UNIVERSITY OF WESTMINSTER

MARKING SCHEME
DO NOT COPY

FACULTY OF
SCIENCE & TECHNOLOGY
Department of Computer Science

Module:	**Reasoning about Programs**
Module Code:	**6SENG001W, 6SENG003C**
Module Leader:	Klaus Draeger
Date:	17th January 2018
Start:	10:00
Time allowed:	2 Hours

Instructions for Candidates:

You are advised (but not required) to spend the first ten minutes of the examination reading the questions and planning how you will answer those you have selected.

Answer ALL questions in Section A and TWO questions from Section B.

Section A is worth a total of 50 marks.

Each question in section B is worth 25 marks.

DO NOT TURN OVER THIS PAGE
UNTIL THE INVIGILATOR INSTRUCTS YOU TO DO SO.

Section A

Answer ALL questions from this section.

Question 1

(a) $SeaBirds \cap ZooBirds \ = \ \{ \ Penguin \ \}$ **[1 mark]**

(b) $SeaBirds - \{ \ Penguin, \ \ Parrot \ \} \ \ = \ \ \{ \ Seagull, Albatross \ \}$
[2 marks]

(c) $\mathrm{card}(\ maxPerZooCage \ ) \ = \ 4$ **[1 mark]**

(d) $SeaBirds \cap \mathrm{dom}(maxPerZooCage) \ = \ \{ \ Penguin \ \}$ **[2 marks]**

(e) $\mathrm{ran}(maxPerZooCage) \ = \ \{ \ 2, \ 4, \ 50 \ \}$ **[1 mark]**

(f) $maxPerZooCage(Ostrich) \ = \ 4$ **[1 mark]**

(g) $SeaBirds \lhd maxPerZooCage \ = \ \{ \ Penguin \mapsto 50 \ \}$ **[2 marks]**

(h) $maxPerZooCage \unrhd \{ \ Emu, \ Penguin \ \}$
$= \ \{ \ Parrot \mapsto 2, \ Ostrich \mapsto 4 \ \}$ **[2 marks]**

(i)

$$
\begin{aligned}
& \mathbb{P} \ \{ \ Seagull, \ Penguin, \ Albatross \ \} \\
= \ & \{ \ \{\}, \ \{Seagull\}, \ \{Penguin\}, \ \{Albatross\}, \\
& \quad \{Penguin, Seagull\}, \ \{Penguin, Albatross\}, \ \{Albatross, Seagull\}, \\
& \quad \{Seagull, Penguin, Albatross\} \ \}
\end{aligned}
$$

[3 marks]

[QUESTION Total 15]

Question 2

Evaluate the following expressions:

(a) $\mathrm{dom}(R_1) \;=\; \{\ a, b, c, d, e, f, g, h\ \}$ $[\mathbf{1\ mark}]$

(b) $\mathrm{ran}(R_2) \;=\; \{\ 1, 2, 3\ \}$ $[\mathbf{1\ mark}]$

(c) $\{\ a, b, g\ \} \lhd R_1 \;=\; \{\ a \mapsto 1,\ b \mapsto 1,\ b \mapsto 2,\ g \mapsto 5\ \}$ $[\mathbf{2\ marks}]$

(d) $R_1 \rhd \{\ 2, 4, 6\ \} \;=\; \{\ b \mapsto 2,\ d \mapsto 2,\ e \mapsto 4,\ f \mapsto 4,\ h \mapsto 6\ \}$ $[\mathbf{2\ marks}]$

(e) $R_2 \ntriangleright \{\ 1, 3\ \} \;=\; \{\ b \mapsto 2,\ d \mapsto 2\ \}$ $[\mathbf{2\ marks}]$

(f) $R_3 \mathbin{\lhd\!\!\!+} \{\ 0 \mapsto s, 4 \mapsto t\ \} \;=\; \{\ 0 \mapsto s, 1 \mapsto x,\ 2 \mapsto y,\ 4 \mapsto t\ \}$ $[\mathbf{3\ marks}]$

(g) $R_2 \,;\, R_3 \;=\; \{\ a \mapsto x,\ b \mapsto x,\ b \mapsto y,\ d \mapsto y\ \}$ $[\mathbf{4\ marks}]$

$[\mathbf{QUESTION\ Total\ 15}]$

Question 3

The *signatures* of the functions are the following, minor mistakes will result in marks being deducted.

(a) $inc \;\in\; \mathbb{N} \rightarrowtail \mathbb{N}$ [Total injective] $[\mathbf{2\ marks}]$

(b) $dec \;\in\; \mathbb{N} \mapsto\!\!\!\!\!\!\shortmid\;\; \mathbb{N}$ [Partial injective] $[\mathbf{2\ marks}]$

(c) $add \;\in\; \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$ [Total surjection] $[\mathbf{3\ marks}]$

(d) $sub \;\in\; \mathbb{N} \times \mathbb{N} \mathrel{+\!\!\!\twoheadrightarrow} \mathbb{N}$ [Partial surjection] $[\mathbf{3\ marks}]$

$[\mathbf{QUESTION\ Total\ 10}]$

Question 4

(a) An Abstract Machine is similar to the programming concepts of: modules, class definition (e.g. Java) or abstract data types. **[1 mark]**

An Abstract Machine is a specification of what a system should be like, or how it should behave (operations); but not how a system is to be built, i.e. no implementation details. **[2 marks]**

The main logical parts of an Abstract Machine are its: *name*, *local state*, represented by "encapsulated" variables, *collection of operations*, that can access & update the state variables. **[3 marks]**

[PART Total 6]

(b) Three categories of system states are: *valid* states, *initial* or *start* states & *error* or *invalid* states. **[1 mark]**

The *valid* states are those that satisfy the *state invariant*. The *invalid* states are those that do not satisfy the *state invariant*. The *state invariant* is the constraints & properties that the states of the machine must satisfy during its lifetime. Defined in the `INVARIANT` clause **[2 marks]**

The *initial state(s)* are the set of possible starting states of the machine. Any initial state must also be a valid state, i.e. one that satisfies the state invariant. Defined in the `INITIALISATION` clause. **[1 mark]**

[PART Total 4]

[QUESTION Total 10]

Section B

Answer TWO questions from this section.

Question 5

The plane's flight route is just a version of a Queue, with no duplicates. A B machine roughly similar to the following is expected.
Some possible acceptable alternatives:
Uses B symbols not ASCII versions, or a mixture.
Combines `ROUTE_STATUS` & `REPORT` or uses string literals.
Also likely that some less important parts are omitted, e.g. preconditions – "`report : REPORT`", use of `UnknownCity`.
Using an ordinary sequence `seq` rather than an injective sequence `iseq`, but this allows duplicates.
Probably outputting the destination city from the "pop", but not important.

(a)

```
MACHINE FlightRoutes

   SETS
     CITY = { Berlin, Dublin, Geneva, London, Madrid,
              New_York, Paris, Rome, Sydney, Washington,
              UnknownCity } ;

     ROUTE_STATUS = { Route_is_Empty,
                      Route_is_Full,
                      Route_Only_Has_Departure_City,
                      Route_Can_Be_Extended } ;

     REPORT = { City_Added_To_Route,
                ERROR_Route_is_Full,
                Departure_City_Removed_From_Route,
                ERROR_Route_Empty    }

    CONSTANTS
      MaxRouteLength, ROUTE, UNDEFINED_ROUTE
```

```
PROPERTIES
    MaxRouteLength : NAT1  &  MaxRouteLength = 5
    &
    ROUTE = iseq(CITY)
    &
    UNDEFINED_ROUTE : ROUTE  &  UNDEFINED_ROUTE = []

VARIABLES
    flightroute

INVARIANT
    flightroute : ROUTE  &  size( flightroute ) <= MaxRouteLength
    &
    UnknownCity /: ran( flightroute )

INITIALISATION
    flightroute := UNDEFINED_ROUTE
```

Marks for each clause: `SETS` **[3 marks]** , `CONSTANTS` & `PROPERTIES` **[3 marks]** , `VARIABLES INVARIANT` & `INITIALISATION` **[3 marks]** .

[PART Total 9]

(b) Append to Queue:

```
OPERATIONS

    report <-- AppendCityToRoute( city ) =
        PRE
            city : CITY  &  city /: ran( flightroute )
            & city /= UnknownCity   &  report : REPORT
        THEN
            IF ( size( flightroute )  < MaxRouteLength )
            THEN
                flightroute  := flightroute <- city  ||
                report := City_Added_To_Route
            ELSE
                report := ERROR_Route_is_Full
            END
        END ;
```

[Subpart (b.i) 7 marks]

Remove from Queue:

```
report <-- RemoveDepartureCityFromRoute =
    PRE
        report : REPORT
    THEN
        IF ( flightroute /= UNDEFINED_ROUTE )
        THEN
            flightroute  := tail( flightroute )         ||
            report := Departure_City_Removed_From_Route
        ELSE
            report := ERROR_Route_Empty
        END
    END ;
```

[Subpart (b.ii) 5 marks]

```
    status <-- RouteStatus =
        PRE
            status : ROUTE_STATUS
        THEN
            IF  ( flightroute = UNDEFINED_ROUTE )
            THEN
                 status := Route_is_Empty
            ELSIF
                 ( card(flightroute) = MaxRouteLength )
            THEN
                 status := Route_is_Full
            ELSIF
                 ( card(flightroute) = 1 )
            THEN
                 status := Route_Only_Has_Departure_City
             ELSE
                 status := Route_Can_Be_Extended
            END
        END
END /* FlightRoutes */
```

[Subpart (b.iii) 4 marks]

[**PART Total 16**]

[QUESTION Total 25]

Question 6

Refer to the `TaxiFirm` B machine given in the exam paper's Appendix ??.

(a) **(i)** `maxpassengers : TAXI --> NAT1`

Every taxi must have a maximum limit on the number of passengers it can take, it can obviously take fewer. **[2 marks]** It is not sensible to use a *partial* function since that would mean at least one taxi hasn't got a limit. **[2 marks]**

[SUBPART Total 4]

(ii) `passengers : TAXI <-> CUSTOMER`

The relationship between a taxi & its passengers is one-to-many, since a taxis can take more than one passenger. **[2 marks]** Using a *function* would mean that all taxis could only take a single passenger, since a function can only map a taxi to one passenger. **[1 mark]**

[SUBPART Total 3]

(iii) `booked : CUSTOMER >+> TAXI`

A customer can book only one taxi at a time & a taxi cannot be booked by more than one person at a time.

[SUBPART Total 3]

(iv)
```
!(taxi).( taxi : dom(passengers)  =>
          ( card( passengers[ { taxi } ] )
                   <= maxpassengers( taxi ) )
        )
```

The number of passengers in any taxi must not exceed the maximum number of passengers for the particular taxi.

[SUBPART Total 3]

[PART Total 13]

(b) **(i)** `bookTaxi` preconditions: the valid customer does not already have a booking & the valid taxi has not been booked.

[SUBPART Total 2]

(ii) `passengersPickedUp` preconditions: the valid taxi has been booked & is waiting for a fare. **[2 marks]** And there is at least one valid customer/passenger, but not more than the taxi can take. **[3 marks]**

[SUBPART Total 5]

(iii) `passengersDroppedOff` preconditions: the valid taxi has been on a fare's journey.

[SUBPART Total 1]

[PART Total 8]

(c) The `passengersPickedUp` operation assignments:

```
passengers := passengers <+ ( { taxi } * customers ) ||
```

The `passengers` relation is updated by either having existing `taxi` mappings replaced or adds new mappings from the `taxi` to each of the passengers that have been picked up by the `taxi`. **[2 marks]**

```
status := status <+ { taxi |-> OnJourney }
```

The `status` function is updated by changing the mapping from the `taxi` to its current status to now being on a fare's journey, i.e. `OnJourney`. **[2 marks]**

[PART Total 4]

[QUESTION Total 25]

Question 7

Marking Scheme for Hoare Logic & Program Verification.

(a) **(i)** The Hoare triple

$$[x > z]\ y := z\ [x > y]$$

means that executing the instruction $y := z$ (i.e. assigning the value of z to y), starting from a state in which x is greater than z, leads to a state in which x is greater than y. **[2 marks]**

[SUBPART Total 2]

(ii) $[x > 0]\ x := x + 1\ [true]$ is valid (since any state satisfies $true$). **[2 marks]**
[SUBPART Total 2]

(iii) $[x > 0]\ x := x + 1\ [x = x + 1]$ is invalid: **[1 mark]** Starting from a state in which $x = 1$, executing $x := x + 1$ leads to a state in which $x = 2$, but $2 \neq 2 + 1$. **[1 mark]**
[SUBPART Total 2]

(iv) $[false]\ x := x + 1\ [x = x + 1]$ is valid (vacuously so, since there are no states satisfying false). **[2 marks]**
[SUBPART Total 2]

(v) $[y > 1]\ x := x+1\ [y > 0]$ is valid, since $x := x+1$ does not change y, so starting from a state in which $y > 1$, we end up in another state in which $y > 1$ and therefore also $y > 0$. **[2 marks]**
[SUBPART Total 2]

(vi) $[y > 1]\ x := x + 1\ [x > 1]$ is invalid: **[1 mark]** Starting from a state in which $x = -1, y = 2$, increasing x by one gets us to a state in which $x = 0, y = 2$, but not $x > 1$. **[1 mark]**
[SUBPART Total 2]

[PART Total 12]

(b) The intermediate assertions are

1. $(x > y \Rightarrow y < 10)\&(x \leq y \Rightarrow x < 10)$ **[1 mark]**
2. $y < 10$ **[1 mark]**
3. $x - (x - y) < 10$ or $y < 10$ **[1 mark]**
4. $x - y < 10$ **[1 mark]**
5. $x < 10$ **[1 mark]**

[PART Total 5]

(c) **(i)** $P \Rightarrow I$, i.e. I needs to follow from the precondition. **[1 mark]**
$[I\&B]\ S\ [I]$ must be valid, i.e. executing the loop body once starting from a state satisfying I and B leads to another state satisfying I. **[1 mark]**
$(I\&\neg B) \Rightarrow Q$, i.e. when we exit the loop, I needs to imply the postcondition. **[1 mark]**
[SUBPART Total 3]

(ii) Choosing I to be $2x + y > 5$ works. $[\mathbf{2\ marks}]$

As for $P \Rightarrow I$, we have $x > 1 \wedge y > 1 \Rightarrow 2x + y > 5$ because when x and y are both at least 2, $2x + y$ is at least 6. $[\mathbf{1\ mark}]$

As for $[I\&B]\ S\ [I]$, the intermediate assertions we get for S and I are

$[2x + y > 5]$
$x := x - 1$
$[2x + y > 3]$
$y := y + 2$
$[2x + y > 5]$,

and obviously $2x + y > 5\ \&\ x > 0$ implies $2x + y > 5$, i.e. the Hoare triple is valid. $[\mathbf{1\ mark}]$

As for $(I\&\neg B) \Rightarrow Q$, from $2x + y > 5$ and $x \leq 0$ we can get $y > 5 - 2x \geq 5$. $[\mathbf{1\ mark}]$

$[\mathbf{SUBPART\ Total\ 5}]$

$[\mathbf{PART\ Total\ 8}]$

$[\mathbf{QUESTION\ Total\ 25}]$