```
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;
        private int lambdaDepth;

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

        private void reset() {
                bytecode = new Bytecode();
                lexEnv = new LexicalEnv();
                globEnv = new GlobalEnv();
                lblCount = 1;
                lambdaDepth = 0;
        }

        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);
                if(lambdaDepth > 0) {
                        bytecode.bcReturn();
                } else {
                        bytecode.pop();
                }
        }

        @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());
                lambdaDepth++;
                expr.getBody().accept(this);
                lambdaDepth--;
                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();
        }

}
```

```
package microjs.jcompiler.backend;

import java.util.HashMap;
import java.util.Map;

public class GlobalEnv {
        private Map<String, Integer> vars;

        public GlobalEnv() {
                vars = new HashMap<String, Integer>();
        }

        public int fetch(String var) throws VarNotFound {
                Integer ref = vars.get(var);
                if(ref==null) {
                        throw new VarNotFound(var);
                }
                return ref;
        }

        public int extend(String var) throws VarAlreadyDefined {
                if(vars.containsKey(var)) {
                        throw new VarAlreadyDefined(var);
                }
                vars.put(var, vars.size());
                return vars.size() - 1;
        }

        public class VarAlreadyDefined extends Exception {
                private static final long serialVersionUID = 5877188650153737752
L;

                public VarAlreadyDefined(String var) {
                        super("Global variable '" + var + "' already defined.");
                }
        }

        public class VarNotFound extends Exception {
                private static final long serialVersionUID = -749734412758070265
8L;

                public VarNotFound(String var) {
                        super("Global variable '" + var + "' not found.");
                }
        }


}
```



```
package microjs.jcompiler.backend;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.List;

public class LexicalEnv {
        private Deque<String> lex;

        public LexicalEnv() {
                lex = new ArrayDeque<String>();
        }

        public int fetch(String var) throws VarNotFound {
                int i = 0;
                for(String v : lex) {
                        if(var.equals(v)) {
                                return i;
                        }
                        i++;
                }
                throw new VarNotFound(var);
        }

        public void extend(List<String> vars) {
                for(int i=vars.size()-1;i>=0;i--) {
                        lex.addFirst(vars.get(i));
                }
        }

        public void drop(int howMany) {
                for(int i=0;i<howMany;i++) {
                        lex.removeFirst();
                }
        }

        @Override
        public String toString() {
                return lex.toString();
        }

        public class VarNotFound extends Exception {
                private static final long serialVersionUID = -428657346019066106
L;

                public VarNotFound(String var) {
                        super("Local variable '" + var + "' not found.");
                }
        }


}
```

```
package microjs.jcompiler.backend;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;

public class PrimEnv {
	private Map<String, Primitive> prims;
	private List<String> primIds;

	public PrimEnv() {
		prims = new TreeMap<String, Primitive>();
		primIds = new ArrayList<String>();
	}

	public void register(Primitive prim) {
		if(prims.containsKey(prim.getName())) {
			throw new PrimAlreadyDefined(prim.getName());
		}
		prims.put(prim.getName(), prim);
		prim.setId(primIds.size());
		primIds.add(prim.getName());
	}

	public Primitive fetch(String name) throws PrimNotFound {
		Primitive prim = prims.get(name);
		if(prim==null) {
			throw new PrimNotFound(name);
		}
		return prim;
	}

	public class PrimAlreadyDefined extends Error {
		private static final long serialVersionUID = -129581314413064877
L;

		public PrimAlreadyDefined(String prim) {
			super("Primitive '" + prim + "' already defined.");
		}
	}

	public class PrimNotFound extends Exception {
		private static final long serialVersionUID = 132383297676147375L
;

		public PrimNotFound(String prim) {
			super("Primitive '" + prim + "' not found.");
		}
	}

	public static PrimEnv defaultPrimEnv() {
		PrimEnv primEnv = new PrimEnv();

		primEnv.register(new Primitive("+", "Addition", "P_ADD"));
		primEnv.register(new Primitive("-", "Subtraction", "P_SUB"));
		primEnv.register(new Primitive("*", "Multiplication", "P_MUL"));
		primEnv.register(new Primitive("/", "Division", "P_DIV"));
		primEnv.register(new Primitive("==", "Equality", "P_EQ"));
		primEnv.register(new Primitive("<", "Lower", "P_INF"));
		primEnv.register(new Primitive(">", "Greater", "P_SUP"));
		primEnv.register(new Primitive("<=", "Lower or equal", "P_INFEQ"
));
		primEnv.register(new Primitive(">=", "Greater or equal", "P_SUPE
Q"));
```



```
		return primEnv;
	}

	public String genCDeclarations() {
		StringBuilder buf = new StringBuilder();
		buf.append("/* Constantes pour les primitives */\n");
		for (String primName : primIds) {
			Primitive prim = prims.get(primName);
			buf.append("\n");
			buf.append(prim.getCDeclaration());
			buf.append("\n");
		}
		buf.append("\n/** Noms des primitives */\n");
		buf.append("extern const char *primitive_names[];\n");
		return buf.toString();
	}

	public String genCDefinitions() {
		StringBuilder buf = new StringBuilder();
		buf.append("/** Noms des primitives */\n");
		buf.append("const char *primitive_names[] = {\n");
		for (String primName : primIds) {
			buf.append("  \""); buf.append(primName); buf.append("\"
,\n");
		}
		buf.append("  \"<unknown>\"\n");
		buf.append("};\n");
		return buf.toString();
	}


}
```

```
package microjs.jcompiler.backend;

public class Primitive {
        private String name;
        private int id;
        private String doc;
        private String cname;

        public Primitive(String name, String doc, String cname) {
                this.name = name;
                this.id = -1;
                this.doc = doc;
                this.cname = cname;
        }

        public String getName() {
                return name;
        }

        public int getId() {
                return id;
        }

        public String getDoc() {
                return doc;
        }

        public String getCName() {
                return cname;
        }

        /* package */ void setId(int id) {
                this.id = id;
        }

        public String getCDeclaration() {
                StringBuilder buf = new StringBuilder();
                buf.append("/** primitive "); buf.append(getName()); buf.append(
" */\n");
                buf.append("#define "); buf.append(getCName()); buf.append(" ");
 buf.append(getId());
                return buf.toString();
        }


}
```



```
package microjs.jcompiler.backend;

import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import microjs.jcompiler.backend.bytecode.BCInstr;
import microjs.jcompiler.backend.bytecode.Bytecode;

public class Serializer {
        private Map<String, Integer> labels;
        private int bcSize;
        private Bytecode bc;
        private StringBuilder buf;

        public Serializer(Bytecode bc) {
                labels = null;
                bcSize = -1;
                this.bc = bc;
                buf = null;
        }

        public void encode(int val) {
                buf.append(" ");
                buf.append(val);
        }

        public int fetchLabel(String lbl) {
                return labels.get(lbl);
        }

        private void computeJumps() {
                List<BCInstr> code = bc.getCode();
                labels = new HashMap<String, Integer>();

                int pc = 0;
                for(int i=0; i < code.size(); i++) {
                        BCInstr instr = code.get(i);
                        if(instr.isLabel()) {
                                labels.put(instr.asLabel().getRef(), pc);
                        } else {
                                pc += instr.getSize();
                        }
                }

                bcSize = pc;
        }

        public String serialize() {
                computeJumps();

                buf = new StringBuilder();

                // preamble
                buf.append("424242");
                encode(bcSize);

                for(BCInstr instr : bc.getCode()) {
                        if(instr.isLabel()) {
                                // rien Ã  faire
                        } else {
                                instr.genBytecode(this);
                        }
                }
```

```
                buf.append(" ");  // need at least one space at the end

                return buf.toString();

        }

        public void serializeToFile(String filename) throws IOException {
                FileWriter writer = new FileWriter(filename);
                String bcStr = serialize();
                writer.write(bcStr);
                writer.close();
        }
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public abstract class BCInstr {
        public abstract int getOpcode();
        public abstract String getOpcodeName();

        public abstract void genBytecode(Serializer gen);

        public abstract int getSize();

        public boolean isLabel() {
                return false;
        }

        public Label asLabel() {
                return (Label) this;
        }

        public String genCDeclaration() {
                StringBuilder buf = new StringBuilder();
                buf.append("/** opcode "); buf.append(getOpcodeName()); buf.appe
nd(" */\n");
                buf.append("#define I_"); buf.append(getOpcodeName()); buf.appen
d(" "); buf.append(getOpcode());
                buf.append("\n");
                return buf.toString();
        }
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public abstract class BCValue {
        public abstract int getOpcode();
        public abstract String getOpcodeName();

        public abstract void genBytecode(Serializer gen);

        public abstract int getSize();

        public String genCDeclaration() {
                StringBuilder buf = new StringBuilder();
                buf.append("/** type "); buf.append(getOpcodeName()); buf.append
(" */\n");
                buf.append("#define T_"); buf.append(getOpcodeName()); buf.appen
d(" "); buf.append(getOpcode());
                buf.append("\n");
                return buf.toString();
        }

}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Bool extends BCValue{
        private boolean value;

        public Bool(boolean value) {
                this.value = value;
        }

        @Override
        public int getOpcode() {
                return 4;
        }

        @Override
        public String getOpcodeName() {
                return "BOOL";
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                if(value == true) {
                        gen.encode(1);
                } else {
                        gen.encode(0);
                }
        }

        @Override
        public int getSize() {
                return 2;
        }

        public String toString() {
                if(value) {
                        return "BOOL TRUE";
                } else {
                        return "BOOL FALSE";
                }
        };
}
```

```
package microjs.jcompiler.backend.bytecode;

import java.util.ArrayList;
import java.util.List;
import java.util.SortedMap;
import java.util.TreeMap;

public class Bytecode {
	private List<BCInstr> code;

	public Bytecode() {
		code = new ArrayList<BCInstr>();
	}

	public List<BCInstr> getCode() {
		return code;
	}

	public Bytecode galloc() {
		code.add(new GAlloc());
		return this;
	}

	public Bytecode gfetch(int ref) {
		code.add(new GFetch(ref));
		return this;
	}

	public Bytecode gstore(int ref) {
		code.add(new GStore(ref));
		return this;
	}

	public Bytecode fetch(int ref) {
		code.add(new Fetch(ref));
		return this;
	}

	public Bytecode store(int ref) {
		code.add(new Store(ref));
		return this;
	}

	public Bytecode push(BCValue value) {
		code.add(new Push(value));
		return this;
	}

	public Bytecode pop() {
		code.add(new Pop());
		return this;
	}

	public Bytecode call(int ref) {
		code.add(new Call(ref));
		return this;
	}

	public Bytecode bcReturn() {
		code.add(new Return());
		return this;
	}

	public Bytecode label(String lbl) {
		code.add(new Label(lbl));
		return this;
```



```
	}

	public Bytecode jump(String lbl) {
		code.add(new Jump(lbl));
		return this;
	}

	public Bytecode jfalse(String lbl) {
		code.add(new JFalse(lbl));
		return this;
	}

	@Override
	public String toString() {
		StringBuilder buf = new StringBuilder();
		for(BCInstr instr : code) {
			buf.append(instr.toString());
			buf.append("\n");
		}
		return buf.toString();
	}

	private static SortedMap<Integer, BCInstr> instructionSet = null;

	public static SortedMap<Integer, BCInstr> getInstructionSet() {
		if(instructionSet == null) {
			BCInstr instr = null;
			instructionSet = new TreeMap<Integer, BCInstr>();
			instr = new Label("lbl");
			instructionSet.put(instr.getOpcode(), instr);
			instr = new GAlloc();
			instructionSet.put(instr.getOpcode(), instr);
			instr = new Push(new Unit());
			instructionSet.put(instr.getOpcode(), instr);
			instr = new GStore(-1);
			instructionSet.put(instr.getOpcode(), instr);
			instr = new Pop();
			instructionSet.put(instr.getOpcode(), instr);
			instr = new Jump("lbl");
			instructionSet.put(instr.getOpcode(), instr);
			instr = new GFetch(-1);
			instructionSet.put(instr.getOpcode(), instr);
			instr = new Call(-1);
			instructionSet.put(instr.getOpcode(), instr);
			instr = new Return();
			instructionSet.put(instr.getOpcode(), instr);
			instr = new Fetch(-1);
			instructionSet.put(instr.getOpcode(), instr);
			instr = new JFalse("lbl");
			instructionSet.put(instr.getOpcode(), instr);
			instr = new Store(-1);
			instructionSet.put(instr.getOpcode(), instr);
		}
		return instructionSet;
	}

	private static SortedMap<Integer, BCValue> valueSet = null;

	public static SortedMap<Integer, BCValue> getValueSet() {
		if(valueSet == null) {
			BCValue val = null;
			valueSet = new TreeMap<Integer, BCValue>();
			val = new Unit();
			valueSet.put(val.getOpcode(), val);
			val = new Int(42);
			valueSet.put(val.getOpcode(), val);
```

```
                        val = new Prim(42);
                        valueSet.put(val.getOpcode(), val);
                        val = new Fun("lbl");
                        valueSet.put(val.getOpcode(), val);
                        val = new Bool(false);
                        valueSet.put(val.getOpcode(), val);
                }
                return valueSet;
        }

        public static String genCDeclarations() {
                StringBuilder buf = new StringBuilder();
                buf.append("/* Constantes pour les opcodes */\n");
                for(Integer id : getInstructionSet().keySet()) {
                        BCInstr instr = getInstructionSet().get(id);
                        buf.append("\n");
                        buf.append(instr.genCDeclaration());
                }
                buf.append("\n");
                buf.append("/** Noms des opcodes */\n");
                buf.append("extern const char *opcode_names[];\n\n");

                buf.append("/* Constantes pour les types */\n");

                for(Integer id : getValueSet().keySet()) {
                        BCValue val = getValueSet().get(id);
                        buf.append("\n");
                        buf.append(val.genCDeclaration());
                }
                buf.append("\n/* type T_PAIR (rÃ©servÃ©) */\n");
                buf.append("#define T_PAIR "); buf.append(getValueSet().size());
 buf.append("\n");
                buf.append("\n");
                buf.append("/** Noms des types */\n");
                buf.append("extern const char *type_names[];\n\n");

                return buf.toString();
        }

        public static String genCDefinitions() {
                StringBuilder buf = new StringBuilder();

                buf.append("/** Noms des opcodes */\n");
                buf.append("const char *opcode_names[] = {\n");
                for(Integer id : getInstructionSet().keySet()) {
                        BCInstr instr = getInstructionSet().get(id);
                        buf.append("  \""); buf.append(instr.getOpcodeName()); b
uf.append("\",\n");
                }
                buf.append("  \"<unknown>\"\n");
                buf.append("};\n\n");

                buf.append("/** Noms des types */\n");
                buf.append("const char *type_names[] = {\n");
                for(Integer id : getValueSet().keySet()) {
                        BCValue val = getValueSet().get(id);
                        buf.append("  \""); buf.append(val.getOpcodeName()); buf
.append("\",\n");
                }
                buf.append("  \"pair\",\n");
                buf.append("  \"<unknown>\"\n");
                buf.append("};\n\n");

                return buf.toString();
        }
```



```
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Call extends BCInstr {
        private int ref;

        public Call(int ref) {
                this.ref = ref;
        }

        @Override
        public String getOpcodeName() {
                return "CALL";
        }

        @Override
        public int getOpcode() {
                return 6;
        }

        @Override
        public int getSize() {
                return 2;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(ref);
        }

        @Override
        public String toString() {
                return "  CALL " + ref;
        }
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Fetch extends BCInstr {
        private int ref;

        public Fetch(int ref) {
                this.ref = ref;
        }

        @Override
        public String getOpcodeName() {
                return "FETCH";
        }

        @Override
        public int getOpcode() {
                return 8;
        }

        @Override
        public int getSize() {
                return 2;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(ref);
        }

        @Override
        public String toString() {
                return "  FETCH " + ref;
        }
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Fun extends BCValue{
        private String lbl;

        public Fun(String lbl) {
                this.lbl = lbl;
        }

        @Override
        public int getOpcode() {
                return 3;
        }

        @Override
        public String getOpcodeName() {
                return "FUN";
        }

        @Override
        public int getSize() {
                return 2;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(gen.fetchLabel(lbl));
        }

        public String toString() {
                return "FUN " + lbl;
        };
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class GAlloc extends BCInstr {
        public GAlloc() {

        }

        @Override
        public String getOpcodeName() {
                return "GALLOC";
        }

        @Override
        public int getOpcode() {
                return 0;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
        }

        @Override
        public int getSize() {
                return 1;
        }


        @Override
        public String toString() {
                return "  GALLOC";
        }
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class GFetch extends BCInstr {
        private int ref;

        public GFetch(int ref) {
                this.ref = ref;
        }

        @Override
        public String getOpcodeName() {
                return "GFETCH";
        }

        @Override
        public int getOpcode() {
                return 5;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(ref);
        }

        @Override
        public int getSize() {
                return 2;
        }


        @Override
        public String toString() {
                return "  GFETCH " + ref;
        }
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class GStore extends BCInstr {
        private int ref;

        public GStore(int ref) {
                this.ref = ref;
        }

        @Override
        public String getOpcodeName() {
                return "GSTORE";
        }

        @Override
        public int getOpcode() {
                return 2;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(ref);
        }

        @Override
        public int getSize() {
                return 2;
        }


        @Override
        public String toString() {
                return "  GSTORE " + ref;
        }
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Int extends BCValue{
        private int value;

        public Int(int value) {
                this.value = value;
        }

        @Override
        public int getOpcode() {
                return 1;
        }

        @Override
        public String getOpcodeName() {
                return "INT";
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(value);
        }

        @Override
        public int getSize() {
                return 2;
        }

        public String toString() {
                return "INT " + value;
        };
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class JFalse extends BCInstr {
        private String ref;

        public JFalse(String ref) {
                this.ref = ref;
        }

        @Override
        public String getOpcodeName() {
                return "JFALSE";
        }

        @Override
        public int getOpcode() {
                return 9;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(gen.fetchLabel(ref));
        }

        @Override
        public int getSize() {
                return 2;
        }


        @Override
        public String toString() {
                return "  JFALSE " + ref;
        }
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Jump extends BCInstr {
        private String ref;

        public Jump(String ref) {
                this.ref = ref;
        }

        @Override
        public String getOpcodeName() {
                return "JUMP";
        }

        @Override
        public int getOpcode() {
                return 4;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(gen.fetchLabel(ref));
        }

        @Override
        public int getSize() {
                return 2;
        }


        @Override
        public String toString() {
                return "  JUMP " + ref;
        }
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Label extends BCInstr {
        private String ref;

        public Label(String ref) {
                this.ref = ref;
        }

        public String getRef() {
                return ref;
        }

        public boolean isLabel() {
                return true;
        }

        @Override
        public String getOpcodeName() {
                return "LABEL";
        }

        @Override
        public int getOpcode() {
                return -1;
        }

        @Override
        public void genBytecode(Serializer gen) {
                // nothing to do (no generation of labels)
        }

        @Override
        public int getSize() {
                return 0;
        }


        @Override
        public String toString() {
                return ref + ":";
        }
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Pop extends BCInstr {
        public Pop() {

        }

        @Override
        public String getOpcodeName() {
                return "POP";
        }

        @Override
        public int getOpcode() {
                return 3;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
        }

        @Override
        public int getSize() {
                return 1;
        }


        @Override
        public String toString() {
                return "  POP";
        }
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Prim extends BCValue{
        private int ref;

        public Prim(int ref) {
                this.ref = ref;
        }

        @Override
        public int getOpcode() {
                return 2;
        }

        @Override
        public String getOpcodeName() {
                return "PRIM";
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(ref);
        }

        @Override
        public int getSize() {
                return 2;
        }


        public String toString() {
                return "PRIM " + ref;
        };
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Push extends BCInstr {
        private BCValue value;

        public Push(BCValue value) {
                this.value = value;
        }

        @Override
        public String getOpcodeName() {
                return "PUSH";
        }

        @Override
        public int getOpcode() {
                return 1;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                value.genBytecode(gen);
        }

        @Override
        public int getSize() {
                return value.getSize() + 1;
        }



        @Override
        public String toString() {
                return "  PUSH " + value.toString();
        }
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Return extends BCInstr {
        public Return() {

        }

        @Override
        public String getOpcodeName() {
                return "RETURN";
        }

        @Override
        public int getOpcode() {
                return 7;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
        }

        @Override
        public int getSize() {
                return 1;
        }

        @Override
        public String toString() {
                return "  RETURN";
        }
}
```

```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Store extends BCInstr {
        private int ref;

        public Store(int ref) {
                this.ref = ref;
        }

        @Override
        public String getOpcodeName() {
                return "STORE";
        }

        @Override
        public int getOpcode() {
                return 10;
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
                gen.encode(ref);
        }

        @Override
        public int getSize() {
                return 2;
        }


        @Override
        public String toString() {
                return "  STORE " + ref;
        }
}
```



```
package microjs.jcompiler.backend.bytecode;

import microjs.jcompiler.backend.Serializer;

public class Unit extends BCValue{
        public Unit() {

        }

        @Override
        public int getOpcode() {
                return 0;
        }

        @Override
        public String getOpcodeName() {
                return "UNIT";
        }

        @Override
        public void genBytecode(Serializer gen) {
                gen.encode(getOpcode());
        }

        @Override
        public int getSize() {
                return 1;
        }

        public String toString() {
                return "UNIT";
        };
}
```