Office Use Only

MONASH University

Semester One 2017
Examination Period

Faculty of Information Technology

EXAM CODES: **FIT5171**

TITLE OF PAPER: **SYSTEM VALIDATION AND VERIFICATION, QUALITY AND STANDARDS - PAPER 1**

EXAM DURATION: 2 hours writing time

READING TIME: 10 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Berwick	☐ Clayton	☐ Malaysia	☐ Off Campus Learning	☐ Open Learning
☒ Caulfield	☐ Gippsland	☐ Peninsula	☐ Monash Extension	☐ Sth Africa
☐ Parkville	☐ Other (specify)			

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam.
Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	☒ **YES**	☐ **NO**
CALCULATORS	☒ **YES**	☐ **NO**
SPECIFICALLY PERMITTED ITEMS **if yes, items permitted are:**	☐ **YES**	☒ **NO**

Candidates must complete this section if required to write answers within this paper

STUDENT ID: __ __ __ __ __ __ __ __ DESK NUMBER: __ __ __ __ __

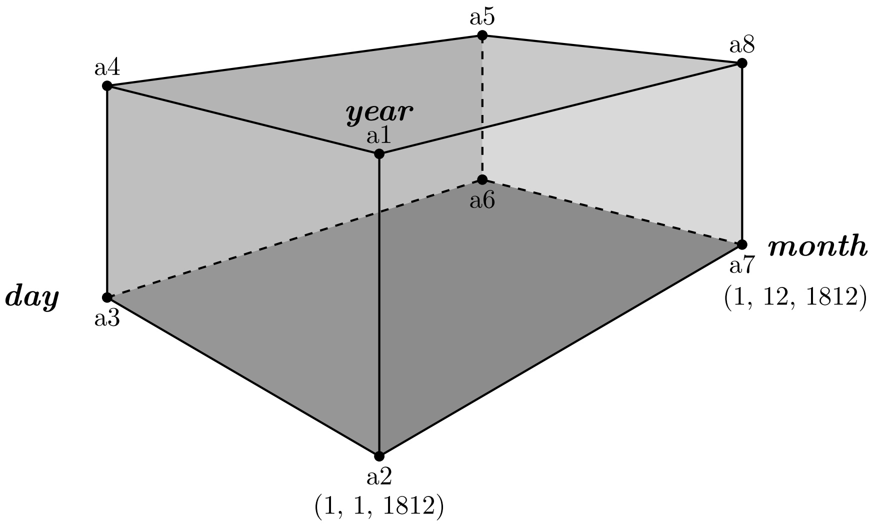
Answer all questions in the space provided here.

Question	Points	Score
Question 1	16	
Question 2	9	
Question 3	9	
Question 4	10	
Question 5	6	
Total	50	

Question 1 .... **16 marks**

In the lecture we discussed the testing of a **NextDate** method, which, given a day, a month, and a year, returns the date of the following day. Assume the year variable ranges over [1812, 2016].

Since there are three variables (day, month, and year), we can visualise the boundaries using a 3D plot, as follows.



Each of the 3 axes represents day, month, and year, respectively, and the internal of the 3D shape represents the valid values. Each of the end points in the rectangular cube, labelled a1, a2 to a8, represents a specific tuple of boundary values for the three variables day, month, and year. Point a2 represents the allowed min values for day, month, as well as year, i.e., (1, 1, 1812). Point a7 represents the allowed min values for day and year, but the allowed max value for month, i.e., (1, 12, 1812).

Given the above visualisation, we can reason about test cases for **strong**, **normal** boundary value testing (BVT) on points, lines, planes, and the cube itself. An example, for points, is given below.

Points. For each point (denoted a1 to a8), there are 8 test cases for the 3 variables. For example, for points a2 and a7, we have the following test cases at and around the min values of the 3 variables:

(a) Test cases for point a2.

day	month	year
1	1	1812
1	1	1813
1	2	1812
1	2	1813
2	1	1812
2	1	1813
2	2	1812
2	2	1813

(b) Test casse for point a7.

day	month	year
1	11	1812
1	11	1813
1	12	1812
1	12	1813
2	11	1812
2	11	1813
2	12	1812
2	12	1813

Hence, for all 8 points, we need 8*8=64 test cases.

Now we'd like to extend the method to **NextHour**, with an additional variable, *hour*, that represents the 24 hours of a day (ranging between 0 and 23). Given an hour, a day, a month, and a year, **NextHour** returns the hour as well as the date of the following hour. The 3D cube now becomes a 4D *tesseract*. A tesseract is a four-dimensional analog of a cube, with 16 points, 32 lines, 24 planes, 8 cubes, and (of course) the tesseract itself.

For **NextHour**, please complete the following tasks.

(a) (14 marks) Identify details of test cases for **strong, robust** BVT testing. Specifically,

1. Give details of test cases for (1) points, (2) (the mid point of) lines, (3) (the centre of) planes, (4) (the centre of) cubes, and (5) the (4D) tesseract.
2. Calculate the total number of test cases.

— blank page for answers if required. Will be marked. —
— Indicate clearly question number. —

(b) (2 marks) Generalising from the previous part. What is the number of total test cases for strong, robust BVT for n variables?

Question 2 .... **9 marks**

The binary search tree (BST) is a data structure that is very efficient in sorting, search and in-order traversal. A BST has the following properties:

- all nodes are comparable,
- all nodes of a node's left subtree are *less than* the node itself,
- all nodes of a node's right subtree are *greater than* the node itself,
- Each subtree is a BST, and
- there are no duplicate nodes.

The insertion of a node into a BST can be specified using the algorithm.

Algorithm 0: The insertion operation of the binary search tree.

```
Input: node                                         ▷ The node to be inserted
Input: root                                         ▷ The root node of the BST
if root = null then
    root ← node
    return
end
while root ≠ null do
    if node = root then                             ▷ Node already in the BST
        return
    else if node < root then                        ▷ Insert left
        if root.left = null then
            root.left ← node
            return
        else
            root ← root.left
        end
    else                                            ▷ Insert right
        if root.right = null then
            root.right ← node
            return
        else
            root ← root.right
        end
    end
end
```

(Continued overleaf)

(a) (4 marks) Calculate the cyclomatic complexity of the algorithm, and show your calculation.

(b) (2 marks) Recall the concept of *structured programming constructs*. For the algorithm above, identify **all** *violations* of structured programming constructs.

(c) (3 marks) Propose changes to the algorithm to make it free of such violations.

Listing 1 below shows, on two pages, the class `Foo` in Java. Answer **Question 3**, **Question 4** and **Question 5** about the class `Foo` and its tests.

Listing 1: The Java class `Foo`.

```java
public class Foo {

    private int min;
    private int max;

    public String fizzBuzz(String input) {
        int x = Integer.parseInt(input);

        boolean hasFizz = x % 3 == 0;
        boolean hasBuzz = x % 5 == 0;
        if (hasFizz && hasBuzz)
            return "FizzBuzz";
        else if (hasFizz)
            return "Fizz";
        else if (hasBuzz)
            return "Buzz";
        else
            return input;
    }

    public String[] fizzBuzzRange(int low, int high) {
        if (low <= 1)
            throw new IllegalArgumentException("low should be >= 1");
        else if (high > 100)
            throw new IllegalArgumentException("high should be <= 100");

        String[] result = new String[high - low + 1];
        for (int i = low; i <= high; i++)
            result[i - low] = fizzBuzz(Integer.toString(i));

        return result;
    }
```

(Continued overleaf)

```java
    private void findMinMax(int[] array) {
        min = array[0];
        max = array[0];

        for (int i : array) {
            if (min > i)
                min = i;
            else if (max < i)
                max = i;
        }
    }

    public int[] unique(int[] array) {
        findMinMax(array);

        boolean[] set = new boolean[max - min + 1];
        for (int i : array)
            set[i - min] = true;

        int size = 0;
        for (boolean i : set) {
            if (i)
                size++;
        }
        int[] result = new int[size];
        int j = 0;
        for (int i = 0; i < set.length; i++) {
            if (set[i])
                result[j++] = i + min;
        }
        return result;
    }

    public int maxOccurrences(int[] array) {
        findMinMax(array);

        int[] set = new int[max - min + 1];
        for (int i : array)
            set[i - min]++;

        max = set[0];
        for(int i : set) {
            max = max < i ? i : max;
        }
        return max;
    }
}
```

Question 3 .... **9 marks**

"Fizz Buzz" has been used as a simple interview question for software developers. In its simplest form, the program takes as input an integer value between 1 and 100 (both inclusive), and prints the number itself when it is not divisible by either three or five. For numbers which are multiples of both three and five the program should print "`FizzBuzz`" instead. Otherwise, for multiples of three the program should print "`Fizz`" instead of the number, for multiples of five the program should print "`Buzz`". Methods `fizzBuzz` and `fizzBuzzRange` in code listing 1 above are a simple implementation in Java.

The following test suite in Listing 2 has been developed for the `fizzBuzz` and `fizzBuzzRange` methods. Answer the following questions about the test suite.

Listing 2: A test suite for the `fizzBuzz()` and `fizzBuzzRange` methods.

```java
@Test
public void testFizzBuzzRange() {
    System.out.println(Arrays.toString(new Foo().fizzBuzzRange(1, 100)));
}

@Test
public void wrongNumbersAreNotProcessed() {
    try {
        new Foo().fizzBuzz("0");
    } catch (Exception e) {
        String message = e.getMessage();
        assertTrue("Contains correct message", message.contains(">= 1"));
    }
}

@Test(expected = Exception.class)
public void illegalInputThrowsException() {
    new Foo().fizzBuzz(" ");
    new Foo().fizzBuzz(" a");
    new Foo().fizzBuzz("   ");
}

@Test
public void threeGetsFizzAndFiveGetsBuzz() {
    try {
        assertEquals("Should return Fizz", "Fizz", new Foo().fizzBuzz("3"));
        assertEquals("Should return Fizz", "Fizz", new Foo().fizzBuzz("6 "));
        assertEquals("Should return Buzz", "buzz", new Foo().fizzBuzz("5"));
        assertEquals("Should return Buzz", "buzz", new Foo().fizzBuzz("10"));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(a) (1 mark) What is the statement coverage of this test suite for these two methods (`fizzBuzz` and `fizzBuzzRange`)?

Note that the lines you need to consider include those lines in the body of the two methods, excluding empty lines and line 17. In other words, the total number of lines to cover is 18.

(b) (8 marks) There are some problems (errors or deficiencies) with some of these test cases. (1) List these problems, and (2) discuss how they can be fixed.

Question 4 .... **10 marks**

The method `unique` returns the unique elements from an input `int` array that possibly contains duplicates. For example, given an input array {`-1, -1, -1, -1, 10`}, `unique` returns the array {`-1, 10`}. It makes use of method `findMinMax` that finds the minimum and maximum values of a given `int` array.

(a) (4 marks) Devise a test suite with the smallest possible number of test cases that achieves 100% *branch coverage*, or *DD-path coverage*, for methods `unique` and `findMinMax`, and argue why it is the smallest test suite.

(b) (6 marks) Mutation testing is a technique to assess the efficacy and quality of a test suite. It works by making *mutants*, syntactic variations of the program under test, and measuring how many of the mutants are *killed* by the test suite. The presence of non-equivalent *live* mutants represents inadequacy of the test suite.

Come up with three *non-equivalent*, first-order mutants of the method `unique`. Each mutant should use one of the following mutation operators. Determine the *kill rate* of your test suite on each of the three mutants.

The mutation operators you can use are:

- `aor`: Arithmetic operator replacement.
- `ror`: Relational operator replacement.
- `sdl`: Statement deletion.
- `uoi`: Unary operator insertion.
- `svr`: Scalar variable replacement.
- `vie`: Scalar variable initialisation elimination.

Question 5 .... **6 marks**

The quality of classes in object-oriented languages such as Java can be measured by different object-oriented metrics. Some metrics measure the *lack of cohesion* of a class based on interactions between its methods and attributes. Answer the following questions on the *cohesion* of the class `Foo`.

(a) (3 marks) Specifically, the metric LCOM1 is defined as

$$LCOM1 = \begin{cases} P - Q, & \texttt{for } P > Q, \\ 0 & \texttt{otherwise} \end{cases}$$

where for each pair of different methods (order of methods irrelevant), P is incremented by 1 if they do not access any common attribute, otherwise Q is incremented by 1. The initial values of P and Q are both 0.

Compute the $LCOM1$ value for the class `Foo`. Include all its methods (`public` and `private`) in your calculation.

(b) (3 marks) $LCOM2$ is another lack of cohesion metric for classes. Specifically, $LCOM2$ is defined as

$$LCOM2 = 1 - \frac{\sum\limits_{\text{for each class attribute} A} \#m_A}{m * a}$$

where m is the number of methods, a is the number of attributes, and $\#m_A$ is the number of methods that access a particular attribute A.

Compute the $LCOM2$ value for the class `Foo`. Include all its methods (`public` and `private`) in your calculation.

— Additional page for answers if required. Will be marked. —
— Indicate clearly question number. —

— Additional page for answers if required. Will be marked. —
— Indicate clearly question number. —

—— End of the paper ——