IVE Information Technology

Information & Communications Technology

Programme Board

Instructions:

(a) This paper has a total of **THIRTEEN** pages including the covering page.

(b) This paper contains TWO Sections.

(c) Section A is WORTH 40 marks and Section B is WORTH 60 marks.

(d) Section A contains FOUR questions. Answer ALL questions in Section A.

(e) Section B contains THREE questions. Answer ALL questions in Section B.

Note: The result of this assessment will not be counted if you do not meet the minimum attendance requirement (if any) governed by the general academic regulations of your programme/course unless approval of the campus principal has been granted.

HIGHER DIPLOMA IN

SOFTWARE ENGINEERING

(IT114105)

MODULE TITLE:

DATA STRUCTURES & ALGORITHMS: CONCEPTS AND IMPLEMENTATION

MODULE CODE: **ITP4510**

SEMESTER TWO
MAIN EXAMINATION

6 May, 2019
1:30 PM TO 3:30 PM (2 hours)

This paper contains TWO sections.

Section A (40 marks)

This section contains 4 questions.

Answer ALL questions.

A1 Consider the following JAVA classes and interface.

```
interface Moveable {
    void moveTo(int x, int y, double speed);
}

abstract class Human implements Moveable {
    int locationX;
    int locationY;
    String name;
    double runningSpeed;
    Human(String name, double runningSpeed) {
        this.name = name;
        this.runningSpeed = runningSpeed;    }
}

abstract class SoccerPlayer extends Human {
    int shirtNum;
    boolean injured;
    public SoccerPlayer(String name, double runningSpeed, int
         shirtNum) {
        super(name, runningSpeed);
        this.shirtNum = shirtNum;
        injured = false;    }

    void dribble(double direction, double speed) {
        System.out.println(name + " is dribbling");    }

    abstract void jump();
}

class StandardPlayer extends SoccerPlayer {
    public StandardPlayer(String name, double runningSpeed, int
         shirtNum) {
        super(name, runningSpeed, shirtNum);    }
    // some codes are hidden
}

class GoalKeeper extends SoccerPlayer {
    public GoalKeeper(String name, double runningSpeed) {
        super(name, runningSpeed, 1);    }
}
```

(a) What method(s) MUST be implemented in class *GoalKeeper* in order to get it complied successfully? [4 marks]

(b) Can each line of the following JAVA code be compiled successfully? For each line, if it can be compiled successfully, state the meaning of the code; if not, state the reason. [6 marks]

```
SoccerPlayer[] team = new SoccerPlayer[11];      //line 1
team[0] = new StandardPlayer("Messi", 8.1, 10);  //line 2
team[1] = new GoalKeeper("Buffon", 6.2, 1);      //line 3
```

A2 Consider the following Java program.

(a) What exception will be generated from the JAVA code below? Is it a checked or unchecked exception? [4 marks]

```
int h[] = {2, 4, 6, 8, 10};
int x = 9;
System.out.println(h[x]);
```

(b) State the output of the JAVA program on the next page. [6 marks]

```
class CustomExpA extends ArithmeticException {
    public CustomExpA() {
        super("cus exp a");
    }
}

public class ExceptionSample1 {
    public static void main(String[] args) {
        try {
            methodC(48, 0);
            System.out.println("main try");
        }
        catch (Exception e) {
            System.out.println("exp");
        }
        System.out.println("main finished");
    }

    public static void methodB(double b, int t) {
        double ans;
        if (t == 0) {
            System.out.println("in b part 1");
            throw new CustomExpA();
        } else
            ans = b / t;
        System.out.println(ans);
        System.out.println("in b part 2");
    }

    public static void methodC(double b, int t) {
        try {
            System.out.println("in c part 1");
            methodB(b, t);
            System.out.println("in c part 2");
        }
        catch (ArithmeticException e) {
            System.out.println("ari exp");
            throw e;
        }
        catch (RuntimeException e) {
            System.out.println("run exp");
        }
        finally {
            System.out.println("fin");
        }
        System.out.println("in c part 3");
    }
}
```

A3 A stack can be implemented by using an array of objects. Consider the following implementation of ***ArrayStack*** class. An *int* variable **top** is the index of the last element. When the stack is empty, the value of **top** is -1.

```
public class ArrayStack implements Stack {
   public static final int CAPACITY=1000;
   private int capacity;
   private Object [ ] array;
   private int top=-1;
   public boolean isEmpty() {  ......  }
   public int size() {  ......  }
   public Object top() throws StackEmptyException {
      /** to be implemented in part (a)(i) **/
   }
   public void push(Object item) throws StackFullException {
      /** to be implemented in part (a)(ii) **/
   }
   public Object pop() throws StackEmptyException {
      ......
   }
}
```

(a) (i) Complete the ***top*** method. The method intends to return the last element in the stack. [2 marks]

(ii) Complete the ***push*** method. The method intends to put the data item on top of the stack. [3 marks]

(b) Copy and complete the following table on your answer book. The table shows the conversion of the following infix expression to postfix expression by using a stack: [5 marks]

Inputted infix expression:
1 - 2 * (4 + 1)

Inputted element	Stack after processing element	Output
1		1
-	-	
2	-	1 2
*		
(		
4		
+		
1		
)		
Last step		

The algorithm of infix to postfix conversion is given in **Appendix I**.

A4 A queue can be implemented by using a circular array. A snapshot of a queue is given below.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Content	**Lemon**	**Lychee**	**Plums**	***null***	***null***	***null***	***null***	**Cherry**	**Apple**

(a) What is the maximum number of data that can be stored? [1 mark]

(b) If the index of front is 7, what is the index of rear? [1 mark]

(c) How to check the queue whether it is empty by using the front and rear? [1 mark]

(d) Copy and complete the following table on your answer book to show the status of the given queue after following operations. [3 marks]

dequeue()
dequeue()
enqueue("Pineapple")
front()
enqueue("Pear")
enqueue("Orange")
enqueue("Peach")
isEmpty()

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Content									

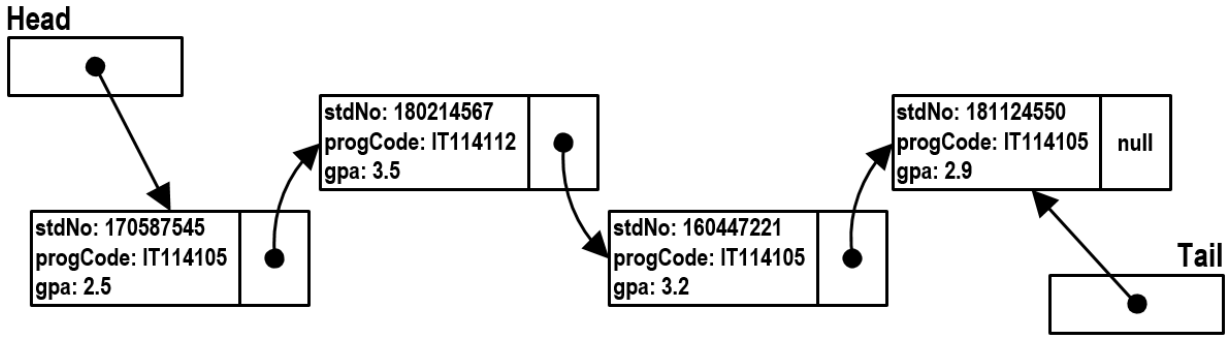
(e) A queue can also be implemented by using Linked List. State **ONE** advantage and **ONE** disadvantage of using Linked List instead of using Array. [4 marks]

Section B (60 marks)

This section contains 3 questions. Each question carries 20 marks.

Answer ALL questions.

B1 (a) The following diagram represents the Linked List used by IT discipline to maintain IT students' GPA in different programmes. The nodes are inserted in random order.




```
public class ListNode {
    int stdNo;  String progCode;  double gpa;
    ListNode next;
    public ListNode(int stdNo, String progCode, double gpa) { ... }
}

public class LinkedList {
    private ListNode head;  private ListNode tail;

    public LinkedList() { head = null;  tail = null; }
    public boolean isEmpty() { ... }
    public void addToHead(int stdNo, String progCode, double gpa) { ... }

    public ListNode removeFromHead() throws EmptyListException {
        /*** TO BE COMPLETED in B1 part a(i) ***/
    }

    public double averageGPA()  {
        /*** TO BE COMPLETED in B1 part a(ii) ***/
    }
}
```

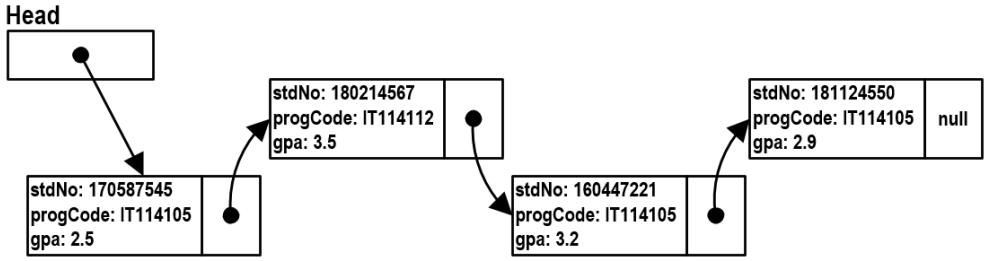
(i) Complete the method **removeFromHead ( )** of the LinkedList class. The method removes the first node from the beginning of the list and returns the deleted node to the caller. [4 marks]

(ii) Complete the method **averageGPA** ( ) of the LinkedList class. The method reports the average GPA of all the IT students in the list. [4 marks]

***** Question B1 continues in next page *****

***** Question B1 continues from previous page *****

(iii) The tail pointer in the linked list is removed and it is assumed that the list will never be empty. Describe the steps in adding a node to the end of the list when the list has no tail pointer. [4 marks]



(b) The list is now re-programmed to maintain the nodes in descending order of progCode. The following illustrates the comparator implementation.

```
public interface Comparator {
    public abstract boolean isLessThan (Object item1, Object item2);
    public abstract boolean isGreaterThan (Object item1, Object item2);
}

public class ProgCodeComparator implements Comparator {
        /*** TO BE COMPLETED in B1 part b(i) ***/
}

public class LinkedList {
    private ListNode head;  private ListNode tail;
    private Comparator comparator;

    public LinkedList(Comparator comparator) {
        head = null;  tail = null;  this.comparator = comparator;
    }
    public boolean isEmpty() { ... }
    public void addToHead(int stdNo, String progCode, double gpa) { ... }
    public void addToTail(int stdNo, String progCode, double gpa) { ... }

    public void insertInOrder (int stdNo, String progCode, double gpa) {
        ListNode newNode = new ListNode(stdNo, progCode, gpa);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            if (comparator.isGreaterThan(progCode, head.progCode)) {
                addToHead(stdNo, progCode, gpa);
            /*** TO BE COMPLETED in B1 part b(ii) ***/
        }
    }

public class ITDSRS {
    public static void main(String [ ] args) {
        LinkedList student = new LinkedList(new ProgCodeComparator());
        student.insertInOrder(170587545,"IT114105",2.5);
        ...
    }
}
```

***** Question B1 continues in next page *****

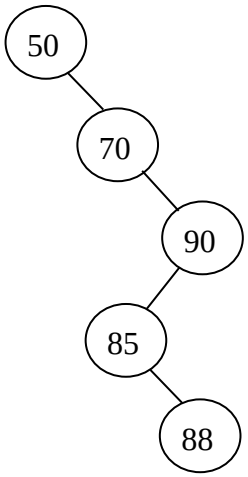
***** Question B1 continues from previous page *****

(i) Complete the **ProgCodeComparator** class. [2 marks]

[Recall that Java String compareTo method compares string values and returns 0, positive or negative integer depending on if the strings are equal, greater than or less than respectively.]

(ii) Complete the method **insertInOrder()** of the LinkedList class. The method uses the program code comparator to insert the node in descending order of program code. [6 marks]

B2 Given a Binary Search Tree (BST) which is inserted in the order of [50, 70, 90, 85, 88].



(a) List the traversal result of the above BST. [6 marks]

(i) Pre-order traversal
(ii) Post-order traversal

(b) Build an AVL Tree using the same data set and inserting sequence. Draw the result of the insertion for the AVL Tree **step by step**. [6 marks]

(c) Complete the following table about search performance for the BST and AVL tree with the above same set of data. [4 marks]

	Number of searching steps needed for Best Case	Number of searching steps needed for Worst Case
Binary Search Tree		
AVL Tree		

(d) Describe the general performances of the BST and AVL trees by evaluating the average steps based on the above set of data. The answer should compare by how many percentage one is better than the other. [4 marks]

B3 (a) Given the following sequence of numbers:

29, 18, 35, 68, 11, 23, 50, 39

(i) Perform an **Insertion Sort** on the above sequence of numbers. Show the result of **each pass** of the sorting. The first pass is shown below for your reference (number(s) in [ ] is/are sorted) :

29, 18, 35, 68, 11, 23, 50, 39
[18, 29], 35, 68, 11, 23, 50, 39 [3 marks]

(ii) What is the time complexity (in Big-O notation) for a **Merge Sort** on data in random order? [1 mark]

(iii) What is the worst time complexity (in Big-O notation) for a **Bubble Sort**? Give an example sequence of 6 data that Bubble sort will achieve the worst performance. [2 marks]

(iv) Perform a **Merge Sort** on the above sequence of numbers. Show your steps in partitioning and merging the list clearly. [4 marks]

(v) Copy and complete the following tables to your answer book to show how the **Quick Sort** algorithm (using the first number as the pivot) partitions the above sequence of number into two sub-lists. The first two steps are shown below for your reference:

index	**0 (pivot)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
data	29	18	35	68	11	23	50	39
Step 1	29	18						
Step 2	29	18	35					
Step 3								
Step 4								
Step 5								
Step 6								
Step 7								
Step 8 (swap pivot)								

storeIndex
1
2
2

[4 marks]

***** Question B3 continues in next page *****

***** Question B3 continues from previous page *****

(vi) You have a very long list of data which is almost sorted (best case) during the collection phase. Suggest which sorting algorithm you will choose to sort this sequence of numbers and explain why you make such choice. [2 marks]

(b) What are the time complexities (in Big-O notation) of the following algorithms?

(i)
```
int sum = 0;
for (int i=n; i>=1; i-=3)
   sum += i;
```
[1 mark]

(ii)
```
int sum = 0;
for (int i=1; i<=n; i++)
    for (int j=1; j<=n-i; j++)
        sum = sum + i*j;
```
[1 mark]

(iii)
```
int sum = 0;
int i = 1;
while (i<=n) {
    sum += i;
    i *= 4;
}
```
[2 marks]

******* END OF PAPER *******

Appendix I - Algorithm of Infix to Postfix Conversion

Algorithm infixToPostfix (exp)
Input: An infix expression exp
Output: The expression is printed in postfix form

```
BEGIN
    create stack S
    WHILE exp is not ended DO
        read next element elm from exp
        IF elm is an operand
            print elm
        ELSE IF elm is ( THEN
                S.push (elm)
            ELSE IF elm is ) THEN
                    DO
                        op = S.pop ()
                        IF op is not ( THEN print  op END IF
                    UNTIL op is (
                ELSE  // elm is an operator
                    WHILE stack S is not empty DO
                        op = S.pop()
                        IF op has higher precedence  than elm THEN
                            print op
                        ELSE
                            S.push (op)
                            BREAK
                        END IF
                    END WHILE
                    S.push (elm)
                END IF
            END IF
        END IF
    END WHILE

    WHILE stack S is not empty DO
        pop operator op from stack S
        print op
    END WHILE
END
```