**Lab Session-1**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-4 Implement different searching and sorting algorithms.**

**Program Title:** Implement Linear and Binary Search.

**Pre Lab:**

1. Write an algorithm for reading a string from the keyboard and count number of vowels in the given string.
2. Write an algorithm to read n integer values through keyboard, find the middle element in the given list. Also find the smallest element in the list from the starting to middle position and find the big element from the mid+1 location to ending of the list.

**In Lab:**

1. If you have the attendance statement of an examination in an array, read student roll number from the keyboard as an integer, find the student attended for the exam or not using linear search.(assume the statement contains roll numbers in random)

**Test Case:**

**Input:** No of students attended: 10

15, 33, 45, 6, 79, 2, 32, 53, 99, 48

**Case 1:** Student roll number: 79

**Output:** Attended for the Exam

**Case 2:** Student roll number: 13

**Output:** Absent for the Exam

1. Assume result is an array contains list of student roll numbers those who clear or pass the Data structures course in the ascending order. Read the student roll number as an integer and check the result of the student whether he/she pass or fail using binary search technique.

**Test Case:**

**Input:** No of students attended: 10

2, 6, 15, 32, 33, 45, 48, 53, 79, 99

**Case 1:** Student roll number: 79

**Output:** Data Structures Course Completed

**Case 2:** Student roll number: 13

**Output:** Data Structures Course Not Completed

**Post Lab:**

1. Compare Linear Search and Binary Search
2. Calculate the time complexity of Linear Search
3. Calculate the time complexity of Binary Search
4. Define Time Complexity and Space Complexity.

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-2**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-1 Apply linear data structures to different applications.**

**Program Title:** Implement Stack Operations

**Pre Lab:**

* 1. Define Data structure and Abstract Data Type (ADT)
  2. Define Stack. List the Operations of Stack
  3. How can you find the top of the element?
  4. Write the condition for stack is full and stack is empty.

**In Lab:**

1. Develop a program to implement Stack Operations using arrays.

**Test Case:**

**Input:** Stack Maximum Size : 5

**List of Operations**

**1.** PUSH

2. POP

3. Display Stack elements

4. Stack is Full

5. Stack is empty

**Case 1:** Initial Stack is empty, if you are trying to perform POP Operation

**Output:** Stack Underflow

**Case 2:** If your stack contains three elements, you are trying to perform PUSH

Operation.

**Output:** Top is updated by one and new element is inserted…

**Case 3:** If your stack contains four elements, you are trying to perform POP

Operation.

**Output:** Top reduced by one and top element is deleted from the stack

**Post Lab:**

* 1. After Completion of the following Stack Operations what are the element on the stack (assume stack size is five)

PUSH (Stack, ’V’)

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

PUSH (Stack, ’R’)

PUSH (Stack, ’S’)

PUSH (Stack, ’E’)

PUSH (Stack, ’C’)

POP (Stack)

POP (Stack)

POP (Stack)

PUSH (Stack, ’C’)

Initially stack is empty

PUSH (Stack, ’E’)

PUSH (Stack, ’S’)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

* 1. If your stack is empty, you are trying to delete an element from the stack what happened?
  2. If your stack is full, you are trying to perform PUSH() operation then what happened?

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-3**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-1 Apply linear data structures to different applications.**

**Program Title:** Implement Stack Applications

**Pre Lab:**

1. Write an algorithm for push and pop operations of the stack.
2. List out real time applications for Stack.
3. Define recursion.
4. What is an expression? What are the different types of expressions?
5. What is need of converting infix expression into postfix expression?

**In Lab:**

1. Develop a program to implement Infix to Postfix Conversion (applications of Stack)

**Test case:**

**Input :** a + b

**Output:** a b +

**Input :** a – b \* c

**Output:** a b c \* -

1. Develop a program to implement Postfix Expression Evaluation (applications of Stack)

**Test case:**

**Input :** 4 5+

**Output:** 9

**Input :** 9 2 3 \* -

**Output:** 3

**Post Lab:**

1. Write the postfix expression for the following infix expression
   1. **a+b\*c/d-e+f**
   2. **a\*(b-c\*d)+(e/f)-g**
2. Evaluate the following postfix expression
3. **3 5 \* 4 6 - / 2 +**
4. **A B – C \* D +**

A=4 B=2 C=6 D=3

1. Develop a program for Towers of Hanoi Problem using stack.

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-4**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-1 Apply linear data structures to different applications.**

**Program Title:** Implement Queue Operations

**Pre Lab:**

1. Define Queue
2. List the Operations of Queue
3. Write real time applications to Queue data structures.

**In Lab:**

1. Develop a program to implement Queue Operations using arrays.

**Input:** Queue Maximum Size : 4

**List of Operations**

**1.** Enqueue

2. Dequeue

3. Display Queue elements

4. Queue is Full

5. Queue is empty

**Case 1:** Initial Queue is empty, if you are trying to perform Enqueu Operation

**Output:** front and rear pointers\variables updated by one

**Case 2:** If your Queue contains three elements, you are trying to perform

Enqueue Operation.

**Output:** Rear position is updated by one and new element is inserted…

**Case 3:** If your Queue contains four elements, you are trying to perform

Dequeue Operation.

**Output:** Front reduced by one and one element is deleted from the Queue

**Post Lab:**

1. Compare Stacks and Queues
2. Write the condition to check whether queue is full or not.
3. After Completion of the following Queue Operations what are the element on the Queue

(Assume queue size is 10)

ENQUEUE (Queue, ’V’)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

ENQUEUE (Queue, ’R’)

ENQUEUE (Queue, ’S’)

DEENQUEUE (Queue)

ENQUEUE (Queue, ’E’)

Initially Queue is empty

Front =0 rear= 0

DE ENQUEUE (Queue)

ENQUEUE (Queue, ’C’)

ENQUEUE (Queue, ’C’)

DE ENQUEUE (Queue)

ENQUEUE (Queue, ’E’)

ENQUEUE (Queue, ’S’)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Front = rear=

Front = rear=

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Front = rear=

Front = rear=

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Front = rear=

Front = rear=

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Front = rear=

Front = rear=

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Front = rear=

Front = rear=

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-5**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-1 Apply linear data structures to different applications.**

**Program Title:** Implement Circular Queue Operations

**Pre Lab:**

1. Define Circular Queue
2. What is the drawback of Linear Queue?
3. List the applications of circular queue

**In Lab:**

1. Develop a program to implement Circular Queue Operations using arrays.

**Input:** Circular Queue Maximum Size : 5

**List of Operations**

**1.** Enqueue

2. Dequeue

3. Display Queue elements

4. Queue is Full

5. Queue is empty

**Case 1:** If your Circular Queue contains four (index 1 to index 5) elements, you

are trying to perform Enqueue Operation.

**Output:** Update rear position to index 0 and new element is inserted…

**Case 2:** If your Queue contains four elements, you are trying to perform

Dequeue Operation.

**Output:** Update front position and one element is deleted from the Queue

**Post Lab:**

* 1. If circular queue implemented using an arrays, size of the circular queue is MAX\_SIZE. Write the conditions for circular queue getting full and empty?
  2. A data structure in which elements can be inserted or deleted at/from both the ends but not in the middle is?

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session- 6**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-2 Solve problems using linked list.**

**Program Title:** ImplementSingly Linked List Operations

**Pre Lab:**

1. What is self referential Structure?
2. Write about Dynamic memory management functions.
3. Define Structure variable with the name college and the structure members are department name, department code and number of students in each department. Define one structure type (and Pointer) variable to read and write the structure members.

**In Lab:**

Develop a Program to implement Singly Linked Operations

a) Insertion at the end b) Delete at the end c) Display the list elements

**Test Case:**

NULL

Head

* + 1. Insert the new node (10) in the empty List

2024

Head

|  |  |
| --- | --- |
| **10** |  |

* + 1. Insert the new node (20) at the end of the list

2024

Head

|  |  |  |
| --- | --- | --- |
| **10** | **20** |  |

* + 1. Insert the new node (30) at the end of the list

2024

Head

|  |  |  |  |
| --- | --- | --- | --- |
| **10** | **20** | **30** |  |

* + 1. Delete a node from the end of the list

|  |  |  |
| --- | --- | --- |
| **10** | **20** |  |

2024

Head

* + 1. Delete a node from the end of the list

2024

Head

|  |  |
| --- | --- |
| **10** |  |

* + 1. Delete a node from the end of the list

NULL

Head

* + 1. Delete a node from the end of the list

List is empty

**Post Lab:**

1. What are the advantages and disadvantages of linked list?
2. What is the time complexity for inserting an element into the Singly linked list (end of the list)?
3. Write a code for inserting an element into the Singly linked list at the specific location.

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-7**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-2 Solve problems using linked list.**

**Program Title:** ImplementStack and Queue Operations using Linked List

**Pre Lab:**

1. Develop a function to insert an element in the Singly linked list at the beginning position.
2. Develop a function to delete an element from the Singly linked list. (from beginning position)

**In Lab:**

Develop a Program to implement Stack Operations using Linked List

**Test Case:**

3024

Head

|  |  |  |  |
| --- | --- | --- | --- |
| **30** | **20** | **10** |  |

1. Perform PUSH Operation (40)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **40** | **30** | **20** | **10** |  |

4024

Head

1. Perform PUSH Operation (50)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **50** | **40** | **30** | **20** | **10** |  |

5024

Head

1. Print Top element in the stack

5024

Head

**Top of the stack is 50**

(Print head node value (i.e value @ 5024))

1. Perform POP Operation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **40** | **30** | **20** | **10** |  |

4024

Head

**50 is deleted from the stack**

1. Perform POP Operation

|  |  |  |  |
| --- | --- | --- | --- |
| **30** | **20** | **10** |  |

3024

Head

**40 is deleted from the stack**

Develop a Program to implement Queue Operations using Linked List

**Test Case:**

3024

Rear

1024

Front

|  |  |  |  |
| --- | --- | --- | --- |
| **10** | **20** | **30** |  |

1. Perform ENQUEUE Operation (40)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **10** | **20** | **30** | **40** |  |

4024

Rear

1024

Front

1. Perform ENQUEUE Operation (50)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **10** | **20** | **30** | **40** | **50** |  |

5024

Rear

1024

Front

1. Print Front and rear elements in the Queue

5024

Rear

1024

Front

Front is 10 and Rear is 50

(i.e value @ 1024 and value @ 5024)

1. Perform DEQUEUE Operation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **20** | **30** | **40** | **50** |  |

5024

Rear

2024

Front

**10 is deleted from the Queue**

1. Perform POP Operation

|  |  |  |  |
| --- | --- | --- | --- |
| **30** | **40** | **50** |  |

5024

Rear

3024

Front

**40 is deleted from the stack**

**Post Lab:**

* 1. Write an algorithm to implement the stack operations using Queue.
  2. Write an algorithm to compare two stacks.
  3. How will you traverse linked list in reverse order

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-8**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-2 Solve problems using linked list.**

**Program Title:** Implement **Doubly** Linked List Operations

**Pre Lab:**

1. What is Doubly linked list?
2. How to represent a node in Doubly linked list?
3. Differentiate between Singly and Doubly linked list.

**In Lab:**

Develop a Program to implement Doubly Linked Operations

* + - 1. Insertion at the end b) Delete from the end c)Display the list elements

**Test Case:**

NULL

Head

1. Insert the new node (10) in the empty List

1024

Head

|  |  |  |
| --- | --- | --- |
|  | **10** |  |

1. Insert the new node (20) at the end of the list

1024

Head

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **10** |  |  |  | **20** |  |

1. Insert the new node (30) at the end of the list

1024

Head

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **10** |  |  |  | **20** |  |  |  | **30** |  |

1. Delete a node from the end of the list

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **10** |  |  |  | **20** |  |

1024

Head

1. Delete a node from the end of the list

1024

Head

|  |  |  |
| --- | --- | --- |
|  | **10** |  |

1. Delete a node from the end of the list

NULL

Head

1. Delete a node from the end of the list List is empty

**Post Lab:**

1. What are the advantages of Doubly linked list compare with Singly linked list (show with diagrammatic form)?
2. Draw Singly and Doubly linked list for the following Operations

Insert(list, ‘A’)

Insert(list, ‘E’)

Insert(List,’I’)

Delete(List)

Insert(List,’O’)

Delete(List)

Insert(List,’U’)

1. What is the time complexity for finding the element in the Doubly linked list?

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-9**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO-3 Implement operations on binary trees and binary search trees.**

**Program Title:** Implement **Binary Search Tree** Operations

**Pre Lab:**

1. Write Tree terminology
2. How can we represent trees in computer memory?
3. Define Binary Search. What are the different ways we can traverse Binary Search Tree?
4. What are the properties of Binary Tree.
5. In a Full Binary Tree, how many nodes exists of a given height ’h’?

**In Lab:**

Develop a program to implement Binary Tree with following Operations

a.    BST - Find, findmax, findmin, insert

b.    BST- Delete, Inorder, Preorder, Postorder traversal

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary Search Tree Operations** | | | |
| **BST** | **Insert(7)** | **Insert(19)** | **Insert(29)** |
|  |  |  |  |
| 31  16  45  24  19  29  **Delete(7)** | 31  16  45  29  19  **Delete(24)** | 45  16  29  19  **Delete(31)** | 45  16  19  **Delete(29)** |
|  |  |  |  |
|  | **Find Minimum Value**: 16  **Find Maximum Value:** 45  **In order Traversals :** 16 24 31 45  **Pre Order Traversals:** 31 16 24 45  **Post Order Traversals:** 24 16 45 16 | | |

**Post Lab:**

1. You are going to delete a node from the Binary search tree, if the selected node has two children’s what we need to do?
2. Define Complete Binary Tree and Full Binary Tree.
3. List different types of tree traversing methods.
4. What are the applications of Binary Search Tree (BST)?
5. Construct a binary tree for the given In-order and Pre order traversal

In Order traversal: D B H E A I F J C G

Pre Order reversal: A B D E H C F I J G

1. Construct a Binary Search Tree for the following datasets
2. Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov and Dec
3. Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday

**(Faculty Use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**(Faculty use only)**

**Lab Session-10**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO\_4 Implement different searching and sorting algorithms.**

**Program Title:** ImplementBubble, selection and Insertion sorting techniques

**Pre Lab:**

Write an algorithm to find the biggest and smallest number in the given array. Exchange biggest and smallest numbers in the array and print the final array to the screen.

**In Lab:**

1. Develop a program to implement Bubble sort technique

**Input:**

Number of elements: 10

Dataset: 15, 19, 2, 78, 53, 48, 32, 64, 40, 9

1. Develop a program to implement Selection Sort technique

**Input:**

Number of elements: 10

Dataset: 64, 1, 32, 17, 45, 84, 13, 76, 50, 2 9

1. Develop a set of programs to implement Insertion Sort technique

**Input:**

Number of elements: 10

Dataset: 56, 21, 42, 17, 25, 38, 62, 94, -1, 79

**Post Lab:**

1. Write the time complexity of selection sort and Bubble sort
2. Compare bubble sort and selection sort
3. What is time complexity of Insertion sort?

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator

**Lab Session-11**

**Date of Session: Time of the Session** : \_\_\_\_\_**to\_\_**\_\_\_

**CO\_4 Implement different searching and sorting algorithms.**

**Program Title:** ImplementQuick Sort and Merge sort techniques

**Pre Lab:**

Develop a Program to merge two different arrays

Example:

**Input:**

int a[5] ={1, 2, 3, 4,5,0} ;

int b[5] ={6, 7, 8, 9, 10};

**Output: 1** c[10]={1, 2, 3, 4, 5, 0, 6, 7, 8, 9, 10}

**Output: 2** c[10]={ 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 0}

**In Lab**

Develop a program to implement Merge Sort technique (Divide and conquer method)

**Input:**

Number of elements: 10

Dataset: 315, 219, 2, 678, 153, 248, 332, 464, 540, 49

Develop a program to implement Quick sort technique (Divide and conquer method)

**Input:**

Number of elements: 10

Dataset: 5, 9, 2, 7, 3, 8, 1, 6, 0, 4,

**Post Lab:**

1. Write the time complexity of Quick sort

Best Case :

Average Case :

Worst Case :

1. Write the time complexity of Merge sort

Best Case :

Average Case :

Worst Case :

1. Write the recurrence for Merge sort
2. Write the recurrence for Quick sort

**(Faculty use only)**

Comments of the Evaluator:

**Evaluator’s Observation**

Marks Secured: \_\_\_\_\_\_\_ out of \_\_\_\_\_\_

Signature of Evaluator