```java
package microjs.jcompiler.main;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import java_cup.runtime.ComplexSymbolFactory;
import java_cup.runtime.Symbol;
import microjs.jcompiler.backend.Compiler;
import microjs.jcompiler.backend.PrimEnv;
import microjs.jcompiler.backend.Serializer;
import microjs.jcompiler.backend.Compiler.CompileError;
import microjs.jcompiler.backend.bytecode.Bytecode;
import microjs.jcompiler.middleend.kast.KProg;
import microjs.jcompiler.frontend.ast.Prog;
import microjs.jcompiler.frontend.lexer.Lexer;
import microjs.jcompiler.frontend.parser.parser;

enum ControlMode {
                PARSE_ONLY,
                PARSE_AND_SHOW_AST,
                PARSE_AND_EXPAND,
                COMPILE_AND_SHOW_BYTECODE,
                COMPILE_AND_GENERATE_TARGET,
                GEN_CONSTANTS
}

public class Main {

        public static void abort(String msg, int errCode) {
                System.out.printf("Now quitting\n  ==> %s\n\nBye bye !\n", msg);
                System.exit(errCode);
        }

        public static Prog parseFile(String filename, boolean do_debug_parse) {
          File file;
          FileReader fr = null;
      //try {

          file = new File(filename);
          try {
          fr = new FileReader(file);
          } catch(FileNotFoundException e) {
                  abort(e.getMessage(), 1);
          }
          // System.out.printf("File encoding = %s\n", fr.getEncoding());

          Lexer lexer = new Lexer(fr);

          parser parser_obj = new parser(lexer, new ComplexSymbolFactory());

          Symbol sprog;
          try {
          if (do_debug_parse)
                  sprog = parser_obj.debug_parse();
          else
                  sprog = parser_obj.parse();
          }
          catch (java.lang.NullPointerException e) {
              return parser_obj.resultat;
          }
          catch(Exception e) {
              throw new Error(e);
          }
```

```java

          return parser_obj.resultat;

//      } catch (Exception e) {
//        throw new Error(e);
//        // abort(e.getMessage(), 1);
//      } finally {
//        try {
//                      fr.close();
//              } catch (IOException e) {
//                      abort("Cannot close file", 1);
//              }
//      }

      //return null;
        }

        private static String usageString() {
                return
"Usage:\n"
+ "  compiler [opts] <source_file>\n"
+ "opts:\n"
+ "  -parse Parse and show parsed program\n"
+ "  -astdot Parse and generate AST graph (dot file)\n"
+ "  -expand Parse, expand and show kernel abstract syntax tree\n"
+ "  -compile Compile and show bytecode\n"
+ "  -gen Compile and generate target (default mode)\n"
+ "  -vmconst Generate the constants for the VM\n"
+ "  -help  Display this list of options\n"
+ "  --help  Display this list of options\n";
        }

        private static ControlMode parseControlMode(String[] args) {
                for(int i=0;i<args.length; i++) {
                        String arg = args[i];
                        if(arg.equals("-parse")) {
                                return ControlMode.PARSE_ONLY;
                        } else if(arg.equals("-astdot")) {
                                return ControlMode.PARSE_AND_SHOW_AST;
                        } else if(arg.equals("-expand")) {
                                return ControlMode.PARSE_AND_EXPAND;
                        } else if(arg.equals("-compile")) {
                                return ControlMode.COMPILE_AND_SHOW_BYTECODE;
                        } else if(arg.equals("-gen")) {
                                return ControlMode.COMPILE_AND_GENERATE_TARGET;
                        } else if(arg.equals("-vmconst")) {
                                return ControlMode.GEN_CONSTANTS;
                        } else if(arg.equals("-help") || arg.equals("--help")) {
                                System.out.println(usageString());
                                System.exit(0);
                        } else if(arg.startsWith("-")) {
                                System.err.println("Warning: don't know option:
" + arg);
                        }
                        // skip other options
                }
                return ControlMode.COMPILE_AND_GENERATE_TARGET; // default mode
        }

        private static String parseFilename(String[] args) {
                if(args.length == 0) {
                        abort("Missing filename on the command line", 1);
                }
                if(args[args.length-1].startsWith("-")) {
                        System.err.println("Last argument must be a filename");
                        System.err.println(usageString());
                }

                return args[args.length-1];
        }

        public static void main(String[] args) {
                System.out.println("Microjs compiler v0.0.1\n");
                System.out.println("-----------------------\n");

                Compiler compiler = new Compiler(PrimEnv.defaultPrimEnv());

                ControlMode mode = parseControlMode(args);

                if(mode==ControlMode.GEN_CONSTANTS) {
                        FileWriter writer = null;
                        System.out.println("Generate header file: constants.h");
                        try {
                                writer = new FileWriter("constants.h");
                                writer.write(compiler.genCDeclarations());
                                writer.close();
                                System.out.println(" ... done.");
                        } catch (IOException e) {
                                abort(e.getMessage(), 0);
                        }

                        System.out.println("Generate source file: constants.c");
                        try {
                                writer = new FileWriter("constants.c");
                                writer.write(compiler.genCDefinitions());
                                writer.close();
                                System.out.println(" ... done.");
                        } catch (IOException e) {
                                abort(e.getMessage(), 0);
                        }

                        abort("Constants generation successful", 0);
                }


                String filename = parseFilename(args);

                System.out.printf("[1] Parsing source file: %s...\n", filename);
                Prog prog = parseFile(filename, false);

                System.out.println("... parsing done.");

                if(mode==ControlMode.PARSE_ONLY) {
                        System.out.printf("Parsed program:\n====================
=======\n%s\n===========================\n", prog.toString());
                        abort("I could compile, you know...", 0);
                } else if(mode==ControlMode.PARSE_AND_SHOW_AST) {
                        System.out.println("  => generating dot file: " + filena
me + ".dot");
                        try {
                                FileWriter writer = new FileWriter(filename + ".
dot");

                                writer.write(prog.genDotGraph().toString());
                                writer.close();
                                abort("I could compile, you know...", 0);
                        } catch(IOException e) {
                                abort(e.getMessage(), 0);
                        }
                }

                System.out.println("[2] Expanding...");
                KProg kprog = prog.expand();
```

```
                System.out.println("... expansion done.");


                if(mode==ControlMode.PARSE_AND_EXPAND) {
                        System.out.printf("Kernal Abstract Syntax Tree:\n=======
====================\n%s\n===========================\n", kprog.toString());
                        abort("I could compile, you know...", 0);
                }

                System.out.println("[3] Compiling ...");

                Bytecode bytecode = null;
                try {
                        bytecode = compiler.compile(kprog);
                } catch(Compiler.CompileError err) {
                        System.err.println("Compilation error at line " + err.ge
tASTNode().getStartPos().getLine() + " column " + err.getASTNode().getStartPos()
.getColumn() + ":");
                        System.err.println("  ==> " + err.getMessage());
                        abort("compilation failed.", 0);
                }

                System.out.println("... compilation done.");

                if(mode == ControlMode.COMPILE_AND_SHOW_BYTECODE) {
                        System.out.printf("Bytecode:\n==========================
=\n%s\n===========================\n", bytecode.toString());
                         abort("I could generate the target, you know...", 0);
                }

                System.out.println("[3] Serializing ...");
                String bcFilename = filename + ".bc";

                Serializer gen = new Serializer(bytecode);
                try {
                        gen.serializeToFile(bcFilename);
                } catch (IOException e) {
                        abort(e.getMessage(), -1);
                }

                System.out.println("... serialized to file '" + bcFilename + "'"
);
                abort("Nothing left to do, I can rest.", 0);

        }

}
```