GCI

SDD

PROGRAM → STATEMENT

STATEMENT.isFor = false

STATEMENT.next = createLabel(“label”)

STATEMENT.breakNext = “”

PROGRAM.code = STATEMENT.code

PROGRAM → FUNCLIST

PROGRAM.code = FUNCLIST.code

PROGRAM → &

PROGRAM.code = “”

FUNCLIST → FUNCDEF FUNCLIST2

FUNCLIST.code = FUNCDEF.code + ‘\n’ + FUNCLSIT2.code

FUNCLIST2 → FUNCLIST

FUNCLIST2.code = FUNCLIST.code

FUNCLIST2 → &

FUNCLIST2.code = “”

FUNCDEF → ‘def’ ‘ident’ ‘(‘ PARAMLIST ‘)’ ‘{‘ STATELIST ‘}’

String funcLabel = createLabel(“func”)

String next = createLabel(“label”)

STATELIST.isFor = false

STATELIST.next = next

STATELIST.breakNext = “”

FUNCDEF.code = funcLabel + “:\n” + STATELIST.code

STATEMENT → VARDECL ‘;’

STATEMENT.code = “”

STATEMENT →PRINTSTAT ‘;’

STATEMENT.code = “”

STATEMENT → READSTAT ‘;’

STATEMENT.code = “”

STATEMENT → RETURNSTAT ‘;’

STATEMENT.code = “”

STATEMENT → ‘;’

STATEMENT.code = “”

STATEMENT → ATRIBSTAT1

STATEMENT.code = ATRIBSTAT1.code

STATEMENT → IFSTAT

IFSTAT.isFor = STATEMENT.isFor

IFSTAT.next = STATEMENT.next

IFSTAT.breakNext = STATEMENT.breakNext

STATEMENT.code = IFSTAT.code

STATEMENT → FORSTAT

FORSTAT.next = STATEMENT.next

STATEMENT.code = FORSTAT.code

STATEMENT →'{' STATELIST '}'

STATELIST.isFor = STATEMENT.isFor

STATELIST.next = STATEMENT.next

STATELIST.breakNext = STATEMENT.breakNext

STATEMENT.code = STATELIST.code

STATEMENT → 'break' ';'

STATEMENT.code = Generator.generateInconditionalDeviationCode(STATEMENT.breakNext)

STATELIST → STATEMENT STATELIST2

String stmtNext = createLabel("label")

STATEMENT.isFor = STATELIST.isFor

STATEMENT.next = stmtNext

STATEMENT.breakNext = STATELIST.breakNext

STATELIST2.isFor = STATELIST.isFor

STATELIST2.next = STATELIST.next

STATELIST2.breakNext = STATELIST.breakNext

STATELIST.code = STATEMENT.code + '\n' + stmtNext + ":\n" + STATELIST2.code

STATELIST2 → STATELIST

STATELIST.isFor = STATELIST2.isFor

STATELIST.next = STATELIST2.next

STATELIST.breakNext = STATELIST.breakNext

STATELIST2.code = STATELIST.code

STATELIST2 → &

STATELIST2.code = “”

IFSTAT → ‘if’ ‘(‘ EXPRESSION ‘)’ ‘{‘ STATELIST ‘}’ IFSTAT1

String exprTrue = createLabel(“exprTrue”)

String exprFalse = createLabel(“exprFalse”)

EXPRESSION.exprFalse = exprFalse

STATELIST.isFor = IFSTAT.isFor

STATELIST.next = IFSTAT.next

STATELIST.breakNext = IFSTAT.breakNext

IFSTAT1.isFor = IFSTAT.isFor

IFSTAT1.next = IFSTAT.next

IFSTAT1.breakNext = IFSTAT.breakNext

IFSTAT.code = EXPRESSION.code + exprTrue + “:\n” + STATELIST.code +
Generator.generateInconditionalDeviationCode(IFSTAT.next) + exprFalse + “:\n” + IFSTAT1.code

IFSTAT1 → ‘else’ STATEMENT

STATEMENT.isFor = IFSTAT1.isFor

STATEMENT.next = IFSTAT1.next

STATEMENT.breakNext = IFSTAT1.breakNext

IFSTAT1.code = STATEMENT.code

IFSTAT1 → &

IFSTAT1.code = “”

FORSTAT → ‘for’ ‘(‘ ATRIBSTAT1 ‘;’ EXPRESSION ‘;’ ATRIBSTAT1) STATEMENT

String begin = createLabel(“begin”)

String exprTrue = createLabel(“exprTrue”)

EXPRESSION.exprFalse = FORSTAT.next

STATEMENT.isFor = true

STATEMENT.next = begin

STATEMENT.breakNext = FORSTAT.next

FORSTAT.code = begin + “:\n” + EXPRESSION.code + exprTrue + “:\n” + STATEMENT.code + Generator.generateInconditionalDeviationCode(begin)

ATRIBSTAT1 → LVALUE ’=’ ATRIBSTAT2

ATRIBSTAT1.code = “”

if (!ATRIBSTAT2.isEmpty()) {

ATRIBSTAT1.code = LVALUE.code + ATRIBSTAT2.code + LVALUE.last + ‘=’

+ ATRIBSTAT2.code

}

ATRIBSTAT2 → ‘ident’ ATRIBSTAT3

ATRIBSTAT2.code = ATRIBSTAT3.code

ATRIBSTAT2.last = ATRIBSTAT3.last

ATRIBSTAT2 → ALLOCEXPRESSION

ATRIBSTAT2.code = “”

ATRIBSTAT.last = “”

ATRIBSTAT.expTree = new ExpressionTree()

ATRIBSTAT2 → ‘+’ FACTOR D C EXPRESSION2

GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree)

ATRIBSTAT2.code = generateReturn.getCode()

ATRIBSTAT2.last = generateReturn.getLast()

ATRIBSTAT2 → ‘-’ FACTOR D C EXPRESSION2

GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree)

ATRIBSTAT2.code = generateReturn.getCode()

ATRIBSTAT2.last = generateReturn.getLast()

ATRIBSTAT2 →’int_constant’ D C EXPRESSION2

GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree)

ATRIBSTAT2.code = generateReturn.getCode()

ATRIBSTAT2.last = generateReturn.getLast()

ATRIBSTAT2 → ‘float_constant’ D C EXPRESSION2

GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree)

ATRIBSTAT2.code = generateReturn.getCode()

ATRIBSTAT2.last = generateReturn.getLast()

ATRIBSTAT2 →’string constant’ D C EXPRESSION2

GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree)

ATRIBSTAT2.code = generateReturn.getCode()

ATRIBSTAT2.last = generateReturn.getLast()

ATRIBSTAT2 → ‘null’ D C EXPRESSION2

ATRIBSTAT2.code = “”

ATRIBSTAT2.last = “”

ATRIBSTAT2.expTree = new ExpressionTree()

ATRIBSTAT2 →’(‘ NUMEXPRESSION ‘)’ D C EXPRESSION2

GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree)

ATRIBSTAT2.code = generateReturn.getCode()

ATRIBSTAT2.last = generateReturn.getLast()

ATRIBSTAT3 → B D C EXPRESSION2

GenerateReturn generateReturn = generateCode(ATRIBSTAT3.expTree)

ATRIBSTAT3.code = generateReturn.getCode()

ATRIBSTAT3.last = generateReturn.getLast()

ATRIBSTAT3 → ‘(‘ PARAMLISTCALL ‘)’

String[] params = PARAMLISTCALL.text.split(“,”)

String[] result = Generator.generateFunctionCallCode(ATRIBSTAT3.h.getValue(), params)

ATRIBSTAT3.code = result[0]

ATRIBSTAT3.last = result[1]

EXPRESSION → NUMEXPRESSION EXPRESSION2

EXPRESSION.code = Generator.generateConditionalDeviationCode(“False “ + NUMEXPRESSION.last + EXPRESSION2.code, EXPRESSION.exprFalse)

EXPRESSION2 → ‘>’ NUMEXPRESSION

EXPRESSION2.code = ‘>’ + NUMEXPRESSION.last

EXPRESSION2 → ‘<’ NUMEXPRESSION

EXPRESSION2.code = ’<’ + NUMEXPRESSION.last

EXPRESSION2 →’<=’ NUMEXPRESSION

EXPRESSION2.code = ‘<=’ + NUMEXPRESSION.last

EXPRESSION2 →’>=’ NUMEXPRESSION

EXPRESSION2.code = ‘>=’ + NUMEXPRESSION.last

EXPRESSION2 →’==‘ NUMEXPRESSION

EXPRESSION2.code = ‘==’ + NUMEXPRESSION.last

EXPRESSION2 →’!=’ NUMEXPRESSION

EXPRESSION2.code = ‘!=’ + NUMEXPRESSION.last

EXPRESSION2 → &

EXPRESSION.code = “”

NUMEXPRESSION → TERM C

GenerateReturn generateReturn = generateCode(NUMEXPRESSION.expTree)

NUMEXPRESSION.code = generateReturn.getCode()

NUMEXPRESSION.last = generateReturn.getLast()

LVALUE → ’ident’ B

GenerateReturn generateReturn = generateCode(LVALUE.expTree)

LVALUE.code = generateReturn.getCode()

LVALUE.last = generateReturn.getLast()

GCI

SDT

PROGRAM → {STATEMENT.isFor = false; STATEMENT.next = createLabel(“label”);

STATEMENT.breakNext = “”} STATEMENT {PROGRAM.code = STATEMENT.code}

PROGRAM → FUNCLIST {PROGRAM.code = FUNCLIST.code;}

PROGRAM → & {PROGRAM.code = “”}

FUNCLIST → FUNCDEF FUNCLIST2 {FUNCLIST.code = FUNCDEF.code + ‘\n’ + FUNCLSIT2.code}

FUNCLIST2 → FUNCLIST {FUNCLIST2.code = FUNCLIST.code}

FUNCLIST2 → & {FUNCLIST2.code = “”}

FUNCDEF → {String funcLabel = createLabel(“func”); String next = createLabel(“label”)

;} ‘def’ ‘ident’ ‘(‘ PARAMLIST ‘)’ ‘{‘ {STATELIST.isFor = false; STATELIST.next = next

;STATELIST.breakNext = “”} STATELIST ‘}’ {FUNCDEF.code = funcLabel + “:\n” + STATELIST.code}

STATEMENT → VARDECL ’;’ {STATEMENT.code = “”}

STATEMENT →PRINTSTAT ‘;’ {STATEMENT.code = “”}

STATEMENT → READSTAT ‘;’ {STATEMENT.code = “”}

STATEMENT → RETURNSTAT ‘;’ {STATEMENT.code = “”}

STATEMENT → ‘;’ {STATEMENT.code = “”}

STATEMENT → ATRIBSTAT1 {STATEMENT.code = ATRIBSTAT1.code}

STATEMENT → {IFSTAT.isFor = STATEMENT.isFor; IFSTAT.next = STATEMENT.next; IFSTAT.breakNext = STATEMENT.breakNext} IFSTAT {STATEMENT.code = IFSTAT.code}

STATEMENT →{FORSTAT.next = STATEMENT.next} FORSTAT {STATEMENT.code = FORSTAT.code}

STATEMENT →’{‘ {STATELIST.isFor = STATEMENT.isFor; STATELIST.next = STATEMENT.next; STATELIST.breakNext = STATEMENT.breakNext} STATELIST ‘}’ {STATEMENT.code = STATELIST.code}

STATEMENT → ‘break’ ‘;’ {STATEMENT.code = Generator.generateInconditionalDeviationCode(STATEMENT.breakNext)}

STATELIST → {String stmtNext = createLabel(“label”)} {STATEMENT.isFor = STATELIST.isFor; STATEMENT.next = stmtNext; STATEMENT.breakNext = STATELIST.breakNext} STATEMENT {STATELIST2.isFor = STATELIST.isFor; STATELIST2.next = STATELIST.next; STATELIST2.breakNext = STATELIST.breakNext} STATELIST2 {STATELIST.code = STATEMENT.code + ‘\n’ + stmtNext + “:\n” + STATELIST2.code}

STATELIST2 → {STATELIST.isFor = STATELIST2.isFor; STATELIST.next = STATELIST2.next; STATELIST.breakNext = STATELIST.breakNext} STATELIST {STATELIST2.code = STATELIST.code}

STATELIST2 → & {STATELIST2.code = “”}

IFSTAT → {String exprTrue = createLabel(“exprTrue”); String exprFalse = createLabel(“exprFalse”)} ‘if’ ‘(‘ {EXPRESSION.exprFalse = exprFalse} EXPRESSION ‘)’ ‘{‘ {STATELIST.isFor = IFSTAT.isFor; STATELIST.next = IFSTAT.next; STATELIST.breakNext = IFSTAT.breakNext} STATELIST ‘}’ { IFSTAT1.isFor = IFSTAT.isFor; IFSTAT1.next = IFSTAT.next; IFSTAT1.breakNext = IFSTAT.breakNext} IFSTAT1 {IFSTAT.code = EXPRESSION.code + exprTrue + “:\n” + STATELIST.code + Generator.generateInconditionalDeviationCode(IFSTAT.next) + exprFalse + “:\n” + IFSTAT1.code}

IFSTAT1 → ‘else’ {STATEMENT.isFor = IFSTAT1.isFor; STATEMENT.next = IFSTAT1.next; STATEMENT.breakNext = IFSTAT1.breakNext} STATEMENT {IFSTAT1.code = STATEMENT.code}

IFSTAT1 → & {IFSTAT1.code = “”}

FORSTAT → {String begin = createLabel(“begin”); String exprTrue = createLabel(“exprTrue”)} ‘for’ ‘(‘ ATRIBSTAT1 ’;’ {EXPRESSION.exprFalse = FORSTAT.next} EXPRESSION ’;’ ATRIBSTAT1 ‘)’ { STATEMENT.isFor = true; STATEMENT.next = FORSTAT.begin; STATEMENT.breakNext = FORSTAT.next} STATEMENT {FORSTAT.code = begin + “:\n” + EXPRESSION.code + exprTrue + “:\n” + STATEMENT.code + Generator.generateInconditionalDeviationCode(begin)}

ATRIBSTAT1 → {ATRIBSTAT1.code = “”} LVALUE ‘=’ ATRIBSTAT2
{if (!ATRIBSTAT.2.isEmpty()) {
        ATRIBSTAT1.code = LVALUE.code + ATRIBSTAT2.code + LVALUE.last + ‘=’ + ATRIBSTAT2.code
}}

ATRIBSTAT2 → ‘ident’ ATRIBSTAT3 {ATRIBSTAT2.code = ATRIBSTAT3.code; ATRIBSTAT2.last = ATRIBSTAT3.last}

ATRIBSTAT2 → ALLOCEXPRESSION {ATRIBSTAT2.code = “”; ATRIBSTAT.last = “”; ATRIBSTAT.expTree = new ExpressionTree()}

ATRIBSTAT2 → ‘+’ FACTOR D C EXPRESSION2 {GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree); ATRIBSTAT2.code = generateReturn.getCode(); ATRIBSTAT2.last = generateReturn.getLast()}

ATRIBSTAT2 → ‘-’ FACTOR D C EXPRESSION2 {GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree); ATRIBSTAT2.code = generateReturn.getCode(); ATRIBSTAT2.last = generateReturn.getLast()}

ATRIBSTAT2 → ‘int_constant’ D C EXPRESSION2 {GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree); ATRIBSTAT2.code = generateReturn.getCode(); ATRIBSTAT2.last = generateReturn.getLast()}

ATRIBSTAT2 → ‘float_constant’ D C EXPRESSION2 {GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree); ATRIBSTAT2.code = generateReturn.getCode(); ATRIBSTAT2.last = generateReturn.getLast()}

ATRIBSTAT2 →’string constant’ D C EXPRESSION2 {GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree); ATRIBSTAT2.code = generateReturn.getCode(); ATRIBSTAT2.last = generateReturn.getLast()}

ATRIBSTAT2 → ‘null’ D C EXPRESSION2 {ATRIBSTAT2.code = “”; ATRIBSTAT.last = “”; ATRIBSTAT.expTree = new ExpressionTree()}

ATRIBSTAT2 →’(‘ NUMEXPRESSION ‘)’ D C EXPRESSION2 {GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree); ATRIBSTAT2.code = generateReturn.getCode(); ATRIBSTAT2.last = generateReturn.getLast()}

ATRIBSTAT3 → B D C EXPRESSION2 {GenerateReturn generateReturn = generateCode(ATRIBSTAT2.expTree); ATRIBSTAT2.code = generateReturn.getCode(); ATRIBSTAT2.last = generateReturn.getLast()}

ATRIBSTAT3 → ‘(‘ PARAMLISTCALL ‘)’ {String[] params = PARAMLISTCALL.text.split(“,”); String[] result = Generator.generateFunctionCallCode(ATRIBSTAT3.h.getValue(), params); ATRIBSTAT3.code = result[0]; ATRIBSTAT3.last = result[1]}

EXPRESSION → NUMEXPRESSION EXPRESSION2 {EXPRESSION.code = Generator.generateConditionalDeviationCode(“False “ + NUMEXPRESSION.last + EXPRESSION2.code, EXPRESSION.exprFalse)}

EXPRESSION2 → ‘>’ NUMEXPRESSION {EXPRESSION2.code = ‘>’ + NUMEXPRESSION.last}

EXPRESSION2 → ‘<’ NUMEXPRESSION {EXPRESSION2.code = ‘<’ + NUMEXPRESSION.last}

EXPRESSION2 →’<=’ NUMEXPRESSION {EXPRESSION2.code = ‘<=’ + NUMEXPRESSION.last}

EXPRESSION2 →’>=’ NUMEXPRESSION {EXPRESSION2.code = ‘>=’ + NUMEXPRESSION.last}

EXPRESSION2 →’==’ NUMEXPRESSION {EXPRESSION2.code = ‘==’ + NUMEXPRESSION.last}

EXPRESSION2 →’!=’ NUMEXPRESSION {EXPRESSION2.code = ‘!=’ + NUMEXPRESSION.last}

EXPRESSION2 → & {EXPRESSION.code = “”}

NUMEXPRESSION → TERM C {GenerateReturn generateReturn = generateCode(NUMEXPRESSION.expTree); NUMEXPRESSION.code = generateReturn.getCode(); NUMEXPRESSION.last = generateReturn.getLast()

LVALUE → ‘ident’ B {GenerateReturn generateReturn = generateCode(LVALUE.expTree); LVALUE.code = generateReturn.getCode(); LVALUE.last = generateReturn.getLast()}