UNIVERSITY OF WESTMINSTER

MARKING SCHEME
DO NOT COPY

School of Computer Science and Engineering

Module:	**Reasoning about Programs**
Module Code:	**6SENG001W, 6SENG003C**
Module Leader:	Klaus Draeger
Date:	9th January 2019
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.

In section B, only the TWO questions with the HIGHEST MARKS will count towards the FINAL MARK for the EXAM.

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

School of Computer Science and Engineering
Module: Reasoning about Programs **[MARKING SCHEME]**
Module Code: 6SENG001W, 6SENG003C

Section A

Answer ALL questions from this section.

Question 1

(a) $BakerlooStations \cap VictoriaStations = \{Oxford\_Circus\}$
[**1 mark**]

(b) $CentralStations - VictoriaStations = \{ Bond\_Street, Tottenham\_Court\_Road \}$
[**1 mark**]

(c) $\text{card}(\ CircleStations\ ) = 3$ [**1 mark**]

(d) $Baker\_Street \mapsto Central \in onLine = FALSE$ [**1 mark**]

(e) $\text{ran}(onLine) = TubeLines$ [**1 mark**]

(f) $colour(Central) = red$ [**1 mark**]

(g) $\bigcup(\ \{\ \{\ Bond\_Street,\ Euston\_Square\ \},\ \{\},\ \{\ Warren\_Street\ \}\ \}\ )$
$= \{Bond\_Street,\ Euston\_Square,\ Warren\_Street\ \}$ [**2 marks**]

(h) $BakerlooStations \subseteq \text{dom}(onLine) = TRUE$ [**2 marks**]

(i) $CentralStations \cap \text{dom}(onLine) = CentralStations$ [**2 marks**]

(j) $\mathbb{P}(BakerlooStations)$

$= \{\ \{\},\ \{\ Baker\_Street\ \},\ \{\ Regents\_Park\ \},\ \{\ Oxford\_Circus\ \},$
$\{Baker\_Street,\ Regents\_Park\},\ \{Baker\_Street,\ Oxford\_Circus\},$
$\{Regents\_Park,\ Oxford\_Circus\},$
$\{Baker\_Street,\ Regents\_Park,\ Oxford\_Circus\}\ \}$

[**3 marks**]

[**QUESTION Total 15**]

Question 2

See the *TubeSystem* B machine of the London Underground System, that is given in an Appendix of the Exam paper.

(a) The relation $onLine$ in cannot be defined as a function because it contains maplets that map one element of the domain to more than one element in the range, e.g.

$$\begin{aligned} = \{\ \ldots,\ & Baker\_Street \mapsto Bakerloo, Baker\_Street \mapsto Circle, \\ & Oxford\_Circus \mapsto Bakerloo, Oxford\_Circus \mapsto Central, \\ & Oxford\_Circus \mapsto Victoria,\ \ldots\ \} \end{aligned}$$

[**2 marks**]

[**PART Total 2**]

(b) Yes, the $colour$ function can be defined as a total injective function,

$$colour \in TubeStation \rightarrowtail Colour$$

Because the domain equals the target set (total) & each element in the domain is mapped to a different element in the range, i.e. it is a total injective function. [**3 marks**]

[**PART Total 3**]

(c) Evaluate the following expressions:

(i) $onLine[\ \{\ Baker\_Street,\ Oxford\_Circus\ \}\ ]$
$= \{\ Circle, Central, Bakerloo, Victoria\ \}$ [**2 marks**]

(ii) $CircleStations \lhd onLine$
$= \{\ Baker\_Street \mapsto Circle,\ Baker\_Street \mapsto Bakerloo,$
$Great\_Portland\_Street \mapsto Circle,\ Euston\_Square \mapsto Circle\ \}$

[**2 marks**]

(iii) $onLine \rhd \{\ Victoria\ \}$
$= \{\ Oxford\_Circus \mapsto Victoria,\ Warren\_Street \mapsto Victoria\ \}$

[**2 marks**]

(iv) $onLine \rhd \{\ Bakerloo,\ Central,\ Victoria\ \}$
$= \{\ Baker\_Street \mapsto Circle,\ Great\_Portland\_Street \mapsto Circle,$
$Euston\_Square \mapsto Circle\ \}$ [**2 marks**]

(v) $colour \lhd\!\!+ \{ Circle \mapsto red,\ Central \mapsto yellow \}$
$= \{ Bakerloo \mapsto brown, Circle \mapsto red,$
$Central \mapsto yellow, Victoria \mapsto lightblue \}$ **[2 marks]**

(vi) $colour^{-1}$
$= \{ brown \mapsto Bakerloo,\ yellow \mapsto Circle,$
$red \mapsto Central,\ lightblue \mapsto Victoria \}$ **[2 marks]**

(vii) $(\ CircleStations \lhd onLine\ )\ ;\ colour$
$= \{ Baker\_Street \mapsto brown),\ Baker\_Street \mapsto yellow,$
$Great\_Portland\_Street \mapsto yellow,\ Euston\_Square \mapsto yellow \}$

[3 marks]

[QUESTION Total 20]

Question 3

(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. **[1 mark]**

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 5]

(b) Explain the purpose of the following B Machine *clauses*:

- `VARIABLES` – declare state variable identifiers. **[1 mark]**
- `INVARIANT` – define the *state invariant* for the system, including the types of the variables & any additional constraints on them. **[2 marks]**
- `INITIALISATION` – initialise all the state variable with values that *satisfy the state invariant*. **[2 marks]**

[PART Total 5]

(c) To add a tube passenger's current *location*, (station and line), to the *TubeSystem* B machine, need to add the following clauses:

```
VARIABLES
    station, line
INVARIANT
    station ∈ Stations  ∧  line ∈ TubeLine ∧
    station ↦ line  ∈  onLine
INITIALISATION
    station := Oxford_Circus  ||
    line := Victoria
```

[5 marks]

Any pair of station & line can be used to initialise the two variables, as long as they satisfy the invariant, i.e. are *consistent* with $onLine$.

[PART Total 5]

[QUESTION Total 15]

Section B

Answer TWO questions from this section.

Question 4

The LuggageRack B machine; basically its a **stack**. So would expect something similar to the following machine, but a student's version is unlikely to include all the details included in this *solution*, as long as its a stack & has the main parts will not penalise for minor errors, e.g. syntax errors, etc.

(a)

```
MACHINE LuggageRack

SETS
  LUGGAGE  = { case1, case2, case3, case4, case5,
               bag1,  bag2,  bag3,  bag4,  bag5,
               null_bag } ;

  ANSWER  = { Yes, No } ;

  MESSAGE = { Luggage_Added,   ERROR_No_Space_Left,
              Luggage_Removed, ERROR_No_Luggage_Left }

CONSTANTS
  MaxItemsOfLuggage, No_Luggage, EMPTY_LuggageRack

PROPERTIES
  MaxItemsOfLuggage : NAT1         &  MaxItemsOfLuggage = 5   &
  No_Luggage        : LUGGAGE      &  No_Luggage = null_bag   &
  EMPTY_LuggageRack : seq(LUGGAGE) &  EMPTY_LuggageRack = []

VARIABLES
  luggageRack

INVARIANT
  luggageRack : seq( LUGGAGE ) &
  size( luggageRack ) <= MaxItemsOfLuggage

INITIALISATION
  luggageRack := EMPTY_LuggageRack
```

Roughly award: SETS [**2 marks**] , CONSTANTS & PROPERTIES [**3 marks**] , VARIABLES & INVARIANT [**3 marks**] , INITIALISATION [**1 mark**] .

[**PART Total 9**]

(b) **(i)** In this version the "top" of the luggage rack (stack) is the front of the sequence, but okay if the end. Might use strings for reporting rather than MESSAGE.

```
OPERATIONS

  report <-- AddLuggage( luggage ) =
      PRE
          report : MESSAGE & luggage : LUGGAGE
      THEN
          IF ( size(luggageRack) < MaxItemsOfLuggage )
          THEN
              luggageRack := luggage -> luggageRack ||
              report      := Luggage_Added
          ELSE
              report := ERROR_No_Space_Left
          END
      END ;
```

[**PART Total 6**]

(ii)

```
  report, topcase <-- RemoveLuggage =
      PRE
          report : MESSAGE & topcase : LUGGAGE
      THEN
          IF ( luggageRack /= EMPTY_LuggageRack )
          THEN
              luggageRack := tail(luggageRack) ||
              report      := Luggage_Removed   ||
              topcase     := first(luggageRack)
          ELSE
              report  := ERROR_No_Luggage_Left ||
              topcase := No_Luggage
          END
      END ;
```

[**PART Total 7**]

(iii)

```
answer <-- AnyLuggageLeft =
    PRE
        answer : ANSWER
    THEN
        IF   ( luggageRack /= EMPTY_LuggageRack )
        THEN
             answer := Yes
        ELSE
             answer := No
        END
    END
END  /* LuggageRack */
```

[**PART Total 3**]

[**PART Total 16**]

[**QUESTION Total 25**]

Question 5

See Exam paper Appendix for the `BirthdayBook` B machine.

(a) The B type of `DATE` is an element of the Cartesian product of days & months, i.e. an ordered pair (or maplet). [**1 mark**]

Today's date is represented: $9 \mapsto Jan$ (exam date). [**1 mark**]

[**PART Total 2**]

(b) The B type of `known` is a set of names. Example value is any subset of `NAME`, e.g. `known = { Jim, Sue, Mon, Zoe }`. [**1 mark**]

[**PART Total 1**]

(c) `birthday` is a *partial function* because the birthday book will not always hold everyone's birthday & everyone has just one birthday. [**1 mark**]

It is not:

- a relation because no one has more than one birthday, (excluding the Queen). [**1 mark**]
- a total function since the birthday book does not always hold everyone's birthday. [**1 mark**]

- an injective function since several people can have the same birthday. [**1 mark**]
- a surjective function since not every possible date will be recorded as someone's birthday. [**1 mark**]
- a bijections, since its neither an injective, surjective or total function. [**1 mark**]

[**PART Total 6**]

(d) `birthday`'s constraints:

- `card( birthday ) <= maximum`
  the birthday book has a maximum limit on how many birthdays it can record. [**1 mark**]
- `known = dom( birthday )`
  the known people are those that have a birthday recorded. [**1 mark**]
- `NonDate /: ran( birthday )`
  no one can have a birthday recorded as being on the 31st February. [**1 mark**]

[**PART Total 3**]

(e) *Pre-conditions* for the following operations:

(i) `AddBirthday`
The name must be a proper name. [**1 mark**] The date must be valid date & it cannot be 31st February. [**2 marks**] There must be room in the birthday book to record at least one more birthday. [**1 mark**]
[**SUBPART Total 4**]

(ii) `Reminder`
The date must be valid date & it cannot be 31st February. [**1 mark**] Only a collection (set) of real names can be output. [**1 mark**]
[**SUBPART Total 2**]

(iii) `NumKnownBirthdays`
Since there is no explicit pre-condition for this operation, it is implicitly just "TRUE". [**2 marks**]
[**PART Total 2**]

[**PART Total 8**]

(f) `AddBirthday` updates the state as follows:

- Provided the person's `name` is not already in the birthday book it updates the state by adding the `name` to the known people & adds the `name` and `date` pair to the birthday book. [**1 mark**]
- If the person's `name` is already in the birthday book it does not add them again, i.e. it does not update the state. [**1 mark**]

[**PART Total 2**]

(g) The state to produce the ProB output: `report = Birthdays_On_Date & cards = { Jim, Sue }` for "`Reminder( 10 |-> Jul )`" is any pair of `known` & `birthday` that satisfy:

$$\{\ Jim,\ Sue\ \} \subseteq known$$
$$\{\ Jim \mapsto (10 \mapsto Jul),\ Sue \mapsto (10 \mapsto Jul)\ \} = birthday \rhd \{\ 10 \mapsto Jul\ \}$$

[**3 marks**]

[**PART Total 3**]

[**QUESTION Total 25**]

Question 6

Marking Scheme for Hoare Logic & Program Verification.

(a) The Hoare triple

$$[x = 0]\ y := z\ [z = x + y]$$

means that executing the instruction $y := z$ (i.e. assigning the value of z to y), starting from a state in which x is 0, leads to a state in which z equals $x + y$. [**2 marks**]

[**SUBPART Total 2**]

(b) **(i)** $[x < y]\ y := 0\ [x < 0]$ is invalid. [**1 mark**] Counterexample: Starting in a state with $x = 1, y = 2$ leads to a state with $x = 1, y = 0$. [**1 mark**]

[**SUBPART Total 2**]

(ii) $[x < y]\ y := y+1\ [x < y+1]$ is valid: the pre-condition is $x < y+2$, which follows from $x < y$. **[2 marks]**

[SUBPART Total 2]

(iii) $[x < y]\ y := y - 1\ [x < y - 1]$ is invalid. **[1 mark]** Counterexample: Starting in a state with $x = 1, y = 2$ leads to a state with $x = 1, y = 0$. **[1 mark]**

[SUBPART Total 2]

(iv) $[true]\ x := 0\ [true]$ is valid, since any post-state satisfies $true$. **[2 marks]**

[SUBPART Total 2]

[PART Total 10]

(c) The intermediate assertions are

1. $0 < z$ **[2 marks]**
2. $0 < y + z$ **[2 marks]**
3. $x < y + z$ **[2 marks]**

[PART Total 6]

(d)
$[y = 10]$
$x := 0;$
$[x + y = 10\ \&\ y \geq 0]$ **[3 marks]**
$WHILE\ y > 0\ DO$
$[x + y = 10\ \&\ y > 0]$ **[2 marks]**
$x := x + 1;$
$[x + y = 11\ \&\ y > 0]$ **[2 marks]**
$y := y - 1$
$[x + y = 10\ \&\ y \geq 0]$ **[2 marks]**
END
$[x = 10]$

[PART Total 9]

[QUESTION Total 25]