Project Description

You will start with a compiler, written in Java, for a subset of Java called MiniJava. The course project is to extend the MiniJava compiler to also support the following additional language constructs:

- comments ("`//`" to end of line, `/*...*/` unnested block comments)
- allow "_" in identifiers
- floating-point values and operators
- arrays and array indexing
- `||` (short-circuiting or) operation
- `if`/`then` statements without `else`
- simple `for` loops
- `break` statements
- `static` class variables

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 keep the MiniJava compiler a lot simpler than the full Java virtual machine!)

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 the initial Java implementation of MiniJava to compile the new features).

Emphasis in the implementation should be placed on making clean, understandable changes that fit in well to the existing compiler structure. This will make reading the changed code easier, both for me, the TA, and you. Gross, low-level hacks, even for better compiler efficiency, will not be appreciated.

You are encouraged to organize yourselves into two-person teams to work on the project. When turning in a part of the project, simply list the names of the members of the project team. The grade for that part of the project will be assigned to all members of the team. Teams can be dissolved at any point, after which the two people will do independent work and receive independent grades. You are not required to be on a team, but you are likely to learn more and work less by being on a team.

The MiniJava compiler implementation is intended to be developed on Unix platforms with the GNU and Sun Java tools, including java, javac, GNU make, and CVS. We will probably be making extensions to the base compiler available over the course of the quarter via CVS, and we use a Makefile to manage building and testing the compiler. The instructional Linux machine, `attu`, is a suitable place to work on the MiniJava project. You can work on any platform you like, using any tools you like, but you will need to provide your own support and mechanisms for handling our CVS-based updates.

Initial 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 | true | false | this | new | String | main |
                 System.out.println
Operator     ::= + | - | * | / | < | <= | >= | > | == | != | && | !
Delimiter    ::= ; | . | , | = | ( | ) | { | } | [ | ]

Whitespace   ::= <space> | <tab> | <newline>
```

Initial MiniJava syntax

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**;**
  | **{** {Stmt} **}**
  | **if (**Expr**)** Stmt **else** Stmt
  | **while (**Expr**)** Stmt
  | **System.out.println(**Expr**);**
  | ID **=** Expr**;**

Expr ::= Expr (**+** | **-** | ***** | **/** | **<** | **<=** | **>=** | **>** | **==** | **!=** | **&&**) Expr
  | **!** Expr
  | Expr **.** ID **(** [Expr {**,** Expr}] **)**
  | ID
  | **this**
  | Integer
  | **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).

Initial 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.
- The argument to `System.out.println` must be an integer.

Unlike Java, MiniJava currently does not perform any "definite assignment" checking to ensure that local variables are assigned to before they are used. The MiniJava evaluator checks this during evaluation, but the target-code compiler does not check it at all.