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

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

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();
        }


}
```

```
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_cup.runtime.ComplexSymbolFactory.Location;
import microjs.jcompiler.middleend.kast.KInt;

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

```
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;

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

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

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

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 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.KCall;
import microjs.jcompiler.middleend.kast.KEVar;
import microjs.jcompiler.middleend.kast.KExpr;

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 void prettyPrint(StringBuilder buf) {
                buf.append("(");
                left.prettyPrint(buf);
                buf.append(name);
                right.prettyPrint(buf);
                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.KProg;
import microjs.jcompiler.middleend.kast.KSeq;
import microjs.jcompiler.middleend.kast.KStatement;

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 void prettyPrint(StringBuilder buf) {
                Statement.prettyPrintStatements(buf, body, 0);
        }
}
```

```
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.KAssign;

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);
    }
}
```

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

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

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

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

import java.util.ArrayList;
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.KVoidExpr;

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 KVoidExpr expand() {
        List<String> params = new ArrayList<String>();
        params.add(name);

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

        return new KVoidExpr(
                        new KClosure(params, kbody, getStartPos(), getEndPos()),
                        getStartPos(), getEndPos());
    }

    @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);
    }
}
```

```java
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;

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 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("}");
    }
}
```