Jim Lambers
CS143 Compilers
Summer Quarter 2000-01
Programming Project 4

This assignment must be submitted electronically by noon PDT on Sunday, August 19.

1 Overview

In this project, you will create an intermediate code generator for Mocha, capable of translating a valid Mocha program into an equivalent program in a given intermediate language.

2 Functional Specification

Your task is to translate a semantically valid Mocha program into the intermediate langauge that is described in the Mocha Specification. This language consists of instructions similar to those used by the Java virtual machine, relying on an operand stack for retrieving and storing values.

The output of your program will be the same as for Project 3, with the following additions:

- With each method, after its declarations are output, a list of the intermediate code that implements the method will be displayed.
- After the class declaration is output, a listing of intermediate statements that perform the initialization of the class' static fields is to be displayed.

3 Implementation

All of your work will be included in the starter file `parser.y`. In this file, you must supply the code for the semantic actions associated with most productions, using the included comments and the Mocha Specification as a guide.

You will be working with the same classes as in Project 3, along with some new additions: the `TacStmt` class, its derived classes, and other helper classes. These classes can be found in the files `tac.cc` and `tac.h`. In

addition, various convenience functions are added to `parser.y`, and some of the classes from Project 3.

Many of the operations performed by Mocha correspond directly to intermediate statements, such as arithmetic operations or variable access. However, some statements and expressions are implemented using flow control statements exclusively. These constructs are: boolean AND and OR operations, `for` statements, `if` statements, `while` statements, `do` statements, and `break` statements. For these statements, you must output a sequence of branching instructions and branch targets that will ensure the proper flow control during execution. In each case, the translation scheme is described for you in the comments in `parser.y`. You will make frequent use of backpatching, as illlustrated in lecture. Methods to create, merge, and backpatch lists of statements are provided for you in the `PatchList` class, described in `tac.cc` and `tac.h`.

You should look through the starter files for any comments labeled "PP4:". These comments either describe code you must supply, or code that you should examine since you may be using it to perform the required tasks.

As for producing the output specified above, the actual display is very simple; we have provided `Print` methods for every kind of declaration. The task for you is to ensure that all of this information is correctly gathered and stored so that the output methods will perform as desired.

You are not required to check for errors of any kind in this project. You may assume that all input programs are semantically correct.

4 Notes and Suggestions

In many cases, you will be reaching into the parsing stack for attributes, and you will need to implement mid-rule actions in many cases as well. Be sure you understand how these work before you proceed.

You should proceed by translating the simplest constructs first, which are the expressions. Then, once you are certain that you are implementing those correctly, you can move on to the more complex ones, such as field access or storage, all the way to the most complex ones, the flow control statements.

5 Testing

Once you are ready to test your intermediate code generator, simply type the command `make` at the prompt (assuming your project files are in your current working directory) to compile your code. If there are no compiler or linker errors, the executable, named `pp4`, will be written to the current directory.

We have provided several sample inputs to help you test your scanner. These may be found in

```
/usr/class/cs143/assignments/pp4/samples
```

With each sample input, there is a corresponding outputs file, with a `.out` extension, containing the output of the re-created declarations of the input program, (just like in Project 3), along with the intermediate code for each method, along with a separate code listing used for initializing static fields. This is the actual output generated by our solution, so you should ensure that your output matches ours. We strongly recommend that you create other input files for testing, as we will test your code on a larger collection of inputs.

To run the intermediate code generator on any of the samples, or any input files that you may create, simply use this command from the directory containing the executable `pp4`:

`(./pp4 <` *filename* `>&` *outputfile*

where *filename* is the name of the input file you are using, *outputfile* is the name of an output file of your choosing. When testing against the sample input files, you should then use the `diff` command to compare your output against the sample output files. Please make sure that you format your output in the same was as the solution, in order to facilitate grading. To that end, the functions used to display output have already been provided in the starter files.

6 Submission

This assignment must be submitted electronically by the given due date. You can find detailed information about the electronic submission process

and our policy on late assignments on the course web site.

7 Grading

This assignment is worth 100 points, and counts for 15% of your overall grade in this course. We will grade this assignment by running your program against all of our input files, and comparing the output against the output generated by our solution using `diff`, and points will be deducted based on the differences that are detected.

NOTE: programs that do not compile, or cause segmentation faults when run, will only receive limited partial credit. We will not try to fix your code in these situations. Please do your best to make sure that your code works before you submit it.

8 Portability

While you have a choice as to where you may work on this assignment, you do not have a choice as to where it will be graded. Your program will be run on the AIR Solaris workstations, and must be submitted from there. Before you submit, please test on these workstations to ensure that your code will work just as well for us, where and when we grade it, as it did for you, wherever you wrote it.