```
package microjs.jcompiler.backend;

import microjs.jcompiler.backend.GlobalEnv.VarAlreadyDefined;
import microjs.jcompiler.backend.bytecode.Bool;
import microjs.jcompiler.backend.bytecode.Bytecode;
import microjs.jcompiler.backend.bytecode.Fun;
import microjs.jcompiler.backend.bytecode.Int;
import microjs.jcompiler.backend.bytecode.Prim;
import microjs.jcompiler.backend.bytecode.Unit;
import microjs.jcompiler.middleend.kast.KASTNode;
import microjs.jcompiler.middleend.kast.KASTVisitor;
import microjs.jcompiler.middleend.kast.KAssign;
import microjs.jcompiler.middleend.kast.KCall;
import microjs.jcompiler.middleend.kast.KClosure;
import microjs.jcompiler.middleend.kast.KEVar;
import microjs.jcompiler.middleend.kast.KFalse;
import microjs.jcompiler.middleend.kast.KIf;
import microjs.jcompiler.middleend.kast.KInt;
import microjs.jcompiler.middleend.kast.KProg;
import microjs.jcompiler.middleend.kast.KReturn;
import microjs.jcompiler.middleend.kast.KSeq;
import microjs.jcompiler.middleend.kast.KStatement;
import microjs.jcompiler.middleend.kast.KTrue;
import microjs.jcompiler.middleend.kast.KVar;
import microjs.jcompiler.middleend.kast.KVoidExpr;

public class Compiler implements KASTVisitor {
        private Bytecode bytecode;
        private PrimEnv primEnv;
        private LexicalEnv lexEnv;
        private GlobalEnv globEnv;
        private int lblCount;

        public Compiler(PrimEnv primEnv) {
                this.primEnv = primEnv;
                reset();
        }

        private void reset() {
                bytecode = new Bytecode();
                lexEnv = new LexicalEnv();
                globEnv = new GlobalEnv();
                lblCount = 1;
        }

        public Bytecode compile(KProg prog) {
                reset();
                prog.accept(this);
                return bytecode;
        }

        private String nextLabel() {
                String lbl = "L" + lblCount;
                lblCount++;
                return lbl;
        }

        @Override
        public void visit(KProg prog) {
                prog.getBody().accept(this);
        }

        @Override
        public void visit(KVoidExpr stmt) {
                stmt.getExpr().accept(this);
                bytecode.pop();
        }
```



```
        }

        @Override
        public void visit(KEVar expr) {
                int ref = -1;
                try {
                        ref = lexEnv.fetch(expr.getName());
                        bytecode.fetch(ref);
                } catch(LexicalEnv.VarNotFound err) {
                        try {
                                ref = globEnv.fetch(expr.getName());
                                bytecode.gfetch(ref);
                        } catch(GlobalEnv.VarNotFound e) {
                                try {
                                        Primitive prim = primEnv.fetch(expr.getN
ame());
                                        bytecode.push(new Prim(prim.getId()));
                                } catch(PrimEnv.PrimNotFound ee) {
                                        throw new CompileError(expr, "Not in sco
pe: " + expr.getName());
                                }
                        }
                }
        }

        @Override
        public void visit(KIf stmt) {
                String onFalseLbl = nextLabel();
                String contLbl = nextLabel();
                stmt.getCond().accept(this);
                bytecode.jfalse(onFalseLbl);
                stmt.getThen().accept(this);
                bytecode.jump(contLbl);
                bytecode.label(onFalseLbl);
                stmt.getElse().accept(this);
                bytecode.label(contLbl);
        }

        @Override
        public void visit(KSeq seq) {
                for(KStatement stmt : seq.getStatements()) {
                        stmt.accept(this);
                }
        }

        @Override
        public void visit(KAssign stmt) {
                stmt.getExpr().accept(this);
                try {
                        int ref = lexEnv.fetch(stmt.getVarName());
                        bytecode.store(ref);
                } catch(LexicalEnv.VarNotFound e) {
                        try {
                                int ref = globEnv.fetch(stmt.getVarName());
                                bytecode.gstore(ref);
                        } catch(GlobalEnv.VarNotFound ee) {
                                throw new CompileError(stmt, "Unknown variable t
o assign to: " + stmt.getVarName());
                        }
                }
        }

        @Override
        public void visit(KReturn stmt) {
                stmt.getExpr().accept(this);
                bytecode.bcReturn();
```

```
	}

	@Override
	public void visit(KInt expr) {
		bytecode.push(new Int(expr.getValue()));
	}

	@Override
	public void visit(KTrue expr) {
		bytecode.push(new Bool(true));
	}

	@Override
	public void visit(KFalse expr) {
		bytecode.push(new Bool(false));
	}

	@Override
	public void visit(KVar stmt) {
		int ref;
		try {
			ref = globEnv.extend(stmt.getName());
		} catch(VarAlreadyDefined err) {
			throw new CompileError(stmt, err.getMessage());
		}

		bytecode.galloc();
		stmt.getExpr().accept(this);
		bytecode.gstore(ref);

	}

	@Override
	public void visit(KCall expr) {
		for(int i=expr.getArguments().size()-1; i>=0; i--) {
			expr.getArguments().get(i).accept(this);
		}
		expr.getFun().accept(this);
		bytecode.call(expr.getArguments().size());
	}

	@Override
	public void visit(KClosure expr) {
		String funLbl = nextLabel();
		String contLbl = nextLabel();
		bytecode.jump(contLbl);
		bytecode.label(funLbl);
		lexEnv.extend(expr.getParams());
		expr.getBody().accept(this);
		lexEnv.drop(expr.getParams().size());
		// par sÃ©curitÃ©  (retour "forcÃ©")
		bytecode.push(new Unit());
		bytecode.bcReturn();
		// continuation
		bytecode.label(contLbl);
		bytecode.push(new Fun(funLbl));
	}

	public class CompileError extends java.lang.Error {
		private static final long serialVersionUID = -723059668318220832
3L;

		private KASTNode kast;

		public CompileError(KASTNode kast, String msg) {
			super(msg);
```



```
			this.kast = kast;
		}

		public KASTNode getASTNode() {
			return kast;
		}

	}

	public String genCDeclarations() {
		StringBuilder buf = new StringBuilder();
		buf.append("/* Fichier gÃ©nÃ©rÃ© automatiquement : ne pas Ã©dite
r. */\n\n");

		buf.append(Bytecode.genCDeclarations());
		buf.append(primEnv.genCDeclarations());

		return buf.toString();
	}

	public String genCDefinitions() {
		StringBuilder buf = new StringBuilder();
		buf.append("/* Fichier gÃ©nÃ©rÃ© automatiquement : ne pas Ã©dite
r. */\n\n");

		buf.append(Bytecode.genCDefinitions());
		buf.append(primEnv.genCDefinitions());

		return buf.toString();
	}

}
```