UNIVERSITY OF THE WEST INDIES
Department of Computing
COMP3652–Language Processors
Sem I, 2016
Lecturer(s): Prof. Daniel Coore

Project: Checkoff by Friday, Dec. 23, 2016

Instructions: This is the first (and only) group assignment. It requires a substantial amount of work, and therefore needs the full participation of each group member. Please note that there are **two** problems on this question paper.

Collaboration between groups is permitted, but each group's solution must be distinctly its own. Any hint of duplication across groups will be dealt with severely.

Problem 1: SMPL *Interpreter* **[90]** Write an SMPL interpreter in Java. The following language features are optional and attract extra credit if implemented correctly.

- Character literals. (Both the simple character literals using the `#c` form as well as the unicode form.) **[5]**
- Tail recursion. You must support recursive procedure calls, but it is optional to prevent tail calls from using extra stack space. **[15]**
- Variable arity procedures. (You must at least support procedures that take a fixed number of arguments.) **[5]**
- Lazy evaluation **[5]**
- Multiple valued expressions and assignment. **[5]**
- Dynamic scoping. **[10]**
- Redefinable built-in procedures. **[10]**

Other optional features mentioned in the language specification will also attract extra credit if implemented. (Do not go overboard, there is a limit of 150% to extra credit).

Use JavaCUP and JLex to help you specify the grammar and the tokens of SMPL. You will need to define your own classes to support the specification of the semantics. (Use the files from the previous assignments as a starting point.) You should also use the visitor design pattern to define an appropriate visitor interface for traversing the abstract syntax tree that arises from parsing.

Note that one significant component of the interpreter that you will need to implement yourself is the representation of values in SMPL. You can use the implementation of FnPlot as a guide for how to do that effectively when you need to support several primitive types, especially when it comes to operating on mixed types.

Problem 2: *Programming in* SMPL **[10]**
Implement the quick sort algorithm in SMPL. Your SMPL quick sort should accept a vector and a comparator (i.e. a function that takes two objects and returns true if the first precedes the second) and return a vector of the same elements in increasing order of precedence, having sorted them using the quick sort algorithm.