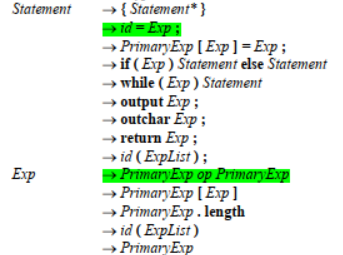
**Description**

We are implementing increment and decrement statements, which adheres to part of the existing java form. This would be limited to integer variables in the form of “variable++;” and “variable--;”. This is a quick alternative to the combinations of the highlighted grammar:



**Grammar**

We decided to add the extension to the statement non-terminal like this:

Statement → id++;

               id--;

**AST Files**

The AST classes would need the variable name using an id of type String like other AST classes. By using the specific variable name rather than an expression, it would be easier to access and directly manipulate the values of the variable in the compiler class.

AST file for increment

|  |
| --- |
| package lpl.ast;  import lpl.ast.util.Visitor;  public class StmInc extends Stm{     public final String id;     // StmInc grammar: Statement → id++      public StmInc(String id) {        this.id = id;    }     public <T> T accept(Visitor<T> v) {        return v.visit(this);    } } |

AST file for decrement

|  |
| --- |
| package lpl.ast;  import lpl.ast.util.Visitor;  public class StmDec extends Stm{     public final String id;    // StmInc Grammar: Statement → id--    public StmDec(String id) {        this.id = id;    }     public <T> T accept(Visitor<T> v) {        return v.visit(this);    } } |

**Type Checker**

We used the following existing method to assist in checking what type the variable is:

|  |
| --- |
| private Type getTypeForVar(String name) {    Type t;    t = locals.get(name);    if (t != null) {        return t;    }    throw new TypeCheckingException("No declaration found for variable: " + name); } |

Using the method, we created the following methods which ensure the type for our increment and decrement is correct and throws an error if the type is not an integer.

The following methods are to be added to TypeChecker.java:

Increment type checker method

|  |
| --- |
| @Override public Type visit(StmInc s) {     Type t1 = getTypeForVar(s.id);    if (!(t1.equals(TYPE\_INT))) {        throw new TypeCheckingException("The argument of increment must be of type int");    }    return null; } |

Decrement type checker method

|  |
| --- |
| @Override public Type visit(StmDec s) {     Type t1 = getTypeForVar(s.id);    if (!(t1.equals(TYPE\_INT))) {        throw new TypeCheckingException("The argument of decrement must be of type int");    }    return null; } |

**Compiler**

To calculate the offset value from the frame pointer, we created a method which is used in the StmAssign and PrimaryExpVar visit methods in our submission for Compiler.java. This method searches through the variable and parameter lists, assigned in the FunDef method, for a specific value and the offset is calculated based on the index value:

|  |
| --- |
| public int offset(String id){     if (var != null && !var.isEmpty()) {        for (VarDecl f : var) {            if (f.id.equals(id)){                int n = var.indexOf(f) \* -1;                return n - 1;            }        }    }    if (param != null && !param.isEmpty()) {        for (Formal v : param) {            if (v.id.equals(id)) {                return param.indexOf(v) + 2;            }        }    }    return 0; } |

We need to move the new value of the variable into the variable location in the visit method for the extensions in the compiler class. This is done by retrieving the location of the variable and fetching the contents to be used in a BINOP operation to either add one or minus one which is stored back in the original variable location.

Increment

|  |
| --- |
| @Override public IRExp visit(StmInc s) {     // Calculating new value from fetching the contents of the original location     IRExp ire = BINOP(MEM(BINOP(TEMP("FP"), IROp.ADD, CONST(offset(s.id)))), IROp.ADD, CONST(1));     // Storing new value in original location    emit(MOVE(MEM(BINOP(TEMP("FP"), IROp.ADD, CONST(offset(s.id)))), ire));    return null; } |
|  |

Decrement

|  |
| --- |
| @Override public IRExp visit(StmDec s) {     // Calculating new value from fetching the contents of the original location    IRExp ire = BINOP(MEM(BINOP(TEMP("FP"), IROp.ADD, CONST(offset(s.id)))), IROp.MINUS, CONST(1));     // Storing new value in original location    emit(MOVE(MEM(BINOP(TEMP("FP"), IROp.ADD, CONST(offset(s.id)))), ire));    return null; } |