Table of Contents

Introduction

About the workbook

This workbook is intended to be used by S. Y. B. Sc (Computer Science) students for the Data Structures using C Lab course and Relational Database Management System in Semester III.

Workbook is divided in two sections.

1. First section contains assignments on Data Structure using C.
2. Second section contains assignments on Relational Database Management System.

Data structures using C is an important core subject of computer science curriculum, and hands-on laboratory experience is critical to the understanding of theoretical concepts studied as part of this course. Study of any programming language is incomplete without hands on experience of implementing solutions using programming paradigms and verifying them in the lab. This workbook provides rich set of problems covering the basic algorithms as well as numerous computing problems demonstrating the applicability and importance of various data structures and related algorithms.

The objectives of this book are

- Defining clearly the scope of the course
- Bringing uniformity in the way the course is conducted across different colleges
- Continuous assessment of the course
- Bring variation and variety in experiments carried out by different students in a batch
- Providing ready reference for students while working in the lab
- Catering to the need of slow paced as well as fast paced learners

How to use this workbook

The Data Structures using C practical syllabus is divided into five assignments. Each assignment has problems divided into three sets A, B and C.

- Set A is used for implementing the basic algorithms or implementing data structure along with its basic operations. **Set A is mandatory.**
- Set B is used to demonstrate small variations on the implementations carried out in set A to improve its applicability. Depending on the time availability the students should be encouraged to complete set B.
- Set C prepares the students for the viva in the subject. Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

Instructions to the students

Please read the following instructions carefully and follow them.

- Students are expected to carry workbook during every practical.
- Students should prepare oneself before hand for the Assignment by reading the relevant material.
- 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 as many problems as possible given in this work book.
- Students will be assessed for each exercise on a scale from 0 to 5
    - Not done 0
    - Incomplete 1
    - Late Complete 2
    - Needs improvement 3
    - Complete 4
    - Well Done 5

Instruction to the Practical In-Charge

- Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.
- Choose appropriate problems to be solved by students. Set A is mandatory. Choose problems from set B depending on time availability. Discuss set C with students and encourage them to solve the problems by spending additional time in lab or at home.
- Make sure that students follow the instruction as given above.
- You should 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.

Instructions to the Lab administrator and Exam guidelines

- You have to ensure appropriate hardware and software is made available to each student.
- Do not provide Internet facility in Computer Lab while examination
- Do not provide pen drive facility in Computer Lab while examination.

The operating system and software requirements are as given below:

- Operating system: Linux
- Editor: Any Linux based editor like vi, gedit etc.
- Compiler: cc or gcc

Assignment Completion Sheet

Sr. No	Assignment Name	Marks (Out of 5)	Signature
1	Assignment on array: Matrix operations		
2	Searching Algorithms a. Sequential Search b. Binary Search		
3	Sorting Algorithms 3A – Bubble, Insertion, Selection 3B – Quick, Merge Sort and Count Sort		
4	Operations on Stack		
5	Applications of Stack – palindrome, Conversion to postfix and evaluation of postfix		
6	Operations on Queue and Implementation of priority queue		
7	Operations on Singly linked list		
8	Operations on Doubly linked list		
9	Tree Traversal		
10	Binary Search tree – create(), insert(), searc()		
Total out of 50			
a. Total out of 5 (DS Assignment marks converted to 5)			
b. Total out of 10 (Relational Database Management System)			
Total (Out of 15) (a+b)			

This is to certify that Mr/Ms _____

University Exam Seat Number _____ has successfully completed the course work for Lab Course I and has scored _____Marks out of 15.

Instructor **Head**

Internal Examiner **External Examiner**

Section I

Data Structures using C

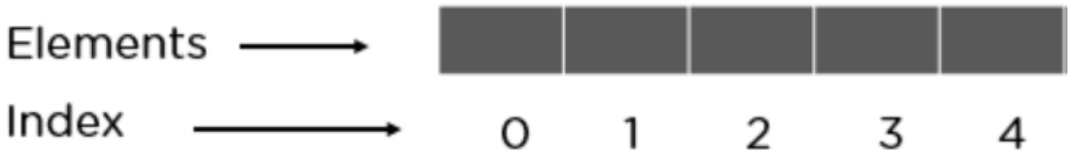
Assignment 1: Assignment on array: Matrix operations

An array is a linear data structure that collects elements of the same data type and stores them in contiguous and adjacent memory locations. Arrays work on an index system starting from 0 to (n-1), where n is the size of the array.

What Are the Types of Arrays?

There are majorly two types of arrays, they are:

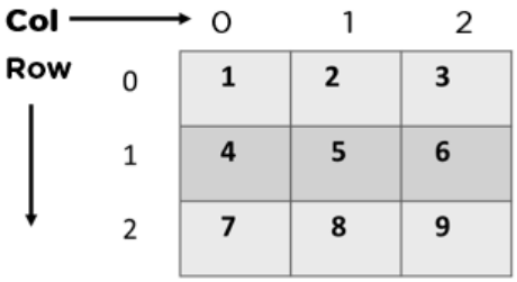
One-Dimensional Arrays:



Multi-Dimensional Arrays:

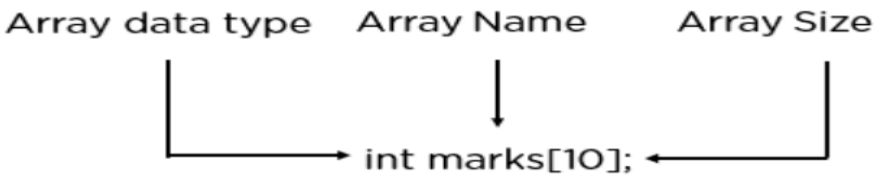
These multi-dimensional arrays are again of two types. They are:

Two-Dimensional Arrays:



You can imagine it like a table where each cell contains elements.

How Do You Declare an Array?



Arrays are typically defined with square brackets with the size of the arrays as its argument. Here is the syntax for arrays:

1D Arrays: int arr[n];
2D Arrays: int arr[m][n];

How Do You Initialize an Array?

You can initialize an array in four different ways:

Method 1:

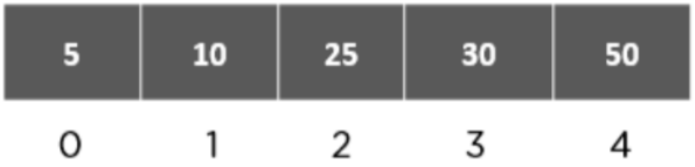
int a[6] = {2, 3, 5, 7, 11, 13};

Method 2:
int arr[]= {2, 3, 5, 7, 11};

Method 3:
int n;
scanf("%d",&n);
int arr[n];
for(int i=0;i<5;i++)
{
scanf("%d",&arr[i]);
}

Method 4:
int arr[5];
arr[0]=1;
arr[1]=2;
arr[2]=3;
arr[3]=4;
arr[4]=5;

How Can You Access Elements of Arrays in Data Structures?



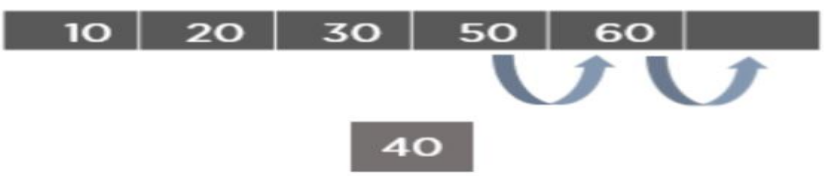
You can access elements with the help of the index at which you stored them.
printf("%d\n",a[0]); // we are accessing first element i.e 5

What Operations Can You Perform on an Array?

- Traversal
- Insertion
- Deletion
- Searching
- Sorting

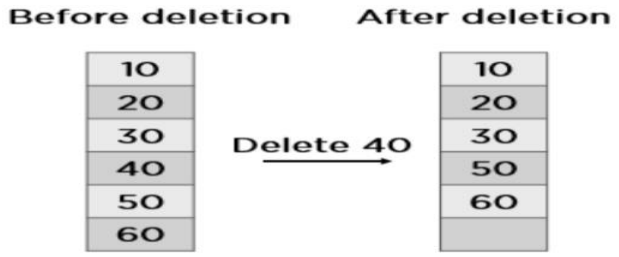
Traversing the Array:
Traversal in an array is a process of visiting each element once.

Insertion:



Insertion in an array is the process of including one or more elements in an array.
Insertion of an element can be done:
At the beginning
At the end and
At any given index of an array.

Deletion:



Deletion of an element is the process of removing the desired element and re-organizing it.

You can also do deletion in different ways:

At the beginning

At the end

Set A

1. Write a program to enter 'n' elements in 1-D Array and to display those elements. (Take the values from user at run time)
2. Write a program to enter elements in 2-D Array (n*m matrix) and to display those elements. (Take the values from user at run time)
3. Write a program to insert the elements at the end of the array

Set B

1. Write a program to insert the elements at the position provided by the user. (All other respective elements will be shifted)
2. Write a program to delete the elements from the given position provided by the user. (All other respective elements will be shifted)

Set C

1. Write the program to multiply 2-D Array.
2. Insert and delete any user specified values from 2-D Array.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 2: Searching algorithms

a. Sequential Search

b. Binary Search

Searching is the process of finding a value in a list of values. The searching algorithms you are to use in this assignment are linear, sentinel and binary search.

In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.

Algorithm:

Step 1: Start
Step 2: Accept n numbers in an array num and a number to be searched
Step 3: se ti=0 and flag=0
Step 4: if i<n then goto next step else goto step 7
Step 5: Compare num[i] and number
    If equal then
        set flag=1 and goto step 7
Step 6: i=i+1 and goto step 4
Step 7: if (flag=1) then
    Print "Required number is found at location i+1"

  else
    Print "Require data not found"

Step 8: Stop

Time Complexity:

Base Case: O (1)
Worst Case: O(n)
Average Case: O(n)

Sentinel search is a type of linear search where the number of comparisons is reduced as compared to a traditional linear search. The value to be searched can be appended to the list at the end as a "sentinel value.

Algorithm:

Step 1: Start
Step 2: Accept n numbers in an array num and a number to be searched
Step 3: set i=0, last=num[n-1], num[n]=number
Step 4: Compare num[i] and number
    If equal then goto step 6
Step 5: i=i+1 and goto step 4

Step 6: if (i<n) then
Print “Required number is found at location i+1”

else
Print “Require data not found”

Step 8: Stop

Time Complexity:

Base Case: O(1)
Worst Case: O(n)
Average Case: O(n)

Binary Search is used with sorted array or list. So a necessary condition for Binary search to work is that the list/array should be sorted. It works by repeatedly dividing in half the portion of the list that could contain the item.

Algorithm:

Step 1: Start
Step 2: Accept n numbers in an array num and a number to be searched
Step 3: set low=0, high=n-1 and flag=0
Step 5: if low <= high then
middle=(low+high)/2
else
goto step 8.
Step 6: if (num[middle]=number)
position=middle, flag=1 goto step 8.
else if (number< num[middle])
high=middle-1
else
low=middle+1
Step 7: goto step 5
Step 8: if flag=1
Print “Required number is found at location position+1”
Else
Print “Required number is not found.
Step 9: Stop

Time Complexity:

Base Case: O(1)
Worst Case: O(log n)

Average Case: O(log n)

The data to be searched is in memory usually in an array. It could be an array of integers, characters, strings or of defined structure type. To test a searching algorithm we require large data set. Data is generated using random (rand()) function. The array of random integers in the range 0 to 99 is generated by using following code:

```
void generate ( int * a , int n)
{
        int i;
        for (i=0; i<n; i++)
                a[i]=rand()%100;
}
```

Set A

1. Create a random array of n integers. Accept a value x from user and use linear search algorithm to check whether the number is present in the array or not and output the position if the number is present.
2. Accept n values in array from user. Accept a value x from user and use sentinel linear search algorithm to check whether the number is present in the array or not and output the position if the number is present.
3. Accept n sorted values in array from user. Accept a value x from user and use binary search algorithm to check whether the number is present in sorted array or not and output the position if the number is present.

Set B

1. Read the data from file 'sycsstudent.txt' containing names of student name and their grade. Accept a name of the student name from user and use linear search algorithm to check whether the name is present in the file and output the grade, otherwise output "students not in the list".
2. Read the data from file 'cities.txt' containing names of cities and their STD codes. Accept a name of the city from user and use sentinel 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".
3. Read the data from file 'sycsstudent.txt' containing sorted names of student name and their grade. Accept a name of the student name from user and use binary search algorithm to check whether the name is present in the file and output the grade, otherwise output "students not in the list".

Set C

1. If the file contains multiple occurrences of a given element, linear search will give the

2. position of the first occurrence, what modifications are required to get the last occurrence?
3. If the file contains multiple occurrences of a given element, will binary search output the position of first occurrence or last occurrence?
4. Which is best case search when searching using linear search, sentinel search and when using binary search?
5. What modifications are required to linear search, sentinel search and binary search algorithm to count the number of comparisons?
6. What modifications are required to binary search so that it returns the position where x can be inserted in the sorted array to retain the sorted order?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 3: Sorting Algorithms

A) Bubble Sort, Insertion Sort, Selection Sort

B) Quick, Merge Sort and count sort

Arranging in an ordered sequence is called Sorting. The sorting algorithms you are to use in this assignment are bubble sort, insertion sort and selection sort. These are non-recursive algorithms.

Bubble Sort
It compares adjacent elements and swaps if required. The pass is repeated until the list is sorted.

Algorithm:

Step1: Start
Step2: Accept ‘n’ numbers in array ‘A’
Step3: set i=0
Step4: if i<n-1 then goto next step else goto step 9
Step4: set j=0
Step5: if j<n-i-1 then goto next step else goto step 8
Step6: if A[j] > A[j+1] then
interchange A[j] and A[j+1]
Step7: j=j+1 and goto step 5
Step8: i=i+1 and goto step 4
Step9:Stop

Time Complexity:

Base Case: O(n)
Worst Case: $O(n^2)$
Average Case: $O(n^2)$

Insertion Sort

Insertion sort removes one element from list, find proper location in the list and insert it in the list. This is repeated until no input element remains.

Algorithm:

Step1: Start
Step2: Accept ‘n’ numbers and store all in array ‘A’
Step3: set i=1
Step4: if i<=n-1 then goto next step else goto step 10
Step5: set Temp=A[i] and j=i-1

Step6: if Temp < A[j] && j>=0 then goto next step else goto step 9
Step7: set A[j+1]=A[j]
Step8: set j=j-1
Step9: set A[j+1]=Temp
Step10: Stop

Time Complexity:

Base Case: O(n)
Worst Case: $O(n^2)$
Average Case: $O(n^2)$

Selection Sort:

In this sort the smallest element is selected from the unsorted array and swapped with the leftmost element.

Algorithm:

Step1: Start
Step2: Accept ‘n’ numbers and store all in array ‘A’
Step3: set i=0
Step4: if i<n-1 then goto next step else goto step 11
Step5: set min=i and j=i+1
Step6: if j<n then goto next step else goto step 9
Step7: if A[j]<A[min] then
min=j
Step8: set j=j+1 and goto step 6
Step9: if(min not equal to i) then
interchange A[i] and A[min]
Step10: i=i+1 and goto step 4
Step11: Stop

Time Complexity:

Base Case: $O(n^2)$
Worst Case: $O(n^2)$
Average Case: $O(n^2)$

In reality data to be sorted is externally stored in files. One need to read data from files and bring it into memory in an array before sorting and sorted array also need to be written back to an external file.

Suppose the records to be sorted containing name, age and salary of a set of employees, is in

a text file “employee.txt” as follows:

Rajiv 43 100000
Prakash 34 29000
Vinay 35 20000
.....

The data is read into memory in an array of structures as follows

Variable declarations & main program

```
typedef struct
{
        char name[30];
        int age;
intsalary;
}RECORD;

RECORD emp[100];
main()
{
         int n; n=readFile(emp);
         sort(emp,n);
writeFile(emp,n);
}
```

Function for reading from a file

```
int readFile(RECORD *a)
{
        int i=0;
FILE *fp;
        if((fp=fopen("emp.txt","r"))!=NULL)
        {
while(! feof(fp))
           {
fscanf(fp,"%s%d%d", a[i].name,  &a[i].age, &a[i].salary);
i++;
            }
        }
return i; // number of records read
}
```

Function for writing to a file

```
void writeFile(RECORD *a, int n)
{
        int i=0;
        FILE *fp;
        if((fp=fopen("sortedemp.txt","w"))!=NULL)
        {
for(i=0;i<n; i++)
                {
fprintf(fp,"%s\t%d\t%d\n", a[i].name, a[i].age, a[i].salary);
                }
         }
}
```

Set A

1. Sort a random array of n integers (accept the value of n from user) in ascending order by using bubble sort algorithm.
2. Sort a random array of n integers (create a random array of n integers) in ascending order by using insertion sort algorithm.
3. Sort a random array of n integers (accept the value of n from user) in ascending order by using selection sort algorithm.

Set B

1. Read the data from the file "employee.txt" and sort on age using bubble sort, insertion sort and selection sort.
2. Read the data from the file "employee.txt" and sort on names in alphabetical order (use strcmp) using bubble sort, insertion sort and selection sort.

Set C

1. What modification is required to bubble sort, insertion sort and selection to sort the integers in descending order?
2. What modifications are required to bubble sort to count the number of swaps?
3. What modifications are required to insertion sort to count the number of key comparisons?
4. What modifications are required to output the array contents after every pass of the sorting algorithm?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 3B: Sorting Algorithms –Quick Sort, Merge Sort and Counting Sort

Sorting techniques to be used in this assignment are Counting Sort, Merge sort, Quick sort. Merge and Quick sort both are recursive and use divide and conquer strategy.

In Quick sort, data is partitioned into two parts in such a way that all elements in first part are less than or equal to elements in second part. Both the parts are then sorted using the same Quick sort technique.

Algorithm:

QuickSort(A,n)
Step1: Start
Step2: Accept 'n' numbers and store all in array 'A'
Step3: lb=0, ub=n-1
Step3: if (lb<ub)
    j=Partition(A,lb,ub)
    QuickSort(A,lb,j-1)
    QuickSort(A,j+1,ub)

Partition(A, lb, ub)
Step1: down=lb, up=ub
Step2: pivot=A[lb]
Step3: while (A[down]<=pivot && down<up)
        down++
Step4: while (A[up]>pivot && up>down)
        up--
Step5: if (down < up)
        Interchange A[down] and A[up] and goto step 3.
Step6: Interchange A[up] and pivot
Step7: return up
Step8: Stop

The average time complexity of quick sort is O(N log(N))
The average space complexity of quick sort **O(N)**

Merge Sort
In Merge sort, data is divided into two parts, each part is sorted by using the same merge sort technique and the sorted files are then merged using a Merge procedure.

Algorithm:
mergesort(a, N) algorithm:
Step1: Start
Step2: Accept N numbers and store into array a
Step3: Assign low=0 and high=N-1
Step4: if low < high
        mid=(low+high)/2

```
        mergesort(a,low,mid);
        mergesort(a,mid+1,high);
        merge(a,low,mid,high);
```

merge(A, low, mid, high) algorithm:

Step1: i=low, j=mid+1, k=0;

Step2: while i<=mid && j<=high

```
        if(a[i]<=a[j])
            b[k]=a[i]
            k=k+1, i=i+1
        else
            b[k]=a[j]
            k=k+1, j=j+1
```

Step3: while(i<=mid)

```
        b[k]=a[i]
        k=k+1, i=i+1
```

Step4: while(j<=high)

```
        b[k]=a[j]
        k=k+1, j=j+1
```

Step5: j=low, k=0

Step6: while j<=high

```
        a[j]=b[k]
        j=j+1, k=k+1
```

Step7: stop

AVERAGE TIME COMPLEXITY :O(n*log(n))

Counting sort is an algorithm for sorting a collection of objects according to keys that are small integers. So, it is an integer sorting algorithm.

```
countingSort(array, size)
    max <- find largest element in array
    initialize count array with all zeros
    for j <- 0 to size
        find the total count of each unique element and
        store the count at jth index in count array
    for i<- 1 to max
        find the cumulative sum and store it in count array itself
    for j <- size down to 1
        restore the elements to array
        decrease count of each element restored by 1
```

Set A

1. Sort a random array of n integers (accept the value of n from user) in ascending order by using recursive Counting sort algorithm.
2. Sort a random array of n integers (accept the value of n from user) in ascending order by using a recursive Merge sort algorithm.
3. Sort a random array of n integers (create a random array of n integers) in ascending order by using recursive Quick sort algorithm.

Set B

1. Read the data from the ‘employee.txt’ file and sort on age using Counting sort, Merge sort, Quick sort and write the sorted data to another file 'sortedemponage.txt'.
2. Read the data from the file and sort on names in alphabetical order (use strcmp) using Counting sort, Merge sort, Quick sort and write the sorted data to another file 'sortedemponname.txt'.

Set C

1. What modifications are required to choose the pivot element randomly instead of choosing the first element as pivot element while partitioning in Quick sort algorithm?
2. Compare the system time taken by Merge sort and bubble sort by using time command on a random array of integers of size 10000 or more.
3. What modification is required to Merge sort to sort the integers in descending order?
4. In 'employee.txt' there are records with same name but different age and salary values. What are the relative positions when the data is sorted on name using Merge sort and what happens in case of quick sort?
5. Sort a random array of integers of large size and store the sorted file. Compare the system time taken by Quicksort on a random file of large size and the sorted file of same size. Repeat the same for Merge sort. Does sorted file give best time?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

How to Make Custom Header File / Library file?

What are the header files?

Header files are helping file of C program which holds the definitions of various functions and their associated variables.These header files are imported into the C program with the help of pre-processor *#include* statement.

The basic syntax of using these header files is:

#include<file>/* This syntax searches for file-name in the standard list of system directories */
OR

#include "file"/* This technique is used to search the file(s) within the directory that contains the file where this include statement is written. */

The **custom header file** [.h extension] is a file that contains user defined functions. The Function should accept input (arguments) and generate the output after processing the received input.

For example, header file [arithmetic.h] has following functions

```
int add(int a.int b)
{
        return(a+b);
}
int mult(int a,int b)
{
        return(a*b);
}
```

- Here, functions only accept value and returns the result after processing the input.
- Functions written inside header file should not have any scanf / gets or printf / puts statements.

Do you remember how the functions already defined in C header file works?

- Let's try to explore math.h header file.
- This header file contains the function called as sqrt.
- The signature of the sqrt function is
  double sqrt(double arg);
- The sqrt function accepts one argument and returns the result. It doesn't directly print the result inside sqrt function.

Note:
Function written inside header file will only accept and return value. It will not have any type of standard input or output statements.

Assignment 04- Operation on Stack

A stack is a linear data structure in which elements are added and removed only from one end, which is called the top. Hence, a stack is called a LIFO (Last-In, First-Out) data structure as the element that is inserted last is the first one to be taken out.

1. Static Implementation:

A stack is implemented statically by using an array of size MAX to hold stack elements and integer top storing the position of topmost element of the stack. A stack represents a structure made up of both the array and the top. The stack elements can be integers, characters, strings or user defined types.

The basic operations to be performed on a stack are:

Init(S)	Create an empty stack and initializes value of top to -1 indicating the stack is empty
Push(S,x)	Insert element x into stack S
Pop(S)	Deletes and returns topmost element from stack S
Peek(S)	Returns topmost element from the stack S without deleting element.
isEmpty(S)	Checks and returns if the stack S is empty or not. Stack isempty when top==-1
isFull(S)	Checks and returns if the stack S is full or not. Stack isfull when top==MAX-1

2. Dynamic Implementation

A stack is implemented dynamically by using a Linked list where each node in the linked list has two parts, the data element and the pointer to the next element of the stack. Stack is a single entity i.e. a pointer pointing to the top node in the linked list. The stack elements can be integers, characters, strings or user defined types. Dynamic lists can grow without limits.

The operations to be performed on a stack are

Init(S)	Create an empty stack and initializing top to NULL indicating the stack is empty
S=Push(S,x)	Adding an element x to the stack S requires creation of node containing x and putting it in front of the top node pointed by S. This changes the top node S and the function should return the changed value ofS.

S=Pop(S)	Deletes top node from stack and returns. Next element will become top of stack and function returns the changed value of S.
X=Peek(S)	Returns topmost element from the stack S without deleting element.
isEmpty(S)	Checks and returns if the stack S is empty or not. Stack isempty when top==NULL

Set A

1. Implement a stack library (ststack.h) of integers using a static implementation of the stack and implementing the above six operations. Write a driver program that includes stack library and calls different stack operations.
2. Implement a stack library (dystack.h) of integers using a dynamic (linked list) implementation of the stack and implementing the above five operations. Write a driver program that includes stack library and calls different stack operations.

Set B

1. Write a program to check whether the contents of two stacks are identical. Use stack library to perform basic stack operations. Neither stack should be changed.
2. Write a program that copies the contents of one stack into another. Use stack library to perform basic stack operations. The order of two stacks must be identical.(Hint: Use a temporary stack to preserve the order).

Set C

1. In dynamic implementation of stack, How to modify pop operation so that it also returns the popped element as an argument of the pop function?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 05: Applications of Stack

Stack can be used in many applications like expression conversion, evaluation, parenthesis checking, checking string is palindrome or not etc. Infix, prefix, and postfix notations are three different but equivalent notations of writing algebraic expressions.Reversing string of characters, converting the infix notation into postfix notation and evaluating the postfix notation make extensive use of stacks as the primary tool. Following are few applications which extensively uses stack for their implementation.

1) String reverse and checking palindrome string:
A string of characters can be reversed by reading each character from a string starting from the first index and pushing it on a stack. Once all the characters have been read, the characters can be popped one at a time from the stack and then stored in the another string starting from the first index.

Algorithm to reverse the string:

1. Read the string character by character.
2. Push every character into the stack of characters.
3. When string becomes empty pop every character from stack and attach to the new string.

2) Infix to Postfix conversion:
Infix, prefix, and postfix notations are three different but equivalent notations of writing algebraic expressions. In postfix notation, operators are placed after the operands, whereas in prefix notation, operators are placed before the operands. While expression conversion precedence of operators is considered.

The precedence of operators can be given as follows:

Highest priority:	^ or $ (Exponential operator)
Higher priority:	*, /, %
Lower priority:	+, –

The algorithm accepts an infix expression that may contain operators. The algorithm uses a stack to temporarily hold operators. The postfix expression is obtained from left-to-right using the operands from the infix expression and the operators which are removed from the stack.

Algorithm for infix to postfix expression conversion:
1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else, If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
4. Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
5. If the scanned character is an '(', push it to the stack.
6. If the scanned character is an ')', pop the stack and and output it until a '(' is encountered, and discard both the parenthesis.

7. Repeat steps 2-7 until infix expression is scanned.
8. Print the output
9. Pop and output from the stack until it is not empty.

3) Evaluation of postfix string:
Postfix notations are evaluated using stacks. Every character of the postfix expression is scanned from left to right. If the character is an operand, it is pushed onto the stack. Else, if it is an operator, then the top two values are popped from the stack and the operator is applied on these values. The result is then pushed onto the stack.

Algorithm for evaluation of postfix expressions:
1) Create a stack to store operands (or values).
2) Scan the given expression and do following for every scanned element.
    2.1) If the element is a variable then , push it's value into the stack
    2.2) If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack
3) When the expression is ended, pop element from stack which is the final answer.

Set A

1. Write a program that reverses a string of characters. The function should use a stack library (cststack.h) of stack of characters using a static implementation of the stack.
2. Write a program to convert an infix expression of the form (a*(b+c)*((p-q)/r)) into its equivalent postfix notation. Consider usual precedence's of operators. Use stack library of stack of characters using static implementation.

Set B

1. A postfix expression of the form ab+cd-*ab/ is to be evaluated after accepting the values of a, b, c and d. The value should be accepted only once and the same value is to be used for repeated occurrence of same symbol in the expression. Formulate the problem and write a C program to solve the problem by using stack
2. Write a program that checks whether a string of characters is palindrome or not. The function should use a stack library (cststack.h) of stack of characters using a static implementation of thestack.

Set C

1. Write a program that checks the validity of parentheses in any algebraic expression. The function should use a stack library (cststack.h) of stack of characters using a static implementation of the stack.
2. Write a program to find all solutions to the four queens problem. Your program will need to be able to handle a search for a configuration that has no solution.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 06: Operation on Queue and Implementation of Priority queue

Queue:

A queue is a linear data structure which follows a particular order in which the operations are performed. The order is First in First out (FIFO) or Last in Last out (LILO).

In queue elements are added from one end called rear and removed from other end called front. Linear queue can be implemented statically using arrays and dynamically using linked lists.

1) Static Implementation of linear queue:
A Queue is implemented statically by using an array of size MAX to hold elements and two integers called front and rear.

A queue is a single entity that is a structure made up of the array, rear and front. Elements are added from rear end of the queue and can be deleted from front end of the queue. The 'front' stores the position of the current front element and 'rear' stores the position of the current rear element of the queue. The Queue elements can be integers, characters, strings or user defined types.

The basic operations to be performed on a Queue are:

init (Q)	Create an empty queue by initializing bothfront and rear to -1 indicating the queueisempty.
add (Q, x)	adds an element x to the rear end of the queue Q
X=delete (Q)	Deletes an element x from front end of the queue Q.
X=peek (Q)	Without deleting ,returns the front element from the queue Q
isEmpty (Q)	Checks whether the queue is empty. Queue becomes empty when rear equals to front.
isFull(Q)	Checks whether the queue is full. Queue becomes full when front reaches to MAX -1.

2. Dynamic Implementation of linear queue:
A Queue is implemented dynamically by using a Linked list. Each node in the linked list has two parts, the data element and the pointer to the next element of the queue. Since Queue should be a single entity, we need to use only one external pointer while here we need two

one for rear and one to the front. To avoid this we use a circular linked list and Queue pointer is pointing to the rear of the queue. Front can be easily accessed as it is next to rear. The Queue elements can be integers, characters, strings or user defined types. There is no restriction on how big the Queue can grow.

The operations to be performed on a Queue:

init (Q)	Create an empty queue as a circular linked list by initializing S to NULL indicating that the queue is empty

add (Q, x)	Adding an element x tso the queue Q requires creation of node containing x and putting it next to the rear and rear points to the newly added element. This changes the rear pointer Q and the function should return the changed value of Q. The function call will be asfollows
X=delete (Q)	Deletes the front node from the queue Q which is actually next element to the rear pointer Q. However if queue contains only one element, (Q->next == Q) then deleting this single element can be achieved by making empty Q (Q =NULL). Since the rear pointer Q is changed in this case, function should return the changed value of Q. The function call will be asfollows Q=delete(Q);
X=peek (Q)	returns the data element in the front (Q->next) node of the Queue Q
isEmpty (Q)	Check if the Queue is empty which is equivalent to checking if Q==NULL
isFull(Q)	Checks whether the queue is full. Queue becomes full when front reaches to MAX -1.

In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue. Even though there is space available, the overflow condition still exist because the condition REAR = MAX – 1 still holds true. This is a major drawback of a linear queue.

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle, hence making the queue behave like a circular data structure or Ring Buffer.

In case of a circular queue, head pointer will always point to the front of the queue, and tail pointer will always point to the end of the queue.

Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty.New data is always added to the location pointed by the tail pointer, and once the data is added, tail pointer is incremented to point to the next available location. In a circular queue, data is not actually removed from the queue. Only the head pointer is incremented by one position when **delete_queue()** is executed. As the queue data is only the data between head and tail, hence the data left outside is not a part of the queue anymore, hence removed. The head and the tail pointer will get reinitialized to **0** every time they reach the end of the queue. Also, the head and the tail pointers can cross each other. In other words, head pointer can be greater than the tail. Sounds odd? This will happen when we delete_queue the queue a couple of times and the tail pointer gets reinitialized upon reaching the end of the queue.

init (Q)	Create an empty queue as a circular linked list by initializing S to NULL indicating that the queue is empty

AddQueue(Q, x)	This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position. **Steps:** 1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)). 2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 &&front != 0) if it is true then set rear=0 and insert element.
X=DeleteQueue**(**Q)	This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position. **Steps:** 1. Check whether queue is Empty means check (front==-1). 2. If it is empty then display Queue is empty. If queue is not empty then step 3 3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.
X=peek (Q)	returns the data element in the front (Q->next) node of the Queue Q
isEmpty (Q)	Check if the Queue is empty which is equivalent to checking if Q==NULL

1) Priority queue:
A priority queue is a data structure in which each element is assigned a priority. The priority of the element will be used to determine the order in which the elements will be processed.

The general rules of processing the elements of a priority queue are:
1. An element with higher priority is processed before an element with a lower priority.
2. Two elements with the same priority are processed on a first-come-first-served (FCFS) basis.

A Priority Queue is different from a normal queue, because instead of being a "first-in-first-out", values come out in order by priority, the highest-priority is retrieved first.Thepriority of the element can be set based on various factors. Priority queues are widely used in operating systems to execute the highest priority process first.

2) Doubly Ended Queue (Dqueue):
Doubly Ended Queue is a Queue data structure in which the insert and delete operations can be performed at both the ends (**front** and **rear**) of the queue. That means, we can insert at both front and rear positions and can delete from both front and rear positions.
AddFront(): Adds an item at the front of Dequeue.
AddRear(): Adds an item at the rear of Dequeue.
RemoveFront(): Deletes an item from front of Dequeue.

deleteRear(): Deletes an item from rear of Dequeue.
getFront(): Gets the front element from queueue.
getRear(): Gets the last element from queueue.
isEmpty(): Checks whether Dequeue is empty or not.
isFull(): Checks whether Dequeue is full or not.

Set A

1. Implement a **linear queue library** (st_queue.h) of integers using a static implementation of the queue and implementing the above six operations. Write a program that includes queue library and calls different queueoperations
2. Implement a **circular queue library** (cir_queue.h) of integers using a dynamic (circular linked list) implementation of the queue and implementing the above five operations. Write a menu driven program that includes queue library and calls different queue operations.
3. Implement a priority queue library (PriorityQ.h) of integers using a static implementation of the queue and implementing the below two operations. Write a driver program that includes queue library and calls different queue operations.
    1) Add an element with its priorityinto the queue.
    2) Delete an element from queue according to its priority.

Set B

1. Implement a **linear queue library** (dyqueue.h) of integers using a dynamic (circular linked list) implementation of the queue and implementing the above five operations. Write a driver program that includes queue library and calls different queue operations.
2. Write a program to reverse the elements of a queue using queue library.
3. A doubly ended queue allows additions and deletions from both the ends that is front and rear. Initially additions from the front will not be possible. To avoid this situation, the array can be treated as if it were circular. Implement a queue library (dstqueue.h) of integers using a static implementation of the circular queue and implementing the nine operations :

    1) init(Q) 2) isempty(Q) 3) isFull(Q) 4) getFront(Q)
    5) getRear(Q) 6) addFront(Q,x) 7) deleteFront(Q) 8) addRear(Q, x)
    9) deleteRear(Q) .

Set C

1. Implement queue library using array or linked list. Use this queue library to simulate waiting list operations of railway reservation system.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

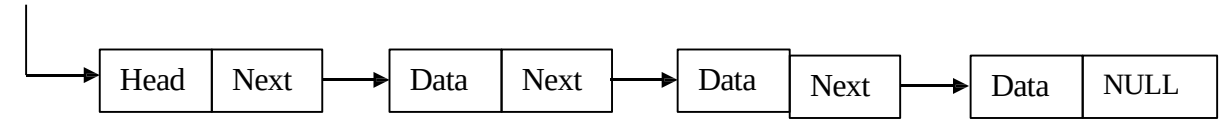
Practical In-charge

Assignment 7: Operations on Singly linked list

An abstract data type List is an ordered set of elements where insertions and deletions are possible from any position. Implementing List statically using an array to store elements is costly as insertions and deletions require moving of array elements. List is efficiently implemented dynamically using Linked list.

The linked list is a series of nodes where each node contains the data element and a link to the node containing the next element. The data element can be integer, character or user defined type. A list is a single entity which is a pointer to the first node of the linked list. A dummy node is used as header of the list so that it is not affected by insertions or deletions.

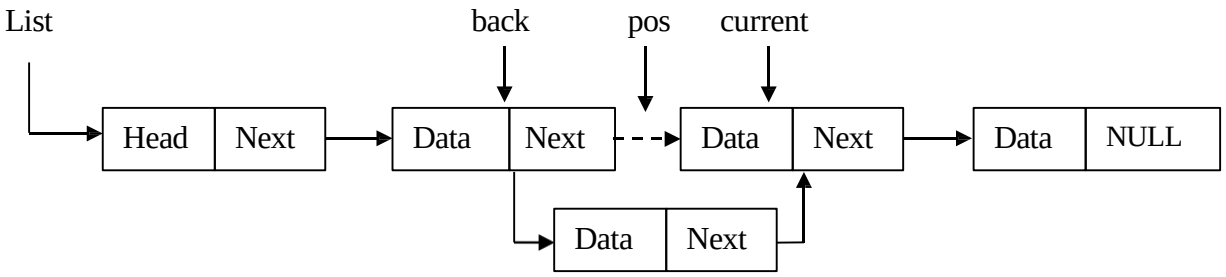
List



Operations on Linked List:

append(L, x)	Insert the data element x by creating the node containing the data element x and inserting it in last position.
insert (L, x, pos)	inserts the data element x by creating the node containing the data element x and inserting it in position pos. The links are appropriately changed. If pos equals 1, the node is inserted in first position immediately after the header. If pos is greater than the nodes present in the list, the node is added at the end of the list.
search (L, x)	Searches for the data element x and returns the pointer to the node containing x if x is present or returns NULL.
delete (L, x)	Deletes the node containing the data element x by appropriately modifying the links.
delete (L, pos)	Delete the node at a position given by pos. If the position pos is invalid then display appropriate message.
display (L)	Displays all the data elements in the list

In a singly linked list there is only a link to the next element. For insertion as well as deletion one need to traverse with two pointers back and current.



Set A

1. Implement a list library (singlylist.h) for a singly linked list with the above six operations. Write a menu driven driver program to call the operations.

2. Implement a list library (singlylist.h) for a singly linked list. Create a linked list, reverse it and display reversed linked list.

Set B

1. There are lists where insertion should ensure the ordering of data elements. Since the elements are in ascending order the search can terminate once equal or greater element is found. Implement a singly linked list of ordered integers(ascending/descending) with insert, search and display operations.
2. There are lists where new elements are always appended at the end of the list. The list can be implemented as a circular list with the external pointer pointing to the last element of the list. Implement singly linked circular list of integers with append and display operations. The operation append(L, n), appends to the end of the list, n integers either accepted from user or randomly generated.

Set C

1. How to divide a singly linked list into two almost equal size lists?
2. The union operation of two disjoint sets takes two disjoint sets S1 and S2 and returns a disjoint set S consisting of all the elements of S1 and S2 and the original sets S1 and s2 are destroyed by the union operation. How to implement union in O(1) time using a suitable list data structure for representing a set?
3. What is the method to reverse a singly linked list in just one traversal?

Assignment Evaluation

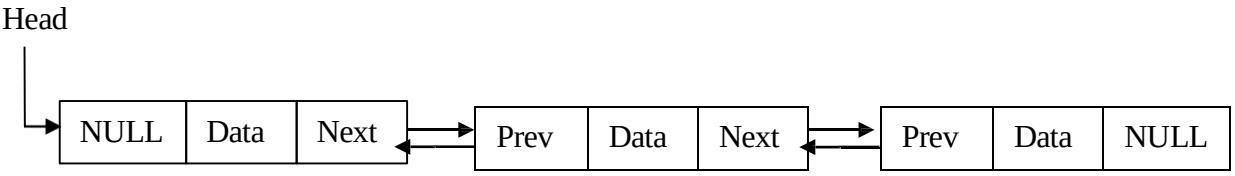
0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 8: Operations on Doubly linked list

In a Doubly linked list there is link to the next element as well as a link to the previous element. For insertion one needs only the pointer to current element and same is true for deletion.

The two links allow traversal of list in both direction i.e. forward and backward. It is easy to reverse doubly linked list as compared to singly linked list.



Set A

1. Implement a list library (doublylist.h) for a doubly linked list with the above four operations. Write a menu driven driver program to call the operationsappend, insert, delete specific node, delete at position, search, display.
2. Implement a list library (doublylist.h) for a doubly linked list. Create a linked list, reverse it and display reversed linked list.

Set B

1. There are lists where insertion should ensure the ordering of data elements. Since the elements are in ascending order the search can terminate once equal or greater element is found. Implement a doubly linked list of ordered integers(ascending/descending) with insert, search and display operations.
2. There are lists where new elements are always appended at the end of the list. The list can be implemented as a circular list with the external pointer pointing to the last element of the list. Implement doubly linked circular list of integers with append and display operations. The operation append(L, n), appends to the end of the list, n integers either accepted from user or randomly generated.

Set C

1. Which operation is performed more efficiently in doubly linked list than singly linked list?
2. What is difference between singly circular linked list and doubly circular linked list?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 09 : Tree Traversal

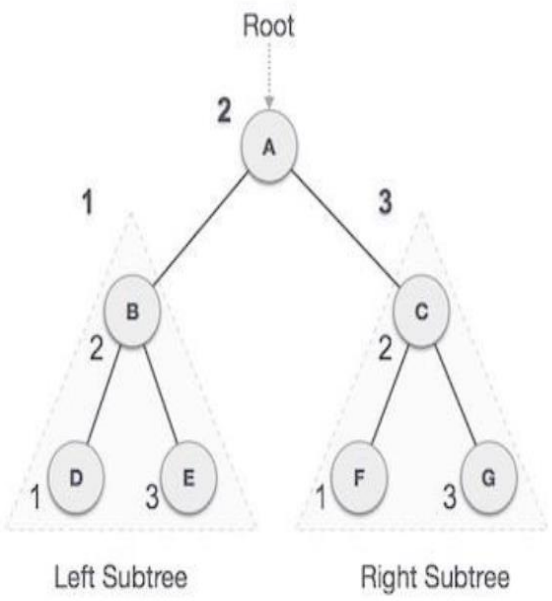
Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal
In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.



We start from A, and following in-order traversal, we move to its left subtree B.B is also traversed in-order. The process goes on until all the nodes are visited. The output of in-order traversal of this tree will be −

D → B → E → A → F → C → G

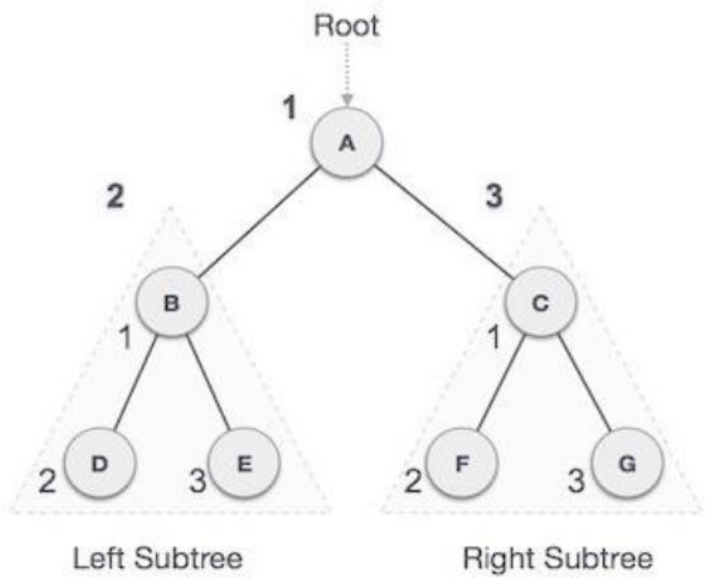
Algorithm
Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree

Post-order Traversal
In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



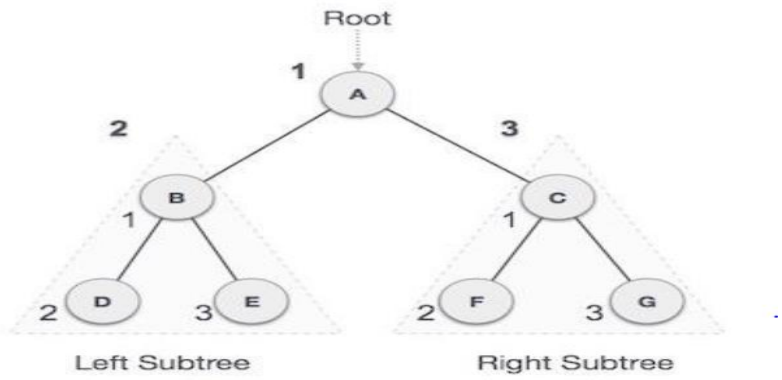
We start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B. B is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

A → B → D → E → C → F → G

Algorithm
Until all nodes are traversed −

Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.

Pre-order Traversal
In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

A → B → D → E → C → F → G

Algorithm

Until all nodes are traversed −

Step 1 − Visit root node.

Step 2 − Recursively traverse left subtree.

Step 3 − Recursively traverse right subtree

Set A

1. Write a program to accept elements of the given BST and display its inorder traversal.
2. Write a program to accept elements of the given BST and display its preorder traversal.
3. Write a program to accept elements of the given BST and display its post order traversal.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

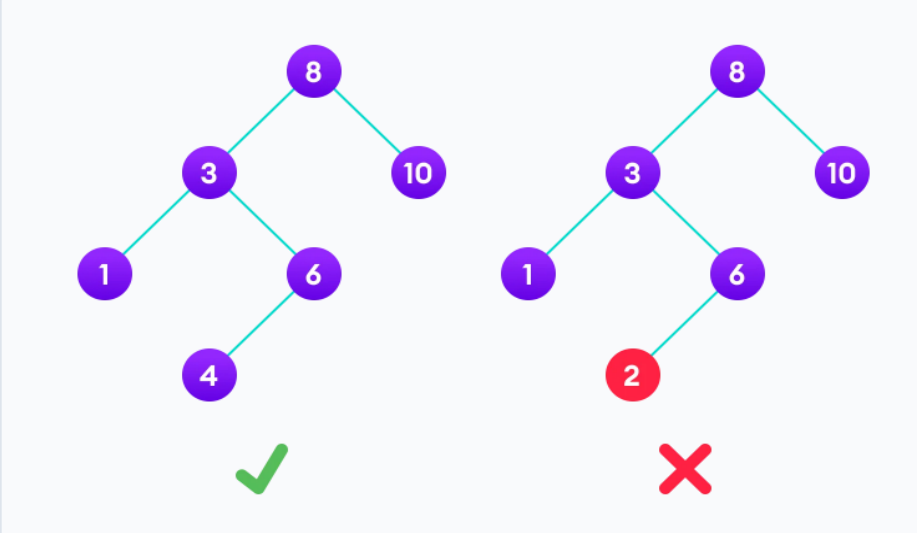
Practical In-charge

Assignment 10: Binary Search tree – create(), insert(), searc()

Binary Search Tree(BST)

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.
It is called a binary tree because each tree node has a maximum of two children.
It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.
The properties that separate a binary search tree from a regular binary tree is

All nodes of left subtree are less than the root node
All nodes of right subtree are more than the root node
Both subtrees of each node are also BSTs i.e. they have the above two properties



A tree having a right subtree with one value smaller than the root is shown to demonstrate that it is not a valid binary search tree

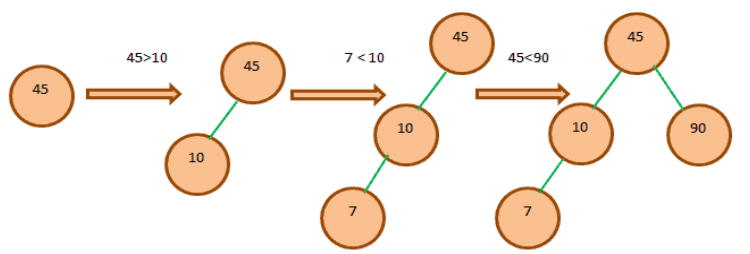
Creating A Binary Search Tree (BST)
Given an array of elements, we need to construct a BST.

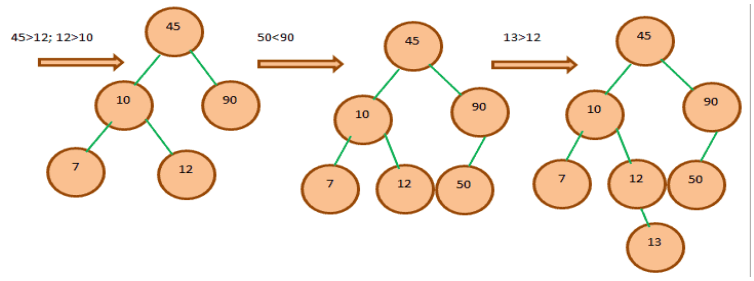
Let's do this as shown below:

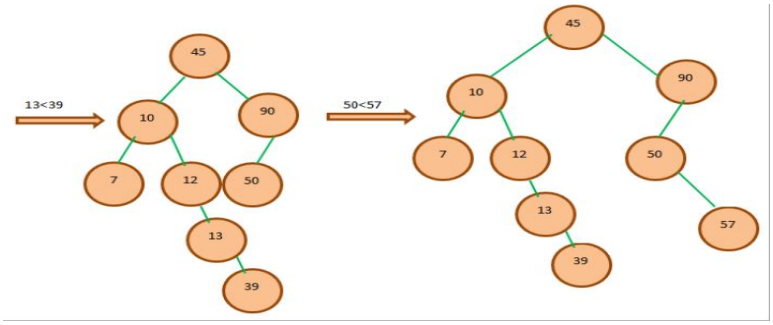
Given array: 45, 10, 7, 90, 12, 50, 13, 39, 57

Let's first consider the top element i.e. 45 as the root node. From here we will go on creating the BST by considering the properties already discussed.

To create a tree, we will compare each element in the array with the root. Then we will place the element at an appropriate position in the tree.
The entire creation process for BST is shown below.







Binary Search Tree Operations
BST supports various operations. The following table shows the methods supported by BST

Insert	Add an element to the BST by not violating the BST properties.
Delete	Remove a given node from the BST. The node can be the root node, non-leaf, or leaf node.
Search	Search the location of the given element in the BST. This operation checks if the tree contains the specified key.

Insert Operation
Inserting an element in a binary search tree is always done at the leaf node. To perform insertion in a binary search tree, we start our search operation from the root node, if the element to be inserted is less than the root value or the root node, then we search for an empty location in the left subtree, else, we search for the empty location in the right subtree.

Algorithm for inserting an element in a binary tree is as follows:

```
if node == NULL
      return createNode(data)
  if (data < node->data)
      node->left  = insert(node->left, data)
  else if (data > node->data)
      node->right = insert(node->right, data)
  return node
```

Search Operation
In Search, we have to find a specific element in the data structure. This searching operation becomes simpler in binary search trees because here elements are stored in sorted order.

Algorithm for searching an element in a binary tree is as follows:

Compare the element to be searched with the root node of the tree.
If the value of the element to be searched is equal to the value of the root node, return the root node.
If the value does not match, check whether the value is less than the root element or not if it is then traverse the left subtree.
If larger than the root element, traverse the right subtree.
If the element is not found in the whole tree, return NULL.

Time Complexity

Insertion, Deletion, Search is $O(logn)$ for Best Case and Average Case

Set A:

1. Write a program to create a binary search tree.Key values will be provided by user in runtime.
2. Write a program to search the given value in a binary search tree

Set B:

1. Write a program to insert the given value in a binary search tree.
2. Write a program to delete the given value in a binary search tree.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge