CIS 500 — Software Foundations
Midterm II

April 1, 2009

Name: ___________________________________________________________

Email: ___________________________________________________________

<table>
<thead>
<tr>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>


Instructions

• This is a closed-book exam: you may not use any books or notes.

• You have 80 minutes to answer all of the questions.

• The exam is worth 80 points. However, questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.

• Partial credit will be given. All correct answers are short. The back side of each page may be used as a scratch pad.

• Good luck!
1. (5 points) Recall the definition of equivalence for while programs:

\[
\text{Definition cequiv (c1 c2 : com) : Prop :=}
\forall (st st' : state), (c1 / st \rightarrow st') \leftrightarrow (c2 / st \rightarrow st').
\]

Which of the following pairs of programs are equivalent? Write “yes” or “no” for each one. (Where it appears, \(a\) is an arbitrary \texttt{aexp} — i.e., you should write “yes” only if the two programs are equivalent for every \(a\).)

(a) \(X ::= A4\)
   \[\text{and} \]
   \[Y ::= A2 ++ A2; \]
   \[X ::= Y\]

(b) \(X ::= a;\)
   \[Y ::= a\]
   \[\text{and} \]
   \[Y ::= a; \]
   \[X ::= a\]

(c) \(\text{while BTrue do (X := !X ++ 1)}\)
   \[\text{and} \]
   \[X ::= !X ++ 1\]

(d) \(\text{while BTrue do (X := !X ++ 1)}\)
   \[\text{and} \]
   \[\text{while BTrue do (X := !X -- 1)}\]

(e) \(\text{while BFalse do (X := !X ++ 1)}\)
   \[\text{and} \]
   \[\text{skip}\]

2
2. (5 points) Is this claim...

Claim: Suppose the command $c$ is equivalent to $c;c$. Then, for any $b$, the command
while $b$ do $c$

is equivalent to
testif $b$ then $c$ else skip.

... true or false? Briefly explain.

3. (5 points) Recall that a program transformation is a function from commands to commands. What does it mean to say that a program transformation is “sound”? (Answer either informally or with a Coq definition.)
4. (7 points) Recall the definition of a valid Hoare triple:

\[
\text{Definition hoare_triple (P:Assertion) (c:com) (Q:Assertion) : Prop :=}
\forall st st',
\begin{align*}
  c / st &\rightarrow st' \\
  \rightarrow P st \\
  \rightarrow Q st'.
\end{align*}
\]

Indicate whether or not each of the following Hoare triples is valid by writing either “valid” or “invalid.” Where it appears, \(a\) is an arbitrary \texttt{aexp}—i.e., you should write “valid” only if the triple is valid for every \(a\).

(a) \(\{\text{True}\}\) \(X ::= a\) \(\{X = a\}\)

(b) \(\{X = 1\}\)
\[
\begin{align*}
  &\text{testif (!X === a) then (while BTrue do Y ::= !X) else (Y ::= A0)} \\
  &\{Y = 0}\}
\end{align*}
\]

(c) \(\{\text{True}\}\)
\[
Y ::= A0; Y ::= A1 \\
\{Y = 1\}
\]

(d) \(\{\text{False}\}\)
\[
X ::= A3 \\
\{X = 0\}
\]

(e) \(\{\text{True}\}\)
\[
\text{skip} \\
\{\text{False}\}
\]

(f) \(\{X = 5 \land Y = X\}\)
\[
Z ::= 0; \text{while BNot (!X === A0) do (Z ::= !Z +++ !Y; X ::= !X --- 1)} \\
\{Z = 25\}
\]

(g) \(\{X = 1\}\)
\[
\text{while BNot (!X === A0) do X ::= !X +++ 1} \\
\{X = 42\}
\]
5. (9 points) Give the weakest precondition for each of the following commands. (Please use the informal notation for assertions rather than Coq notation—i.e., write \( X = 5 \), not \( \text{fun st \Rightarrow st X = 5} \).)

(a) \( \{ ? \} \ X := A5 \ \{ \ X = 5 \} \)

(b) \( \{ ? \} \ X := A0 \ \{ \ X = 5 \} \)

(c) \( \{ ? \} \ X := !X \+++!Y \ \{ \ X = 5 \} \)

(d) \( \{ ? \} \ \text{while } A1 \Leftarrow !X \ \text{do} \ (X := !X\---A1; \ Y := !Y\---A1) \ \{ \ Y = 5 \} \)

(e) \( \{ ? \} \ \text{while } !X \equiv A0 \ \text{do} \ Y := A1 \ \{ \ Y = 1 \} \)

(f) \( \{ ? \} \)
\[
\text{testif } !X \equiv A0 \\
\quad \text{then } Y := IZ \\
\quad \text{else } Y := IW \\
\{ \ Y = 5 \} \]
6. (5 points) The notion of weakest precondition has a natural dual: given a precondition and a command, we can ask what is the strongest postcondition of the command with respect to the precondition. Formally, we can define it like this:

Q is the strongest postcondition of c for P if:
(a) {{P}} c {{Q}}, and
(b) if Q′ is an assertion such that {{P}}c{{Q′}}, then Q st implies Q′ st, for all states st.
Q is the strongest (most difficult to satisfy) assertion that is guaranteed to hold after c if P holds before.

For example, the strongest postcondition of the command skip with respect to the precondition \( Y = 1 \) is \( Y = 1 \). Similarly, the postcondition in...

\[
\begin{align*}
&\{\{ Y = y \}\} \\
&\text{if } !Y \equiv A0 \text{ then } X ::= A0 \text{ else } Y ::= !Y ** A2 \\
&\{\{ (Y = y = X = 0) \lor (Y = 2 \times y \land y \neq 0) \}\}
\end{align*}
\]

...is the strongest one.

Complete each of the following Hoare triples with the strongest postcondition for the given command and precondition.

(a) \{\{ Y = 1 \}\} X ::= !Y +++ A1 \{\ ? \}\n
(b) \{\{ True \}\} X ::= A5 \{\ ? \}\n
(c) \{\{ True \}\} skip \{\ ? \}\n
(d) \{\{ True \}\} while BTrue do skip \{\ ? \}\n
(e) \{\{ X = x \land Y = y \}\} \\
while BNot (!X \equiv A0) do ( \\
\ Y ::= !Y +++ A2; \\
\ X ::= !X --- A1 \\
) \{\ ? \}\n
6
7. (12 points) The following program performs integer division:

```plaintext
div =
    Q ::= A0;
    R ::= ANum x;
    while (ANum y) <<= !R do (  
        R ::= !R --- (ANum y);
        Q ::= !Q +++ A1
    )
```

If x and y are numbers, running this program will yield a state where Q is the quotient of x by y and R is the remainder. (We assume that program variables Q and R are defined.)

Fill in the blanks in the following to obtain a correct decorated version of the program:

```plaintext
{ 0<y } =>
{ 0=0 ∧ x=x ∧ 0<y } =>
Q ::= A0;
{ Q=0 ∧ x=x ∧ 0<y } =>
R ::= ANum x;
{ Q=0 ∧ R=x ∧ 0<y } =>
{ __________________________________________ } =>
while (ANum y <<= !R) do (  
{ __________________________________________ } =>
{ __________________________________________ } =>
R ::= !R --- (ANum y);
{ __________________________________________ } =>
Q ::= !Q +++ A1
{ __________________________________________ } =>
{ __________________________________________ } =>
{ x=Q*y+R ∧ R<y } =>
```

8. (4 points) Suppose we change the initial pre-condition in problem 7 from 0<y to True (i.e., we allow y to be zero). Does the specification now make an incorrect claim — i.e., is the Hoare triple

```plaintext
{{ True }} div {{ x=Q*y+R ∧ R<y }}
```

valid, or is it valid? Briefly explain your answer.
9. (6 points) Recall the syntax...

\[
\text{Inductive \textit{com} : \textit{Set} :=} \\
\quad \ldots \\
\quad | \text{CWhile} : \textit{bexp} \rightarrow \textit{com} \rightarrow \textit{com}
\]

...and operational semantics of the while...do... construct:

\[
\text{Inductive \textit{ceval} : \textit{state} \rightarrow \textit{com} \rightarrow \textit{state} \rightarrow \text{Prop} :=} \\
\quad \ldots \\
\quad | \text{CEWhileEnd} : \forall \text{b1 st c1}, \text{beval st b1 = false} \rightarrow \text{ceval st (CWhile b1 c1) st} \\
\quad | \text{CEWhileLoop} : \forall \text{st st' st'' b1 c1}, \text{beval st b1 = true} \rightarrow \text{ceval st c1 st'} \rightarrow \text{ceval st' (CWhile b1 c1) st''} \rightarrow \text{ceval st (CWhile b1 c1) st''}
\]

Suppose we extend the syntax with one more constructor...

\[
| \text{CLoopWhile} : \textit{com} \rightarrow \textit{bexp} \rightarrow \textit{com}
\]

...written loop c while b:

\[
\text{Notation "'loop c 'while' b" := (CLoopWhile c b).}
\]

The intended behavior of this construct is almost like that of while...do... except that the condition is checked at the \textit{end} of the loop body instead of the beginning (so the body always executes at least once). For example,

\[
\begin{align*}
X & := A1; \\
\text{loop} \quad X & := !X +++ A1 \\
\text{while} \quad !X & \lll A1
\end{align*}
\]

will leave \textit{X} with the value 2.

To define the operational semantics of \textit{loop}...\textit{while}... formally, we need to add two more rules to the \textit{Inductive} declaration of \textit{ceval}. Write these rules in the space below.
10. (6 points) Having extended the language of commands with `loop...while...`, the next thing we want is a Hoare rule for reasoning about programs that use this construct. Recall the rule for `while...do...`:

\[
\frac{\{P \land b\} \quad c \quad \{P\}}{\{P\} \quad \text{while } b \quad \text{do } c \quad \{P \land \neg b\}}
\]

Write an analogous rule for `loop...while...`.
11. (4 points) Recall (from the review session on Monday) the small-step variant of the operational semantics of IMP. The \texttt{astep} and \texttt{bstep} relations (not shown here) are small-step reduction relations for aexps and bexps. The small-step relation for commands is defined as follows:

\[
\text{Inductive } \texttt{cstep} : \text{state} \rightarrow \text{com} \rightarrow \text{com} \rightarrow \text{state} \rightarrow \text{Prop} := \\
| \texttt{CSAssStep} : \forall \text{st} \; \text{i a a'}, \\
\quad \text{astep st a a'} \rightarrow \\
\quad \texttt{cstep st (CAss i a) (CAss i a')} \text{ st} \\
| \texttt{CSAss} : \forall \text{st} \; \text{i n}, \\
\quad \texttt{cstep st (CAss i (ANum n)) CSkip (extend st i n)} \\
| \texttt{CSSeqStep} : \forall \text{st} \; \text{c1 c1' st' c2}, \\
\quad \texttt{cstep st c1 c1' st' c2} \\
| \texttt{CSSeqFinish} : \forall \text{st} \; \text{c2}, \\
\quad \texttt{cstep c2 CSkip c2 c2 st} \\
| \texttt{CSIfTrue} : \forall \text{st} \; \text{cl c2}, \\
\quad \texttt{cstep st (CIf BTrue c1 c2) c1 st} \\
| \texttt{CSIfFalse} : \forall \text{st} \; \text{cl c2}, \\
\quad \texttt{cstep st (CIf BFalse c1 c2) c2 st} \\
| \texttt{CSIfStep} : \forall \text{st} \; \text{b b' cl c2}, \\
\quad \texttt{bstep st b b'} \\
\quad \texttt{cstep st (CIf b cl c2) (CIf b' c1 c2) st} \\
| \texttt{CSWhile} : \forall \text{st} \; \text{b cl}, \\
\quad \texttt{cstep st (CWhile b cl) (CIf b (CSeq c1 (CWhile b c1)) CSkip) st}.
\]

Suppose we extend the syntax of commands with \texttt{loop...while...}, as in the previous two problems. What needs to be added to the definition of \texttt{cstep}?
12. (12 points) Recall the following definitions from Smallstep.v:

\[
\text{Inductive tm : Set := }
\begin{align*}
| \text{tm_const : nat } \rightarrow \text{tm} \\
| \text{tm_plus : tm } \rightarrow \text{tm } \rightarrow \text{tm}.
\end{align*}
\]

\[
\text{Inductive value : tm } \rightarrow \text{Prop := }
\begin{align*}
| \text{v_const : forall n, value (tm_const n)}.
\end{align*}
\]

\[
\text{Inductive step : tm } \rightarrow \text{tm } \rightarrow \text{Prop := }
\begin{align*}
| \text{ES_PlusConstConst : forall n1 n2, } \\
& \quad \text{step (tm_plus (tm_const n1) (tm_const n2))} \\
& \quad \quad \text{(tm_const (plus n1 n2))}
\end{align*}
\]

\[
\begin{align*}
| \text{ES_Plus1 : forall t1 t1' t2, } \\
& \quad \text{(step t1 t1')} \\
& \quad \quad \text{step (tm_plus t1 t2)} \\
& \quad \quad \quad \text{(tm_plus t1' t2)}
\end{align*}
\]

\[
\begin{align*}
| \text{ES_Plus2 : forall v1 t2 t2', } \\
& \quad \text{(value v1)} \\
& \quad \quad \text{(step t2 t2')} \\
& \quad \quad \quad \text{step (tm_plus v1 t2)} \\
& \quad \quad \quad \quad \text{(tm_plus v1 t2')}.
\end{align*}
\]

In class, we discussed the Progress Theorem:

*Theorem:* If \( t \) is a term, then either \( t \) is a value or else there exists some term \( t' \) such that \( t \) steps to \( t' \).

Write a careful informal proof of this theorem.