CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Semester 1 Examinations 2017/18

Module Title: Linear Data Structures & Algorithms

Module Code:	COMP7035
School:	Maths and Computing
Programme Title:	BSc. (Honours) in Software Development BSc. (Honours) in Computer Systems BSc. (Honours) in Web Development BSc. in Software Development Higher Certificate in Software Development
Programme Code:	KSDEV_8_Y2 KDNET_8_Y2 KWEBD_8_Y2 KCOMP_7_Y2 KCOME_6_Y2
External Examiner(s):	Dr Luca Longo
Internal Examiner(s):	Dr Laura Climent Ms Triona McSweeney Dr Ignacio Castineiras
Instructions:	Answer **all** the questions.
Duration:	2 hours
Sitting:	Winter 2017

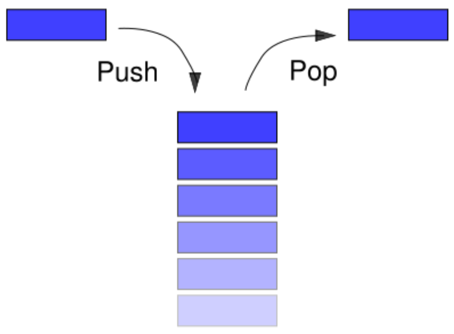
Requirements for this examination:

Note to Candidates: Please check the Programme Title and the Module Title to ensure that you have received the correct examination.
If in doubt please contact an Invigilator.

SECTION A: DATA STRUCTURES [50 Marks]

Background:

The stack is a fundamental data-structure used extensively in algorithm design and program implementation. At an abstract level it can be described very simply, as it only allows for addition (pushing) of new elements and removal (popping) of existing elements from the top of the stack. This description can be abbreviated to LIFO, which stands for Last-In-First-Out.



Problem Specification:

The ADT Stack has the following operations:

• The operation push(int item) places an integer item onto the top of the stack (if there is space for it). If there is not space, it prints on the screen an error message informing that the stack is full.

• The operation pop() removes the top integer item from the stack (if the stack is non-empty) and returns that item. If the stack is empty, it prints on the screen an error message informing that the stack is empty.

A.1 Exercises for the static implementation of the ADT Stack:

Consider that you can only use arrays as data structures for the following exercises. You can not use the ArrayList Java package neither the LinkedList Java package. Implement in Java a class MyStaticStack with:

(i) The necessary attributes and the constructor of the class. **[10 Marks]**
(ii) The pop operation. **[10 Marks]**

```
public class MyStaticStack {

       //--------------------------------------------------
       // Attributes
       //--------------------------------------------------
       /// CODE MISSING: YOU MUST DECLARE THE NECESSARY ATTRIBUTES

       //------------------------------------------------------------------
       // Constructor
       //------------------------------------------------------------------
       /// CODE MISSING: YOU MUST IMPLEMENT THE CONSTRUCTOR

       //------------------------------------------------------------------
       // Pop Operation
       //------------------------------------------------------------------
       /// CODE MISSING: YOU MUST IMPLEMENT THIS OPERATION
}
```

A.2 Exercises for the dynamic implementation of the ADT Stack:

Consider that you can only use objects of type MyNode for representing your items in the stack (see the code of the class MyNode below). You can not use the ArrayList Java package neither the LinkedList Java package. Implement in Java a class MyDynamicStack with:

(iii) The necessary attributes and the constructor of the class. **[10 Marks]**
(iv) The push operation. **[10 Marks]**

```
public class MyDynamicStack {

             //--------------------------------------------------
       // Attributes
       //--------------------------------------------------
       /// CODE MISSING: YOU MUST DECLARE THE NECESSARY ATTRIBUTES


       //------------------------------------------------------------------
       // Constructor
       //------------------------------------------------------------------
       /// CODE MISSING: YOU MUST IMPLEMENT THE CONSTRUCTOR

       //------------------------------------------------------------------
       // Push Operation
       //------------------------------------------------------------------
       /// CODE MISSING: YOU MUST IMPLEMENT THIS OPERATION

}

/**
* This class models the concept (a type of objects) of a single linked node<br>.
*/
public class MyNode {

       //--------------------------------------------------
       // Attributes
       //--------------------------------------------------
       /**
       * @param info: It represent the element contained in MyNode.
       * @param next: It represents the MyNode placed next (referenced) from this object.
       */
       private int info;
       private MyNode next;

       //--------------------------------------------------
       // Constructor
       //--------------------------------------------------
```

```
        public MyNode(int i, MyNode n){
                this.info = i;
                this.next = n;
        }

        //-----------------------------------------------------
        // Get methods
        //-----------------------------------------------------
        public int getInfo(){
                return this.info;
        }

        public MyNode getNext(){
                return this.next;
        }

        //-----------------------------------------------------
        // Set methods
        //-----------------------------------------------------
        public void setInfo(int i){
                this.info = i;
        }

        void setNext(MyNode n){
                this.next = n;
        }

}
```

A.3 Exercise for comparing both implementations:

(v) Explain the advantages and disadvantages of both implementations (dynamic and static) in terms of: memory usage, time efficiency of the pop operation and time efficiency of the push operation. After such explanation, answer the question: Which implementation (static or dynamic) is better for implementing the ADT Stack described in this exercise?

[10 Marks]

SECTION B: ALGORITHMS [50 Marks]

(i) Write the general schema followed by any divide and conquer-based algorithm. Include a brief description of the components *divide&conquer, small*, *solve*, *divide*, *map* and *reduce*) of such schema.

[10 Marks]

(ii) Implement the divide and conquer algorithm *isInList* which, given the integer element *e* and the list *m*, uses recursion to return whether the element is in the list.

```
public boolean isInList(int e, MyDynamicList m){
                        ???
}
```

Note: The following operations are available to be used by MyDynamicList:

```
public int length();
public int  getElement(int index);
public void addElement(int index, int  element);
public void removeElement(int index);
```

[10 Marks]

(iii) Describe the concept of *selection function*/*get candidate* followed by any greedy algorithm.

[10 Marks]

(iv) Given a Greedy algorithm optimally solving the following Knapsack problem:

- Knapsack capacity: m
- Number of items: n
- Items weight: [w0, w1, …, wn-1]
- Objective: Maximise the amount of items being picked.

Implement the method *selection function*/*get candidate* of such this Greedy algorithm.
The input remainingItems is the list of items that still have not been considered.
The method returns the index of the selected item.
Note: The method must use a policy/strategy leading to an optimal solution.

```
public int selectionFunction(MyDynamicList remainingItems){
                    ???
}
```

[10 Marks]

(v) Briefly compare and contrast the advantages and disadvantages of the following sorting algorithms:

- bubbleSort.
- mergeSort.

[10 Marks]