```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KAssign;
import microjs.jcompiler.utils.DotGraph;

public class Assign extends Statement {
    private String name;
    private Expr expr;

    public Assign(String name, Expr expr, Location startPos, Location endPos) {
        super(startPos, endPos);
        this.name = name;
                this.expr = expr;
    }

    @Override
    public KAssign expand() {
        return new KAssign(name, expr.expand(), getStartPos(), getEndPos());
    }

    @Override
    protected void prettyPrint(StringBuilder buf, int indent_level) {
        indent(buf, indent_level);
        buf.append(name);
        buf.append(" = ");
        expr.prettyPrint(buf);
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String assignNode = graph.addNode("Assign[" + name + "]");
                String exprNode = expr.buildDotGraph(graph);
                graph.addEdge(assignNode, exprNode, "expr");

                return assignNode;
        }



}
```



```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KASTNode;
import microjs.jcompiler.utils.DotGraph;

public abstract class ASTNode {
        Location startPos;  // objects with getLine and getColumn
        Location endPos;


        /* package */ ASTNode(Location startPos, Location endPos) {
                this.startPos = startPos;
                this.endPos = endPos;
        }

        public Location getStartPos() {
                return startPos;
        }

        public Location getEndPos() {
                return endPos;
        }

        public abstract KASTNode expand();

        protected abstract void prettyPrint(StringBuilder buf);

        @Override
        public final String toString() {
                StringBuilder buf = new StringBuilder();
                prettyPrint(buf);
                return buf.toString();
        }

        public DotGraph genDotGraph() {
                DotGraph graph = new DotGraph();
                buildDotGraph(graph);
                return graph;
        }

        protected abstract String buildDotGraph(DotGraph graph);


}
```

```
package microjs.jcompiler.frontend.ast;

import java.util.ArrayList;
import java.util.List;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KCall;
import microjs.jcompiler.middleend.kast.KEVar;
import microjs.jcompiler.middleend.kast.KExpr;
import microjs.jcompiler.utils.DotGraph;

public class BinOp extends Expr {
        private String name;
        private Expr left;
        private Expr right;

        public BinOp(String name, Expr left, Expr right, Location startPos, Loca
tion endPos) {
                super(startPos, endPos);
                this.name = name;
                this.left = left;
                this.right = right;
        }

        @Override
        public KCall expand() {
                List<KExpr> args = new ArrayList<KExpr>();
                args.add(left.expand());
                args.add(right.expand());
                return new KCall(new KEVar(name, getStartPos(), getEndPos()), ar
gs, getStartPos(), getEndPos());
        }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String binopNode = graph.addNode("Binop[" + name + "]");

                String leftNode = left.buildDotGraph(graph);
                graph.addEdge(binopNode, leftNode, "left");

                String rightNode = right.buildDotGraph(graph);
                graph.addEdge(binopNode, rightNode, "right");

                return binopNode;
        }

        @Override
        protected void prettyPrint(StringBuilder buf) {
                buf.append("(");
                left.prettyPrint(buf);
                buf.append(name);
                right.prettyPrint(buf);
                buf.append(")");
        }

}
```



```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KExpr;
import microjs.jcompiler.middleend.kast.KFalse;
import microjs.jcompiler.middleend.kast.KTrue;
import microjs.jcompiler.utils.DotGraph;

public class BoolConst extends Expr {
        private boolean value;

        public BoolConst(boolean value, Location startPos, Location endPos) {
                super(startPos, endPos);
                this.value = value;
        }

        @Override
        public KExpr expand() {
                if(value == true) {
                        return new KTrue(getStartPos(), getEndPos());
                } else {
                        return new KFalse(getStartPos(), getEndPos());
                }
        }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String boolNode = graph.addNode("Bool[" + value +"]");
                return boolNode;
        }

        @Override
        protected void prettyPrint(StringBuilder buf) {
                buf.append(value);
        }
}
```

```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KEVar;
import microjs.jcompiler.utils.DotGraph;

public class EVar extends Expr {
        private String name;

        public EVar(String name, Location startPos, Location endPos) {
                super(startPos, endPos);
                this.name = name;
        }

        @Override
        public KEVar expand() {
                return new KEVar(name, getStartPos(), getEndPos());
        }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String varNode = graph.addNode("EVar[" + name +"]");
                return varNode;
        }

        @Override
        protected void prettyPrint(StringBuilder buf) {
                buf.append(name);
        }

}
```



```
package microjs.jcompiler.frontend.ast;

import java.util.ArrayList;
import java.util.List;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KExpr;

public abstract class Expr extends ASTNode {

        /* package */ Expr(Location startPos, Location endPos) {
                super(startPos, endPos);
        }

        @Override
        public abstract KExpr expand();

        protected static List<KExpr> expandExprs(List<Expr> exprs) {
                List<KExpr> kexprs = new ArrayList<KExpr>();
                for(Expr expr : exprs) {
                        kexprs.add(expr.expand());
                }
                return kexprs;
        }
}
```

```
package microjs.jcompiler.frontend.ast;

import java.util.List;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KCall;
import microjs.jcompiler.middleend.kast.KExpr;
import microjs.jcompiler.utils.DotGraph;

public class Funcall extends Expr {
    private Expr fun;
    private List<Expr> arguments;

    public Funcall(Expr fun, List<Expr> arguments, Location startPos, Location e
ndPos) {
        super(startPos, endPos);
                this.fun = fun;
                this.arguments = arguments;
    }

    @Override
    public KCall expand() {
        List<KExpr> kargs = Expr.expandExprs(arguments);
        return new KCall(fun.expand(), kargs, getStartPos(), getEndPos());
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String callNode = graph.addNode("Funcall[" + fun.toString() + "]
");
                for(int i=0; i<arguments.size(); i++) {
                        Expr arg = arguments.get(i);
                        String argRoot = arg.buildDotGraph(graph);
                        graph.addEdge(callNode, argRoot, "arg[" + i + "]");
                }
                return callNode;
        }

    @Override
    protected void prettyPrint(StringBuilder buf) {
        fun.prettyPrint(buf);
        buf.append("(");
        String sep = "";
        for(Expr arg : arguments) {
                buf.append(sep);
                arg.prettyPrint(buf);
                if(sep.equals("")) {
                        sep = ", ";
                }
        }
        buf.append(")");
    }
}
```



```
package microjs.jcompiler.frontend.ast;

import java.util.List;
import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KClosure;
import microjs.jcompiler.middleend.kast.KSeq;
import microjs.jcompiler.middleend.kast.KStatement;
import microjs.jcompiler.middleend.kast.KVar;
import microjs.jcompiler.utils.DotGraph;

public class Function extends Statement {
    private String name;
    private List<String> params;
    private List<Statement> body;

    public Function(String name, List<String> params, List<Statement> body, Loca
tion startPos, Location endPos) {
        super(startPos, endPos);
        this.name = name;
        this.params = params;
        this.body = body;
    }

    @Override
    public KVar expand() {
        List<KStatement> kstmts = Statement.expandStatements(body);
        Location startBody = getStartPos(); // XXX: good approx for empty body ?
        Location endBody = getEndPos();
        KStatement kbody = KSeq.buildKSeq(kstmts, startBody, endBody);
        return new KVar(name,
                        new KClosure(params, kbody, getStartPos(), getEndPos()),
                        getStartPos(), getEndPos());
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String funNode = graph.addNode("Fun[" + name + "](" + params + "
)");
                for(int i=0; i<body.size(); i++) {
                        Statement st = body.get(i);
                        String stRoot = st.buildDotGraph(graph);
                        graph.addEdge(funNode, stRoot, "body[" + i + "]");
                }
                return funNode;
        }

    @Override
    protected void prettyPrint(StringBuilder buf, int indent_level) {
        indent(buf, indent_level);
        buf.append("function ");
        buf.append(name);
        buf.append("(");
        String sep = "";
        for(String param : params) {
                buf.append(sep);
                buf.append(param);
                if(sep.equals("")) {
                        sep = ", ";
                }
        }
        buf.append(") {\n");
        Statement.prettyPrintStatements(buf, body, indent_level + 1);
        indent(buf, indent_level);
        buf.append("}");
    }
}
```

```
package microjs.jcompiler.frontend.ast;

import java.util.List;
import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KIf;
import microjs.jcompiler.middleend.kast.KSeq;
import microjs.jcompiler.middleend.kast.KStatement;
import microjs.jcompiler.utils.DotGraph;

public class If extends Statement {
    private Expr cond;
    private List<Statement> thens;
    private List<Statement> elses;

    public If(Expr cond, List<Statement> thens, List<Statement> elses, Location
startPos, Location endPos) {
        super(startPos, endPos);
        this.cond = cond;
        this.thens = thens;
        this.elses = elses;
    }

    @Override
    public KIf expand() {
        // then part
        Location thenStartPos = getStartPos(); // XXX: good approximation ?
        Location thenEndPos = getStartPos();
        List<KStatement> kthens = Statement.expandStatements(thens);
        KStatement kthen = KSeq.buildKSeq(kthens, thenStartPos, thenEndPos);

        // else part
        Location elseStartPos = thenEndPos; // XXX: good approximation ?
        Location elseEndPos = thenEndPos;
        List<KStatement> kelses = Statement.expandStatements(elses);
        KStatement kelse = KSeq.buildKSeq(kelses, elseStartPos, elseEndPos);
        return new KIf(cond.expand(), kthen, kelse, getStartPos(), getEndPos());
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String ifNode = graph.addNode("If");
                String condNode = cond.buildDotGraph(graph);
                graph.addEdge(ifNode, condNode, "cond");
                String thenNode = cond.buildDotGraph(graph);
                graph.addEdge(ifNode, thenNode, "then");
                String elseNode = cond.buildDotGraph(graph);
                graph.addEdge(ifNode, elseNode, "else");

                return ifNode;
        }

    @Override
    protected void prettyPrint(StringBuilder buf, int indent_level) {
        indent(buf, indent_level);
        buf.append("if (");
        cond.prettyPrint(buf);
        buf.append(") {\n");
        Statement.prettyPrintStatements(buf, thens, indent_level + 1);
        indent(buf, indent_level);
        buf.append("} else {\n");
        Statement.prettyPrintStatements(buf, elses, indent_level + 1);
        indent(buf, indent_level);
        buf.append("}");
    }
}
```



```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KInt;
import microjs.jcompiler.utils.DotGraph;

public class IntConst extends Expr {
        private int value;

        public IntConst(int value, Location startPos, Location endPos) {
                super(startPos, endPos);
                this.value = value;
        }

        @Override
        public KInt expand() {
                return new KInt(value, getStartPos(), getEndPos());
        }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String intNode = graph.addNode("Int[" + value +"]");
                return intNode;
        }

        @Override
        protected void prettyPrint(StringBuilder buf) {
                buf.append(value);
        }
}
```

```
package microjs.jcompiler.frontend.ast;

import java.util.List;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KClosure;
import microjs.jcompiler.middleend.kast.KSeq;
import microjs.jcompiler.middleend.kast.KStatement;
import microjs.jcompiler.utils.DotGraph;

public class Lambda extends Expr {
    private List<String> params;
    private List<Statement> body;

    public Lambda(List<String> params, List<Statement> body, Location startPos,
Location endPos) {
        super(startPos, endPos);
        this.params = params;
        this.body = body;
    }

    @Override
    public KClosure expand() {
        List<KStatement> kstmts = Statement.expandStatements(body);
        Location startBody = getStartPos(); // XXX: good approx for empty body ?
        Location endBody = getEndPos();
        KStatement kbody = KSeq.buildKSeq(kstmts, startBody, endBody);
        return new KClosure(params, kbody, getStartPos(), getEndPos());
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String lamNode = graph.addNode("Lambda(" + params + ")");
                for(int i=0; i<body.size(); i++) {
                        Statement st = body.get(i);
                        String stRoot = st.buildDotGraph(graph);
                        graph.addEdge(lamNode, stRoot, "body[" + i + "]");
                }
                return lamNode;
        }


    @Override
    protected void prettyPrint(StringBuilder buf) {
        buf.append("lambda");
        buf.append("(");
        String sep = "";
        for(String param : params) {
                buf.append(sep);
                buf.append(param);
                if(sep.equals("")) {
                        sep = ", ";
                }
        }
        buf.append(") {\n");
        Statement.prettyPrintStatements(buf, body, 2);
        buf.append("}");
    }
}
```



```
package microjs.jcompiler.frontend.ast;

import java.util.ArrayList;
import java.util.List;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KCall;
import microjs.jcompiler.middleend.kast.KClosure;
import microjs.jcompiler.middleend.kast.KExpr;
import microjs.jcompiler.middleend.kast.KReturn;
import microjs.jcompiler.middleend.kast.KSeq;
import microjs.jcompiler.middleend.kast.KStatement;
import microjs.jcompiler.utils.DotGraph;

public class Let extends Statement {
    private String name;
    private Expr expr;
    private List<Statement> body;

    public Let(String name, Expr expr, List<Statement> body, Location startPos,
Location endPos) {
        super(startPos, endPos);
        this.name = name;
                this.expr = expr;
        this.body = body;
    }

    @Override
    public KReturn expand() {
        List<String> params = new ArrayList<String>();
        params.add(name);

        List<KStatement> kstmts = Statement.expandStatements(body);
        KStatement kbody = KSeq.buildKSeq(kstmts, getStartPos(), getEndPos());

        KClosure fun = new KClosure(params, kbody, getStartPos(), getEndPos());

        List<KExpr> kargs = new ArrayList<>();
        kargs.add(expr.expand());

        KCall call = new KCall(fun, kargs, startPos, endPos);

        return new KReturn(call, startPos, endPos);
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String letNode = graph.addNode("Let[" + name + "]");
                String exprNode = expr.buildDotGraph(graph);
                graph.addEdge(letNode, exprNode, "expr");
                for(int i=0; i<body.size(); i++) {
                        Statement st = body.get(i);
                        String stRoot = st.buildDotGraph(graph);
                        graph.addEdge(letNode, stRoot, "body[" + i + "]");
                }
                return letNode;
        }


    @Override
    protected void prettyPrint(StringBuilder buf, int indent_level) {
        indent(buf, indent_level);
        buf.append("let ");
        buf.append(name);
        buf.append(" = ");
        expr.prettyPrint(buf);
```

```
        buf.append(";\n");
        Statement.prettyPrintStatements(buf, body, indent_level);
    }
}
```



```
package microjs.jcompiler.frontend.ast;

import java.util.ArrayList;
import java.util.List;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KProg;
import microjs.jcompiler.middleend.kast.KSeq;
import microjs.jcompiler.middleend.kast.KStatement;
import microjs.jcompiler.utils.DotGraph;

public class Prog extends ASTNode {
        private String filename;
        private List<Statement> body;

        public Prog(String filename, List<Statement> body, Location startPos, Lo
cation endPos) {
                super(startPos, endPos);
                this.filename = filename;
                this.body = body;
        }

        @Override
        public KProg expand() {
                List<KStatement> kseq = Statement.expandStatements(body);
                KStatement kbody = KSeq.buildKSeq(kseq, getStartPos(), getEndPos
());
                return new KProg(filename, kbody, getStartPos(), getEndPos());
        }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String progNode = graph.addNode("Prog[" + filename + "]");
                for(int i=0; i<body.size(); i++) {
                        Statement st = body.get(i);
                        String stRoot = st.buildDotGraph(graph);
                        graph.addEdge(progNode, stRoot, "body[" + i + "]");
                }
                return progNode;
        }

        @Override
        protected void prettyPrint(StringBuilder buf) {
                Statement.prettyPrintStatements(buf, body, 0);
        }

}
```

```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KReturn;
import microjs.jcompiler.utils.DotGraph;

public class Return extends Statement {
    private Expr expr;

    public Return(Expr expr, Location startPos, Location endPos) {
        super(startPos, endPos);
                this.expr = expr;
    }

    @Override
    public KReturn expand() {
        return new KReturn(expr.expand(), getStartPos(), getEndPos());
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String retNode = graph.addNode("Return");
                String exprNode = expr.buildDotGraph(graph);
                graph.addEdge(retNode, exprNode, "expr");

                return retNode;
        }


    @Override
    protected void prettyPrint(StringBuilder buf, int indent_level) {
        indent(buf, indent_level);
        buf.append("return ");
        expr.prettyPrint(buf);
    }
}
```



```
package microjs.jcompiler.frontend.ast;

import java.util.ArrayList;
import java.util.List;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KStatement;

public abstract class Statement extends ASTNode {
        public static final int INDENT_FACTOR = 2;

        /* package */ Statement(Location startPos, Location endPos) {
                super(startPos, endPos);
        }

        @Override
        public abstract KStatement expand();


        protected static List<KStatement> expandStatements(List<Statement> stmts
) {
        List<KStatement> kstmts = new ArrayList<KStatement>();
        for(Statement stmt : stmts) {
                kstmts.add(stmt.expand());
        }
        return kstmts;
        }

        protected void indent(StringBuilder buf, int level) {
                for(int i=0; i< level * INDENT_FACTOR ; i++) {
                        buf.append(' ');
                }
        }

        protected abstract void prettyPrint(StringBuilder buf, int indent_level)
;

        @Override
        protected void prettyPrint(StringBuilder buf) {
                prettyPrint(buf, 0);
        }

        protected static void prettyPrintStatements(StringBuilder buf, List<Stat
ement> stmts, int indent_level) {
                for(Statement stmt : stmts) {
                        stmt.prettyPrint(buf, indent_level);
                        buf.append(";\n");
                }
        }

}
```

```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KVar;
import microjs.jcompiler.utils.DotGraph;

public class Var extends Statement {
    private String name;
    private Expr expr;

    public Var(String name, Expr expr, Location startPos, Location endPos) {
        super(startPos, endPos);
        this.name = name;
                this.expr = expr;
    }

    @Override
    public KVar expand() {
        return new KVar(name, expr.expand(), getStartPos(), getEndPos());
    }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String varNode = graph.addNode("Var[" + name +"]");
                String exprNode = expr.buildDotGraph(graph);
                graph.addEdge(varNode, exprNode, "expr");
                return varNode;
        }


    @Override
    protected void prettyPrint(StringBuilder buf, int indent_level) {
        indent(buf, indent_level);
        buf.append("var ");
        buf.append(name);
        buf.append(" = ");
        expr.prettyPrint(buf);
    }
}
```



```
package microjs.jcompiler.frontend.ast;

import java_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KVoidExpr;
import microjs.jcompiler.utils.DotGraph;

public class VoidExpr extends Statement {
        private Expr expr;

        public VoidExpr(Expr expr, Location startPos, Location endPos) {
                super(startPos, endPos);
                this.expr = expr;
        }

        @Override
        public KVoidExpr expand() {
                return new KVoidExpr(expr.expand(), getStartPos(), getEndPos());
        }

        @Override
        protected String buildDotGraph(DotGraph graph) {
                String voidNode = graph.addNode("VoidExpr");
                String exprNode = expr.buildDotGraph(graph);
                graph.addEdge(voidNode, exprNode, "expr");
                return voidNode;
        }

        @Override
        protected void prettyPrint(StringBuilder buf, int indent_level) {
                indent(buf, indent_level);
                expr.prettyPrint(buf);
        }

}
```