Wright State University
Department of Computer Science and Engineering

CS 780 Fall 2010 Prasad

## Assignment 1 (Due: Sept 22) (10 pts)

This assignment asks you to write a short Cool program. The purpose is to acquaint you with the Cool language and to give you experience with some of the tools used in the course.

In this assignment, you are to implement a very simple type inference engine for postfix expressions in Cool. The input is a sequence of postfix expressions built using integer variables (i and j), real variables (a and b) and overloaded binary operators (+ and *). The output is the sequence of inferred types. To accomplish this task, you can use a stack as follows.

A machine with only a single stack for storage is a *stack machine*. Consider the following very primitive language for programming a stack machine and the corresponding action for inferring types:

| *Command* | *Meaning* |
|---|---|
| i | push *int* on the stack |
| j | push *int* on the stack |
| a | push *real* on the stack |
| b | push *real* on the stack |
| $*, +$ | infer the type of the associated postfix expression using the top two stack entries, and pushing the result type back on stack |
| ! | print contents of the stack top |
| q | quit |

Here are some additional details. The 'i' and 'j' commands simply push the string "int" on top of the stack. The 'a' and 'b' commands simply push the string "real" on top of the stack. The behavior of the '+' (resp. '*') command is to pop the top two stack entries corresponding to the two sub-expresssions, and to push the type of the resulting postfix expression on the stack. Specifically, the type of $e_1$ *op* $e_2$ is *int* if both the sub-expressions $e_1$ and $e_2$ are of type *int*, otherwise, it is *real*. The '!' command prints the contents of the stack top.

The following examples show the effect of the various commands; the input is read from left to right and the top of the stack is on the left (that is, stack values are the mirror image of the corresponding input expression types):

| *input* | *stack* |
|---|---|
| $j\ i\ *\ \ldots$ | int ... |
| $a\ i\ j\ +\ \ldots$ | int real ... |
| $a\ b\ *\ i\ +\ \ldots$ | real ... |
| $i\ b\ !\ \ldots$ | real int |

Furthermore, the '!' command has the side-effect of printing the type of $b$: *real*.

You are to implement this translator in Cool. Input to the program is a series of commands, one command per line. Your interpreter should prompt for commands with `>>`. To simplify coding, your program need not do any sophisticated error checking. However, it is always a good idea to write robust programs, to facilitate debugging.

You are free to implement this program in any style you choose. However, in preparation for building a Cool compiler, we recommend that you try to develop an object-oriented solution. One approach is to define a class `Stack` of strings with suitable operations.

We wrote a robust solution using about 120 lines of formatted Cool source code. This information is provided to you as a rough measure of the amount of work involved in the assignment—your solution may be either shorter or substantially longer.

**Sample session.** The following is a sample compile and run of our solution on gandalf.cs.wright.edu.

```
gandalf % coolc type.cl
gandalf % spim -file type.s
SPIM Version 6.3a of January 14, 2001
Copyright 1990-2000 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
spim: (parser) immediate value (-4) out of range (0 .. 65535) on line 806 of file /usr/local/lib/cool/li
          andi  $gp $gp 0xfffffffc              # word align $gp
                                                   ^
spim: (parser) immediate value (-4) out of range (0 .. 65535) on line 870 of file /usr/local/lib/cool/li
          andi  $a0 $a0 0xFFFFFFFC
          ^
spim: (parser) immediate value (-4) out of range (0 .. 65535) on line 990 of file /usr/local/lib/cool/li
          andi  $gp $gp 0xfffffffc
                                ^
...                                  ^
spim: (parser) immediate value (-4) out of range (0 .. 65535) on line 1680 of file /usr/local/lib/cool/l
          andi  $t1 $t1 0xFFFFFFFC
                                ^
Loaded: /usr/local/lib/cool/lib/trap.handler
Warning: local symbol main was not defined
>>a
>>i
>>j
>>*
>>!
int
>>+
>>!
real
>>q
COOL program successfully executed
```

The executables, coolc and spim, are compiled for the SPARC-Solaris and installed on gandalf.cs.wright.edu in the directory /usr/local/lib/cool/bin/. As it stands, due to some incompatibilities between the expectations of standard spim and the changes to spim to run Cool, there are "benign" error messages complaining about "`...immediate value (-4) out of range (0 .. 65535)...`", and about "**main being undefined**". *Ignore them for this assignment.*

**Important:** You should make sure to place /usr/local/lib/cool/bin at the beginning in your path variable setting to use the executables meant for this assignment. To change the path variable, add the line

```
setenv PATH /usr/local/lib/cool/bin:${PATH}
```

at the end of your .login file (or .tcshrc file) if you use tcsh; add the line

```
path=/usr/local/lib/cool/bin:$path
export path
```

at the end of your .profile file (or .bashrc file) if you use bash.

The assembly code for a working version of type.cl is in the file type_eg.s. Login to **gandalf** and run as shown in the example above. You can compare the results from your program with the results of type_eg.s to test whether your program is operating correctly. You can run on a test input file as follows:

```
spim -file type_eg.s < type.test
```

**Getting and turning in the assignment.**

Create a working directory and `cd` into it. From there, at the prompt `gandalf%` , type

```
gandalf% make -f /usr/local/lib/cool/assignments/PA1/Makefile
```

This command creates several files you will need in the directory. Follow the directions in the README file, which explains how to turn in your assignment when you are finished; it also contains a few questions you are to answer as part of the assignment. Upon turning in your program, you will receive an automatic email acknowledgement providing a long listing of copied files.