

In this assignment you will extend the MiniJava language and compiler to enable the double data type. The report is a crucial part of this project and all other projects in this course. You should proof read your report, use a spell checker and possibly a grammar checker, and rewrite it a couple of times. **EACH spelling error will result in 5% off the grade for this project.** Grammar errors will also be costly. If the same grammar errors appear in more than two reports, the resulting penalty will increase. This assignment should be done in groups of two. Please speak to me at least a week before the assignment is due if you cannot find a partner.

1 Motivation

The MiniJava compiler will be used throughout this course for hands-on projects such as implementing a garbage collector and implementing data-flow based code optimizations. By extending the MiniJava language and compiler to include the double data type, you will review all of the phases of a typical compiler and become comfortable with the MiniJava compiler.

2 The Assignment

Your assignment is to extend the MiniJava language (see the grammar at <http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html>) and compiler to include the double data type. DoubleTest1.java is a simple example of a MiniJava program that uses a double data type.

```
/*
  DoubleTest1.java

  A program that prints a constant double.

MMS Aug 07
*/

class DoubleTest1 {
    public static void main(String[] a){
        System.out.println(1.1);
    }
}
```

Your group will need to create MiniJava programs that use doubles and compare the output of those test programs when compiled with `javac` and run with `java` versus when compiled with the

MiniJava compiler and run with MARS. For example,

```
java -jar --two-pass-mips MJ.jar DoubleTest1.java
// note creation of .s and .dot files
java -jar Mars.jar DoubleTest1.java.s > mars.output
javac DoubleTest1.java
java DoubleTest1 > sun.output
diff sun.output mars.output
0a1,2
> MARS 3.7 Copyright 2003-2009 Pete Sanderson and Kenneth Vollmar
>
1a4
>
```

The MiniJava-double compiler should be able to handle the following:

- double constants of the form digit* dot digit+
- declarations of a double scalar variable
- usage of a double in addition, subtraction, multiplication, and less than expressions
- System.out.println() should accept an int or a double

For type rules, assume that doubles can only be in arithmetic and comparison operations with other doubles. Feel free to share such test cases with other groups in the class, but each group must submit their own test cases.

Extra Credit opportunities:

- extend the interpreter called with the command-line option `-two-pass-interpret` so that it can interpret double expressions and print them out with System.out.println()
- assignment to double variables
- usage of double variables in an expression
- general parameter passing of doubles
- doubles as return values from methods
- arrays of doubles

3 Getting Started

The course mailing list was sent a starting MiniJava compiler called MJFull.tgz and a file called double-mars.s, which contains example double MIPS instructions. For further MIPS information,

you should use the help provided in the MARS simulator help (<http://courses.missouristate.edu/KenVollmar/MARS>) and Appendix A from Hennessy and Patterson, for which a link can be found at <http://pages.cs.wisc.edu/~larus/spim.html>.

1. Untar the the MJFull.tgz file. It will expand into the subdirectory MJFull/.

```
tar xzf MJFull.tgz
```

2. To create the file MJ.jar, do the following:

```
cd MJFull
make
```

3. To work with the MJ compiler within Eclipse, create a project as instructed in PA0. The main driver is in MJFull/src/MJDriver.java. To enable running MJDriver.java as a Java application you will need to generate the lexer and parser source files and add the JavaCUP runtime library to your class path. The following command will create the lexer and parser source files.

```
cd MJFull
./MakeSource.sh
```

To add the JavaCUP runtime library to your class path, do the following in Eclipse:

```
Project --> Properties --> Java Build Path --> Add JARs
select java-cup-11a-runtime.jar
```

Run the MJDriver.java with the following command line in the run configuration:

```
--two-pass-mips TestCases/BoolExp.java
```

4. Look at the output from various phases of the compiler. The output is in the ../TestCases/ subdirectory and in files ending with a .dot and .s. See project 0 for how to visualize and use these files.

Step through the compiler by starting in MJDriver.java and using the Eclipse debugger. You are responsible for understanding how the provided MiniJava compiler works and you will be using it in all the projects in this course. Learning someone else's code is not easy, but it is a task that you will find yourself doing quite often in industry and/or academia. I am open to any ideas you might have about modifying the compiler. You can modify your own copy as much as you want as long as the intermediate outputs remain the same.

Modifications needed to lex, parse, and generate the AST:

- Modify src/mjparser/mj.lex to include new tokens like double literals and the double keyword. Make sure to declare those tokens in src/mjparser/mj_ast.cup.

- Modify `src/mjparser/mj_ast.cup` to include grammar rules to parse declarations of double variables and parameters.
- Add AST nodes for double literals and the double type. Use the `IntegerExp` and `IntType` nodes as examples.
- Modify `src/mjparser/mj_ast.cup` to construct your new AST nodes for double literals and the double type.
- Modify all of the visitor classes in `src/ast/analysis` so that they visit the new node types.

At this point you can test that input MiniJava files with doubles in it are parsed into an AST correctly. **FIXME:** If you are using Eclipse to do your development, do a "make source" on the command line to generate the new lexer and parser anytime you modify the `mj.lex` and/or `mj_ast.cup` files. In Eclipse you will need to do a refresh for the project. You should get the following type error for the example `DoubleTest1.java`:

```
[11,26] Invalid argument type for method println
```

You can check that the AST was constructed by looking at the files `DoubleTest1.java.ast.dot` and `DoubleTest1.java.astlines.dot`.

Use `dot` to visualize the symbol table currently being generated into the file `DoubleTest1.java.ST.dot`. If your example has a declaration of a double variable, that variable and its type won't show up in the symbol table unless you fix how the symbol table is being constructed. Modifications needed for building the symbol table and performing semantic type analysis:

- `symtable/Type.java` needs a double type.
- `ast_visitors/BuildSymTable.java`, modify `getType()` and `outVarDecl()` so that they handle the double type.
- `ast_visitors/CheckTypes.java`, doubles should only be in arithmetic and comparison operations with other floats and/or as an input to `println()`.

Test the modifications you made to the creation of the symbol table and the type checking.

Translating the AST to MIPS (`mipsGenVisitor.java`):

- Make a version of all the code generation being done for integers that will work for doubles. Consult the `double-mars.s` file for the MIPS doubles syntax.
- Put a copy of `_printdouble` from `double-mars.s` into the `mips.rtl.s` file in the `src/` subdirectory.
- When implementing the less than operator for doubles, note how the `Label` data structure is used in other places in `mipsGenVisitor.java`.

I recommend a test-driven approach. Write a test for each feature the extended compiler must handle and then make each test work. Also, make sure you understand how integers and integer operations are implemented in the current MiniJava compiler. Doubles will be implemented very much like integers, except that doubles are 8 bytes instead of 4 bytes and the MIPS instructions that manipulate doubles are different than those that manipulate integers.

MJFull includes a regression testing suite that can be run with the following commands:

```
cd MJFull
make clean
make
cp MJ.jar Testcases
cd TestCases/
./regress.sh
```

You can add your own tests to the regression suite just by putting the test .java files into the TestCases subdirectory.

4 Your Report

The report is an essential part of your completed assignment. Writing is one of the most important things you will do as a graduate student. If you can't communicate what you have done, then what's the point? Use your report to describe the phases of the MiniJava compilers and how you implemented the double data type. Organize and present your document as if it were the only basis for your assignment's grade. The format of your writeup is up to you, but it should minimally include the following:

- Introduce the main goals of the project and in a couple of sentences summarize what you have accomplished.
- Briefly describe how you augmented each phase of the MiniJava compiler. Indicate any new tokens and grammar rules added to the MiniJava language. What new semantic type checking rules were needed? What new MIPS instructions are generated?
- How did you test your MiniJava extensions?
- What problems did you encounter while implementing the double data type. If you knew someone who was just about to start work on this assignment, what advice would you give them?
- What, if any, outside sources did you use (e.g., articles, books, other students)? This is particularly important. It's OK to look at books and articles and speak with your professor and fellow students (although sharing code and working with another group is strictly forbidden), but as with any scientific document, you should always cite your references and collaborations. You can either cite collaborations in footnotes or in a separate Acknowledgment section.

There's no exact number of pages you should write, but if you have between two and three then you are in the right ballpark.

5 Hints for doing the assignment

You should start this assignment right now!! You only have 14 days to complete this assignment. This assignment is not particularly difficult, because I step you through the modifications HOWEVER debugging the lexer and parser changes can take a LOT of time. I recommend spending a couple hours a day on the project starting now. Start writing your report as you are planning the implementation. A well-written report with some missing implementation guarantees a much higher grade than a poorly written report with all the implementation.

I can look at your code and help point you in the right direction, but the amount of help I can give may be inversely proportional to the amount of time until the due date. By no means should you spend several hours trying to figure out a weird bug; consult me for help. When e-mailing me about the project, send a copy of the relevant section of your code (not as an attachment; send it as text pasted into your message). Give a good description of the problem including information about the stack in a debugger. Also, indicate possible solutions you are considering.

6 What to turn in

Turn in a hard copy of the report and email a copy in pdf format to mstrout@cs.colostate.edu.

Create a jar file that includes all of the bytecode files and the source files. Submit a README file, the jar file, and two test cases via email to mstrout@cs.colostate.edu. Note that the provided Makefile in MJFull does not include the source files in the generated jar file, so you will have to make some minor modifications to the Makefile. The README should include specific command-lines for running the jar file on your provided test cases.

7 Due date

This assignment is due Tuesday September 8th, **at 3pm** and is worth 10% of your final grade. Late assignments will be penalized 10% per day.