编译实习实验报告

游凌云(1700012838), 张家硕(1700012848)

lab1：Type Checking

语义检查主要检查的语义错误分类及其处理方法, 各类错误处理的具体时机如下：

- 类、方法、变量的重复定义，建符号表时处理
- 类的循环继承，遍历符号表时处理
- 声明时类型的未定义错误，遍历符号表时处理
- 方法的重载错误，遍历符号表时处理
- 使用方法、变量时的未定义错误，遍历语法树时处理
- 类型匹配错误，遍历语法树时处理

类、方法、变量的重复定义

在向符号表中插入相应元素时进行判断即可

类的循环继承

在符号表中进行遍历，寻找循环继承的类

声明时类型的未定义错误

检查范围包括

- 类继承的父类有没有定义
- 类的成员变量的类型有没有定义
- 方法的返回值的类型有没有定义
- 方法的参数类型有没有定义
- 方法的局部变量的类型有没有定义

方法的重载错误

方法的重载指父类和子类有相同名字的方法，但基调（返回值类型，参数顺序、数量、类型）不同。此处的不同是严格的不同，不考虑继承的情况。据此，检查是否有重载。

使用方法、变量时的未定义错误

检查语法树节点以及检查内容：

语法树节点	检查内容
AssignmentStatement	被赋值的变量是否被定义
ArrayAssignmentStatement	被赋值的数组变量是否被定义
MessageSend	调用方法的变量是否被定义，变量调用的方法是否被定义
PrimaryExpression	当PrimaryExpression是Identifier时，说明PrimaryExpression是一个变量。变量是否被定义
AllocationExpression	类是否被定义

类型匹配错误

检查语法树节点以及检查内容：

语法树节点	检查内容
MethodDeclaration	方法返回值类型与定义是否匹配
AssignmentStatement	赋值左右值类型是否匹配
ArrayAssignmentStatement	右值类型是否是int，左值类型是否是int[]，下标类型是否是int
IfStatement	条件表达式类型是否是boolean
WhileStatement	条件表达式类型是否是boolean
PrintStatement	输出表达式类型是否是int
AndExpression	子表达式类型是否是boolean
CompareExpression	子表达式类型是否是int
PlusExpression	子表达式类型是否是int
MinusExpression	子表达式类型是否是int
TimesExpression	子表达式类型是否是int
ArrayLookup	变量类型是否是int[]，下标类型是否是int
ArrayLength	变量类型是否是int[]
MessageSend	变量是否是类类型，方法的传入参数是否和声明参数匹配
ArrayAllocationExpression	下标是否是int
NotExpression	子表达式类型是否是boolean

lab2: Minijava to Piglet

设计思路

从面向对象到面向过程

用VTable和DTable表示一个类的实例，然后把实例的地址也作为参数传入函数调用中。具体来说，地址放在第一个参数的位置，即`TEMP 0`。

在ClassList中调用buildVDTable，即可在java中显式地创建出VTable和DTable。注意到此处的VTable的第一项还没有指向DTable的指针，在AllocationExpression中需要进一步组装。

为了实现多态，具体的设计是，一个类的VTable就是其父类的VTable，再加上自己类的成员变量。一个类的DTable就是其父类的DTable，再加上自己类的方法，如果自己类的方法与父类方法有同名，就用自己类的方法覆盖掉父类的方法。偏移量容易算出。

数组的处理

数组用地址来表示。长度为n的数组需要n+1个位置存储，第一个位置存放长度。

函数参数大于等于20个的处理

设参数有n个（n>=20），那么前18个参数放在`TEMP 1`到`TEMP 18`，然后`MOVE TEMP 19 HALLOCATE TIMES 4 MINUS n 18`，假设现在要放第k个参数（k>=19），就`HSTORE TEMP 19 TIMES 4 MINUS k 19 第k个参数的值`。在翻译时遇到参变量需要考虑其位置。

翻译

把所有的语法节点看成不同的函数。那么还需要考虑函数什么时候需要返回值，什么时候不需要。例如在赋值节点`a = b`，就可以看成这样一个函数：

```
void assignmentStatement(a, b) {
  a = b;
}
```

所以赋值语句不需要返回值。又如在ArrayLookup节点`a[i]`,就可以看成这样一个函数：

```
int arrayLookup(a, i) {
  c = a[i];
  return c;
}
```

所以在ArrayLookup节点需要返回值。又如在加法节点`a+b`,就可以看成这样一个函数：

```
int plusExpression(a, b) {
  c = a + b;
  return c;
}
```

但是piglet语言的`PLUS`语句自带返回值（不像汇编语句，即`PLUS a b`等价于`RETURN PLUS a b`），所以说虽然上面的翻译也对，但不需要，可以直接翻译成`PLUS a b`。以上的三种策略贯穿整个翻译过程。最后，遍历语法树生成piglet代码了。

代码实现

生成代码需要一些额外的格式控制，为了实现更简洁和自动的格式控制，我们实现了CodeManager类，以支持代码自动缩进等多种功能：

```
class CodeManager {

  StringBuffer sb;
  int currentTemp, currentTab, currentLabel;
  boolean flag;

  public CodeManager() {
    sb = new StringBuffer();
    currentTemp = 0;
    currentTab = 0;
    currentLabel = 0;
    flag = false;
  }
  public CodeManager(int tmp) {
    sb = new StringBuffer();
    currentTemp = tmp;
    currentTab = 0;
    currentLabel = 0;
    flag = false;
  }
  public void write(Object... ar) {
    if (ar.length == 0)
      return;
    if (flag)
      sb.append(" ");
    else {
      for (int i = 0; i < currentTab; ++i)
        sb.append(" ");
      flag = true;
    }
    sb.append(ar[0]);
    for (int i = 1; i < ar.length; ++i) {
      sb.append(" ");
      sb.append(ar[i].toString());
    }
  }
  public void writeLabel(String lbl) {
    sb.append(lbl);
    sb.append("\n");
  }
```

```
  public String getNewLabel() {
    return "label_" + (++currentLabel);
  }
  public String getNewTemp() {
    return "TEMP " + (currentTemp++);
  }
  public void writeEnd() {
    for (int i = 0; i < currentTab; ++i)
      sb.append(" ");
    sb.append("END");
    currentTab = currentTab - 1;
    flag = true;
  }
  public void writeBegin() {
    sb.append("\n");
    currentTab = currentTab + 1;
    for (int i = 0; i < currentTab; ++i)
      sb.append(" ");
    sb.append("BEGIN");
    flag = true;
  }

  public void newline() {
    sb.append("\n");
    flag = false;
  }

}
```

CodeManager类就是StringBuffer的封装，可以把它看成一个文档。在遍历语法树的时候向CodeManager中写入代码即可。

在MessageSend时，需要得知是何种类的实例在调用方法，此时由于缺少返回值，实现起来比较麻烦。实现中，我们在MessageSend节点new一个TraverseVisitor，用TraverseVisitor得知是何种类的实例。但是这体现了当前设计的弊端。后续我们在处理的时候考虑了这一问题，此后继承的都是有返回值的visitor。

以下列出所有需要考虑的语法树节点和处理流程：

Goal

把主类和其他类的代码串起来就行。

MainClass

在翻译PrintStatement的基础上加外壳。

MethodDeclaration

翻译语句块Statement，翻译返回值表达式Expression，然后加外壳。

AssignmentStatement

把右值的表达式Expression赋给左值Identifier。用`HSTORE 左值Identifier 右值Expression`或者`MOVE 左值Identifier 右值Expression`。

具体来说，考虑左值的三种情况：

1. 是类（或者考虑继承，父类）的成员。此时左值的地址应该在`TEMP 0`加一个偏移量的地方寻找。此时使用`HSTORE`。例如`HSTORE TEMP 0 4 5`。
2. 是参变量。那么需要根据参变量在何处，用`MOVE`，或者用`HSTORE`。具体在后文Identifier中分析。
3. 是方法中的局部变量。此时左值应该在`TEMP 20`到`TEMP 9999`之间。使用`MOVE`。例如`MOVE TEMP 20 5`。

在考虑左值是地址还是值的时候要注意java语言是按值传递的。对于第一种情况，类的成员变量在堆上。对于后两种情况，参变量和局部变量在栈上。对于数组或者类的情况，数组或者类的地址是参变量或者局部变量，所以在栈上，而内容在堆上。

ArrayAssignmentStatement

由于这是`a[i] = b`的情况，所以不管a是上述三种情况中的哪一种，都可以先取一个寄存器，把a[0]的地址放到新的寄存器中。 然后再计算表达式Expression，进而算出偏移量。再组装得到a[i]的地址。再用`HSTORE a[i]的地址 右值Expression`。 对于`Identifier[IndexExpression] = RightExpression`，翻译如下：

```
MOVE NEWTEMP 1 Identifier
MOVE NEWTEMP 2 PLUS 4 TIMES 4 IndexExpression
MOVE NEWTEMP 1 PLUS NEWTEMP 1 NEWTEMP 2
HSTORE NEWTEMP 1 0 RightExpression
```

IfStatement

对于`if Expression then ThenStatement else ElseStatement`，翻译如下：

```
  CJUMP Expression label_1
  ThenStatement
  JUMP label_2
label_1
  ElseStatement
label_2
  NOOP
```

WhileStatement

对于`while Expression BodyStatement`，翻译如下：

```
label_1
  CJUMP Expression label_2
  BodyStatement
  JUMP label_1
label_2
  NOOP
```

PrintStatement

对于`System.out.println(Expression)`，翻译如下：

```
Print Expression
```

AndExpression

考虑到短路机制，所以这里要采取积极的求值策略。 对于`Expression1 && Expression2`，翻译如下：

```
  BEGIN
  MOVE NEWTEMP 1 Expression1
  CJUMP NEWTEMP 1 label_1
  MOVE NEWTEMP 2 Expression2
  CJUMP NEWTEMP 2 label_1
  MOVE NEWTEMP 3 1
  JUMP label_2
label_1
  MOVE NEWTEMP 3 0
label_2
  NOOP
  RETURN NEWTEMP 3
  END
```

CompareExpression

对于`Expression1 < Expression2`，翻译如下：

```
LT Expression1 Expression2
```

PlusExpression

对于`Expression1 + Expression2`，翻译如下：

```
PLUS Expression1 Expression2
```

MinusExpression

对于`Expression1 - Expression2`，翻译如下：

```
Minus Expression1 Expression2
```

TimesExpression

对于`Expression1 * Expression2`，翻译如下：

```
Times Expression1 Expression2
```

ArrayLookup

对于`PrimaryExpression[IndexExpression]`，翻译如下：

```
BEGIN
MOVE NEWTEMP 1 PrimaryExpression
MOVE NEWTEMP 2 PLUS 4 TIMES 4 IndexExpression
MOVE NEWTEMP 1 PLUS NEWTEMP 1 NEWTEMP 2
HLOAD NEWTEMP 3 NEWTEMP 1 0
RETURN NEWTEMP 3
END
```

注意到这里数组是PrimaryExpression而不是Identifier。所以除了Identifier的上述三种情况外，还有一种情况是，PrimaryExpression是ArrayAllocationExpression的情形。显然ArrayAllocationExpression需要`RETURN`在堆上为数组创建的地址。后文会讨论Identifier和ArrayAllocationExpression的翻译。

ArrayLength

数组长度存放在第一个位置，因此偏移量为0。

对于`PrimaryExpression.length`，翻译如下：

```
BEGIN
MOVE NEWTEMP 1 PrimaryExpression
HLOAD NEWTEMP 2 NEWTEMP 1 0
RETURN NEWTEMP 2
END
```

MessageSend

对于message send，翻译如下：(省略了Param的获取以及对于20个以上Params的处理)

```
CALL
BEGIN
MOVE NEWTEMP 1 IDENTIFIER
HLOAD NEWTEMP 2 NEWTEMP 1 0
HLOAD NEWTEMP 3 NEWTEMP 2 Method.offset
RETURN NEWTEMP 3
END
(NEWTEMP 1 [.PARAMS])
```

IntegerLiteral

数字本身即是代码。

TrueLiteral

翻译为数字1。

FalseLiteral

翻译为数字0。

Identifier

分类讨论：

1. Identifier是类（或者考虑继承，父类）的成员。对于`Identifier`，翻译如下：

   BEGIN HLOAD NEWTEMP 1 TEMP 0 OFFSET RETURN NEWTEMP 1 END

2. 是参变量。首先要判断该参变量是不是在堆上。如果该参变量所在的方法的参数个数>=20个，且该参变量不是前18个，那么该参变量在堆上。 对于不在堆上的情形，`Identifier`的翻译应该在`TEMP 1`到`TEMP 19`之间。对于在堆上的情形，设该参变量是第k个，那么`Identifier`的翻译如下：

   ```
   BEGIN
   HLOAD NEWTEMP 1 TEMP 19 TIMES 4 MINUS k 19
   RETURN NEWTEMP 1
   END
   ```

3. 是方法中的局部变量。此时对于`Identifier`的翻译应该在`TEMP 20`到`TEMP 9999`之间。在ClassList中调用alloc即可为所有局部变量分配一个offset。

ThisExpression

对于`this`的翻译就是`TEMP 0`。

ArrayAllocationExpression

对于`new int[Expression]`，翻译如下：

```
BEGIN
MOVE NEWTEMP 1 Expression
MOVE NEWTEMP 2 PLUS 4 TIMES 4 NEWTEMP 1
MOVE NEWTEMP 3 HALLOCATE NEWTEMP 2
HSTORE NEWTEMP 3 0 NEWTEMP 1
RETURN NEWTEMP 3
END
```

AllocationExpression

对于new identifier( ), 翻译如下：

```
BEGIN
MOVE NEWTEMP 1 HALLOCATE 4*ClassMethodNumber
for each method:
  HSTORE NEWTEMP 1 Method.offset Method.PigletName
MOVE NEWTEMP 2 HALLOCATE 4*ClassVarNumber+4
for each var:
  HSTORE NEWTEMP 2 var.offset 0
HSTORE NEWTEMP 2 0 NEWTEMP 1
RETURN NEWTEMP 2
END
```

NotExpression

对于`!Expression`，翻译如下：

```
MINUS 1 Expression
```

lab3 : Piglet to Spiglet

设计思路

主要的翻译思路是：对expression先求值，再new一个temp，把求出的值move到new temp中（同时输出代码），然后返回new temp。对statement，先遍历其中的expression，得到存放其值的temp，然后再按照语法规则翻译statement。如对于`op() exp1() exp2()`的翻译如下：

```
String tmp1 = n.f1.accept(this);
String tmp2 = n.f2.accept(this);
String tmp3 = this.document.getNewTemp();
document.write("MOVE", tmp3, op, tmp1, tmp2);
```

代码实现

输出代码同样需要一些额外的格式控制（CodeManager类）。由于spiglet嵌套求值几乎消失，所以输出代码时一行一行地输出，而不是像piglet中一样考虑生成statement代码才换行，生成expression代码不换行。

对于语法树节点Temp和Label需要特殊处理。Temp节点不应该用上述的翻译思路，new一个temp，然后把当前temp的值move到new temp中，而是应该直接返回当前temp。Label节点应该分情况讨论是函数的名字还是jump的标签，对于函数的名字，应该和数字常量同等对待，即new一个temp，把常量值move进new temp，然后再返回new temp。对于jump的标签，应该直接生成同样的代码。

具体的实现位于Piglet2SpigletVisitor.java:

```
package visitor;

import java.util.Enumeration;

import syntaxtree.*;

public class Piglet2SpigletVisitor extends GJNoArguDepthFirst<String> {
  CodeManager document = null;
  boolean inStmt=false;
  public Piglet2SpigletVisitor() {
    this.document = new CodeManager();
  }

  public void setDocumentCurrentTemp(int a) {
    this.document.currentTemp = a;
    return;
  }

  public int getDocumentCurrentTemp() {
    return this.document.currentTemp;
  }

  public String getCode() {
    return this.document.sb.toString();
  }

  /**
   * f0 -> "MAIN" f1 -> StmtList() f2 -> "END" f3 -> ( Procedure() )* f4 ->
<EOF>
   */
  public String visit(Goal n) {
    this.document.write("MAIN");
    document.newline();

    n.f1.accept(this);

    document.write("END");
    document.newline();
```

```
    n.f3.accept(this);

    return null;
  }

  /**
   * f0 -> ( ( Label() )? Stmt() )*
   */
  public String visit(StmtList n) {
    n.f0.accept(this);
    return null;
  }

  /**
   * f0 -> Label() f1 -> "[" f2 -> IntegerLiteral() f3 -> "]" f4 -> StmtExp()
   */
  public String visit(Procedure n) {
    document.newline();

    document.write(n.f0.f0.tokenImage, "[", n.f2.f0.tokenImage, "]");
    document.writeBegin();
    document.newline();

    String tmp1 = n.f4.accept(this);

    document.write("RETURN", tmp1);
    this.document.newline();

    document.writeEnd();
    this.document.newline();

    return null;
  }

  /**
   * f0 -> NoOpStmt() | ErrorStmt() | CJumpStmt() | JumpStmt() | HStoreStmt()
|
   * HLoadStmt() | MoveStmt() | PrintStmt()
   */
  public String visit(Stmt n) {
    this.inStmt=true;
    n.f0.accept(this);
    this.inStmt=false;
    return null;
  }

  /**
   * f0 -> "NOOP"
```

```
   */
  public String visit(NoOpStmt n) {
    this.document.write("NOOP");
    this.document.newline();

    return null;
  }

  /**
   * f0 -> "ERROR"
   */
  public String visit(ErrorStmt n) {
    this.document.write("ERROR");
    this.document.newline();

    return null;
  }

  /**
   * f0 -> "CJUMP" f1 -> Exp() f2 -> Label()
   */
  public String visit(CJumpStmt n) {
    String tmp1 = n.f1.accept(this);

    document.write("CJUMP", tmp1, n.f2.f0.tokenImage);
    this.document.newline();

    return null;
  }

  /**
   * f0 -> "JUMP" f1 -> Label()
   */
  public String visit(JumpStmt n) {
    this.document.write("JUMP", n.f1.f0.tokenImage);
    this.document.newline();

    return null;
  }

  /**
   * f0 -> "HSTORE" f1 -> Exp() f2 -> IntegerLiteral() f3 -> Exp()
   */
  public String visit(HStoreStmt n) {
    String tmp1 = n.f1.accept(this);

    String tmp2 = n.f3.accept(this);

    this.document.write("HSTORE", tmp1, n.f2.f0.tokenImage, tmp2);
```

```
    this.document.newline();

    return null;
  }

  /**
   * f0 -> "HLOAD" f1 -> Temp() f2 -> Exp() f3 -> IntegerLiteral()
   */
  public String visit(HLoadStmt n) {
    String tmp1 = n.f1.accept(this);

    String tmp2 = n.f2.accept(this);

    this.document.write("HLOAD", tmp1, tmp2, n.f3.f0.tokenImage);
    document.newline();

    return null;
  }

  /**
   * f0 -> "MOVE" f1 -> Temp() f2 -> Exp()
   */
  public String visit(MoveStmt n) {
    String tmp1 = n.f1.accept(this);

    String tmp2 = n.f2.accept(this);

    this.document.write("MOVE", tmp1, tmp2);
    this.document.newline();

    return null;
  }

  /**
   * f0 -> "PRINT" f1 -> Exp()
   */
  public String visit(PrintStmt n) {
    String tmp1 = n.f1.accept(this);

    this.document.write("PRINT", tmp1);
    this.document.newline();

    return null;
  }

  /**
   * f0 -> StmtExp() | Call() | HAllocate() | BinOp() | Temp() |
IntegerLiteral()
   * | Label()
```

```
   */
  public String visit(Exp n) {
    return n.f0.accept(this);
  }

  /**
   * f0 -> "BEGIN" f1 -> StmtList() f2 -> "RETURN" f3 -> Exp() f4 -> "END"
   */
  public String visit(StmtExp n) {
    n.f1.accept(this);

    String tmp1 = n.f3.accept(this);

    return tmp1;
  }

  /**
   * f0 -> "CALL" f1 -> Exp() f2 -> "(" f3 -> ( Exp() )* f4 -> ")"
   */
  public String visit(Call n) {
    String tmp1 = n.f1.accept(this);
    String tmp2 = this.document.getNewTemp();

    StringBuffer argList = new StringBuffer();
    if (n.f3.present()) {
      for (Enumeration<Node> e = n.f3.elements(); e.hasMoreElements();) {
        String arg = e.nextElement().accept(this);
        argList.append(arg + " ");
      }
    }

    this.document.write("MOVE", tmp2, "CALL", tmp1, "(",
argList.toString().trim(), ")");
    this.document.newline();

    return tmp2;
  }

  /**
   * f0 -> "HALLOCATE" f1 -> Exp()
   */
  public String visit(HAllocate n) {
    String tmp1 = n.f1.accept(this);
    String tmp2 = this.document.getNewTemp();

    document.write("MOVE", tmp2, "HALLOCATE", tmp1);
    this.document.newline();

    return tmp2;
```

```
}

/**
 * f0 -> Operator() f1 -> Exp() f2 -> Exp()
 */
public String visit(BinOp n) {
  String tmp1 = n.f1.accept(this);

  String tmp2 = n.f2.accept(this);

  String tmp3 = this.document.getNewTemp();
  String op = new String();
  if (n.f0.f0.which == 0) {
    op = "LT";
  } else if (n.f0.f0.which == 1) {
    op = "PLUS";
  } else if (n.f0.f0.which == 2) {
    op = "MINUS";
  } else {
    op = "TIMES";
  }

  document.write("MOVE", tmp3, op, tmp1, tmp2);
  this.document.newline();

  return tmp3;
}

/**
 * f0 -> "TEMP" f1 -> IntegerLiteral()
 */
public String visit(Temp n) {
  String res = "TEMP " + n.f1.f0.tokenImage;
  return res;
}

/**
 * f0 -> <INTEGER_LITERAL>
 */
public String visit(IntegerLiteral n) {
  String tmp = this.document.getNewTemp();
  this.document.write("MOVE", tmp, n.f0.tokenImage);
  this.document.newline();
  return tmp;
}

/**
 * f0 -> <IDENTIFIER>
 */
```

```
  public String visit(Label n) {
    String s = n.f0.tokenImage;
    if (!this.inStmt) {
      this.document.writeLabel(s);
      return s;
    } else {
      String tmp = this.document.getNewTemp();
      this.document.write("MOVE", tmp, n.f0.tokenImage);
      this.document.newline();
      return tmp;
    }
  }


}

class CodeManager {

  StringBuffer sb;
  int currentTemp, currentTab, currentLabel;
  boolean flag;

  public CodeManager() {
    sb = new StringBuffer();
    currentTemp = 0;
    currentTab = 0;
    currentLabel = 0;
    flag = false;
  }

  public CodeManager(int tmp) {
    sb = new StringBuffer();
    currentTemp = tmp;
    currentTab = 0;
    currentLabel = 0;
    flag = false;
  }

  public void write(Object... ar) {
    if (ar.length == 0)
      return;
    if (flag)
      sb.append(" ");
    else {
      for (int i = 0; i < currentTab; ++i)
        sb.append("\t");
      flag = true;
    }
    sb.append(ar[0]);
```

```
    for (int i = 1; i < ar.length; ++i) {
      sb.append(" ");
      sb.append(ar[i].toString());
    }

  }

  public void writeLabel(String lbl) {
    sb.append(lbl);
    sb.append("\n");
  }

  public String getNewLabel() {
    return "label_" + (++currentLabel);
  }

  public String getNewTemp() {
    return "TEMP " + (currentTemp++);
  }

  public void writeEnd() {
    for (int i = 0; i < currentTab; ++i)
      sb.append("\t");
    sb.append("END");
    currentTab = currentTab - 1;
    flag = true;
  }

  public void writeBegin() {
    sb.append("\n");
    currentTab = currentTab + 1;
    for (int i = 0; i < currentTab; ++i)
      sb.append("\t");
    sb.append("BEGIN");
    flag = true;
  }

  public void newline() {
    sb.append("\n");
    flag = false;
  }

}
```

lab4 : Spiglet to Kanga

kanga简介

局部变量变成了寄存器。寄存器有限，共24个。用途有：传递参数，获得返回值，存放局部/临时值。

寄存器是全局的。可以理解成一个全局的数组。这意味着在函数嵌套调用中，内层的函数对寄存器的修改会对外层函数造成影响。一方面，这使得参数和返回值的传递成为可能；另外一方面，内层的函数应该有保存/恢复存放局部/临时值的寄存器的责任，外层的函数应该有保存/恢复存放参数的寄存器的责任（在每次函数调用前后），当然如果后面再也不使用存放参数的寄存器可以不保存/恢复（事实上也就是这么实现的）。

有运行时栈。每层函数调用有自己独立的运行时栈。可以理解成一个局部的数组，但外层的函数可以且仅可以通过PASSARG来写入内层函数的运行时栈。内层函数可以用HLOAD和HSTORE对自己的运行时栈进行读写。

设计思路

寄存器有限就意味着要设计一些方法，建立spiglet中的局部变量与寄存器的映射关系。当两个局部变量不冲突，他们就可以使用同一个寄存器。而如何建立这种映射关系就变成一种图染色问题。而如何求两个局部变量是否冲突则是活性分析的内容。活性分析的准备工作是建立流图，对流图中的基本块求出def和use集合。流图是以函数为单位的（有多少个函数，就有多少个流图），因此可以得到一个层次结构：整体环境/代码 -> 流图/函数 -> 基本块/statement（为了实现的简便，不求最大基本块，而是简单地把一条statement当作一个基本块）。所以最初步的目标是建立层次结构，求出基本块的def和use集合。再建立流图并进行活性分析，求出基本块的in和out集合。最后根据上述信息建立干涉图，做图染色进行寄存器分配。

代码实现

建立层次结构

遍历spiglet代码的语法树，建立层次结构。最基本的单位是statement（包括call和return），对于每个statement，求出def和use集合。按照定义（定义见龙书P390），对于`MOVE TEMP EXP`和`HLOAD TEMP TEMP2 INT`，`TEMP`都在def集中。其余的所有statement中用到的TEMP都在use集中。除了求statement的def和use集合外，还可以记录一下statement的类型，跳转的label等其他信息。

建立流图

主要工作是求出执行每条statement后下一条执行什么语句（即后继，CJUMP有两条可能的语句）。建立了层次结构后，由于记录了一些额外的信息，所以可以比较容易地求出每个statement的后继。

除此以外，还可以求一些额外的信息，如记录该函数中所有用到的变量（所有statement的use和def集合的并集，为了方便后续的干涉图的建立），记录这些变量是否在循环中被用到（如果一条语句的后继在它的前面，说明从它的后继到自己这段代码很可能在循环中，这段代码中被用到的所有的局部变量（即use和def集合）就也可能循环中）。

活性分析

由于我们已经求出每条statement的后继，use和def集合，所以直接应用数据流后向分析算法即可。

建立干扰图

每个函数的第一条statement的in集合中任意两个变量是相互冲突的。每条语句的out集合并def集合中任意两个变量是相互冲突的。

把每个变量看成一个点（在流图中我们记录了所有的变量，因此建图是方便的），对于冲突的两个变量连边。最后就转化成一个图染色问题。注意要去重边。

另外一个问题是要根据变量的作用不同而不同地对待。对于参变量，由于只能用寄存器传递前4个参数，所以后面的参变量直接用PASSARG溢出到栈上。由于我们不愿意在外层的函数保存/恢复存放参数的寄存器，所以内层函数应该拿到参数后直接move到其他用于保存局部/临时变量的寄存器中。所以前4个参数不仅被分配到用于传递参数的4个寄存器，还应该为前4个参数分配用于保存局部/临时变量的寄存器。所以被当作干涉图中点的变量包含前4个参数和局部/临时变量，不包含后面的参数。并且，由于我们必须留几个（实现时，保留了4个）寄存器来存放从运行时栈中取出的值，所以实际上可供分配的寄存器一共有16个。

总结：对于参变量，前4个参变量只在参数传递时是a0-a3，在子函数中马上会被move到其他寄存器（或者是栈上），后续所有对该参变量的使用都映射到其他寄存器，而非a0-a3。后面的所有参变量都被溢出到栈上。所有的局部变量和前4个参变量一起参与寄存器分配（共16个）。剩余的4个寄存器（t8 t9 v0 v1）用于存放从栈中取出的值。v0还用来存放函数返回值。运行时栈的前一部分（0, paraCnt-4）用于存放溢出的参数，后一部分（paraCnt-4, spilledCnt)用于存放图染色算法决定的从前4个参数和局部/临时变量中溢出的变量。

图的k染色

如果存在一个点的度数 < k，那么从图上把这个点删去，把这个点入栈。（实现中，如果有多个点的度数 < k，就选一个度数最大的）

如果所有的点的度数都 >= k，那么选一个点代表的变量溢出到栈上，从图上把这个点删去，不把这个点入栈。（实现中，优先溢出顺序：在循环中且度数大 > 在在循环中且度数小 > 不在循环中且度数大 > 不在循环中且度数小）

这样循环地进行，最终图会被删光。然后从栈顶开始向图中加点，加点时为这个点染色，注意不要染成和图中已有的邻接点一个颜色。

翻译

主要原理为在每次遇到Temp时，都查询为其分配的寄存器，以将其替换。

代码实现为：

```
package visitor;
import spiglet2kanga.*;
import syntaxtree.BinOp;
import syntaxtree.CJumpStmt;
import syntaxtree.Call;
import syntaxtree.ErrorStmt;
import syntaxtree.Exp;
import syntaxtree.Goal;
import syntaxtree.HAllocate;
import syntaxtree.HLoadStmt;
import syntaxtree.HStoreStmt;
import syntaxtree.IntegerLiteral;
import syntaxtree.JumpStmt;
import syntaxtree.Label;
import syntaxtree.MoveStmt;
```

```
import syntaxtree.NoOpStmt;
import syntaxtree.Operator;
import syntaxtree.PrintStmt;
import syntaxtree.Procedure;
import syntaxtree.SimpleExp;
import syntaxtree.Stmt;
import syntaxtree.StmtExp;
import syntaxtree.StmtList;
import syntaxtree.Temp;
public class spiglet2kangaVisitor extends GJDepthFirst<String, Environment>
{
  //
     // User-generated visitor methods below
     //

     /**
      * f0 -> "MAIN"
      * f1 -> StmtList()
      * f2 -> "END"
      * f3 -> ( Procedure() )*
      * f4 -> <EOF>
      */
     public String visit(Goal n, Environment argu) {
        String _ret=null;
        argu.setFunc("MAIN");
        String decl="MAIN[0]["+argu.currentFunc.spilledCnt+"]"+"[20]";
        argu.document.write(decl);
        argu.document.newline();
        n.f1.accept(this, argu);
        argu.document.writeEnd();
        argu.document.newline();
        n.f3.accept(this, argu);
        return _ret;
     }

     /**
      * f0 -> ( ( Label() )? Stmt() )*
      */
     public String visit(StmtList n, Environment argu) {
        String _ret=null;
        n.f0.accept(this, argu);
        return _ret;
     }

     /**
      * f0 -> Label()
      * f1 -> "["
      * f2 -> IntegerLiteral()
      * f3 -> "]"
```

```
      * f4 -> StmtExp()
      */
     public String visit(Procedure n, Environment argu) {
        String _ret=null;
        argu.setFunc(n.f0.f0.tokenImage);///TODO:spilledCnt?
        String decl=n.f0.f0.tokenImage+"["+argu.currentFunc.paraCnt+"]"+"["+
(argu.currentFunc.spilledCnt+16)+"]";
        decl += "[20]";
        argu.document.write(decl);
        argu.document.newline();
        int para=Math.max(0,argu.currentFunc.paraCnt-4);
        for(int i=0;i<16;i++)
        {
          argu.document.write("ASTORE SPILLEDARG "+
(argu.currentFunc.spilledCnt+i)+" "+argu.currentFunc.getRegName(i));
          argu.document.newline();
        }
        for (int i = 0; i < argu.currentFunc.paraCnt; i++)
        {
        if (i < 4)
        {
            if(argu.currentFunc.isSpilled(i)==1)
            {

 argu.document.write("ASTORE",argu.currentFunc.queryReg(i),"a"+i);
              argu.document.newline();
            }
            else if(argu.currentFunc.isSpilled(i)==0)
            {
              argu.document.write("MOVE",argu.currentFunc.queryReg(i),"a"+i);
              argu.document.newline();
            }
        }
        /*else
        {
          if(argu.currentFunc.isSpilled(i)==1)
            {
            argu.document.write("ALOAD","v0","SPILLEDARG "+(i-4));
            argu.document.newline();
              argu.document.write("ASTORE",argu.currentFunc.queryReg(i),"v0");
              argu.document.newline();
            }
            else if(argu.currentFunc.isSpilled(i)==0)
            {

 argu.document.write("ALOAD",argu.currentFunc.queryReg(i),"SPILLEDARG "+(i-
4));
              argu.document.newline();
            }
```

```
         }*/
       }
         n.f4.accept(this, argu);

         for(int i=0;i<16;i++)//TODO:18?
         {

 argu.document.write("ALOAD",argu.currentFunc.getRegName(i),"SPILLEDARG",
(para+i));
           argu.document.newline();
         }
         argu.document.writeEnd();
         argu.document.newline();
         return _ret;
      }

      /**
       * f0 -> NoOpStmt()
       *       | ErrorStmt()
       *       | CJumpStmt()
       *       | JumpStmt()
       *       | HStoreStmt()
       *       | HLoadStmt()
       *       | MoveStmt()
       *       | PrintStmt()
       */
      public String visit(Stmt n, Environment argu) {
         String _ret=null;
         argu.isInStmt=true;
         argu.isParam=-1;
         n.f0.accept(this, argu);
         argu.isInStmt=false;
         return _ret;
      }

      /**
       * f0 -> "NOOP"
       */
      public String visit(NoOpStmt n, Environment argu) {
         String _ret=null;
         argu.document.write("NOOP");
         argu.document.newline();
         return _ret;
      }

      /**
       * f0 -> "ERROR"
       */
      public String visit(ErrorStmt n, Environment argu) {
```

```
      String _ret=null;
      argu.document.write("ERROR");
      argu.document.newline();
      return _ret;
   }

   /**
    * f0 -> "CJUMP"
    * f1 -> Temp()
    * f2 -> Label()
    */
   public String visit(CJumpStmt n, Environment argu) {
      String _ret=null;
      String tmp =n.f1.accept(this, argu);
      argu.document.write("CJUMP",tmp,n.f2.f0.tokenImage);
      argu.document.newline();
      return _ret;
   }

   /**
    * f0 -> "JUMP"
    * f1 -> Label()
    */
   public String visit(JumpStmt n, Environment argu) {
      String _ret=null;
      argu.document.write("JUMP",n.f1.f0.tokenImage);
      argu.document.newline();
      return _ret;
   }

   /**
    * f0 -> "HSTORE"
    * f1 -> Temp()
    * f2 -> IntegerLiteral()
    * f3 -> Temp()
    */
   public String visit(HStoreStmt n, Environment argu) {
      String _ret=null;
      String tmp1=n.f1.accept(this,argu);
      argu.document.write("MOVE","v1",tmp1);
      argu.document.newline();
      String tmp2=n.f3.accept(this, argu);//////TODO
      argu.document.write("HSTORE","v1",n.f2.f0.tokenImage,tmp2);
      argu.document.newline();
      return _ret;
   }

   /**
    * f0 -> "HLOAD"
```

```
     * f1 -> Temp()
     * f2 -> Temp()
     * f3 -> IntegerLiteral()
     */
    public String visit(HLoadStmt n, Environment argu) {
       String _ret=null;

       String tmp2=n.f2.accept(this, argu);
       int num1=Integer.valueOf(n.f1.f1.f0.tokenImage);
       if(argu.currentFunc.isSpilled(num1)==1)//spilled
       {
         String tmp;
         if(tmp2=="t9")
         {
           tmp="v0";
         }
         else
         {
           tmp="t9";
         }
         argu.document.write("HLOAD",tmp,tmp2,n.f3.f0.tokenImage);
         argu.document.newline();
         argu.document.write("ASTORE",argu.currentFunc.queryReg(num1),tmp);
         argu.document.newline();
       }
       else  if(argu.currentFunc.isSpilled(num1)==0)
       {

argu.document.write("HLOAD",argu.currentFunc.queryReg(num1),tmp2,n.f3.f0.token
Image);
         argu.document.newline();
       }
       return _ret;
    }

    /**
     * f0 -> "MOVE"
     * f1 -> Temp()
     * f2 -> Exp()
     */
    public String visit(MoveStmt n, Environment argu) {
       String _ret=null;
       int num1=Integer.valueOf(n.f1.f1.f0.tokenImage);
       //default: Move t8 Expr;
       String tmp=n.f2.accept(this,argu);
       //argu.document.write("MOVE");
      // argu.document.newline();

       if(argu.currentFunc.isSpilled(num1)==1)//spilled
```

```
      {
        argu.document.write("ASTORE",argu.currentFunc.queryReg(num1),tmp);
      }
      else   if(argu.currentFunc.isSpilled(num1)==0)
      {
       argu.document.write("MOVE",argu.currentFunc.queryReg(num1),tmp);
      }
      argu.document.newline();

      return _ret;
   }

   /**
    * f0 -> "PRINT"
    * f1 -> SimpleExp()
    */
   public String visit(PrintStmt n, Environment argu) {
      String _ret=null;
     // n.f0.accept(this, argu);
      String tmp= n.f1.accept(this, argu);
      argu.document.write("PRINT",tmp);
      argu.document.newline();
      return _ret;
   }

   /**
    * f0 -> Call()
    *       | HAllocate()
    *       | BinOp()
    *       | SimpleExp()
    */
   public String visit(Exp n, Environment argu) {

      return  n.f0.accept(this, argu);
   }

   /**
    * f0 -> "BEGIN"
    * f1 -> StmtList()
    * f2 -> "RETURN"
    * f3 -> SimpleExp()
    * f4 -> "END"
    */
   public String visit(StmtExp n, Environment argu) {
      String _ret=null;
     // n.f0.accept(this, argu);
      n.f1.accept(this, argu);
     // n.f2.accept(this, argu);
      String tmp=n.f3.accept(this, argu);
```

```
      argu.document.write("MOVE","v0",tmp);
      argu.document.newline();
      _ret="v0";
      //n.f4.accept(this, argu);
      return _ret;
   }

   /**
    * f0 -> "CALL"
    * f1 -> SimpleExp()
    * f2 -> "("
    * f3 -> ( Temp() )*
    * f4 -> ")"
    */
   public String visit(Call n, Environment argu) {
      String _ret=null;
      argu.isParam=0;/////-1 is not
      n.f3.accept(this, argu);
      argu.isParam=-1;
      String tmp=n.f1.accept(this, argu);
      argu.document.write("CALL",tmp);//TODO Move r1 Label?
      argu.document.newline();

      //argu.document.write("MOVE","t8","v0");///TODO: 2 return value?
     // argu.document.newline();
      _ret="v0";

      return _ret;
   }

   /**
    * f0 -> "HALLOCATE"
    * f1 -> SimpleExp()
    */
   public String visit(HAllocate n, Environment argu) {
      String _ret=null;
    //  n.f0.accept(this, argu);
      String tmp=n.f1.accept(this, argu);
      argu.document.write("MOVE","t8","HALLOCATE",tmp);
      argu.document.newline();
      _ret="t8";
      return _ret;
   }

   /**
    * f0 -> Operator()
    * f1 -> Temp()
    * f2 -> SimpleExp()
    */
```

```
   public String visit(BinOp n, Environment argu) {
      String _ret=null;
      String tmp1=n.f1.accept(this,argu);
      argu.document.write("MOVE","v1",tmp1);
      argu.document.newline();
      String tmp2=n.f2.accept(this, argu);
      String op=n.f0.accept(this,argu);
      argu.document.write("MOVE","t8",op,"v1",tmp2);
      argu.document.newline();
      _ret="t8";
      return _ret;
   }

   /**
    * f0 -> "LT"
    *       | "PLUS"
    *       | "MINUS"
    *       | "TIMES"
    */
   public String visit(Operator n, Environment argu) {
      String _ret=null;
      _ret=n.f0.choice.toString();
      return _ret;
   }

   /**
    * f0 -> Temp()
    *       | IntegerLiteral()
    *       | Label()
    */
   public String visit(SimpleExp n, Environment argu) {

      return  n.f0.accept(this, argu);
   }

   /**
    * f0 -> "TEMP"
    * f1 -> IntegerLiteral()
    */
   public String visit(Temp n, Environment argu) {
      String _ret=null;
      if(argu.isParam>=0)
      {
        int t=Integer.valueOf(n.f1.f0.tokenImage);
        if(argu.isParam<4)
        {
          if(argu.currentFunc.isSpilled(t)==1)
          {
```

```
 argu.document.write("ALOAD","a"+argu.isParam,argu.currentFunc.queryReg(t));
              argu.document.newline();
              _ret="a"+argu.isParam;
              argu.isParam += 1;
            }
            else if(argu.currentFunc.isSpilled(t)==0)
            {


 argu.document.write("MOVE","a"+argu.isParam,argu.currentFunc.queryReg(t));
              argu.document.newline();
              _ret="a"+argu.isParam;
              argu.isParam += 1;
            }
          }
          else
          {
            if(argu.currentFunc.isSpilled(t)==1)
            {
              argu.document.write("ALOAD v0",argu.currentFunc.queryReg(t));
              argu.document.newline();
              argu.document.write("PASSARG",argu.isParam-3,"v0");
              argu.isParam+=1;
              argu.document.newline();
            }
            else if(argu.currentFunc.isSpilled(t)==0)
            {
              argu.document.write("PASSARG",argu.isParam-
3,argu.currentFunc.queryReg(t));
              argu.isParam+=1;
              argu.document.newline();
            }

          }
        }
        else {
          int t=Integer.valueOf(n.f1.f0.tokenImage);
          if(argu.currentFunc.isSpilled(t)==1)
          {
            argu.document.write("ALOAD","t8",argu.currentFunc.queryReg(t));
            argu.document.newline();
            _ret="t8";
          }
          else if(argu.currentFunc.isSpilled(t)==0)
          {
            _ret=argu.currentFunc.queryReg(t);
          }
        }
```

```
        return _ret;
     }

     /**
      * f0 -> <INTEGER_LITERAL>
      */
     public String visit(IntegerLiteral n, Environment argu) {
        String _ret=n.f0.tokenImage;
        argu.document.write("MOVE","a2",_ret);
        argu.document.newline();
        return "a2";
     }

     /**
      * f0 -> <IDENTIFIER>
      */
     public String visit(Label n, Environment argu) {
        String _ret=n.f0.tokenImage;
        //n.f0.accept(this, argu);
        argu.document.write("MOVE","v0",_ret);
        argu.document.newline();
        if(argu.isInStmt==false)
        {
          argu.document.writeLabel(_ret);
        }
        return _ret;
     }
}
```

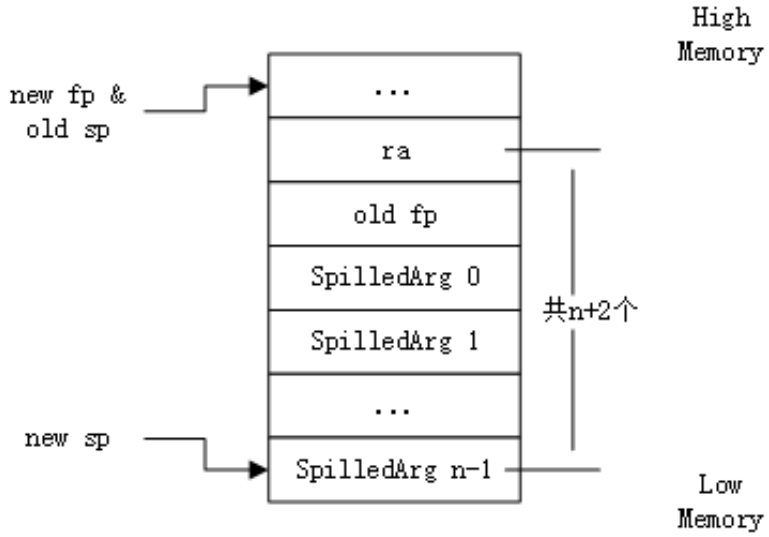
lab5：Kanga to Mips

Mips和kanga相比，最大的区别主要是需要自己维护栈。

设计思路

栈的维护

如图。其中SpilledArg 0到SpilledArg n-1是如何安排的参考作业4。具体来说，从SpilledArg 0到SpilledArg (paraCnt-5)放置溢出的参变量。从SpilledArg (paraCnt-4)到SpilledArg (n-17)放置溢出的其他变量。从SpilledArg n-16到SpilledArg n-1是保存/恢复用于寄存器分配的16个寄存器（s0-s7, t0-t7）。



翻译思路

常用的抽象成函数（halloc, print, abort）。

对于stmt级别的直接翻译。

对于exp级别的，返回一个String，表示exp的内容。特别注意的是对于二元运算的翻译，直接翻译成四元式形式，对于四元式的dst先用一个"hole"占位，在外层的MOVE中再填这个hole。

代码实现

```
package visitor;

import kanga2mips.Environment;
import syntaxtree.ALoadStmt;
import syntaxtree.AStoreStmt;
import syntaxtree.BinOp;
import syntaxtree.CJumpStmt;
import syntaxtree.CallStmt;
import syntaxtree.ErrorStmt;
import syntaxtree.Exp;
import syntaxtree.Goal;
import syntaxtree.HAllocate;
import syntaxtree.HLoadStmt;
import syntaxtree.HStoreStmt;
import syntaxtree.IntegerLiteral;
import syntaxtree.JumpStmt;
import syntaxtree.Label;
import syntaxtree.MoveStmt;
import syntaxtree.NoOpStmt;
import syntaxtree.Node;
import syntaxtree.NodeToken;
import syntaxtree.Operator;
import syntaxtree.PassArgStmt;
import syntaxtree.PrintStmt;
```

```
import syntaxtree.Procedure;
import syntaxtree.Reg;
import syntaxtree.SimpleExp;
import syntaxtree.SpilledArg;
import syntaxtree.Stmt;

public class Kanga2MipsVisitor extends GJDepthFirst<String, Environment> {
  void init(Environment argu) {
    argu.document.writeln(".text");
    argu.document.writeln(".globl _halloc");
    argu.document.writeLabel("_halloc:");
    argu.document.writeln("li $v0 9");
    argu.document.writeln("syscall");
    argu.document.writeln("jr $ra");
    argu.document.newline();

    argu.document.writeln(".text");
    argu.document.writeln(".globl _print");
    argu.document.writeLabel("_print:");
    argu.document.writeln("li $v0 1");
    argu.document.writeln("syscall");
    argu.document.writeln("la $a0 newl");
    argu.document.writeln("li $v0 4");
    argu.document.writeln("syscall");
    argu.document.writeln("jr $ra");
    argu.document.newline();

    argu.document.writeln(".text");
    argu.document.writeln(".globl _abort");
    argu.document.writeLabel("_abort:");
    argu.document.writeln("la $a0 str_er");
    argu.document.writeln("li $v0 4");
    argu.document.writeln("syscall");
    argu.document.writeln("li $v0 10");
    argu.document.writeln("syscall");
    argu.document.newline();

    argu.document.writeln(".data");
    argu.document.writeln(".align 0");
    argu.document.writeLabel("newl:");
    argu.document.writeln(".asciiz \"\\n\"");
    argu.document.newline();

    argu.document.writeln(".data");
    argu.document.writeln(".align 0");
    argu.document.writeLabel("str_er:");
    argu.document.writeln(".asciiz \"ERROR: abnormal termination\\n\"");
    argu.document.newline();
  }
```

```
  void enterFunc(String name, int spilledCnt, Environment argu) {
    argu.document.writeln(".text");
    argu.document.writeln(".globl", name);
    argu.document.writeLabel(name + ":");
    argu.document.writeln("sw $fp -8($sp)");
    argu.document.writeln("sw $ra -4($sp)");
    argu.document.writeln("move $fp $sp");
    argu.document.writeln("subu $sp $sp", (spilledCnt + 2) * 4);
  }

  void exitFunc(int spilledCnt, Environment argu) {
    argu.document.writeln("lw $ra -4($fp)");
    argu.document.writeln("lw $fp -8($fp)");
    argu.document.writeln("addu $sp $sp", (spilledCnt + 2) * 4);
    argu.document.writeln("jr $ra");
    argu.document.newline();
  }

  /**
   * f0 -> "MAIN" f1 -> "[" f2 -> IntegerLiteral() f3 -> "]" f4 -> "[" f5 ->
   * IntegerLiteral() f6 -> "]" f7 -> "[" f8 -> IntegerLiteral() f9 -> "]" f10
->
   * StmtList() f11 -> "END" f12 -> ( Procedure() )* f13 -> <EOF>
   */
  public String visit(Goal n, Environment argu) {
    init(argu);

    int spilledCnt = Integer.valueOf(n.f5.f0.tokenImage);
    enterFunc("main", spilledCnt, argu);
    n.f10.accept(this, argu);
    exitFunc(spilledCnt, argu);

    n.f12.accept(this, argu);
    return null;
  }

  /**
   * f0 -> Label() f1 -> "[" f2 -> IntegerLiteral() f3 -> "]" f4 -> "[" f5 ->
   * IntegerLiteral() f6 -> "]" f7 -> "[" f8 -> IntegerLiteral() f9 -> "]" f10
->
   * StmtList() f11 -> "END"
   */
  public String visit(Procedure n, Environment argu) {
    String name = n.f0.f0.tokenImage;
    int spilledCnt = Integer.valueOf(n.f5.f0.tokenImage);
    enterFunc(name, spilledCnt, argu);
    n.f10.accept(this, argu);
    exitFunc(spilledCnt, argu);
```

```
    return null;
  }

  /**
   * f0 -> NoOpStmt() | ErrorStmt() | CJumpStmt() | JumpStmt() | HStoreStmt()
|
   * HLoadStmt() | MoveStmt() | PrintStmt() | ALoadStmt() | AStoreStmt() |
   * PassArgStmt() | CallStmt()
   */
  public String visit(Stmt n, Environment argu) {
    argu.isInStmt = true;
    n.f0.accept(this, argu);
    argu.isInStmt = false;
    return null;
  }

  /**
   * f0 -> "NOOP"
   */
  public String visit(NoOpStmt n, Environment argu) {
    argu.document.writeln("nop");
    return null;
  }

  /**
   * f0 -> "ERROR"
   */
  public String visit(ErrorStmt n, Environment argu) {
    argu.document.writeln("jal _abort");
    return null;
  }

  /**
   * f0 -> "CJUMP" f1 -> Reg() f2 -> Label()
   */
  public String visit(CJumpStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    String r2 = n.f2.accept(this, argu);
    argu.document.writeln("beqz", r1, r2);
    return null;
  }

  /**
   * f0 -> "JUMP" f1 -> Label()
   */
  public String visit(JumpStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    argu.document.writeln("j", r1);
    return null;
```

```
  }

  /**
   * f0 -> "HSTORE" f1 -> Reg() f2 -> IntegerLiteral() f3 -> Reg()
   */
  public String visit(HStoreStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    String r2 = n.f3.accept(this, argu);
    int offset = Integer.valueOf(n.f2.f0.tokenImage);
    argu.document.writeln("sw", r2, offset + "(" + r1 + ")");
    return null;
  }

  /**
   * f0 -> "HLOAD" f1 -> Reg() f2 -> Reg() f3 -> IntegerLiteral()
   */
  public String visit(HLoadStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    String r2 = n.f2.accept(this, argu);
    int offset = Integer.valueOf(n.f3.f0.tokenImage);
    argu.document.writeln("lw", r1, offset + "(" + r2 + ")");
    return null;
  }

  /**
   * f0 -> "MOVE" f1 -> Reg() f2 -> Exp()
   */
  public String visit(MoveStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    String exp = n.f2.accept(this, argu);
    Node choice = n.f2.f0.choice;
    if (choice instanceof SimpleExp) {
      Node choice2 = ((SimpleExp) n.f2.f0.choice).f0.choice;
      String op = null;
      if (choice2 instanceof IntegerLiteral) {
        op = "li";
      } else if (choice2 instanceof Label) {
        op = "la";
      } else {
        op = "move";
      }
      argu.document.writeln(op, r1, exp);
    } else if (choice instanceof HAllocate) {
      argu.document.writeln("move", r1, "$v0");
    } else {
      r1 = r1.replaceAll("\\$", "RDS_CHAR_DOLLAR");
      exp = exp.replaceFirst("hole", r1);
      exp = exp.replaceAll("RDS_CHAR_DOLLAR", "\\$");
      argu.document.writeln(exp);
```

```
    }
    return null;
  }

  /**
   * f0 -> "PRINT" f1 -> SimpleExp()
   */
  public String visit(PrintStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    Node choice = n.f1.f0.choice;
    if (choice instanceof IntegerLiteral) {
      argu.document.writeln("li $a0", r1);
    } else if (choice instanceof Label) {
      argu.document.writeln("la $a0", r1);
    } else {
      argu.document.writeln("move $a0", r1);
    }
    argu.document.writeln("jal _print");
    return null;
  }

  /**
   * f0 -> "ALOAD" f1 -> Reg() f2 -> SpilledArg()
   */
  public String visit(ALoadStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    String r2 = n.f2.accept(this, argu);
    argu.document.writeln("lw", r1, r2);
    return null;
  }

  /**
   * f0 -> "ASTORE" f1 -> SpilledArg() f2 -> Reg()
   */
  public String visit(AStoreStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    String r2 = n.f2.accept(this, argu);
    argu.document.writeln("sw", r2, r1);
    return null;
  }

  /**
   * f0 -> "PASSARG" f1 -> IntegerLiteral() f2 -> Reg()
   */
  public String visit(PassArgStmt n, Environment argu) {
    String r1 = n.f2.accept(this, argu);
    int offset = -4 * (2 + Integer.valueOf(n.f1.f0.tokenImage));
    argu.document.writeln("sw", r1, offset + "($sp)");
    return null;
```

```
  }

  /**
   * f0 -> "CALL" f1 -> SimpleExp()
   */
  public String visit(CallStmt n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    Node choice = n.f1.f0.choice;
    if (choice instanceof IntegerLiteral) {
      argu.document.writeln("li $v0", r1);
      argu.document.writeln("jalr $v0");
    } else if (choice instanceof Label) {
      argu.document.writeln("jal", r1);
    } else {
      argu.document.writeln("jalr", r1);
    }
    return null;
  }

  /**
   * f0 -> HAllocate() | BinOp() | SimpleExp()
   */
  public String visit(Exp n, Environment argu) {
    return n.f0.accept(this, argu);
  }

  /**
   * f0 -> "HALLOCATE" f1 -> SimpleExp()
   */
  public String visit(HAllocate n, Environment argu) {
    String r1 = n.f1.accept(this, argu);
    Node choice = n.f1.f0.choice;
    if (choice instanceof IntegerLiteral) {
      argu.document.writeln("li $a0", r1);
    } else if (choice instanceof Label) {
      argu.document.writeln("la $a0", r1);
    } else {
      argu.document.writeln("move $a0", r1);
    }
    argu.document.writeln("jal _halloc");
    return "$v0";
  }

  /**
   * f0 -> Operator() f1 -> Reg() f2 -> SimpleExp()
   */
  public String visit(BinOp n, Environment argu) {
    String op = n.f0.accept(this, argu);
    String r1 = n.f1.accept(this, argu);
```

```
    String r2 = n.f2.accept(this, argu);
    Node choice = n.f2.f0.choice;
    if (choice instanceof IntegerLiteral) {
      argu.document.writeln("li $a0", r2);
    } else if (choice instanceof Label) {
      argu.document.writeln("la $a0", r2);
    } else {
      argu.document.writeln("move $a0", r2);
    }
    String _ret = op + " hole " + r1 + " $a0";
    return _ret;
  }

  /**
   * f0 -> "LT" | "PLUS" | "MINUS" | "TIMES"
   */
  public String visit(Operator n, Environment argu) {
    String op = ((NodeToken) n.f0.choice).tokenImage;
    if (op.equals("LT"))
      return "slt";
    if (op.equals("PLUS"))
      return "add";
    if (op.equals("MINUS"))
      return "sub";
    return "mul";
  }

  /**
   * f0 -> "SPILLEDARG" f1 -> IntegerLiteral()
   */
  public String visit(SpilledArg n, Environment argu) {
    int offset = -4 * (3 + Integer.valueOf(n.f1.f0.tokenImage));
    return offset + "($fp)";
  }

  /**
   * f0 -> Reg() | IntegerLiteral() | Label()
   */
  public String visit(SimpleExp n, Environment argu) {
    return n.f0.accept(this, argu);
  }

  /**
   * f0 -> "a0" | "a1" | "a2" | "a3" | "t0" | "t1" | "t2" | "t3" | "t4" | "t5"
|
   * "t6" | "t7" | "s0" | "s1" | "s2" | "s3" | "s4" | "s5" | "s6" | "s7" |
"t8" |
   * "t9" | "v0" | "v1"
   */
```

```
  public String visit(Reg n, Environment argu) {
    return "$" + ((NodeToken) n.f0.choice).tokenImage;
  }

  /**
   * f0 -> <INTEGER_LITERAL>
   */
  public String visit(IntegerLiteral n, Environment argu) {
    return n.f0.tokenImage;
  }

  /**
   * f0 -> <IDENTIFIER>
   */
  public String visit(Label n, Environment argu) {
    String lbl = n.f0.tokenImage;
    if (!argu.isInStmt) {
      argu.document.writeLabel(lbl + ":");
    }
    return lbl;
  }
}
```