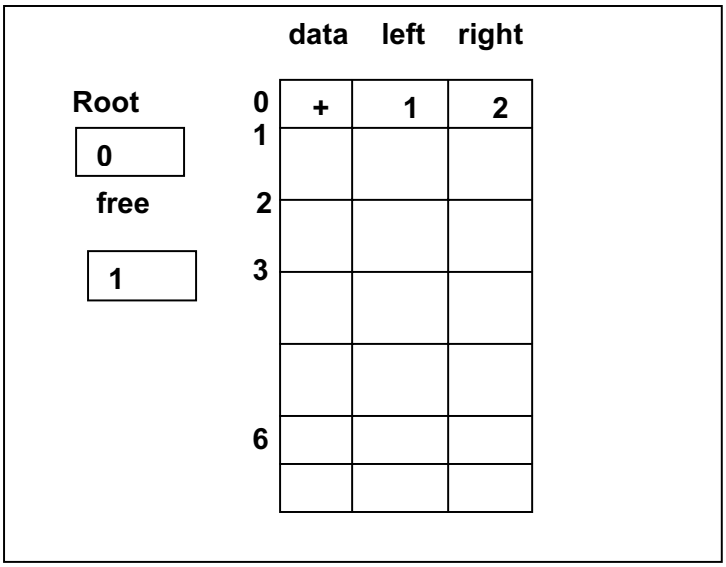
Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 1 compulsory

(a) Draw a diagram to illustrate an array –based implementation of a binary tree. [3]



1m for root and free
1m for array [can be any size]

(b) Discuss **one** difference between a reference-based and an array-based implementation when accessing an item. [2]

An array-based implementation uses random access to retrieve data [1] while a referenced-based implementation must transverse the dynamic links [1].

(c) How does an interface differ from a class? [2]

An Interfaces can only can have static final member variables [1], and all declared methods are without any body [1].

(d) Briefly explain why the Java language is platform independent. [2]

Java compiles a source program into bytecode [1] that any computer running a Java Virtual Machine (i.e. a Java bytecode interpreter) can execute[1].

(e) Identify the two conditions in which automatic conversion between types will take place. If not then a programmer must use the Java keyword **cast** to obtain a conversion between incompatible types. [2]

the two types are compatible [1]
the destination type is larger than the source type [1]

(f) Given the fragment of an employee class as below.

```
class employee {
        protected String name;
        protected int salary;}
```

(i) Create a subclass class called **manager** that inherits the employee class, and which has a private member variable **dept** of type **String**. [2]

```
class manager extends employee  [1 mark]{
        private String dept; [1 mark; no mark is awarded if missing
        private] }
```

(ii) Can a method, say **PrintDetails(),** in manger class directly access the member variables **name** and **salary**? Briefly explain your solution. [2]

Yes [1 mark]

Member variables name and salary are "protected access" which allows subclasses to access them. [1 mark]

(iii) Create a class named **LinkedEmp** that has two member variables:

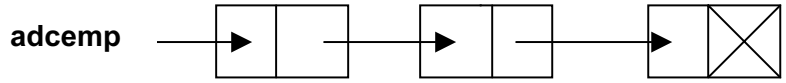
- EmpNode of type manager
- next for connecting to next employee on the list.

Both member variables are private access. [2]

```
class LinkedEmp {
                private manager EmpNode; [1 mark]
                private LinkedEmp next; [1 mark] }
```

(iv) Write a Java instruction that creates an instance of **LinkedEmp** called **abcemp**. [1]

LinkedEmp abcemp = new LinkedEmp( );



(v) Implement a method **PrintEmp( )** in **LinkedEmp** class that displays the linked list starting from the first node through to the last node. Assume the existence of a method **PrintDetails()** in manager class. The signature of the method is as follows: [4]

private void PrintEmp(LinkedEmp adcemp)

```
private void PrintEmp( LinkedEmp adcemp)
{   LinkedEmp tempPtr; [1 mark]
    for(  tempPtr = adcemp; tempPtr != null; tempPtr = tempPtr.GetNext())
                                                                    [2 marks]
                                tempPtr.PrintDetail(); [1 mark]  }
```

(vi) Write a Java instruction to link a new employee **newemp** to **adcemp** at the beginning of the linked list [1]

newemp.SetNext(adcemp); [1 mark]

(g) Given the fragment of **JavaStudent** class as follows:

```
class JavaStudent {
        private String Name;
        private int  test_mark;
public JavaStudent( String Sname) { Name = new String(Sname);
                                            test_mark =0; }
public void SetMark(int m) { test_mark = m; }
}
```

(i) Create a class called **JavaTest** that contains an array of 20 instances of **JavaStudent** class and an integer **StudNum** for the number of students. [3]

class JavaTest {
JavaStudent StudJava = new JavaStudent[20];
[2 marks; deduct 1m for each error up to a max 2m]
int StudNum; [1 mark] }

(ii) Implement a method **AdjustMark** for the **JavaTest** class that takes in two arguments **markadj** and **muchadj** of type integer. The method adds **muchadj** to students' marks that are equal to or less than **markadj**. Assume the existence of a method **int GetTestMark()** in **JavaStudent** class. [4]

```
public void AdjustMark( int markadj, int muchadj)
{    for (int x =0; x< StudNum ; x++) [1 mark]
                if (StudJava[x].GetTestMark() <= markadj)  [2 marks]
                        StudJava[x].SetMark(muchadj); [1 mark]}
```

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 2

The abstract data type *Stack* is a *last in first out* data structure. Data elements are added to and removed from the top of the stack.

Given the following definition of a **StackElement**:

```
public class StackElement{

        // data members
        private int data;
        private StackElement next;

        public StackElement (int x){
                data = x;
                next = null;
                }
        public int getData(){ return data; }

        public StackElement getNext(){ return next; }

        public void setNext(Node e){ next = e; }

}// end class
```

(a) Define a class called **DynamicStack** that has one data member called **top**, which is of **StackElement** type. Also, include a constructor that takes in no arguments and assigns the top to null. Ensure that the data members are only accessible by members of that class. [2]

```
public class DynamicStack{
        private StackElement top;                    [1]
        public DynamicStack(){                       [1]
                top = null;
                }
```

(b) Write a member function called **IsEmpty** that returns true when the dynamic stack is empty and false otherwise. [2]

```
public boolean IsEmpty(){                                    [1]
        return (top == null);                                [1]
        }
```

(c) Write a member function called **Push** that takes in an integer argument and adds it to the top of the Dynamic Stack. [4]

```
public void Push(int x){
        StackElement newNode = new StackElement(x);          [1]
        if IsEmpty()
                top = newNode;                               [1]
        else{
             newNode.setNext(top);                           [1]
             top = newNode;                                  [1]
          }
        }
```

(d) Write a function called **Pop** that takes in no arguments but removes and returns the data at the top of the dynamic stack. However, if the stack is empty then the return value should be –1; [4]

```
public int Pop(){
        int temp=-1;                                         [1]
        if (!IsEmpty()){                                     [1]
                 temp = top.getData();                       [1]
                 top = top.getNext();                        [1]
                }
        return temp;
        }
```

(e) Write the main routine that instantiates a *DynamicStack* object, adds two elements to the stack and then removes and displays the top element from the stack. [3]

```
public static void main(String args[ ]){
        DynamicStack stk = new DynamicStack();               [1]
        stk.push(5);
        stk.push(6);                                         [1]
        System.out.println("The removed element is " + stk.pop());   [1]
        }
```

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 3

(a) Briefly explain what is a two dimensional array. [2]

A two dimensional array stores information using rows and columns format. [1]
It requires the use of two indexes to access elements inside the array, [1]

(b) Write a class called **Teaching** which contains two members. The first member consists of the lecturer's name. The second member is a two dimensional array of integer type describing a teaching schedule. [3]

```
public class Teaching [1 mark]
{
        String lecturer_name; [1 mark]
        int[ ][ ]   Schedule ; [1 mark]
}
```

(c) Implement a constructor which receives a parameter **lecturer_name**. The purpose of this constructor is to initializes both members of the above class. Thus, it must initialize the lecturer_name and the schedule array. The schedule must be able to support 7 weeks with 3 teaching slots per week. [3]

```
Teaching (String  lecturer_name  ) [1 mark]
{
    this.lecturer_name = lecture_name;   [1 mark for correct use of this]
    Schedule = new int[7][3];   [1 mark]
}
```

(d) Implement an iterative method called **Find_Average** which returns the average number of hours spent by the lecturer. Each slot of the two-dimensional array will contain the number of hours needed to prepare for the lecture. [3]

```
double Find_Average (  )
{   int sum=0;
    for( int row=0; row < 7 ;row++)
        for( int col =0; col< 3; col++) [1 mark for both loops]
            sum += Schedule[row][col]                    [1 mark]
    return sum /21 ; [1 mark]
}
```

(e) Implement a method **CountHighProfile** which returns the number of teaching slots that have hours above the calculated average preparation. You have to use the method in part (d) as part of the solution. [4]

```
int CountHighProfile (  )
{       int count=0;
        for( int row=0; row < 7 ;row++)
                for( int col =0; col< 3; col++) [1 mark]
                        if(   Schedule[row][col] > FindAverage())          [1 mark]
                                count ++; [1 mark]
        return count; [1 mark]
}
```

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 4

(a) Briefly explain the properties of a queue and explain one of the error conditions that can occur when manipulating a queue. [3]

A Queue makes use of a First In First Out Protocol.
It requires the use of front and rear pointers.
To prevent an overflow condition – check the queue is full before adding an element to the stack.
To prevent an underflow condition – check the queue is empty before deleting is carried out.

Award one mark for each point. Maximum of 3 marks

(b) Declare a data structure called **QueueNode** which supports the dynamic implementation of a *stack*. Each node in the queue contains integer values. [3]

```
public class QueueNode {
        int data;
        QueueNode  next; }
```

Award one mark for each correct definition.

(c) State any three methods which can be found as routines to interface to the queue (of integers) abstract data type. You can assume that the front and rear pointers are stored in the **QueueList** class definition shown below. You are only asked to provide a signature specification for each queue method. [3]

```
public class QueueList {

QueueNode Front, Rear;
}

QueueList CreateQueue(QueueList Queue);

void Enqueue(QueueList Queue , int  element);

int Pop(QueueNode Queue) ;

boolean IsEmpty( QueueNode Queue);
```

Award one mark for each correct method, Maximum up to 3 marks.

(d) Implement a method **tripleItem** that receives a stack of **QueueList** as its argument. The purpose of this method is to triple the value of any element in the queue that has a value of less than 5 points. [6]

```
void tripleItem(QueueList Queue) [1 mark]
        QueueList temp;
        int item;
{
        temp := Queue.Front; [1 mark]
        while (temp != null ) [1 mark]
        {
                if ( temp.get_data( ) < 5)  [1 mark]
                        temp.get_data = temp.get_data() * 3; [1 mark]
                                temp = temp.get_next( ); [1 mark]
        }

}
```

Award marks for a correct algorithm and attempt at following the recursive structure of the datatype.

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 5

(a) Describe a binary search tree. [2]

A binary search tree is a sorted tree such that:
1) Left children of a parent node precede the parent node in sequence;
2) Right children of a parent node succeed the parent node in sequence;
The purpose of a binary search tree is to ensure efficient retrieval by enabling the search space to be halved at each comparison.

Award one mark for each correct point.

(b) Given the following formulas below, convert each formula into infix, post, and prefix notation.

(i) A * B / T – K [3]

Award 1 mark for each correct notation.

A * B / T - K (INFIX)
- / * A B T K (PRE FIX)
A B * T / K – (POSTFIX)

(ii) G * H + J / K + D * H [3]

G * H + J / K + D * H (INFIX)
+ + * G H / J K * D H (PRE FIX)
G H * J K / + D H * + (POSTFIX)

(c) (i) Define a class **BinaryTreeNode** which describes a dynamic data type for the branches and nodes in a binary tree. Each node in the binary tree contains data of integer type. [3]

```
public class BinaryTreeNode {

BinaryTreeNode Left; [1 mark]
BinaryTreeNode Right; [1 mark]
Int Data; [1 mark]
}
```

(ii) Write an iterative method, called **SearchItem**, the signature of which is given below, that takes in a binary search tree named **Root**, and an **item** of type integer. The method should perform a binary search on the tree. It will return true if the item is found and false otherwise. You can assume that the relevant selector methods, such as **get_item**, **getLeft**, **getRight**, are provided. [4]

boolean SearchIteml(BinaryTreeNode Root, int item)

A sample recursive definition of the method PreFixTraversal is as follows:

```
boolean SearchItem (BinaryTreeNode  Root, int item)

    { if (Root=null)
          return false;        [1 mark]
      else
          {
                if ((Root.get_item() == item) [1 mark]
                                          return true;
                else
                if (Root.get_item() < item) [1 mark]
                          SearchItem (Root.getLeft(), item);
                else
                          SearchItem (Root.getRight(), item);  [1 mark]
           };
    }
```

An iterative solution is equally valid.

- END OF PAPER -