# _LITfinalLOGO

# **SUMMER EXAMINATIONS 2015**

**Tuesday, 12th May 2015, 14.30 p.m. – 16.30 p.m.**

**KSDEM\_8\_Y2**

**Course:** Bachelor of Science (Hons) in Software Development

**Year:** Two

**Subject:** Data Structures & Algorithms

**Time Allowed:** 2 Hours

**Instructions:**

**1.** Answer any **THREE (3)** Questions.

1. All questions carry equal marks
2. The examination paper is marked out of 75 marks.
3. Start each question on a new page.
4. Write the question number at the top of each page.
5. Circle the numbers of the questions you answer at the

front of your answer book.

**Additional Attachments or Exam Material to accompany this paper:**

**A.** None

**Internal Examiners: External Examiners:**  Mr. Des O’Carroll Mr Brian Gillespie

**Q.1.**

**(a)** Define what is meant by a Stack data structure.

**(5 marks)**

**(b)** Given the following class declaration for a Stack:

template <class ItemType>

struct NodeType;

template<class ItemType>

class StackType

{

public:

StackType();

~StackType();

void MakeEmpty();

bool IsFull() const;

bool IsEmpty() const;

**void Push(ItemType newItem);**

**void Pop(ItemType& item);**

private:

NodeType<ItemType>\* topPtr;

};

implement the following member functions:

(i) Push()

// Function: Adds newItem to the top of the stack.

// Pre: Stack has been initialized and is not

// full.

// Post: newItem is at the top of the stack.

(ii) Pop()

// Function: Removes top item from the stack and

// returns it in item.

// Pre: Stack has been initialized and is not

// empty.

// Post: Top element has been removed from

// stack.

// item is a copy of the removed item.

**(10 marks)**

**(c)** Consider the following pseudo code. Assume that IntQueue is an integer queue. What does the function fun do?

void fun(int n)

{

    IntQueue q = new IntQueue();

    q.enqueue(0);

    q.enqueue(1);

    for (int i = 0; i < n; i++)

    {

        int a = q.dequeue();

        int b = q.dequeue();

        q.enqueue(b);

        q.enqueue(a + b);

        print(a);

    }

}

1. Prints numbers from 0 to n-1
2. Prints numbers from n-1 to 0
3. Prints first n Fibonacci numbers
4. Prints first n Fibonacci numbers in reverse order.

**(10 marks)**

**(Total 25 Marks)**

**Q.2.**

**(a)** Explain what is meant by

**(i)** base case

**(ii)** general (or recursive) case

**(iii)** tail recursion

**(iv)** procedure activation record (also known as **stack frame**)

**(8 marks)**

**(b)** Given the following recursive implementation of the QuickSort algorithm, implement the **Split** function:

void QuickSort (ItemType values[], int first, int last)

{

int splitPt1, splitPt2;

if (first < last)

{

**Split(values, first, last, splitPt1, splitPt2);**

//values[first]..values[splitPt2]<= splitVal

//values[splitPt1]..values[last]>= splitVal

QuickSort(values, first, splitPt2);

QuickSort(values, splitPt1, last);

}

}

Additional Information: QuickSort( ) calls function Split( ) to partition the array values between index **first** and index **last** into two segments, one containing elements less than or equal to a **splitVal**, the other segment containing elements greater than or equal to **splitVal**.

On return, **splitPt2** is the upper bound of the left segment and **splitPt1** is the lower bound of the right segment.

Implement function **Split( )** given the following prototype:

**void Split (ItemType values[], int first, int last, int & splitPt1, int & splitPt2)**

**(10 marks)**

**(c)** Trace through the following data using the **QuickSort** and **Split** functions procedure (from part B of this question):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
| 25 | 40 | 33 | 23 | 19 | 46 | 8 | 44 |

**(7 marks)**

**(Total 25 Marks)**

**Q.3.**

**(a)** Give the recursive definitions for the following:

1. a binary tree,
2. a binary search tree.

**(5 marks)**

**(b)** The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the postorder traversal sequence of the same tree?  
(i) 10, 20, 15, 23, 25, 35, 42, 39, 30  
(ii)15, 10, 25, 23, 20, 42, 35, 39, 30  
(iii) 15, 20, 10, 23, 25, 42, 35, 39, 30  
(iv) 15, 10, 23, 25, 20, 35, 42, 39, 30

**(10 marks)**

**(c)** Assuming the following class declaration for a Binary Search Tree ADT:

template <class ItemType>

struct TreeNode

{

ItemType info;

TreeNode \*left;

TreeNode \*right;

};

// Assume “<” and “==” operators have been overloaded for

// the class ItemType

template <class ItemType>

class TreeType

{

public:

TreeType();

~TreeType();

TreeType (const TreeType<ItemType>& originalTree);

// copy constructor

void operator=(const TreeType<ItemType>& originalTree);

void MakeEmpty();

bool IsEmpty() const;

bool IsFull() const;

int NumberOfNodes() const;

void RetrieveItem(ItemType &item, bool &found);

void InsertItem(ItemType item);

void DeleteItem(ItemType item);

void PrintTree(ostream &outFile) const;

private:

TreeNode<ItemType> \*root;

};

Implement the following recursive member functions, including any functions which they, in turn, call:

(i) void DeleteItem(ItemType item);

// Function: Deletes the item whose key matches

// item's key.

// Pre: Key of item has been initialized.

// One and only one item in the tree has a key

// matching item's key.

// Post: No element in tree has key matching item's // key **(10 marks)**

**(Total 25 Marks)**

**Q.4.**

**(a)** Define what is meant by:

(i) a heap

(ii) a priority queue

**(5 marks)**

**(b)** The priority queue algorithms on heaps all work by first making simple

structural modification which could violate the order property of the

heap, then travelling through the heap modifying it to ensure that the

heap order property is satisfied everywhere, using either ReHeapUp()

or ReHeapDown().

Assuming the following declarations:

template <class ItemType>

// Assumes ItemType is either a built-in simple type or a

// class with overloaded relational operators.

struct HeapType

{

**void ReheapDown (int root, int bottom);**

void ReheapUp (int root, int bottom);

ItemType \* elements; // Array to be allocated

// dynamically

int numElements;

};

template<class ItemType>

class PQType

{

public:

PQType(int);

~PQType();

void MakeEmpty();

bool IsEmpty() const;

bool IsFull() const;

void Enqueue(ItemType newItem);

**void Dequeue(ItemType& item);**

private:

int numItems;

HeapType<ItemType> items;

int maxItems;

};

**(i)** Implement DeQueue()

**(ii)** Implement ReHeapDown().

**(20 marks)**

**(Total 25 Marks)**