Programming Languages
Qualifying Exam

Fall 2011
New Mexico State University

May 3, 2011

NOTE: this exam is open book and open notes.

Question 1 [.40 Points]

Consider the following program:

$$
\begin{array}{l}
\{x \geq 2 \wedge y \geq 2\} \\
\quad fl = 1; \\
\quad \textbf{for } (i = 2; i \leq x \wedge i \leq y; i++) \\
\quad\quad \textbf{if } (x \ mod \ i == 0 \wedge y \ mod \ i == 0) \\
\quad\quad\quad fl = 0; \\
\quad\quad \textbf{endif} \\
\quad \textbf{endfor} \\
\{fl == 0 \Rightarrow \neg p \ \wedge \\
\quad fl == 1 \Rightarrow p\}
\end{array}
$$

1. Formulate the post-condition p as a condition on the variables x, y ([10 Points]);
2. Convert this program into a while-program (as in the syntax used by Gumb's book); provide a loop invariant for the resulting while-loop ([10 Points]);
3. Prove partial correctness using Hoare's method ([10 Points]);

4. The idea of Floydian expression can be generalized to the case of while loops—as expressions associated to a loop invariant, that should satisfy the same two conditions as in the case of Floydian Expressions. Develop a Floydian expression for the example program and prove that it satisfies the two required conditions to prove termination ([10 Points]).

Question 2 [60 Points]

Let us consider the following syntax for an imperative language

```
<program>     ::= <statement>
<statement>   ::= <statement>  ; <statement>
                |    <identifier> = <expression>
                |    if <expression> then <statement>
<expression>   ::= <number>
                |     nil
                |     [ <expression> | <expression> ]
                |    (<unaryop> <expression>)
                |    (<binaryop> <expression> <expression>)
                |    (map <unaryop> <expression>)
<unaryop>      ::= head
                |    tail
                |    square
                |    double
<binaryop>      ::= add
                |     times
```

This language manipulates numbers and lists. A list can be either empty (denoted by *nil*) or not empty (denoted by $[exp_1|exp_2]$, where exp_1 is the head of the list and exp_2 is the tail of the list). For example, $[1|[2|nil]]$ denotes the two-element list containing 1 followed by 2.

The expressions $(\langle unaryop \rangle \quad exp)$ denote the application of the unary function $\langle unaryop \rangle$ to the argument *exp*. In a similar manner, the expression $\langle binaryop \rangle \ exp_1 \ exp_2)$ denotes the application of the binary function $\langle binaryop \rangle$ to the two arguments exp_1, exp_2.

The final case of expression, $(map \, \langle unaryop \rangle \, exp)$ is an iterative construct that repeats the application of the function $\langle unaryop \rangle$ to each element of the list represented by *exp*. For example, $(map \ double \ [1|[2|[3|nil]]])$ applies the function *double* to each element of the list $[1|[2|[3|nil]]]$, producing the list $[2|[4|[6|nil]]]$.

1. Provide the denotational semantics of the language. You can avoid worrying about type checking ([30 Points]).

2. Discuss how the language features could be implemented; in particular, describe the memory representation of the lists and how the *map* construct could be implemented taking advantage of concurrency ([15 Points]).

3. Describe how the *map* construct could be translated using traditional iterative (e.g., while) and conditional (e.g., if) constructs ([15 Points]).