MC97 Lab 5: Run-time Environment

Due April 29, 1996; Max Hailperin

You only have to turn in the final compiler you have at the end of this lab. However, I strongly urge you to stop and test your compiler at several intermediate points.

1. Address any grading remarks from the previous lab that are still relevant.

2. Change the overall syntax of a program to be a list of items, each of which can be either a procedure declaration or a procedure definition. A procedure declaration consists of the keyword `extern`, a result type, the name of the procedure, a parenthesized, comma-separated list of zero or more argument types, and then a semicolon. As in C, each argument type in a procedure declaration may be optionally followed by an identifier, which serves merely as documentation. Also as in C, the result type may be `void` to indicate that no value is returned. Unlike in C (but like in C++), we will use an empty list of argument types to mean that no arguments are passed. A procedure definition is a result type name, the procedure name, a parenthesized, comma-separated list of parameter specifiers, a left brace, zero or more variable declarations, zero or more statements, and then a right brace. Each parameter specifier is a a type followed by an identifier. As in C, the scope of the parameters is the procedure body, not an extra scope wrapped around the procedure body for the purpose.

   There may only be one definition per procedure name, but there may be any number of declarations, so long as they all agree with each other and the definition regarding the number and types of arguments and results. Your generated code should follow the conventions described in the earlier handout.

   Remember that each procedure gets to use the registers for its own purposes. (The conventions regarding caller-saves and callee-saves ensure that no conflicts result.) Therefore, when your compiler is done compiling a particular procedure, it should deallocate all the `$s` registers so that they can be reallocated to different variables in the next procedure that is compiled. (The `$t` registers should naturally have all been deallocated by the time the procedure ended.) [A program that consists only of the procedure `main` should work at this point in the lab.]

3. Add procedure calls to the syntax of expressions and statements, and add a return statement (which should have an optional expression, as in C). Provide appropriate compile-time checks for type errors and the like, including calling a procedure that has not previously been declared or defined. [At this point, you should be able to have your main procedure call other procedures.]

4. Remove the `print` statement type. In its place, you'll now just use procedure calling. For example, instead of `print x;` you can now do `print_int(x);`, provided that a suitable declaration of `print_int` appears earlier in the program and a suitable definition of the `print_int` procedure is separately loaded into SPIM. In particular, if you put the following code into a file called `print_int.s`, then you can load it as well as your compiler's output into SPIM (by using the load command twice) and your programs should be able to call `print_int`, which was written directly in assembly language, just the same way as calling a procedure written in your language and compiled with your compiler:

```
        .globl  print_int
print_int:
        li      $v0, 1
        syscall
        j       $ra
```

5. Add global variables; the overall program can now be any mixture of procedure declarations, procedure definitions, variable declarations, and variable definitions. The earlier run time environment handout explains the difference between a global variable declaration and a global variable definition, and specifies the appropriate assembly code to generate for each. That handout also specifies the appropriate assembly code for uses of and assignments to global variables.

6. Add any optimizations or other improvements that seem worthwhile. If you have time but lack inspiration, see me for suggestions.