CIS 500 — Software Foundations

Midterm II

March 28, 2012

Name: ______________________________

Pennkey: ______________________________

Scores:

1	
2	
3	
4	
5	
6	
Total (80 max)	

This exam concentrates on the material on the Imp programming language, program equivalence, and Hoare Logic. Some of the key definitions are repeated, for easy reference, in the accompanying handout.

The version of Imp we consider in this exam only has arithmetic expressions that reduce to numbers; you don't need to worry about lists.

1. (8 points) Indicate whether or not each of the following Hoare triples is valid by writing either "valid" or "invalid." Also, for those that are invalid, give a counter-example. The definition of Hoare triples is given on page 9, for reference.

(a)
```
{{ X = 2 \/ X = 3 }}
X ::= 5
{{ X = 0 }}
```

(b)
```
{{ X = 2 /\ X = 3 }}
X ::= 5
{{ X = 0 }}
```

(c)
```
{{ True }}
IFB X = 0 THEN Y ::= 1 ELSE Y ::= 2 FI
{{ Y > X }}
```

(d)
```
{{ True }}
X ::= 0;
WHILE X <> X DO
  X ::= X + 1
END
{{ False }}
```

2. (12 points) Given the following programs, group together those that are equivalent in Imp by drawing boxes around their names. For example, if you think programs a through h are all equivalent to each other, but not to i, your answer should look like this: $\boxed{a, b, c, d, e, f, g, h}$ $\boxed{i}$.

The definition of program equivalence is repeated on page 9, for reference.

(a)
```
WHILE X > 0 DO
  X ::= X + 1
END
```

(b)
```
IFB X = 0 THEN
  X ::= X + 1;
  Y ::= 1
ELSE
  Y ::= 0
FI;
X ::= X - Y;
Y ::= 0
```

(c)
```
SKIP
```

(d)
```
WHILE X <> 0 DO
  X ::= X * Y + 1
END
```

(e)
```
Y ::= 0
```

(f)
```
Y ::= X + 1;
WHILE X <> Y DO
  Y ::= X + 1
END
```

(g)
```
WHILE BTrue DO
  SKIP
END
```

(h)
```
WHILE X <> X DO
  X ::= X + 1
END
```

(i)
```
WHILE X <> Y DO
  X ::= Y + 1
END
```

3. (20 points) Write a careful informal proof showing that if boolean expression `b` is equivalent to `BTrue`, then the command `IFB b THEN c1 ELSE c2 FI` is equivalent to `c1`—i.e., formally:

```
forall b c1 c2, bequiv b BTrue  ->
  cequiv (IFB b THEN c1 ELSE c2 FI) c1.
```

The definitions of `bequiv` and `cequiv` are given on page 9, for reference.

4. (16 points) The following Imp program calculates the sum of three numbers `a`, `b`, and `c`.

```
X ::= 0;
Y ::= 0;
Z ::= c;
WHILE X <> a DO
  X ::= X + 1;
  Z ::= Z + 1
END;
WHILE Y <> b DO
  Y ::= Y + 1;
  Z ::= Z + 1
END
```

Note that we're using informal notations as usual in Imp examples, for example writing this

```
WHILE (X <> a)
```

instead of this:

```
WHILE (BNot (BEq (AId X) (ANum a)))
```

On the next page, add appropriate annotations to the program in the provided spaces to show that the Hoare triple given by the outermost pre- and post-conditions is valid. Use informal notations for mathematical formulae and assertions, but please be completely precise and pedantic in the way you apply the Hoare rules — i.e., write out assertions in *exactly* the form given by the rules (rather than logically equivalent ones). The provided blanks have been constructed so that, if you work backwards from the end of the program, you should only need to use the rule of consequence in the places indicated with `=>`.

The Hoare rules are provided on page 10, for reference.

```
  {{ True                                                                    }} =>
  {{                                                                         }}
X ::= 0;
  {{                                                                         }}
Y ::= 0;
  {{                                                                         }}
Z ::= c;
  {{                                                                         }}
WHILE X <> a DO
    {{                                                                       }} =>
    {{                                                                       }}
  X ::= X + 1;
    {{                                                                       }}
  Z ::= Z + 1
    {{                                                                       }}
END;
  {{                                                                         }} =>
  {{                                                                         }}
WHILE Y <> b DO
    {{                                                                       }} =>
    {{                                                                       }}
  Y ::= Y + 1;
    {{                                                                       }}
  Z ::= Z + 1
    {{                                                                       }}
END
  {{                                                                         }} =>
  {{ Z = a + b + c                                                           }}
```

5. (12 points) In this exercise we consider extending Imp with " one-sided conditionals" of the form

```
IF1 b THEN c FI
```

where `b` is a boolean expression, and `c` is a command. If `b` evaluates to `false`, then `IF1 b THEN c FI` does nothing.

To formalize the extended language, we first add a clause to the definition of commands:

```
Inductive com : Type :=
  ...
  | CIf1 : bexp -> com -> com.

Notation "'IF1' b 'THEN' c 'FI'" := (CIf1 b c) (at level 60).
```

(a) Refer to the definition of `ceval` (page 9) for the evaluation relation of Imp. What rule(s) must be added to this definition to formalize the behavior of `IF1`? Write out the additional rule(s) in formal Coq notation.

(b) Write a Hoare proof rule for one-sided conditionals. (For reference, the standard Hoare rules for Imp are provided on page 10.)

Try to come up with a rule that is both sound and as precise as possible. For full credit, make sure your rule can be used to prove the following valid Hoare triple:

```
{{ X + Y = Z }}
IF1 Y <> 0 THEN
  X ::= X + Y
FI
{{ X = Z }}
```

6. (12 points) In this exercise we define an asymmetric variant of program equivalence we call *program approximation.* We say that program `c1` *approximates* program `c2` when, for each of the initial states for which `c1` terminates, `c2` also terminates and produces the same final state. Formally, program approximation is defined as follows:

```
Definition capprox (c1 c2 : com) : Prop :=
  forall (st st' : state),
    (c1 / st || st') -> (c2 / st || st').
```

For instance the program `c1 = WHILE X <> 1 DO X ::= X - 1 END` approximates `c2 = X ::= 1`, but `c2` does not approximate `c1` since `c1` does not terminate when `X = 0` but `c1` does. If two programs approximate each other in both directions, then they are equivalent.

(a) Find two programs, `c3` and `c4`, such that neither approximates the other. Formally, the following two propositions should be provable: ∼`(capprox c3 c4)` and ∼`(capprox c4 c3)`. Your programs should be short (3 lines max).

```
c3 =
```

```
c4 =
```

(b) Find a program `cmin` that approximates every other program. Formally, the proposition `forall c', capprox cmin c'` should be provable. (Again, 3 lines max.)

```
cmin =
```

(c) Find a non-trivial property that is preserved by program approximation (when going from left to right). Formally, your `zprop` should satisfy the following condition:

```
forall c c', zprop c -> capprox c c' -> zprop c'
```

Write `zprop` using formal Coq notation. (The correct answer fits on one line.)

```
Definition zprop c : Prop :=
```

Formal definitions for Imp

Syntax

```
Inductive aexp : Type :=
  | ANum : nat -> aexp
  | AId : id -> aexp
  | APlus : aexp -> aexp -> aexp
  | AMinus : aexp -> aexp -> aexp
  | AMult : aexp -> aexp -> aexp.

Inductive bexp : Type :=
  | BTrue : bexp
  | BFalse : bexp
  | BEq : aexp -> aexp -> bexp
  | BLe : aexp -> aexp -> bexp
  | BNot : bexp -> bexp
  | BAnd : bexp -> bexp -> bexp.

Inductive com : Type :=
  | CSkip : com
  | CAss : id -> aexp -> com
  | CSeq : com -> com -> com
  | CIf : bexp -> com -> com -> com
  | CWhile : bexp -> com -> com.

Notation "'SKIP'" :=
  CSkip.
Notation "l '::=' a" :=
  (CAss l a) (at level 60).
Notation "c1 ; c2" :=
  (CSeq c1 c2) (at level 80, right associativity).
Notation "'WHILE' b 'DO' c 'END'" :=
  (CWhile b c) (at level 80, right associativity).
Notation "'IFB' e1 'THEN' e2 'ELSE' e3 'FI'" :=
  (CIf e1 e2 e3) (at level 80, right associativity).
```

Evaluation relation

```
Inductive ceval : com -> state -> state -> Prop :=
  | E_Skip : forall st,
      SKIP / st || st
  | E_Ass  : forall st a1 n X,
      aeval st a1 = n ->
      (X ::= a1) / st || (update st X n)
  | E_Seq : forall c1 c2 st st' st'',
      c1 / st  || st' ->
      c2 / st' || st'' ->
      (c1 ; c2) / st || st''
  | E_IfTrue : forall st st' b1 c1 c2,
      beval st b1 = true ->
      c1 / st || st' ->
      (IFB b1 THEN c1 ELSE c2 FI) / st || st'
  | E_IfFalse : forall st st' b1 c1 c2,
      beval st b1 = false ->
      c2 / st || st' ->
      (IFB b1 THEN c1 ELSE c2 FI) / st || st'
  | E_WhileEnd : forall b1 st c1,
      beval st b1 = false ->
      (WHILE b1 DO c1 END) / st || st
  | E_WhileLoop : forall st st' st'' b1 c1,
      beval st b1 = true ->
      c1 / st || st' ->
      (WHILE b1 DO c1 END) / st' || st'' ->
      (WHILE b1 DO c1 END) / st || st''

  where "c1 '/' st '||' st'" := (ceval c1 st st').
```

Program equivalence

```
Definition bequiv (b1 b2 : bexp) : Prop :=
  forall (st:state), beval st b1 = beval st b2.

Definition cequiv (c1 c2 : com) : Prop :=
  forall (st st' : state),
    (c1 / st || st') <-> (c2 / st || st').
```

Hoare triples

```
Definition hoare_triple (P:Assertion) (c:com) (Q:Assertion) : Prop :=
  forall st st', c / st || st' -> P st  -> Q st'.

Notation "{{ P }}  c  {{ Q }}" := (hoare_triple P c Q)
```

```
                                       (at level 90, c at next level)
                                       : hoare_spec_scope.
```

Implication on assertions

```
Definition assert_implies (P Q : Assertion) : Prop :=
  forall st, P st -> Q st.

Notation "P ~~> Q" := (assert_implies P Q) (at level 80).
```

(ASCII `~~>` is typeset as $\leadsto$ in the rules below.)

Hoare logic rules

$$\frac{}{\{\{\,\texttt{assn\_sub X a}\ Q\,\}\}\ \texttt{X := a}\ \{\{\,Q\,\}\}}\ (\texttt{hoare\_asgn})$$

$$\frac{}{\{\{\,P\,\}\}\ \texttt{SKIP}\ \{\{\,P\,\}\}}\ (\texttt{hoare\_skip})$$

$$\frac{\begin{array}{c}\{\{\,P\,\}\}\ \texttt{c1}\ \{\{\,Q\,\}\}\\ \{\{\,Q\,\}\}\ \texttt{c2}\ \{\{\,R\,\}\}\end{array}}{\{\{\,P\,\}\}\ \texttt{c1; c2}\ \{\{\,R\,\}\}}\ (\texttt{hoare\_seq})$$

$$\frac{\begin{array}{c}\{\{\,P \wedge b\,\}\}\ \texttt{c1}\ \{\{\,Q\,\}\}\\ \{\{\,P \wedge \sim b\,\}\}\ \texttt{c2}\ \{\{\,Q\,\}\}\end{array}}{\{\{\,P\,\}\}\ \texttt{IFB b THEN c1 ELSE c2 FI}\ \{\{\,Q\,\}\}}\ (\texttt{hoare\_if})$$

$$\frac{\{\{\,P \wedge b\,\}\}\ \texttt{c}\ \{\{\,P\,\}\}}{\{\{\,P\,\}\}\ \texttt{WHILE b DO c END}\ \{\{\,P \wedge \sim b\,\}\}}\ (\texttt{hoare\_while})$$

$$\frac{\begin{array}{c}\{\{\,P'\,\}\}\ \texttt{c}\ \{\{\,Q'\,\}\}\\ P \leadsto P'\\ Q' \leadsto Q\end{array}}{\{\{\,P\,\}\}\ \texttt{c}\ \{\{\,Q\,\}\}}\ (\texttt{hoare\_consequence})$$

$$\frac{\begin{array}{c}\{\{\,P'\,\}\}\ \texttt{c}\ \{\{\,Q\,\}\}\\ P \leadsto P'\end{array}}{\{\{\,P\,\}\}\ \texttt{c}\ \{\{\,Q\,\}\}}\ (\texttt{hoare\_consequence\_pre})$$

$$\frac{\begin{array}{c}\{\{\,P\,\}\}\ \texttt{c}\ \{\{\,Q'\,\}\}\\ Q' \leadsto Q\end{array}}{\{\{\,P\,\}\}\ \texttt{c}\ \{\{\,Q\,\}\}}\ (\texttt{hoare\_consequence\_post})$$