**S. Y. B. C. A. (Science)**

**SEMESTER III**

**Lab Course – I**

**Data Structures**

**Workbook**

**Name: --------------------------------------**

**Roll No.:------------------------------------**

**Academic Year: --------------------------**

**Introduction**

1. **About the Workbook:**

This workbook is intended to be used by SYBCA (Science) students for the Data Structure (DS) assignments in Semester–III. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus.

1. **The objectives of this Workbook are:**
   * 1. Defining the scope of the course.
     2. To have continuous assessment of the course and students.
     3. Providing ready reference for the students during practical implementation.
     4. Provide more options to students so that they can have good practice before facing the examination.
     5. Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.
2. **How to use this Workbook:**

The Labbook is divided into eight assignments. Each DS assignment has several SETs. It is mandatory for students to complete all the SET in given slot.

1. **Instructions to the students:**

Please read the following instructions carefully and follow them.

Students are expected to carry this workbook every time they come to the lab for practical.

Students should prepare for the assignment by reading the relevant material which is mentioned in ready reference.

Instructor will specify which problems to solve in the lab during the

allotted slot and student should complete them and get verified by the instructor. However, student should spend additional hours in Lab and at home to cover all workbook assignments if needed.

Students will be assessed for each assignment on a scale from 0 to 5

|  |  |
| --- | --- |
| Not done | 0 |
| Incomplete | 1 |
| Late Complete | 2 |
| Needs improvement | 3 |
| Complete | 4 |
| Well Done | 5 |

**5.Instruction to the Instructors:**

Make sure that students should follow above instructions.

Explain the assignment and related concepts using white board if required or by demonstrating the software.

Give specific input to fill the blanks in queries which can vary from student to student.

Evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.

The value should also be entered on assignment completion page of the respective Lab course.

1. **Instructions to the Lab administrator:**

You have to ensure appropriate hardware and software is made available to each student.

The operating system and software requirements on server side and also client side areas given below:

Server and Client Side-(Operating System) Fedora Core Linux

**Table of Contents for Section-I**

Assignment No. 1 Sorting Techniques (Non-Recursive)

* Bubble sort, Insertion sort

Assignment No. 2 Sorting Techniques (Recursive)

* Quick sort, Merge sort

Assignment No. 3 Assignment No. 4

Searching Techniques Linear search, binary search Linked List

Assignment No. 5

Stack

Assignment No. 6

Queue

Assignment No. 7

Trees

Assignment No. 8

Graph

**Assignment Completion Sheet**

Lab Course I

Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.** | **Assignment Name** | **Marks** | **Teachers** |
| **No** |  | **(Out of 5)** | **Sign** |
|  |  |  |  |
| 1 | Sorting Techniques (Non Recursive) |  |  |
|  |  |  |  |
| 2 | Sorting Techniques (Recursive) |  |  |
|  |  |  |  |
| 3 | Searching Techniques |  |  |
|  |  |  |  |
| 4 | Linked List |  |  |
|  |  |  |  |
| 5 | Stack |  |  |
|  |  |  |  |
| 6 | Queue |  |  |
|  |  |  |  |
| 7 | Trees |  |  |
|  |  |  |  |
| 8 | Graph |  |  |
|  |  |  |  |
| Total (Out of 40 ) | |  |  |
|  | |  |  |
| Total (Out of 10) | |  |  |
|  |  |  |  |

**Assignment No : 1**

**Sorting Techniques (Non Recursive)**

**SORTING**

* Sorting means arranging a set of data in some given order or ordering a list of items.

‘Or’

Sorting is a process of ordering a list of elements in either ascending or descending order.

* List is a collection of record each contains one or more fields. The field which contains unique value for each record is called key field.
* Definition:-
  + Sorting is the operation of arranging the records of a table according to the key value of each record
  + e.g. consider a telephone directory which consists of 4 field phone number, name, address, pin code .
  + So a large data is maintained in the form of records. If we want to search a phone no and name it should be alphabetically sorted then we can search easily. It would be very difficult if records were unsorted.
* The sorting algorithm are divided into two categories

1) Internal sorting-

Sorting is done on data which is sorted in main memory.

2) External sorting –

Sorting is done on data which is stored on auxiliary storage device.

e.g. hard disk, floppy, tape etc.

**BUBBLE SORT**

* + This is one of the simplest and most popular sorting methods. The basic idea is to pass through the file sequentially several times.
  + In each pass we compare successive pairs of elements(x[i] with x[i+1]) and interchange the two if they are not in the required order.
  + One element is placed in its correct position in each pass.
  + In first pass, the largest element will sink to the bottom, second largest in the second pass and so on. Thus a total of n-1 passes are required to sort n keys
  + Efficiency of bubble sort

Worst case = average case = best case complexity =O(n2)

**INSERTION SORT**

* + Insertion sort inserts each item into its proper place in the final list
  + In this the first iteration starts with comparison of 1st element with 0th
  + In second iteration 2nd element is compared with the 0th and 1st element and so on.
  + In every iteration an element is compared with all elements
  + The basic idea of this method is to place an unsorted element into its correct position in a growing sorted list of data. We select one element from the unsorted data at a time and insert it into its correct position in the sorted set.

* 1. E.g. in order to arrange playing cards we pick one card at a time and insert this card hold in the hand.
* Efficiency of insertion sort Best case = O(n)

Worst case = average case = O(n2)

**SET A:**

* 1. Write a C program to accept and sort n elements in ascending order by using bubble sort.
  2. Write a C program to accept and sort n elements in ascending order by using insertion sort.

**SET B:**

1. Write a C program to read the data from the file “employee.txt” which contains empno and empname and sort the data on names alphabetically (use strcmp) using Bubble Sort.
2. Write a C program to read the data from the file “person.txt” which contains personno and

personage and sort the data on age in ascending order using insertion Sort.

**SET C:**

1. Write a C program to sort a random array of n integers (value of n accepted from user) by using Bubble Sort algorithm in ascending order
2. Write a C program to sort a random array of n integers (value of n is accepted from user) by using Insertion Sort algorithm in ascending order .
3. Add the code to Print Time complexity for SET A programs.
4. Write a C program to sort the elements by initializing the array (e.g int A[5] = {10, 20, 35, 23, 12}) using bubble sort.

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: Well Done [ ]

**Signature of Instructor**

**Assignment No : 2**

**Sorting Techniques (Recursive)**

**QUICK SORT**

* It is also called “partition Exchange sort. The strategy used here is “divide and conquer” i.e we successively partition the list in smaller lists and apply the same procedure to the sub-list. The procedure is as follows:-

**Procedure –**

* We will consider one element at a time (pivot) and place it in its correct position.
* The pivot is placed in a position such that all elements to the left of the pivot are less than the pivot and all elements to the right are greater.
* The array is partitioned into two parts:- left partition and right partition.
* The same method is applied for each of the partition.
* The process continues till no more partition can be made. We shall be considering the first element of the partition as the pivot element.

**Algorithm:**

Step 1: start.

Step 2: A is an array of n element.

Step 3: lb=0 lb = lower bound

ub = n-1 ub = upper bound.

Step 4: if(lb<ub)

i.e. if the array can be partitioned

j=partition(A,lb,ub) // j is the pivot position

quicksort(A,lb,j-1);

quicksort(A,j+1,ub);

* Now, we must write the function to partition the array. There are many methods to do the partitioning depending upon which element is chosen as the pivot.
* We will be selecting the first element as the pivot element and do the partitioning accordingly.
* We shall choose the first element of the sub- array as the pivot and find its correct position in the sub- array.
* We will be using two variables down and up for moving down and up array.

**Algorithm for partitioning**

Step 1: down=lb+1

Step 2: up=ub

Step 3: pivot=A[lb]

Step 4: perform step 5 to 7 as long as down<up else go to step 8.

Step 5: while (A[down]<=pivot && down<ub) down++;

Step 6: while (A[up]>pivot) up--;

Step 7: if (down<up) interchange A[down] and A[up]

Step 8: interchange A[up] and pivot, j=up, i.e. pivot position=up

Step 9: return up

Step 10: stop.

In this algorithm we want to find the position of pivot i.e. A[lb].

* We use two pointers up and down initialized to the first and last elements respectively.
* We repeatedly increase down as long as the element is < pivot.
* We repeatedly decrease up as long as the element is > pivot.
* If up and down cross each other i.e. up<=down, the correct position of the pivot is up and A[up and pivot are interchanged.
* If up and down do not cross A[up] and A[down] are interchanged and process is repeated till they do not cross or coincide.
* Efficiency of quick sort.

Best case = average case = O(nlogn)

Worst case = O(n2)

**MERGE SORT.**

* Merging is the process of combining two or more sorted data lists into a third list such that it is also sorted.
* Merge sort follows Divide and Conquer strategy.
* Divide :- divide an n element sequence into n/2 subsequence.
* Conquer :- sort the two sequences recursively.
* Combine :- merge the two sorted sequence into a single sequence.
* In this two list are compared and the smallest element is stored in the third array.
* **Algorithm**:-

Step 1: start

Step 2: initially the data is considered as a single array of n element .

Step 3: divide the array into n/2 sub-array each of length 2i (I is 0 for 0th iteration). i.e. array is divided into n sub-arrays each of 1 element.

Step 4: merge two consecutive pairs of sub-arrays such that the resulting sub-array is also sorted.

Step 5: The sub-array having no pairs is carried a sit is

Step 6: step 3 and 4 are repeated till there is only one sub-array remaining of size n.

Step 7: stop.

**SET A:**

1. Write a C program to accept and sort n elements in ascending order by using Quick sort.
2. Write a C program to sort a random array of n integers (value of n accepted from user) by using Quick Sort algorithm in ascending order

**SET B:**

1. Write a C program to accept and sort n elements in ascending order by using Merge sort.
2. Write a C program to sort a random array of n integers (value of n is accepted from user) by using Merge Sort algorithm in ascending order

**SETC:**

1. Add the code in SET A (Q1) and SET B (Q1) to Print Time complexity for Quick sort and Merge sort.
2. Write a C program to sort the elements by initializing the array (e.g. int A[5] = {11, 12,15, 16, 17}) using Merge sort .
3. Write a C program to sort the elements by initializing the array (e.g. int A[5] = {11, 12,15, 16, 17}) using Quick Sort .

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: Well Done [ ]

**Signature of Instructor**

**Assignment No : 3**

**Searching Techniques**

The commonly used searching methods used are linear search and Binary search

**SET A:**

1. Write a C program to accept n elements from user store it in an array. Accept a value from the user and use linear/Sequential search method to check whether the value is present in array or not. Display proper message.
2. Write a C program to accept n elements from user store it in an array. Accept a value from the user and use recursive binary search method to check whether the value is present in array or not. Display proper message.

**SET B:**

1. Add the code to print the number of Comparisons for SET A programs
2. Write a C program to accept n student names from user store it in an array. Accept a student name from the user and use linear/Sequential search method to check whether the student name is present in array or not. Display proper message.

**SET C:**

1. Write a C program to read the data from file 'cities.txt' containing names of 10 cities and their STD codes. Accept a name of the city from user and use linear search algorithm to check whether the name is present in the file and output the STD code, otherwise output “city not in the list”.
2. Write a C program to accept n elements from user store it in an array. Accept a value from the user and use Non- recursive binary search method to check whether the value is present in array or not. Display proper message.

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: Well Done [ ]

**Signature of Instructor**

**Assignment No : 4**

**Linked List**

**Linked list:-**

A linked list is an ordered collection of items which is dynamic in nature i.e. its size varies and each item is ‘linked” or connected to another item. It is a linear collection of data elements called nodes.

**LINKED LIST IMPLEMENTATION-:**

A linked list may be implemented in two ways:

* + 1. Static representation
    2. Dynamic representation.

1. **Static representation:-**

An array is used to store the elements of the list. The elements may not be stored in a sequential order. The correct order can be stored in another array called “link”

The values in this array are pointers to elements in the disk array.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data array | |  |  |  | Link array | |
| 0 | Blue |  | | 0 |  | 4 |
| 1 | Red |  | | 1 |  | -1 |
| 2 |  |  | | 2 |  |  |
| 3 | Violet |  | | 3 |  | 0 |
|  |  |  |  |  |  |  |
| 4 | Green |  | | 4 |  | 6 |
| 5 |  |  | | 5 |  |  |
| 6 | Orange |  |  | 6 |  | 1 |

Data[3] = violet

Data[0] = Blue

Data[4] = Green

Data[6] = Orange

Data[1] = Red

Link[3] = 0

Link[0] = 4

Link[4] = 6

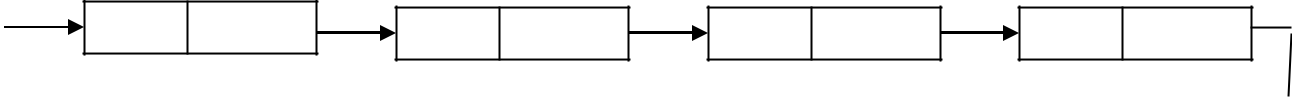
Link[6] = 1

Link[1] = -1 list end

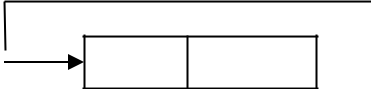
1. **Dynamic Representation:-**
   * The static representation uses arrays which is a static data structure and has its own limitations.
   * A linked list is a dynamic data structure i.e. the size should increase and when elements are deleted, its size should decrease.
   * This cannot be possible using an array which uses static memory allocation i.e. memory is allocated during compile time. Hence we have to use “dynamic memory allocation” where memory can be allocated and de-allocated during run-time.

* Another way of storing a list in memory is by dynamically allocating memory for each node and linking them by means of pointers since each node will be at random memory location. We will need a pointer to store the address of the first node.

list=3



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| violet 0 | Blue | 4 | Green | 6 | orange | 1 |



red NULL

List is an external pointer which stores the address of the first node of the list.

**TYPES OF LINKED LIST**

1. Singly Linked list
2. Circular linked list
3. Doubly linked list
4. Circular doubly linked list

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Linked list | |  |  |  |
|  |  | Linear | |  | Circular | |
|  |  |  |  |  |  |  |
| Singly linked list | | Doubly linked list | | Singly linked listDoubly linked list | | |



1. **Linear linked list**

In this list the elements are organized in a linear fashion and list terminates at some point i.e. the last node contains a NULL pointer.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| list |  |  |  |  |  |  |  |  |  |  |
| Data |  | next |  | Data | next |  | Data | NULL |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | node | |  |  |  | node | |  | node |  |



* Singly linked list-

Each node in this list contains only one pointer which points to the next node.



list

Data next



Data next  Data next  Data NULL

* Doubly linked list

Each node in this contains two pointers, one pointing to the previous node and the other pointing to the next node. This list is used when traversing in both directions is required.

Prev data next



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Prev | data | next |  |  | Prev | | data | | next | |  | Prev | | data | next | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| list | NULL | 10 | 200 |  | 100 |  | 20 |  | 300 |  |  | 200 |  | 30 | NULL |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 100 | | |  |  |  |  |  | 200 | |  |  |  |  |  | 300 |  |  |  |



2) Circular list

In this list, the last node does not contain a NULL pointer but points back to the first node i.e. it contains the address of the first node. Each of these lists can be either a singly linked or a doubly list.

Circular singly linked list

Data

next

Data

next

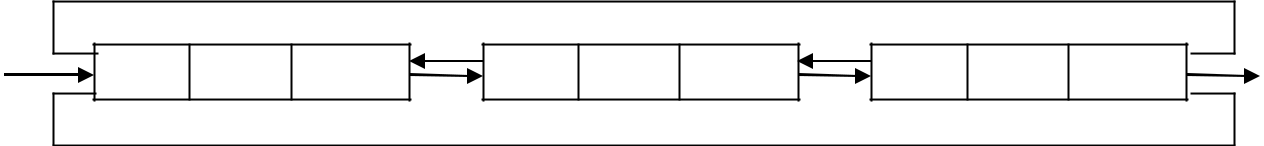
Data

next

Prev

data

next



List

Circular doubly linked list

**OPERATIONS ON A LIST**

The following are some of the basic list operations:-

1. Traversing a list:-

Visiting each node of the list is called traversal

1. Insertion:-

A node can be inserted at the beginning, end or in between two nodes of the list.

1. Deletion:-

Deletion from a list may be done either position-wise or element-wise

1. Display:-

Display each element of the list.

1. Searching:-

This process searches for a specific element in the list.

1. Reversing or inversion:-This process reverses the order of nodes in the list
2. Concatenation:-This process appends the nodes of the second list at the end of the first list i.e. it joins two lists.
3. Computation of length:-Count the total no. of nodes in the list
4. Creating a linked list
5. Intersection, union, difference.

**SET A:**

1. Write a C program to implement a singly linked list with Create and Display operation.
2. Write a C program to implement a Circular Singly linked list with Create and Display operation.
3. Write a C program to implement a doubly linked list with Create and Display operation.

**SET B:**

1. Implement the following programs by adding the functions one by one in SET A (for all questions)
   1. Add the function for Append operation.
   2. Add the function for Search operation.
   3. Add the function to insert node at the beginning.
   4. Add the function to insert node at the end.
   5. Add the function to insert node in the list by accepting the value in the specified linked list. (Note: accept a number; search the number in the list, then insert the new node after that searched node)
   6. Add the function to insert node in the list by accepting the position.

(Note: accept the Position, and then insert the new node in that particular position.)

* 1. Add the function to find length of linked list

**SET C:**

1. Implement the following programs by adding the functions one by one in SET A (for Q1 and Q3)
   1. Add the function to delete the first node.
   2. Add the function to delete the last node.
   3. Add the function to delete the node by accepting the value.

(Note: accept the number, search the number in the list, then delete that node from the linked list)

* 1. Add the function to delete the node by accepting the position.

(Note: accept the position, search the node at that position in the list and then delete the node from that position.)

1. Write a C program to add two polynomial.
2. Write a C program to find intersection of two linked list.
3. Write a C program to reverse a singly linked list.
4. Write a C program to display the data of list in reverse.
5. Write a C program to implement static linear linked list using array for following operation
   * Append() , Insert(), Display()

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: Well Done [ ]

**Signature of Instructor**

**Assignment No : 5:** **Stack**

A stack is an ordered collection of items into which items may be inserted and deleted from one end called the top of the stack.

The stack operates in a LIFO (last in first out) manner. i.e. the element which is put in last is the first to come out. That means it is possible to remove elements from stack in reverse order from the insertion of elements into the stack.

The real life e.g. of the stack are stack of coins, stack of dishes etc. only the topmost plate can be taken and any new plates has to be put at the top.

**PRIMITIVE OPERATIONS ON A STACK.**

1. Create
2. Push
3. Pop
4. Isempty
5. Isfull
6. Peek

**STACK IMPLEMENTATION:-**

The stack implementation can be done in two ways:-

**1)** **Static implementation:-**

* + It can be achieved using arrays. Though it is very simple method it has few limitations.
  + Once a size of an array is declared, its size cannot be modified during program execution.
  + The vacant space of stack also occupies memory space.

In both cases, if we store less argument than declared, memory is wasted and if we want to store more elements than declared array cannot be expanded. It is suitable only when we exactly know the number of elements to be stored.

**Operations on static stack:-**

1) Declaring of a stack :-

* An array of a fixed size.
* An integer called top which stored the index or position of the topmost element. We can use a structure for the above purpose.

2) Creating a stack:-

This declaration only specifies the template. The actual stack can be declared as-STACK s1;

3) initialize a stack:-

When a stack variable is declared the integer top has to be initialized to indicate an empty stack. Since we are using an array the first element will occupy position 0. Hence to indicate an empty stack top has to be initialized to -1

4) Checking whether stack is empty:-

An empty stack can be tested from the value contained in top. If top contains -1 it indicates an empty stack.

5) Checking whether stack is full:-

If the value of top reaches the maximum array index i.e. MAX-1 no more elements can be pushed into the stack.

6) The push operation:-

The element can be pushed into the stack only if it is not full. In such case the top has to be incremented first and the element has to be put in this position.

7) The pop operation:

An element can be removed from the stack if it is not empty. The topmost element can be removed after which top has to decremented

* 1. The peek operation:

It displays the topmost element of the stack without decrementing the top.

1. **Dynamic implementation:-**

* Pointers are used for implementation of stack. The linked list is an e.g. of this implementation.
* The limitations noticed in static implementation can be removed using dynamic implementation. The dynamic implementation is achieved using pointers.
* Using pointer implementation at runtime there is no restriction on the no. of elements. The stack may be expandable.
* The memory is efficiently utilized with pointers.
* Memory is allocated only after element is pushed to the stack
* In static representation there is a limitation on the size of the array if less elements are stored, memory will be wasted. To overcome the program the stack can be implemented using linked list.
* In the linked organization
  + The stack can grow to any size.
  + We need not have prior knowledge of the number of elements.
* When an element is popped the memory can be freed. Thus memory is not unnecessary occupied.
* Since random access to any element is not required in a stack, the linked representation is preferred over the sequential organization.

**SETA:**

1. Write a C program to implement Static implementation of stack of integers with following operation:

-Initialize(), push(), pop(), isempty(), isfull(), display(), peek()

1. Write a C program to implement Dynamic implementation of stack of integers with following operation:

-Initialize(), push(), pop(), isempty(), display(), peek()

**SETB:**

1. Write a C program to reverse the given string by using static and dynamic implementation of stack.
2. Write a C program to reverse the given number by using static and dynamic implementation of stack.

**SET C:**

1. Write a C program to reverse the stack by using recursive function.
2. Write a C program to convert infix expression to postfix expression
3. Write a C program to evaluate postfix expression.

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: Well Done [ ]

**Signature of Instructor**

**Assignment No : 6**

**Queue**

A queue is an ordered collection of items from which items may be deleted from (or removed from ) one end called the front and into which items may be inserted at the other end called rear.

**BASIC OPERATIONS ON QUEUE**

* 1. Create:

Creates a new queue. This operation creates an empty queue.

2) Add or insert:

Adds an element to the queue. A new element can be added to the queue at the

rear.

3) Delete:

Remove an element from the queue. This operation removes the elements, which is at the front of the queue. This operation can only be performed if the queue is not empty.

The result of an illegal attempt to remove an element from an empty queue is called underflow.

4) isempty:

Checks whether a queue is empty. The operation return true if the queue isempty and false otherwise

5) isfull:

Checks whether a queue is full. The operation return true if the queue isfull and false otherwise

**REPRESENTATION OF LINEAR QUEUES.**

There are two ways to represent a queue in memory.

* + 1. Static (using an array)
    2. Dynamic (using an linked list)
  1. Static implementation of queue
* An array to hold queue elements.
* A variable to hold the index of the front element.
* A variable to hold the index of the rear element.

The implementation of a queue using sequential representation is done by using some size MAX and two integer variable front and rear. Initially front and rear is set to -1. Whenever new element is added it is added from the rear and whenever an element is to be removed from the front. The queue full condition is when rear reaches to MAX - 1. Queue empty condition is when front is equal to rear.

2) Dynamic implementation of linear queue (using an linked list)

A queue can be considered as a list in which all insertions are made at one end called the rear and all deletions from the other end from front.

A queue can be easily represented using a linked list. The front and rear will be two pointers pointing to the first and last node respectively.

**SETA:**

1. Write a C program to Implement Static implementation of Queue of integers with following operation:

-Initialize(), insert(), delete(), isempty(), isfull(), display(), peek()

**SET B:**

1. Write a C program to Implement Dynamic implementation of Queue of integers with following operation:

-Initialize(), insert(), delete(), isempty(), display(), peek()

**SET C:**

1. Write a C program to Implement Static implementation of circular queue of integers with following operation:

-Initialize(), insert(), delete(), isempty(), isfull(), display(), peek()

1. Write a C program to Implement Dynamic implementation of circular queue of integers with following operation:

-Initialize(), insert(), delete(), isempty(), display(), peek()

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: WellDone [ ]

**Signature of Instructor**

**Assignment No : 7**

**Trees**

**Definition of tree:-**

A tree is a finite set of one or more nodes such that:- there is a specially designated node called the root. The remaining nodes are partitioned into n>=0 disjoint sets T1……Tn where each of

these sets is a tree. T1…..Tn are called as sub-trees of the root.

The operations on binary search tree are

init (T) – creates an empty Binary search tree by initializing T to NULL

insert (T, x) – inserts the value x in the proper position in the Binary search tree search (T, x) – searches if the value x is present in the search tree

inOrder (T) – displays the node using inorder traversal of binary search tree postOrder (T) – displays the node using postorder traversal of binary search tree preOrder (T) – displays the node using preorder traversal of binary search tree

**SET A:**

1. Write C programs to implement create and display operation for binary tree.
2. Write C programs to implement create and display operation for binary search tree.

**SET B:**

1. Implement the following programs by adding the functions one by one in Q2 of SET A
   1. Add a function to insert a new element in the tree.
   2. Add a function to search an element in tree and give the proper message.
   3. Add a function to create mirror image of the tree.
   4. Add a function to count non-leaf nodes.
   5. Add a function to count leaf nodes.
   6. Add a function to count total number of nodes.

**SET C:**

1. Write a C program to Implement Binary search tree with following operations
   * Create(), inorder(), preorder(), postorder()

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: WellDone [ ]

**Signature of Instructor**

**Assignment No : 8**

**Graph**

A graph consists of a set of vertices and a set of edges. The two main ways of representing graphs are adjacency matrix representation and adjacency list representation. In adjacency matrix representation of a Graph with n vertices and e edges, a two dimensional nxn array , say a , is used , with the property that a[i,j] equals 1 if there is an edge from i to j and a[i,j] equals 0 if there is no edge from i to j.

In adjacency list representation of a graph with n vertices and e edges, there are n linked lists, one list for each vertex in the graph.

The usual operations on graph are:

Indegree(i) – returns the indegree (the number of edges ending on) of the ith vertex Outdegree(i) – returns the outdegree(the number of edges moving out) of the ith vertex displayAdjMatrix – displays the adjacency matrix for the graph

**SET A:**

1. Write a C program to read a graph as adjacency matrix and display the adjacency matrix.
2. Add a function in Q1 (above question) to count total degree , indegree and outdegree of the graph.

**SET B:**

1. Write a C program to convert adjacency matrix into adjacency list. Display the adjacency list.
2. Write a C program to traverse graph by using BFS.

**SET C:**

1. Write a C program to traverse graph by using DFS.
2. Implement a program for simple transpose of the matrix.

**Assignment Evaluation**

0: Not Done [ ]

3: Needs Improvement [ ]

1: Incomplete [ ]

4: Complete [ ]

2: Late Complete [ ]

5: WellDone [ ]

**Signature of Instructor**