

Term Project

PROJECT OVERVIEW

You will be writing a complete compiler for a subset of Java that we'll call MiniJava.¹ You will also be designing and implementing some small extension to MiniJava. Nearly all decisions regarding implementation technologies will be left to you.

ABOUT MINIJAVA

All legal MiniJava programs are also legal Java programs, with the same meaning. (Of course, the converse is not true: there are plenty of Java programs that are not legal MiniJava programs. This is to simplify the MiniJava compiler. It's enough to write 6,000–8,000 lines of code in a term!)

Emphasis in the implementation should be placed on writing clean, understandable code. This will make reading the code easier, both for me and you. Gross, low-level hacks, even for better compiler efficiency, will not be appreciated.

MiniJava Lexical Structure

```

Program ::=      (Token | Whitespace)*
Token ::=       ID | Integer | ReservedWord | Operator | Delimiter
ID ::=         Letter (Letter | Digit)*
Letter ::=      a | ... | z | A | ... | Z
Digit ::=      0 | ... | 9
Integer ::=     Digit+
ReservedWord ::= class | public | static | extends | void | int | boolean | if |
                 else | while | return | null | true | false | this | new |
                 String | main | System.out.println
Operator ::=    + | - | * | / | < | <= | >= | > | == | != | && | || | !
Delimiter ::=   ; | . | , | = | ( | ) | { | } | [ | ]
Whitespace ::=  <space> | <tab> | <newline> | Comment
Comment ::=     (“//” to end of line, “/*...*/” unnested block comments)
  
```

Comments are specified informally above. Part of your first task is to figure out how to formalize them. (By “formalize” here I mean implement.)

¹ Thanks to Craig Chambers at the University of Washington for the project design on which this version of MiniJava is based.

MiniJava Syntax

In the following, a token or tokens enclosed in lightweight square brackets, like [**extends** ID] is optional. A token or tokens enclosed in lightweight braces, like {ClassDecl} may appear 0 or more times.

```

Program ::=      MainClassDecl {ClassDecl}
MainClassDecl ::= class ID { public static void main (String[ ] ID) { {Stmt} } }
ClassDecl ::=    class ID [extends ID] { {ClassVarDecl} {MethodDecl} }
ClassVarDecl ::= Type ID;
MethodDecl ::=   public Type ID ( [Formal {, Formal}] ) { {Stmt} return Expr; }
Formal ::=       Type ID
Type ::=         int | boolean | ID
Stmt ::=         Type ID = Expr;
                  | { {Stmt} }
                  | if (Expr) Stmt else Stmt
                  | while (Expr) Stmt
                  | System.out.println(Expr);
                  | ID = Expr;
Expr ::=         Expr (+ | - | * | / | < | <= | >= | > | == | != | && | || ) Expr
                  | ( - | ! ) Expr
                  | Expr . ID ( [Expr {, Expr}] )
                  | new ID ( )
                  | ID
                  | this
                  | Integer
                  | null
                  | true
                  | false
                  | (Expr)

```

The precedence and associativity of the operators and other expression forms are the same as in Java, except that the == and != operations are non-associative (Java's are left-associative).

MiniJava Typechecking Rules

In addition to Java's normal typechecking requirements, MiniJava imposes the following additional restrictions:

- Programs are defined in a single input file.
- There are no library classes available, including `Object` and `String`.
- The main method's `String[]` formal parameter is a mere syntactic placeholder, and cannot be accessed in the main method's body.
- If a class extends another, that other class must have been declared earlier in the program.
- No method overloading is allowed, i.e., only a single method with a given name can be declared in any class, and if a class inherits a method of some name from a superclass, it can only declare a method locally that directly overrides it, with identical argument and result types.
- No class variable shadowing is allowed, i.e., if a class inherits an instance class variable of a given name from a superclass, it cannot also declare an instance class variable of the same name.
- The argument to `System.out.println` must be an integer.

LANGUAGE EXTENSIONS

Towards the end of the term, you'll work with me to choose an extension or set of extensions to MiniJava. Each language addition will require language design and specification work (extending the initial MiniJava language specification to include the new features) as well as implementation work (extending your MiniJava compiler to compile the new features).

Some extension ideas include:

- floating-point values and operators
- arrays and array indexing
- if/then statements without else
- simple for loops
- break statements
- static class variables
- multi-valued return statements
- closures

Another sort of extension might be to implement an optimization pass or passes in your compiler.

I'll adjust the complexity of the required extensions depending on how the projects are progressing. We have a lot to cover in a 10 week term, so we may need to make adjustments along the way.

IMPLEMENTATION TECHNOLOGY

Nearly all decisions about implementation technology will be left to you. I intend to use examples written in Java, with the JFlex and CUP generator tools. My target platform will be the Java Virtual Machine. I will be able to help more if your choices align with mine, though I will try to help as much as I can if you choose to follow your muse instead. (I can also offer significant help with the ANTLR family of tools.)

Your team will need to decide on:

- implementation language
- whether to hand code your scanner or use a scanner generator (and if so, which one)
- which parser generator to use
- what platform to target

MILESTONES

Milestone dates are listed on the course schedule. The dates are subject to change, but will be no earlier than those listed. Specific test cases and more detailed instructions will be provided during the course of the term.