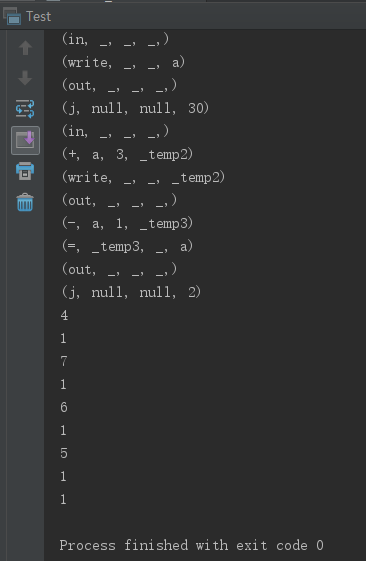
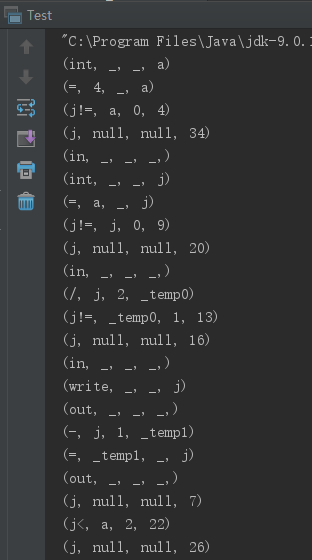
1. Test4\_算术运算（太长，截取成两部分）
2. test5\_IF\_ELSE.cmm（太长，截取成两部分）



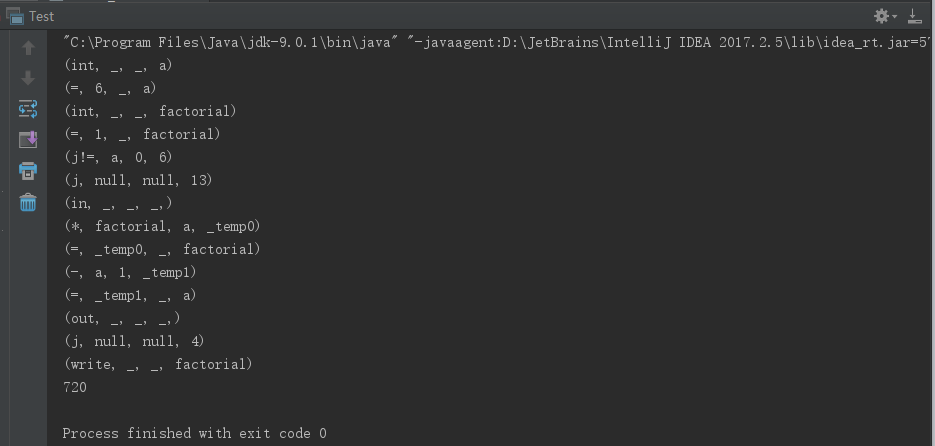
1. Test6\_WHILE（太长，截取成两部分）



1. Test7\_IF\_ELSE与WHILE（太长，截取成两部分）



（11）test8\_阶乘.cmm



（12）test9\_数组排序.cmm

