SINHGAD ACADEMY OF ENGINEERING

LAB MANUAL

Subject: Data Structure & Algorithms Lab (214447)

Department of Information Technology

ASSIGNMENT LIST

Assignment No.	Name of the Assignment /Experiment
1	**Searching and Sorting** Consider a student database of SEIT class (at least 15 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of structure)
2	**Stack** Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix and prefix expression.
3	**Circular Queue** Implement Circular Queue using Array. Perform following operations on it. a) Insertion (Enqueue) b) Deletion (Dequeue) c) Display
4	**Expression Tree** Construct an Expression Tree from postfix and prefix expression. Perform recursive and non- recursive In-order, pre-order and post-order traversals.
5	**Binary Search Tree** Implement binary search tree and perform following operations: a) Insert (Handle insertion of duplicate entry) b) Delete c) Search d) Display tree (Traversal) e) Display - Depth of tree f) Display - Mirror image g) Create a copy h) Display all parent nodes with their child nodes

	i) Display leaf nodes j) Display tree level wise
6	**Threaded Binary Tree** Implement In-order Threaded Binary Tree and traverse it in In-order and Pre-order.
7	**Graph: Minimum Spanning Tree** Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find minimum spanning tree a) Using Kruskal's algorithm. b) Using Prim's algorithm.
8	**Graph: Shortest Path Algorithm** Represent a graph of city using adjacency matrix /adjacency list. Nodes should represent the various .
9	**Heap Sort** Implement Heap sort to sort given set of values using max or min heap.
10	**FILE Handling** Department maintains student's database. The file contains roll number, name, division and address. Write a program to create a sequential file to store and maintain student data. It should allow the user to add, delete information of student. Display information of particular student. If record of student does not exist an appropriate message is displayed. If student record is found it should display the student details.

Course Objectives

1. To study data structures and their implementations and applications.
2. To learn different searching and sorting techniques.
3. To study some advanced data structures such as trees, graphs and tables.
4. To learn different file organizations.
5. To learn algorithm development and analysis of algorithms.

The CO-PO mapping for the course												
PO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	3	3	2	3	-	3	-	-	-	-	-	-
CO2	1	3	3	3	-	3	-	-	-	-	-	-
CO3	2	1	2	3	-	3	-	-	-	-	-	-
CO4	2	3	3	3	-	3	-	-	-	-	-	-
CO5	3	3	2	3	-	3	-	-	-	-	-	-
CO6	1	3	3	3	-	3	-	-	-	-	-	-

Assignment No	**01**
Title	**Searching & Sorting**
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

Assignment No : 01

Title : Bubble sort, Quick Sort, & Binary Search

Problem Statement :

Consider a student database of SEIT class (at least 15 records) .

Database contains different fields of every student like Roll No, Name and SGPA.(array of structure) a) Design a roll call list, arrange list of students according to roll numbers in ascending order (Use Bubble Sort)

b) Arrange list of students alphabetically. (Use Insertion sort)

c) Arrange list of students to find out first ten toppers from a class. (Use Quick sort)

d) Search students according to SGPA. If more than one student having same SGPA, then print list of all students having same SGPA.

e) Search a particular student according to name using binary search without recursion. (all the student records having the presence of search key should be displayed) (Note: Implement either Bubble sort or Insertion Sort.)

Theory :

Array of structure:

C/C++ arrays allow you to define variables that combine several data items of the same kind, but structure is another user defined data type which allows you to combine data items of different kinds.

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member, for your program. The format of the struct statement is this −

```
struct [structure tag] {
  member definition;
  member definition;
  ...
  member definition;
```

} [one or more structure variables];

Array of structure example:

```
typedef struct SEIT_student_Details
{
   int roll;  // 4 bytes
   char name[10]; //10 bytes
   float SGPA; //4 bytes
}stud;

stud s[2]; //array of structure
```

---

roll	name	SGPA
4 bytes	10 bytes	4 bytes
1000-1003	1004-1013	1014-1017
Total=18 bytes		

//array of structure

int arr[10]; //int array

stud s[2]; //size=36 bytes =18*2=36 bytes

S[0]			S[1]		
roll	name	SGPA	roll	name	SGPA
4 bytes	10 bytes	4 bytes	4 bytes	10 bytes	4 bytes
1000	1004	1014	1018-1021	1022-1031	1032-1035
18 bytes			18 bytes		
Total=36 bytes					

Searching : It is a process of retrieving or locating a record with a particular key value.

Binary Search :

This is a very efficient searching method used for linear / sequential data. In this search, Data has to be in the sorted order either ascending or descending. Sorting has to be done based on the key values.

. If the key is less than the record key, the search proceeds in the left half of the table. If the key is greater than In this method, the required key is compared with the key of the middle record. If a match is found, the search terminates record key, search proceeds in the same way in the right half of the table. The process continues till no more partitions are possible. Thus every time a match is not found, the remaining table size to be searched reduces to half.

As the process continues, we will eliminate from consideration, one-half of what is left of the list with

each comparison. This technique is called as binary search.

Advantage : Time complexity is O(logn) which is very efficient.

Disadvantage : Data has to be in sorted manner.

Analysis:

After 0 comaparisons Remaining file size = n

After 1 comaparisons Remaining file size = $n / 2 = n / 2^1$

After 2 comaparisons Remaining file size = $n / 4 = n / 2^2$

After 3 comaparisons Remaining file size = $n / 8 = n / 2^3$

……

After k comaparisons Remaining file size = $n / 2^k$

The process terminates when no more partitions are possible i.e. remaining file size = 1

$n / 2^k = 1$

$k = \log_2 n$

Thus, the time complexity is $O(\log_2 n)$. which is very efficient.

Sorting : It is the process of arranging or ordering information in the ascending or descending order of the key values.

Bubble Sort :

This is one of the simplest and most popular sorting methods. The basic idea is to pass through the array sequentially several times. In each pass we compare successive pairs of elements (A[i] with A[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 the 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.

Advantages:

It is a simple sorting method.

No additional data structure is required.

Disadvantage:

It is very inefficient method – O(n2)

Even if the elements are in the sorted order, all n-1 passes will be done.

Analysis :

In this sort, n-1 comparisons takes place in 1^{st} pass , n-2 in 2^{nd} pass , n-3 in 3^{rd} pass , and so on.

Therefore total number of comparisons will be

(n-1) + (n-2) + (n-3) + ……. + 3 + 2 + 1

This is an Arithmetic series in decreasing order

Sum = n/2 [2a + (n-1)d]

Where, n = no. of elements in series , a = first element in the series

d = difference between second element and first element.

Sum = (n-1) / 2 [2 * 1 + ((n-1) - 1) * 1]

= n(n-1) / 2

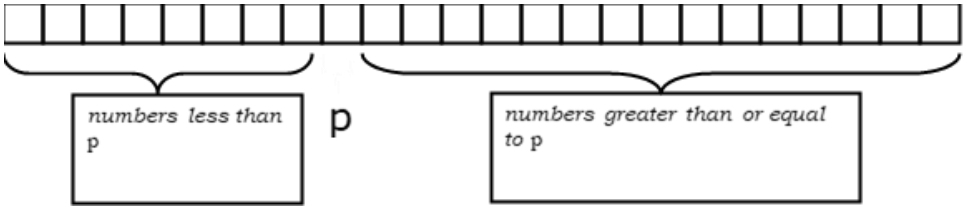
which is of order n2 i.e $O(n^2)$

The main advantage is simplicity of algorithm and it behaves as O(n) for sorted array of element. Additional space requirement is only one temporary variable.

Quick Sort also known as partition exchange sort. Original algorithm was developed by C.A.R. Hoare in 1962. Quicksort sorts by employing a divide and conquer strategy to divide a list into two sub-lists.It is one of the fastest sorting algorithms available. QuickSort is especially convenient with large arrays that contain elements in random order.

The steps are:

1. Pick an element, called a pivot, from the list.
2. Reorder the list so that all elements which are less than the pivot, come before the pivot and all elements greater than the pivot come, after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation. -
3. Recursively sort the sub-list of smaller elements and the sub-list of greater elements.



Thus quicksort is an recursive sorting algorithm which chooses an element of the list, called the pivot element, and then rearranges the list so that all of the elements smaller than the pivot are moved before it and all of the elements larger than the pivot are moved after it.

Advantage:

- Efficient sorting technique
- Time complexity is O(nlogn)

Disadvantage:

- Worst case time complexity is O(n2)
- Time requirement depends on the position of the pivot in the list.

Analysis of Quicksort:

Time requirement of Quicksort depends on the position of pivot in the list, how pivot is dividing list into sublists. It may be equal division of list or may be it will not divide also.

Best case:

In Best case, we assume that list is equally divided means, list1 is equally divided in two sublists, these two sublists in four sublists, and so on. So the total no. of elements at particular level (l) will be 2^{l-1} so total number of steps will be $\log_2 n$. The no. of comparison at any level will be maximum n. so we say run time of Quicksort will be $\Omega(n\log 2n)$.

Worst Case:

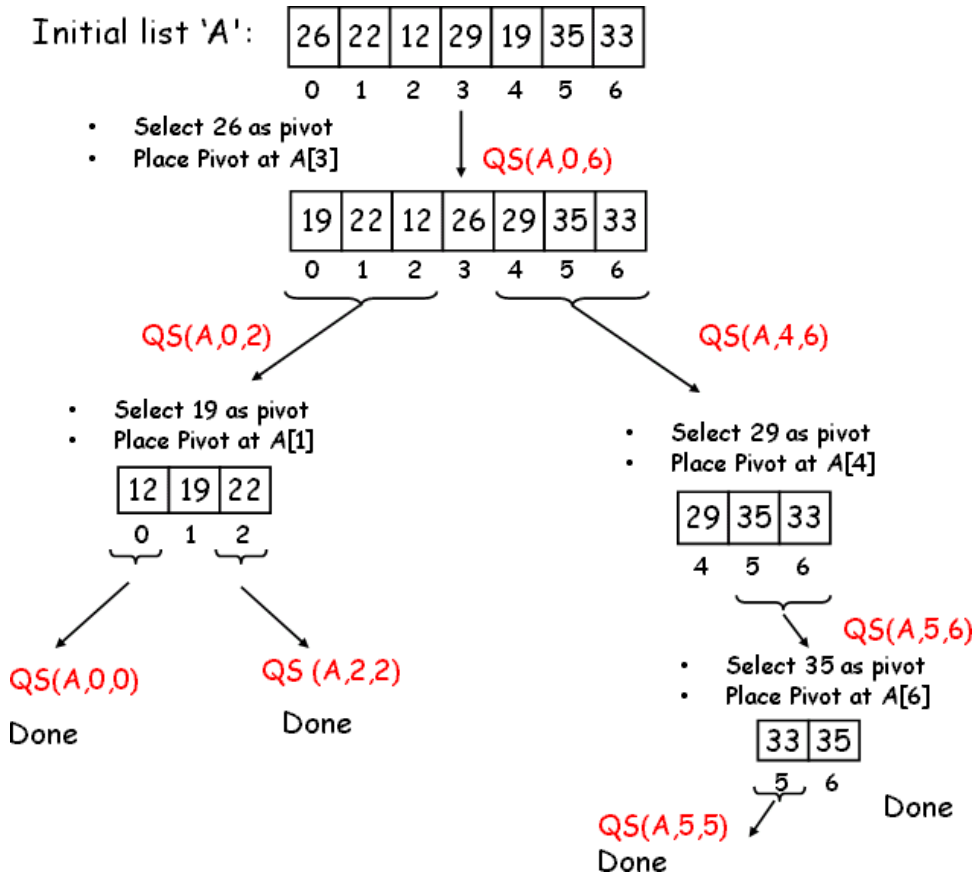
Suppose list of elements are already in sorted order. When we find the pivot then it will be first element. So here it produces only 1 sublist which is on right side of first element and starts from second element. Similarly other sublists will be created only at right side. The no. of comparisons for first element is n, second element requires n-1 comparisons and so on.. So the total no. of comparisons will be n + n-1 + …. 2 + 1

= n (n-1) / 2 = $O(n^2)$.

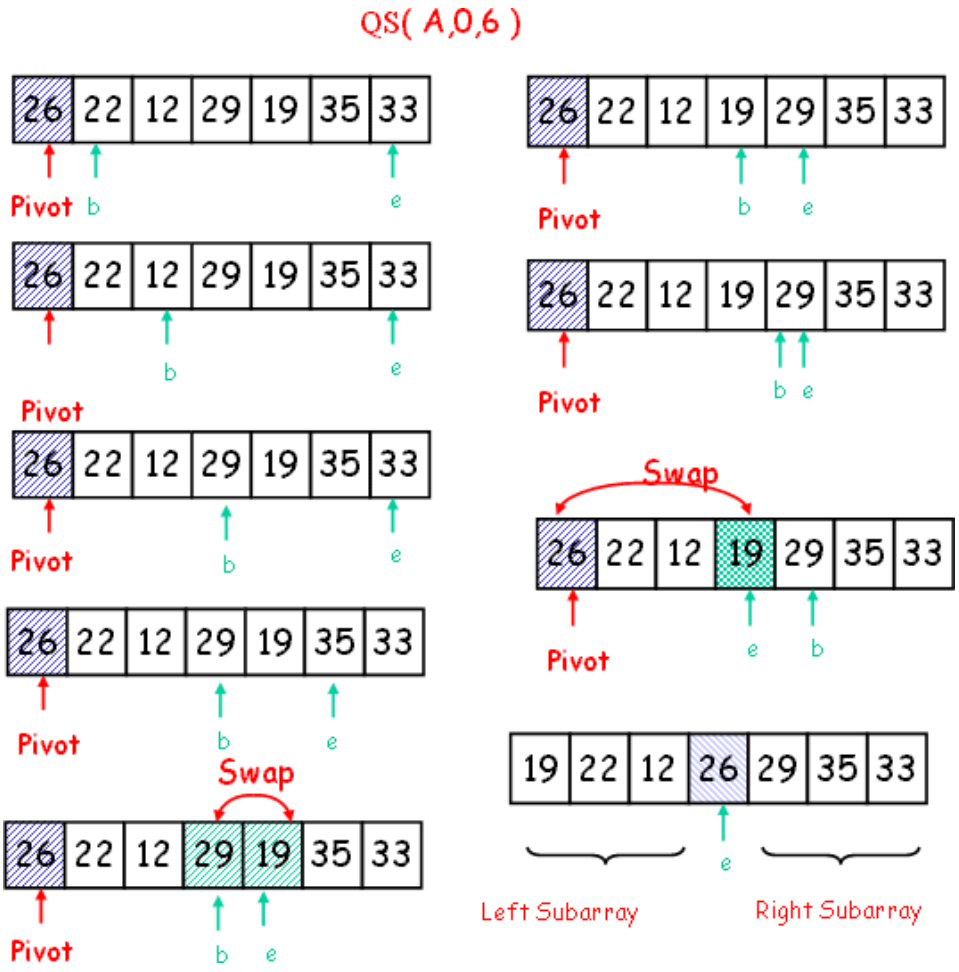
Average Case :

The average case performance of QuickSort is Θ(nlogn)

Example:



Step by Step Process for QS (A, 0, 6)


QS(A,0,6)
26 22 12 29 19 35 33
Pivot b e
26 22 12 29 19 35 33
b e
Pivot
26 22 12 29 19 35 33
Pivot b e
26 22 12 29 19 35 33
Pivot b e
26 22 12 29 19 35 33
Pivot b e
Swap
26 22 12 19 29 35 33
Pivot e b
26 22 12 29 19 35 33
Pivot b e
Swap
26 22 12 29 19 35 33
Pivot b e
19 22 12 26 29 35 33
e
Left Subarray
Right Subarray

Algorithms :

Algorithm for Quicksort

Algorithm Quicksort (int A[], int s, int l) where A is the array , s is the index of starting element and l is the index of last element.

```
1       Start
2       Select A[s]  as the pivot element
3       Set b = s + 1
        Set e = l
5       While  b ≤ e
6           Increment b till A[b] ≤ pivot element
7           Decrement e till A[e] > pivot element
8           If b < e
9               Swap(A[b], A[e])
10          End if
11      End while
12      Swap( A[s] , A[e]) // Place the pivot element at index e
13      If s < e-1
14          Recursively call Quicksort(A,s,e-1)
15      End if
16      If e+1 < l
17          Recursively call Quicksort(A,e+1,l)
18      End if
19      Stop
```

Algorithm for Binary search (Iterative)

Algorithm Binary_Search (A, n , Key) Where A is an 2D character array , n is the no of records, and key iselement to be searched	
1	Start

2	Set b = 0 & e = n-1
3	While b ≤ e
4	mid = (b + e) / 2
5	If A[mid] == key (compare two strings for equality)
6	Return 1 // Found
7	Else
8	If key < A[mid]
9	e = mid – 1
10	Else
11	b = mid + 1
12	End if
13	End if
14	End while
15	Return 0 // Not found

Algorithm for Binary search (Recursive)

Algorithm Binary_Search (A, b , e, Key) Where A is an 2D character array , b is the index of startingelement and e is the index of last element in the array and key is element to be searched	
1	If b > e
2	Return 0 // Not Found
3	Else
4	mid = (b + e) / 2
5	If A[mid] == key (Compare two strings for equality)
6	Return 1 // Found
7	Else
8	If key < A[mid]
9	Recursively call Binary_search(A, b,mid – 1, key) and return the returned value
10	Else
11	Recursively call Binary_search(A, mid + 1, e,key) and return the returned value
12	End if
13	End if
14	End if

Algorithm for Bubble Sort

Algorithm Bubble_Sort (A,n) Where A is an 2D character array , n is the total number of strings to sort	
1	Start
2	For Pass = 1 to n-1

3	For i = 0 to (n – pass – 1)
4	If A[i] < A[i+1] (Check is A[i] string is less than A[i+1] string)
5	Swap (A[i], A[i+1])
6	End if
7	End for
8	End For
9	Stop

Conclusion:

Thus we have implemented bubble sort , selection sort and binary search technique and analyzed the results for best, worst and average cases.

FAQs:

Q1. List the advantages and disadvantages of Binary Search?

Q2. Explain the analysis of Binary Search with respect to best case, worst case & Average case?

Q3. What is In-place & Stable Features of Sorting Algorithms?

Q4. Explain what you mean by Internal & External Sorting.

Q5. Explain the advantages and disadvantages of bubble sort

Q6. Analyze Bubble sort with respect to Time complexity?

Q7. How to improve the efficiency of Bubble sort if the array gets sorted before the n-1 passes?

Q8. What is sorting stability?

Q9. Explain the analysis of Selection Sort?

Q10. Give applications of Binary Search?

Note: Don't write FAQs .

Assignment No	**02**
Title	Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix and prefix expression
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

2. Implementation of Stack

AIM: Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix and prefix expression

.

OBJECTIVE:

1) To understand the concept of abstract data type.
2) How different data structures such as arrays and a stacks are represented as an ADT.

THEORY:

1) What is an abstract data type?

An Abstract Data type is defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects. There are no standard conventions for defining them. A broad division may be drawn between "imperative" and "functional" definition styles. In general terms, an abstract data type is a *specification* of the values and the operations that has two properties:

- It specifies everything you need to know in order to use the data type
- It makes absolutely no reference to the manner in which the data type will be implemented.

When we use abstract data types, our programs divide into two pieces:

- The Application: The part that uses the abstract data type.
- The Implementation: The part that implements the abstract data type.

What is stack? Explain stack operations with neat diagrams.

In computer science, a **stack** is a last in, first out (LIFO) abstract data type and data structure. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: push and pop. The push operation adds an item to the top of the stack, hiding any items already on the stack, or initializing the stack if it is empty. A pop either reveals previously

concealed items, or results in an empty stack. A stack is a restricted data structure, because only a small number of operations are performed on it. The nature of the pop and push operations also mean that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition: therefore, the lower elements are those that have been on the stack the longest. A collection of items in which only the most recently added item may be removed. The latest added item is at the top. Basic operations are push and pop. Often top and isEmpty are available, too. Also known as "last-in, first-out" or LIFO.

Operations

An abstract data type (ADT) consists of a data structure and a set of **primitive**

- **Push** adds a new element
- **Pop** removes a element

Additional primitives can be defined:

- **IsEmpty** reports whether the stack is empty
- **IsFull** reports whether the stack is full
- **Initialise** creates/initialises the stack
- **Destroy** deletes the contents of the stack (may be implemented by re-initialising
- the stack)

Explain how stack can be implemented as an ADT.

User can Add, Delete, Search, and Replace the elements from the stack. It also

checks for Overflow/Underflow and returns user friendly errors. You can use this stack implementation to perform many useful functions. In graphical mode, this C program displays a startup message and a nice graphicto welcome the user.

The program performs the following functions and operations:

- **Push**: Pushes an element to the stack. It takes an integer element asargument. If the stack is full then error is returned.
- **Pop**: Pop an element from the stack. If the stack is empty then error isreturned. The element is deleted from the top of the stack.
- **DisplayTop** : Returns the top element on the stack without deleting. Ifthe stack is empty then error is returned.

ALGORITHM:

Abstract Data Type Stack:

Define Structure for stack(Data, Next Pointer)

Stack Empty:

Return True if Stack Emptyelse
False.
Top is a pointer of type
structure stack.
Empty(Top)
Step 1: If Top
= = NULL
Step 2: Return
1;
Step 3: Return 0;

Push Opertion:

Top & Node pointer of
structure Stack.
Push(element)

Step1:

Node->data=element;

Step 2:

Node->Next =top;

Step 3: Top =Node

Algorithm for infix to postfix conversion

Initialize result as a blank string, Iterate through given expression, one character ata time

1. If the character **is an operand, add it to the result.**
2. **If the character is an operator.**
    - If the operator stack is empty then push it to the operator stack.
    - Else If the operator stack is not empty,
        - If the **operator's precedence is greater than or equal to the precedence of the stack top of the operator stack, then push the character to the operator stack.**

          (ICP >= ISP) -------->push

        - If the **operator's precedence is less than the precedence of the stack top of operator stack then "*pop*** *out an operator from the stack and add it to the result until the stack is empty oroperator's precedence is greater than or equal to the precedence of the stack top of operator stack*". then push the operator to stack.
        - (**ICP<ISP) ------>POP**
3. If the character is "(", then push it onto the operator stack.
4. If the character is ")", then "*pop out an operator from the stack and add it tothe result until the corresponding "(" is encountered in operator stack. Now just pop out the "(".*

A * B + C			
current symbol	**operator stack**	**postfix string**	**OUTPUT**
1	**A**		**A**
2	*****	*****	**A**
3	**B**	*****	**A B**
4	**+**	**+**	**A B * {pop and print the '*' before pushing the '+'}**
5	**C**	**+**	**A B * C**
6			**A B * C +**

A + B * C			
current symbol	**operator stack**	**postfix string**	**OUTPUT**
1	**A**		**A**
2	**+**	**+**	**A**
3	**B**	**+**	**A B**
4	*****	**+ ***	**A B**
5	**C**	**+ ***	**A B C**
6			**A B C * +**

Algorithm to convert infix expression to prefix

To convert an infix to prefix expression alogorithm

Step 1:

Reverse the infix expression i.e A+B*C will become C*B+A. Note whilereversing each '(' will become ')' and each ')' becomes '('.

Step 2:

Obtain the postfix expression of the modified expression i.e CB*A+.

Step 3:

Reverse the postfix expression. Hence in our example prefix is +A*BC.

(a+b)*c			
Step 1) c*(b+a) **Step 2)**			
	current symbol	**operator stack**	**postfix string**
1	**c**		**c**
2	*****	*****	**c**
3	**(**	**(** *****	**c**
4	**b**	**(** *****	**cb**
5	**+**	**+** **(** *****	**cb**
6	**a**	**+** **(** *****	**cba**
7	**)**	*****	**Cba+**
8	**\0**		**Cba+***

Step 3:Reverse string- *+abc

Algorithm for postfix Evalution

EVALUATE_PREFIX(STRING)

Step 1: Put a pointer P at the end of the end

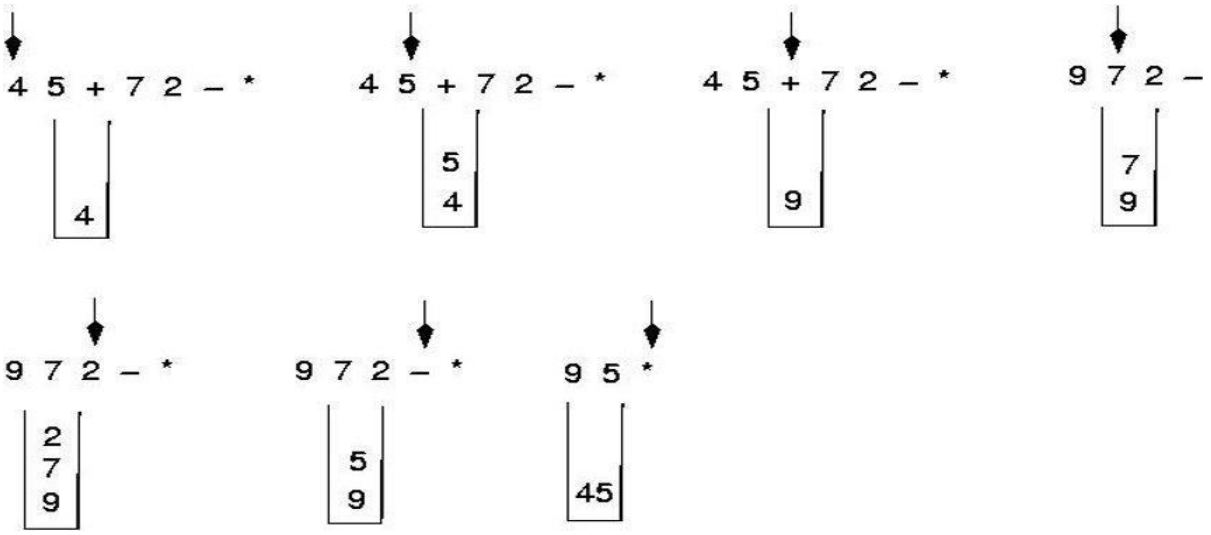
Step 2: If character at P is an operand push it to Stack

Step 3: If the character at P is an operator pop two elements from the Stack. Operate on these elements according to the operator, and push the result back to the Stack

Step 4: Decrement P by 1 and go to Step 2 as long as there are characters left to be scanned in the expression.

Step 5: The Result is stored at the top of the Stack, return it

Step 6: End



Algorithm for prefix Evalution

EVALUATE_PREFIX(STRING)

Step 1: Put a pointer P at the end of the end

Step 2: If character at P is an operand push it to Stack

Step 3: If the character at P is an operator pop two elements from the Stack. Operate on these elements according to the operator, and push the result back to the Stack

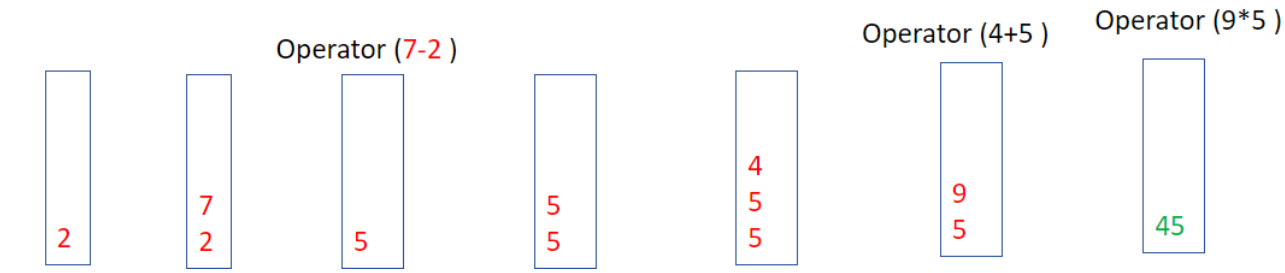
Step 4: Decrement P by 1 and go to Step 2 as long as there are characters left to be scanned in the expression.

Step 5: The Result is stored at the top of the Stack, return it

Step 6: End

Prefix evaluation

- * + 4 5 –7 2



Conclusion:

Thus we implemented infix to postfix and prefix using stacksuccessfully

FAQ:

1. What is data structure?
2. What are Types of data structure?
3. Examples of linear data structure & Non-linear data structure?
4. What are the operations can implement on stack?
5. Explain recursion using stack.
6. Explain how a string can be reversed using stack.
7. How does a stack similar to list? How it is different?
8. List advantages and disadvantages of postfix and prefix expressionover infix expression.

Assignment No	**03**
Title	**Circular Queue**
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

Assignment No : 03

Title : Circular Queue

Problem Statement :

Implement Circular Queue using Array. Perform following operations on it. a) Insertion (Enqueue) b) Deletion (Dequeue) c) Display (Note: Handle queue full condition by considering a fixed size of a queue.)

Theory :

What is Queue?

Queue is list of elements where elements are inserted at one end (rear) & deleted from other end (front).Queue is FIFO data structure.The element that is inserted first into the queue will be the element that will deleted first, and the element that is inserted last is deleted last.A waiting line is a good real- life example of a queue.

Types for Queues:

1) Linear queue
2) Circular queue
3) Priority queue
4) Dequeue
5) Multiqueue

Operations on Queues

- **enqueue**(item):
  - It adds a new item to the rear of the queue
- **dequeue( )**:
  - It deletes the front item of the queue, and returns to the caller.
- **isEmpty( )**
  - Returns true if the queue has no items

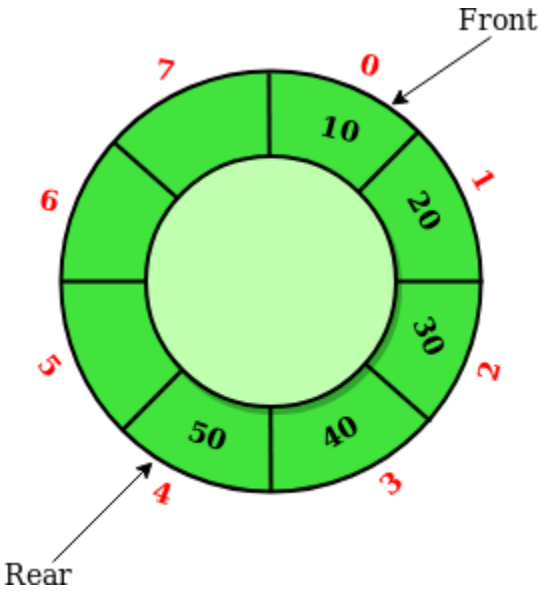
- **isFull()**
- **getfront( ):**
    - Returns true if the queue is full

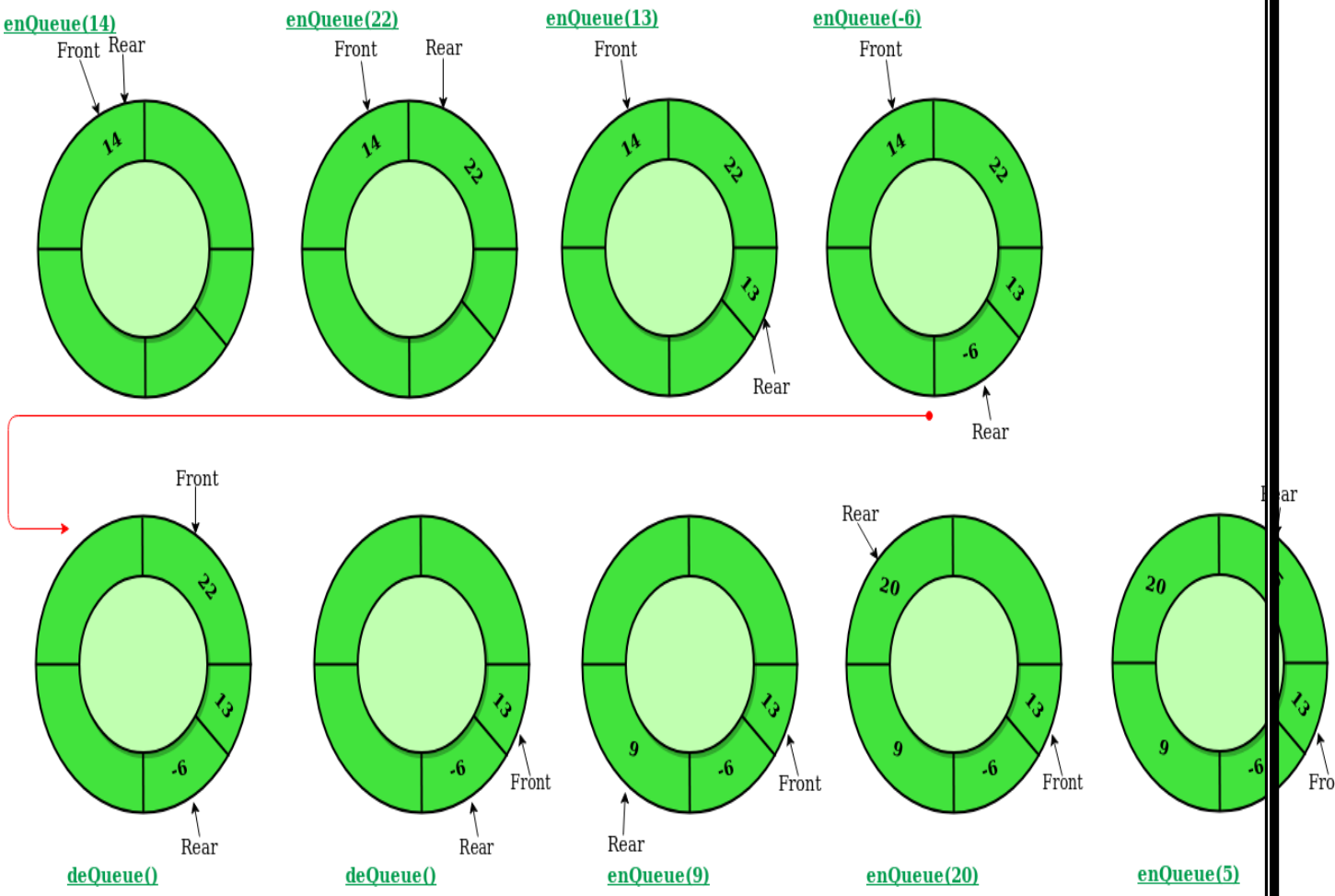
- Returns the value in the front element of the queue

- **getrear( ):**
  - Returns the value in the rear element of the queue

- **size( )**
  - Returns the number of items in the 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. It is also called 'Ring Buffer'.




enQueue(14)
Front
Rear
14
enQueue(22)
Front
Rear
14
22
enQueue(13)
Front
14
22
13
Rear
enQueue(-6)
Front
14
22
13
-6
Rear
Front
22
13
-6
Rear
deQueue()
13
-6
Front
Rear
deQueue()
9
13
-6
Front
Rear
enQueue(9)
20
9
13
-6
Rear
Front
enQueue(20)
20
9
13
-6
Front
enQueue(5)

Implementation of Circular Queue

To implement a circular queue data structure using an array, we first perform the following steps before we implement actual operations.

Step 1 - Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.

Step 2 - Declare all **user defined functions** used in circular queue implementation.

Step 3 - Create a one dimensional array with above defined SIZE (**int cQueue[SIZE]**)

Step 4 - Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)

Step 5 - Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

enQueue(value) - Inserting value into the Circular Queue

In a circular queue, enQueue() is a function which is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as parameter and inserts that value into the circular queue. We can use the following steps to insert an element into the circular queue...

Step 1 - Check whether **queue** is **FULL**. (**(rear == SIZE-1 && front == 0) || (front == rear+1)**)

Step 2 - If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT FULL**, then check **rear == SIZE - 1 && front != 0** if it is **TRUE**, then set **rear = -1**.

Step 4 - rear = (rear + 1) % SIZE; set **queue[rear]** = **value** and check '**front == -1**' if it is **TRUE**, then set **front = 0**.

+

deQueue() - Deleting a value from the Circular Queue

In a circular queue, deQueue() is a function used to delete an element from the circular queue. In a circular queue, the element is always deleted from **front** position. The deQueue() function doesn't take any value as a parameter. We can use the following steps to delete an element from the circular queue...

Step 1 - Check whether **queue** is **EMPTY**. (**front == -1 && rear == -1**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then display **queue[front]** as deleted element and

front = (front + 1) % SIZE;

. Then check whether **front == SIZE**, if it is **TRUE**, then set **front = 0**.

Then check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

Conclusion:

Thus we have implemented circular queue successfully.

FAQs:

Q1. Define queue.

Q2. What are different types of queues.

Q3. What is circular queue?

Q4. What are applications of circular queue?

Q5. What are applications of circular queue?

Q6. Explain ADT of Queue

Q7. Explain different functions of circular queue.

Note: Don't write FAQs .

Assignment No	**04**
Title	**Expression tree Traversals**
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

04. Expression tree Traversals

AIM: Construct and expression tree from postfix/prefix expression and perform recursive and non-recursive In-order, pre-order and post-order traversals.

OBJECTIVE:

a. Understand the concept of expression tree and binary tree.
b. Understand the different type of traversals (recursive & non-recursive).

THEORY:

- **Definition of an expression tree with diagram.**

Algebraic expressions such as a/b + (c-d) e

÷The terminal nodes (leaves) of an expression tree are the variables or constants in the expression (a, b, c, d, and e). The non-terminal nodes of an expression tree are the

×operators (+, -, , and). Notice that the parentheses which appear in Equation do not appear in the tree. Nevertheless, the tree representation has captured the intent of the parentheses since the subtraction is lower in the tree than the multiplication.

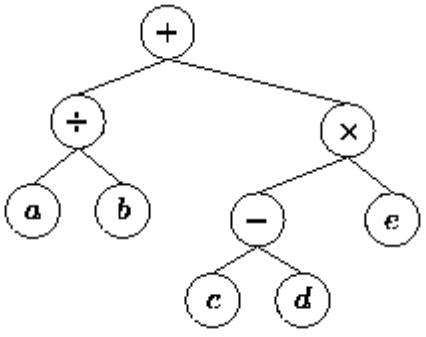


Figure: Tree representing the expression a/b+(c-d)e.

Show the different type of traversals with example

To traverse a non empty binary tree in pre order.

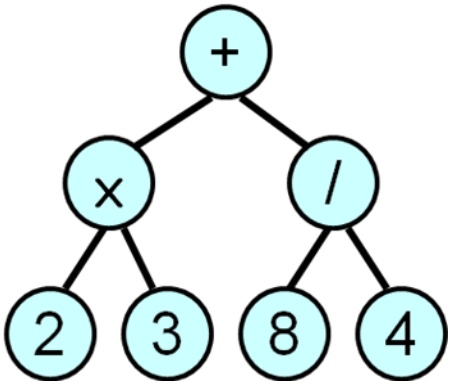
1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right sub tree.

To traverse a non empty binary tree in in order.

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right sub tree.

To traverse a non empty binary tree in post order.

1. Traverse the left subtree
2. Traverse the right subtree.
3. Visit the root.



Pre-order (prefix)

+ □ 2 3 / 8 4

- **In-order (infix)**

2 □ 3 + 8 / 4

- **Post-order (postfix)**

2 3 □ 8 4 / +

ALGORITHM:

Define structure for Binary Tree (Information, Left Pointer & Right Pointer)

Create Expression Tree:

CreateTree()

Root& Node pointer variable of type structure. Stack is an pointer array of type structure.String is character array which contains postfix expression. Top & I are variables of type integer.

Step 1: Top = -1 , I = 0;

Step 2: Do Steps 3,4,5,6 While String[I] != NULL

Step 3: Create Node of type of structure

Step 4: Node->Data=String[I];

Step 5: If isalnum(String[I])

Then Stack[Top++] = Node;

Else

Node->Right = Stack [--Top];

Node->Left = Stack[--Top];

Stack[Top++] = Node;

Step 6: Increment I;

Step 7: Root = Stack[0];

Step 8: Return Root

Inorder Traversal Recursive :

Tree is pointer of type structure.

InorderR(Tree)

Step 1: If Tree != NULL

Step 2: InorderR(Tree->Left)

Step 3: Print Tree->Data

Step 4: InorderR(Tree->Right)

Postorder Traversal Recursive:

Tree is pointer of type structure.

PostorderR(Tree)

Step 1: If Tree != NULL

Step 2: PostorderR(Tree->Left)

Step 3: PostorderR(Tree->Right)

Step 4: Print Tree->Data

Preorder Traversal Recursive:

Tree is pointer of type structure.PreorderR(Tree)

Step 1: If Tree != NULL Step

2: Print Tree->Data

Step 3: PreorderR(Tree->Left)

Step 4: PreorderR(Tree->Right)

Postorder Traversal Nonrecursive :

NonR_Postorder(Tree)

Tree, Temp is pointer of type structure. Stack is pointer array of type structure.Top variableof type integer.

Step 1: Temp = Tree // current pointer pointing to root

Step 2: if Temp != NULL then push current pointer along with its initial flag value _L' onto the stack and traverse on the left side.

Step 3: Otherwise if the stack is not empty then pop an address from stack along withits flag value.

Step 4: For the current pointer if the flag value is _L' then change it to _R' and push current pointer along with its flag value _R' on to the stack and traverse on theright side.

Step 5: otherwise, For the current pointer if the flag value is _R' then display the element and make the current pointer NULL (i.e.temp=NULL)

Step 6: repeat steps 2, 3, 4, 5 until the stack becomes empty.

Preorder Traversal Nonrecursive :

NonR_Preorder(Tree)

Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Topvariable of type integer.

Step 1: Temp = Tree

Step 2: Do Steps3,4,5,6,7,& 8 While Temp != NULL And Stack is not Empty Step3: Do

Steps 4,5&6 While Temp != NULL

Step 4: Print Temp->Data

Step 5: Stack[++ Top] = Temp //Push ElementStep

6: Temp = Temp->Left

Step 7: Temp = Stack [Top --] //Pop ElementStep

8: Temp = Temp->Right

Inorder Traversal Nonrecursive :

NonR_Inorder(Tree)

Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Top variableof type integer.

Step 1: Temp = Tree

Step 2: Do Steps3,4,5,6,7,&8 While Temp != NULL And Stack is not EmptyStep

3: Do Steps 4,5 While Temp != NULL

Step 4: Stack[++ Top] = Temp;//Push ElementStep

5: Temp = Temp->Left

Step 6: Temp = Stack[Top --]//Pop Element

Step 7: Print Temp->Data

Step 8: Temp = Temp->Right

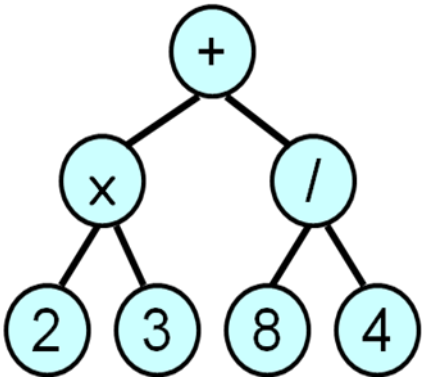
INPUT:

Postfix Expression:2 3 □ 8 4 / +

OUTPUT:

Display result of each operation with error checking.

Expression tree



Conclusion:

Thus we have implemented Constructed and expression tree from postfix/prefix expression and perform recursive and non- recursive In-order, pre-order and post-order traversals.

FAQS:

1. What is tree? What are properties of trees?
2. What is Binary tree, Binary search tree, Expression tree & General tree?
3. What are the members of structure of tree & what is the size of structure?
4. What are rules to construct binary tree?
5. What is preorder, postorder, inorder traversal?
6. Difference between recursive & Nonrecursive traversal?
7. What are rules to construct binary search tree?
8. What are rules to construct expression tree?
9. How binary tree is constructed from its traversals?

Assignment No	**05**
Title	Implement binary search tree and perform following operations:
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

5. Operations on Binary search tree

AIM: Implement binary search tree and perform following operations:

A. Insert
B. Delete
C. Search
D. Mirror image
E. Display
F. Display leaf nodes
G. Copy tree

OBJECTIVE:

To understand the concept of binary search tree as a data structure.

Applications of BST.

THEORY:

Definition of binary search tree

A binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

Illustrate the above operations graphically.

Searching

Searching a binary tree for a specific value can be a recursive or iterative process. This explanation covers a recursive method.We begin by examining the root node. If the tree is null, the value we are searching for does not exist in the tree. Otherwise, if the value equals the root, the search is successful. If the value is less than the root, search the left subtree. Similarly, if it is greater than the root, search the right subtree. This process is repeated until

the value is found or the indicated subtree is null. If the searched value is not found before a null subtree is reached, then the item must not be present in the tree.

Insertion

Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root.

Deletion

There are three possible cases to consider:

Deleting a leaf (node with no children): Deleting a leaf is easy, as we can simply remove it from the tree.

Deleting a node with one child: Remove the node and replace it with its child.

Deleting a node with two children: Call the node to be deleted N. Do not delete N. Instead, choose either its in-order successor node or its in-order predecessor node, R. Replace the value of N with the value of R, then delete R.

As with all binary trees, a node's in-order successor is the left-most child of its right subtree, and a node's in-order predecessor is the right-most child of its left subtree. In either case, this node will have zero or one children. Delete it according to one of the two simpler cases above.

ALGORITHM:

Define structure for Binary Tree (Mnemonics, Left Pointer, Right Pointer)

Insert Node:

Insert (Root, Node)

Root is a variable of type structure, represent Root of the Tree. Node is a variable of type structure ,represent new Node to insert in a tree.

Step 1: Repeat Steps 2,3 & 4 Until Node do not insert at appropriate position.

Step 2: If Node Data is less that Root Data & Root Left Tree is NULL

Then insert Node to

Left. Else Move

Root to Left

Step 3: Else If Node Data is Greater that equal that Root Data & Root Right Tree is NULL

Then insert Node to

Right. Else Move Root

to Right.

Step 4: Stop.

Search Node:

Search (Root, Mnemonics)

Root is a variable of type structure, represent Root of the Tree. Mnemonics is array of character. This function search Mnemonics in a Tree.

Step 1: Repeat Steps 2,3 & 4 Until Mnemonics Not find&&Root != NULL

Step 2: If Mnemonics Equal to Root Data

Then print message Mnemonics present.

Step 3: Else If Mnemonics Greater that Equal that Root Data Then Move Root to Right.

Step 4: Else Move Root

to Left.

Step 5: Stop.

Delete Node:

Dsearch(Root, Mnemonics)

Root is a variable of type structure ,represent Root of the Tree. Mnemonics is array of character. Stack is an pointer array of type structure. PTree(Parent of Searched Node),Tree(Node to be deleted), RTree(Pointg to Right Tree),Temp are pointer variable of type structure;

Step 1: Search Mnemonics in a Binary Tree

Step 2: If Root = = NULL Then Tree is NULL

Step 3: Else //Delete Leaf Node

Definition of binary search tree

A binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

Illustrate the above operations graphically.

Searching

Searching a binary tree for a specific value can be a recursive or iterative process. This explanation covers a recursive method.We begin by examining the root node. If the tree is null, the value we are searching for does not exist in the tree. Otherwise, if the value equals the root, the search is successful. If the value is less than the root, search the left subtree. Similarly, if it is greater than the root, search the right subtree. This process is repeated until the value is found or the indicated subtree is null. If the searched value is not found before a null subtree is reached, then the item must not be present in the tree.

Insertion

Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root.

Deletion

There are three possible cases to consider:

Deleting a leaf (node with no children): Deleting a leaf is easy, as we can simply remove it from the tree.

Deleting a node with one child: Remove the node and replace it with its child.

```
If Tree->Left = = NULL && Tree->Right =
= NULL Then a) If Root = = Tree Then Root
= NULL;
    b)If Tree is a Right Child PTree-
      >Right=NULL; Else PTree-
      >Left=NULL;
Step 4: Else // delete Node with Left and Right
    children If Tree->Left != NULL && Tree-
    >Right != NULL Then a)
    RTree=Temp=Tree->Right;
    b) Do steps i && ii while Temp->Left !=NULL
     i)  RTree=Temp;
     ii)      Temp=Temp-
    >Left; c)RTree-
    >Left=Temp->Right;
```

a) If Root == Tree//Delete

Root Node Root=Temp;

b)If Tree is a Right Child PTree->Right=Temp; Else PTree->Left=Temp;

c) Temp->Left=Tree->Left; g)If RTree!=Temp

Then Temp->Right = Tree->Right; Step 5: Else //with Right child

If Tree->Right!= NULL

Then a) If Root==Tree Root = Root->Right;

b) If Tree is a Right Child PTree->Right=Tree->Right; Else PTree->Left=Tree->Left;

Step 6: Else //with Left child If Tree->Left != NULL

Then a) If Root==Tree Root = Root->Left;

b) If Tree is a Right Child PTree->Right=Tree->Left; Else PTree->Left=Tree->Left;

Step 7: Stop.

Mirror Image:

Root is a variable of type structure ,represent Root of the Tree.

Queue is an pointer array of type structure. Front & Rear variable of type integer.

Mirror(Root)

Step 1: Queue[0]=Root;//Insert Root Node in a
Queue Step 2: Repeat Steps 3,4,5,6,7 & 8 Until
Queue is Empty Step 3: Root = Queue[Front++];
Step 4: Temp1 = Root-
>Left; Step 5: Root->Left
= Root->Right; Step 6:
Root->Right = Temp1;
Step 7: If Root->Left !=
NULL
Then Queue[Rear++] = Tree->Left;//insert Left
SubTree Step 8: If Root->Right != NULL
Then Queue[Rear++] = Root->Right;//insert Right
SubTree Step 9: Stop.

Display Leaf-Nodes:

This function displays all the leaf nodes in the linked binary tree.

1. Start from the ROOT node and store the address of the root node in **ptr.**
2. Check whether **ptr** contains NULL or not. If **ptr** is NULL then tree is empty.
3. Otherwise

    a. Check whether the left child and right child of **ptr** are empty or not. If the left child and right child of **ptr** are NULL then display the content of the node.
    b. Traverse the left subtrees and the right subtrees and repeat step a for all the nodes.

4. Stop

Create a Copy of a Tree:

This function will make a copy of the linked binary tree. The function should allocate memory for necessary nodes and copy respective contents in it.

1. Start from the ROOT node. The address of the root node will be in the **ptr.** Let newroot be the root of the new tree after the copy.

1. Check whether **ptr** contains NULL or not. If **ptr** is NULL then newroot = NULL
2. Otherwise

    (i) Allocate memory space for the new node
    (ii) Copy the data from current node from the old tree to node in the new tree.
    (iii) Traverse each node in the left subtrees from the old tree and repeat step (i) and (ii).
    (iv) Traverse each node in the right subtrees from the old tree and repeat step (i) and (ii).

3. Stop

INPUT:

Accept the nodes from the user

OUTPUT:

Display result of each operation with error checking.

Conclusion:

Thus we have implemented binary search tree and perform following operations.

FAQS:

1. What is Binary search tree?
2. What are the members of structure of tree & what is the size of structure?
3. What are rules to construct binary search tree?
4. How general tree is converted into binary tree?
5. What is binary threaded tree?
6. What is use of thread in traversal?

Assignment No	**06**
Title	Threaded Binary Tree
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

Experiment No.6

TITLE: THREADED BINARY TREE

OBJECTIVES:

1. To understand threaded binary tree.
2. To understand inorder, preorder and postorder traversals on it.

‘

PROBLEM STATEMENT:

To construct an inorder threaded binary tyyyyree from the input prefix arithmetic expression and to perform inorder, preorder and postorder traversals on it.

PROBLEM DEFINITION:

THREADED BINARY TREE0

We know that for binary tree representation using linked list there are more NULL links than actual links i.e. (n+1) and (n-1) respectively. A.J.Perlis and C. Thornton have devised a way of using these NULL links by replacing them with links to other nodes called threads. Such trees are called as threaded binary trees.

IN ORDER THREADED BINARY TREE

Using above concept, if right link of node p is NUL, then we can replace it with a pointer (thread) to a node which would immediately succeed the node p while traversing the binary tree in order. Similarly if left link of node p id NULL, then we have to replace it with a pointer (thread) to a node which would immediately precede the node p while traversing the binary tree in order. Now to distinguish between normal pointers of a thread we need extra one bit fields LBIT and RBIT in the node structure.

REPRESENTATION OF THREADED BINARY TREE USING LINKED LIST

==========

In threaded binary tree representation using linked lists, each node has fine fields namely LBIT, RBIT, DATA, RCHILD and LCHILD as shown

LBIT	LCHILD	DATA	RCHILD	RBIT

LCHILD: A pointer (address) to root node of LEFT binary tree or left thread

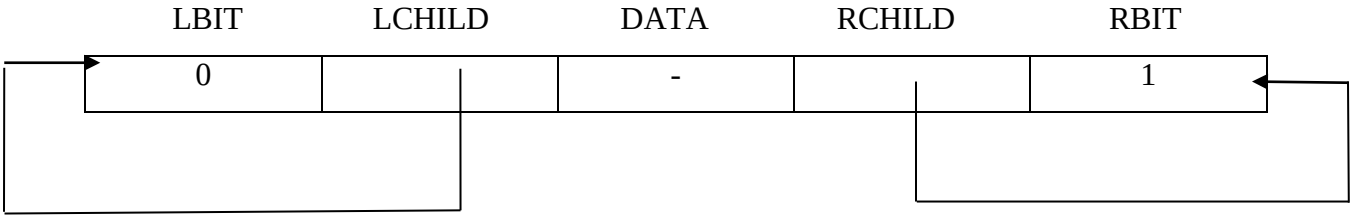
RCHILD: A pointer (address) to root node of RIGHT binary tree or right thread

DATA: Atom or data item or information at node

LBIT: '0' if LCHILD is a thread i.e. pointer to predecessor; '1' if LCHILD is pointer to root of LEFT binary tree

RBIT: '0' if RCHILD is a thread i.e. pointer to successor; '1' if RCHILD is pointer to root of RIGHT binary tree

The empty in order threaded tree binary tree will have only one HEAD node as shown in the following diagram



CREATION OF IN ORDER THREADED BINARY TREE

It is easy to write an algorithm to create a binary tree in computer of memory using the node structure given above. If the arithmetic expression is in prefix or postfix form. A stack (array of pointer) can be used to store the address of nodes containing operands or the intermediate result. By scanning the input string from left to right for postfix and by applying the corresponding expression evaluation algorithm. We can construct the binary tree .please refer the procedure construct_bt in section algorithm. After the construction of binary tree we can convert the same

into in-order threaded binary tree. Here we add ahead into node and convert the null links into in-order thread.

IN-ORDER THREADED BINARY TREE TRAVERSAL

Once we construct the in-order threaded binary tree as above the algorithm for in-order post-order and preorder. Traversal are very simple and here we don't need a stack. First we shall develop a function to get the successor node for the given node and then this function can be used to traverse tree. Please refer the procedure insuc (t), presuc (T) and respective procedure .for inorder, postorder traversal. For the threaded binary tree above following are the result for various traversals.

/

In-order Traversal: A/B-C*D+E
Pre-order Traversal: +*/A-BCDE
Post-order Traversal: ABC^/D*E+

ANALYSIS OF ALGORITHM:

1. Time Complexity:

Time complexity for the construction of binary tree from input prefix expression is O (n), while loop is executed for length (STR) time which is equal to number of nodes in constructed binary tree.

Time complexity for normal binary tree traversal whether recursiveor non recursive is also O(n) ,where 'n' is equal to number of nodes in binary tree, since every node is visited only once in both. For the threaded binary tree it is still O(n), but it is constant times less than recursive and non recursive using stack.

6rf

2. Space Complexity:

In recursive and non-recursive traversals of normal binary tree. We need a stack of depth ‘k’ which is equal to height or depth of a binary tree. But this can be achieved without using stack if the tree is a threaded binary tree.

PROBLEM SPECIFICATION:

1. Data structures to be used

a) Doubly linked list

```
typedef struct linked_list
{
        int lbit, rbit;
        int data;
        struct linked _list *left;
        struct linked _list *right;
} node;
```

b) One dimensional array of pointers to store addresses of nodes

```
struct st
{
        int top;
        node *stack [20];
}
```

2. Concepts to be used

- Link list

- Array of pointers
- Function to construct binary tree for input prefix arithmetic expression.
- Function to obtain in order threaded binary tree
- Functions for pre order, in order and post order traversals of in order threaded binary tree.

3. Input

Prefix arithmetic expression

4. Output

In order threaded binary tree and its in order, pre order and post order traversals

5. Sample

Input

Enter the prefix arithmetic expression: +*/A-BCDE

Output

Pre order traversal: +*/A-BCDE
In order traversal: A/B-C*D+E
Post order traversal: ABC-/D*E+

ALGORITHM:

1. **Pseudo "C" code for preorder traversal**

```
void preorder( tht head)
{
        tht temp;
        temp = head->left;
        printf("\nPREORDER");
        do
        {
                while(temp->lbit)
                {
                        printf("\t%d",temp->left);
                        temp=temp->left;
                }
                printf("\t%d",temp->data);
                while(temp->rbit && temp->right!=head)
                {
                        temp= temp->right;
                }
                if( temp=right->head)
                        return
                temp=temp->right;
        }while(1);
}
```

2. Pseudo code for inorder traversal

```
void inorder( tht head)
{
        tht temp;
        temp = head->left;
        printf("\nINORDER");
        do
```

```
        {
                while(temp->lbit)
                {
                        temp=temp->left;
                        printf("\t%d",temp->data);
                }
                if(temp->right==head)
                        return;
                temp=temp->right;
        }while(1);
}
```

3. Pseudo code for postorder traversal

```
void postorder( the head)
{
        tht temp;
        int array[MAX],i=o,k;
        temp=head->left;
        printf("\nPOSTORDER");
        do
        {
                while(temp->rbit)
                {
                        array[i]=temp->data;
                        i++;
                }
                temp=temp->right;
                array[i]=temp->data;
                i++;
                while(!temp->rbit && temp->left!=head)
```

```
                temp=temp->left;
            if(temp->left==head)
                break;
        }
        temp=temp->left;
    }
```

TEST CASE:

Test your program for following cases

For each test cases

a) Display the total number of comparisons required to construct the in order threaded binary tree.
b) Display the total number of nodes required.
c) Display the depth of the binary tree constructed
d) For each traversal

    Total number of comparisons

    Finally conclude on time and space complexity for the constitution of the threaded binary tree and traversals on it

CONCLUSION:

Consider e.g.: + * / a -b c d e

Operations performed	Theoretical value	Practical value
1. For construction of In-order threaded	10	10

binary tree.

2.	In-order traversal	10	10
3.	Pre-order traversal	10	10
4.	Post-order traversal	10	10

Time complexity = O (n)

APPLICATIONS:

1. It makes tree traversals easier by making uses of NULL pointer to store address at previously visited nodes.
2. Game tree: Trees can be used in playing of the games such as tic-tac-toe, chess. Etc!

FAQS:

1.What are the applications of threaded binary tree?

2.What are the advantages and disadvantages of threaded binary tree?

Assignment No	**07**
Title	**Graph: Minimum Spanning Tree**
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

7. Minimum Spanning Tree

AIM: Represent a graph of your college campus using adjacency list /adjacencymatrix. Nodes should represent the various departments/institutes and links should represent the distance between them.

Find minimum spanning tree

a) Using Kruskal's algorithm.

b) Using Prim's algorithm.

OBJECTIVE:

- Learn the concepts of graph as a data structure and their applications in everyday life.
- Understand graph representation (adjacency matrix, adjacency list, adjacency multi list)
- To understand minimum spanning tree of a Graph
- To understand how Kruskal and Prim's algorithm works

THEORY:

- **What is a graph? Various terminologies and its applications. Explain inbrief.**

A graph G is defined as follows:

G=(V,E)

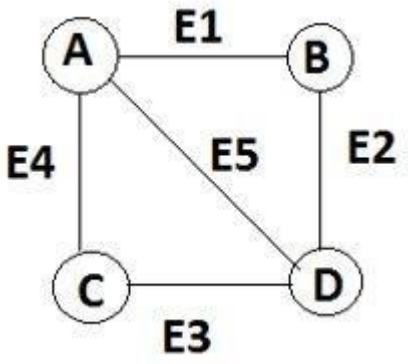
V(G): a finite, nonempty set of vertices

E(G): a set of edges (pairs of vertices)

Example:

V={A,B,C,D}

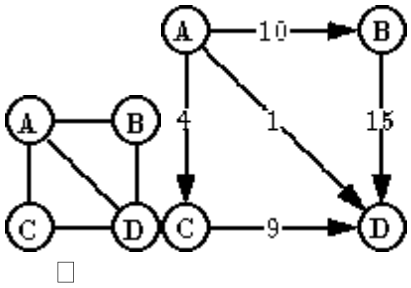
E={E1,E2,E3,E4,E5}



Different representations of graph.

- **Adjacency matrix:** Graphs $G = (V, E)$ can be represented by adjacency matrices $G[v1..v|V|, v1..v|V|]$, where the rows and columns are indexed by the nodes, and the entries $G[vi, vj]$ represent the edges. In the case of unlabeled graphs, the entries are just boolean values.

	A	**B**	**C**	**D**
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

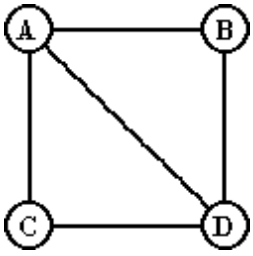


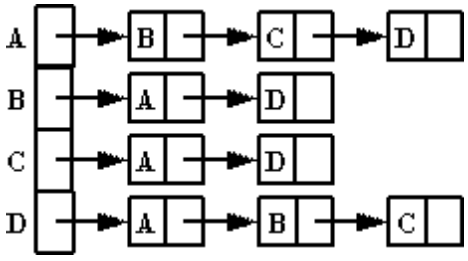
In case of labeled graphs, the labels themselves may be introduced into theentries.

	A		**B**	**C**	**D**
A	∞		10	4	1
B					15

	∞	∞		∞	
C	∞	∞		∞	9
D	∞	∞		∞	∞

Adjacency List: A representation of the graph consisting of a list of nodes,with each node containing a list of its neighboring nodes.





- **A spanning tree** is a sub-graph of G that is a tree containing all the verticesof G.

- **A minimum spanning tree (MST)** is a spanning tree with minimum weight.

 There are two basic algorithms for finding minimum-cost spanning trees

1. Kruskal's algorithm:

2. Prim's algorithm:

Kruskal's Algorithm:

Two steps:

- Sort edges by increasing edge weight
- Select the edges that do not generate a cycle

Step 1: Choose the arc of least weight.

Step 2: Choose from those arcs remaining the arc of least weight which doesnot form a cycle with already chosen arcs. (If there are several such arcs, choose one arbitrarily.)

Step 3: Repeat Step 2 until *n – 1* arcs have been chosen.

- **Prim's Algorithm:**

All vertices of a connected graph are included in the minimum spanning tree. Prim's algorithm starts from one vertex and grows the rest of tree by adding one vertex at a time by adding associated edge in T. This algorithm iteratively adds edges until all vertices are visited.

```
void prims (vertex i)
    1. Start
    2. Initialize
       visited [ ] to 0
       for   (i=0;i<n;
       i++)
             visited [ i ] = 0;
    3. Find   minimum
       edge  from  i  for
       (j=0;j<n; j++)
          {
            if (min > a [i] [j])
            {
```

```
                Min=a[I][j]
                ;x= i;
                y = j;
            }
        }
```

4. Print the edge between i and j with weight.
5. Make visit [i++] = x

   visit [j++] = y

6. Find next minimum edge starting from nodes of visit array.
7. Repeat step 6 until all the nodes are visited.
8. End.

Conclusion :

Implemented Kruskal and prims algorithm successfully

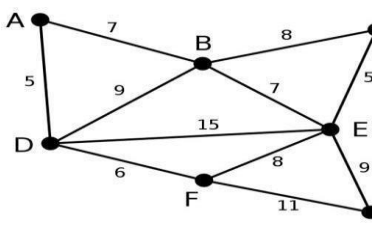
INPUT:

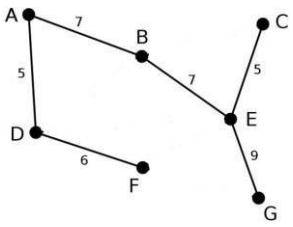
Enter the no. of nodes and edges with weight in graph. Create theadjacency matrix.

OUTPUT:

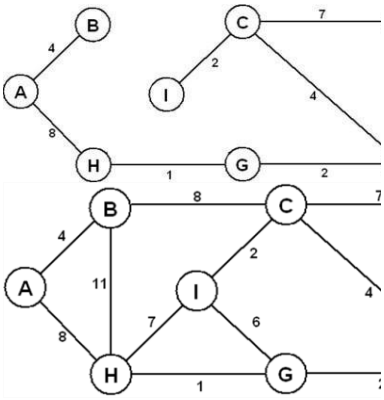
Display result of each operation with error checking.

INPUT **OUTPUT** Remark





Costof MST - 39



Costof MST - 37

FAQS:

1. What is graph?
2. Application of Prim‘s & Kruskal‘s algorithm.
3. What are the traversal techniques?
4. What are the graph representation techniques?
5. What is adjacency Matrix?
6. What is adjacency list?
7. Explain the PRIM‘s algorithm for minimum spanning tree.
8. What are the traversal techniques?

Assignment No	**08**
Title	Shortest Path Algorithm
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

8. Graph: Shortest Path Algorithm

AIM: Represent a graph of city using adjacency matrix /adjacency list. Nodes should represent the various landmarks and links should represent the distance between them. Find the shortest path using Dijkstra's algorithm from single source to all destination.

OBJECTIVE:

1. To understand the application of Dijkstra‘s algorithm

THEORY:

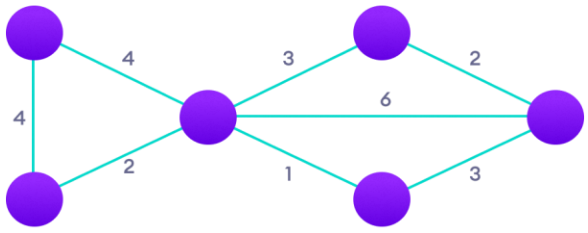
1. Explain in brief with examples how to find the shortest path using Dijkstra‘s algorithm.

Definition of Dijkstra's Shortest Path

1. To find the shortest path between points, the weight or length of a path is calculated as the sum of the weights of the edges in the path.
2. A path is a shortest if there is no path from x to y with lower weight.
3. Dijkstra's algorithm finds the shortest path from x to y in order of increasing distance from x. That is, it chooses the first minimum edge, stores this value and adds the next minimum value from the next edge it selects.
4. It starts out at one vertex and branches out by selecting certain edges that lead to new vertices.
5. It is similar to the minimum spanning tree algorithm, in that it is "greedy", always choosing the closest edge in hopes of an opt*imal solution.

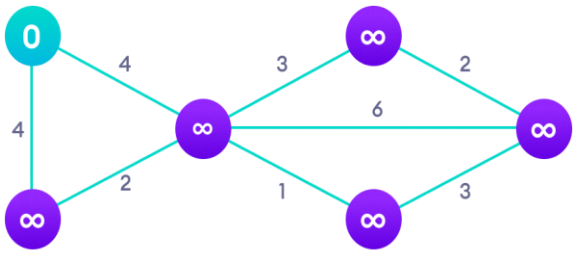
Example:

It is easier to start with an example and then think about the algorithm.



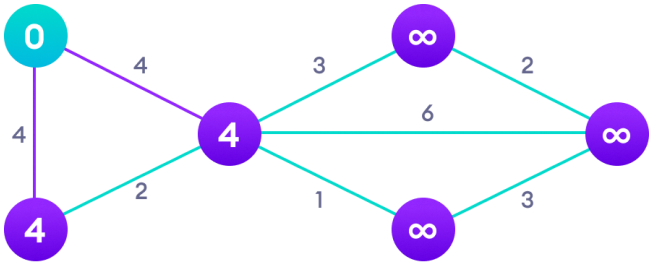
Step: 1

Start with a weighted graph



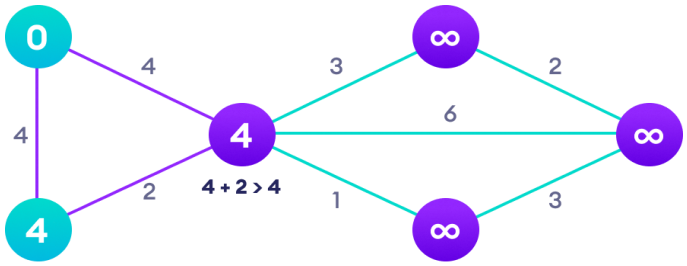
Step: 2

Choose a starting vertex and assign infinity path values to all other devices



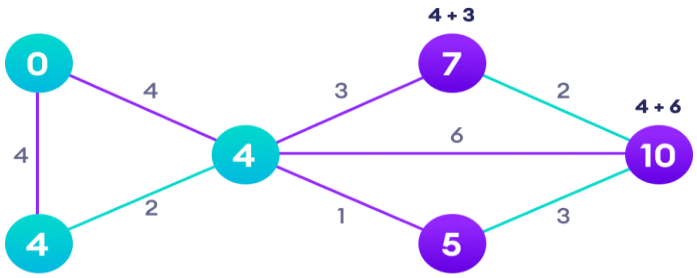
Step: 3

Go to each vertex and update its path length



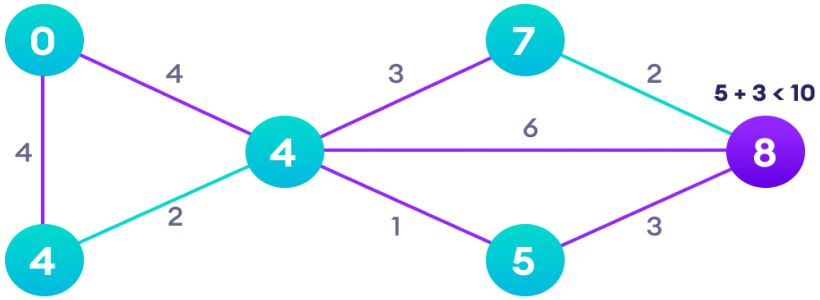
Step: 4

If the path length of the adjacent vertex is lesser than new path length, don't updateIt



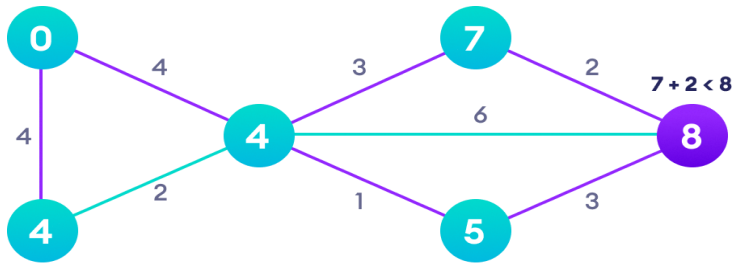
Step: 5

Avoid updating path lengths of already visited vertices



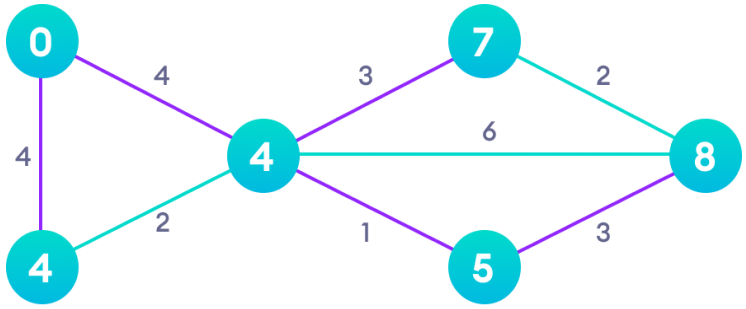
Step: 6

After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7



Step: 7

Notice how the rightmost vertex has its path length updated twice



Step: 8

Repeat until all the vertices have been visited

ALGORITHM:

College Area represented by Graph.

A graph G with N nodes is maintained by its adjacency matrix Cost. Dijkstra's algorithm find shortest path matrix D of Graph G.

Staring Node is 1.

Step 1: Repeat Step 2 for

I = 1 to N D[I] =

Cost[1][I].

Step 2: Repeat Steps 3 & 4 forI = 1

to N Step 3: Repeat Steps4 for J = 1

to N Step 4: If D[J]

> D[I] + D[I][J]

Then D[J] = D[I] + D[I][J]

Step 5: Stop.

INPUT:

The graph in the form of edges, nodes and corresponding weights, thesource node and destination node.

OUTPUT:

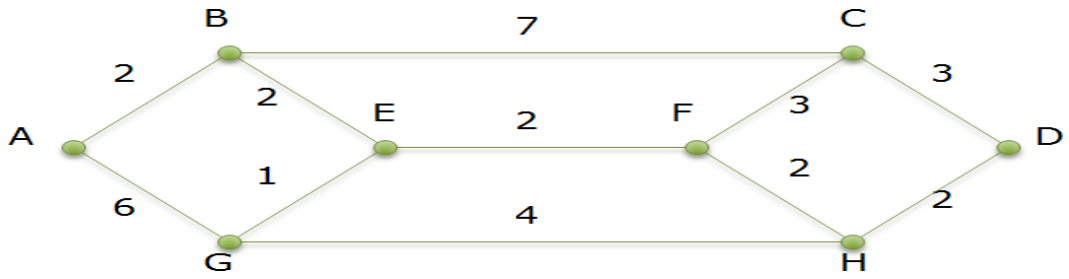
The shortest path and length of the shortest path

INPUT	**OUTPUT**	Remark

Consider Source Vertex as node A and Destination Vertex as node D

Shortest Path is A-B- E-F-H-D

Cost - 10



FAQs:

1. What is shortest path?
2. What are the graph representation techniques?
3. What is adjacency Matrix?
4. What is adjacency list?
5. Explain dijistras algorithm

Assignment No	**09**
Title	Implement Heap sort to sort given set of values using max or min heap.
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

Implement Heap sort to sort given set of values using max or min heap.

AIM: Implement Heap sort to sort a given set of values using max or min heap.

OBJECTIVE:

To understand the concept of a Heap.

THEORY:

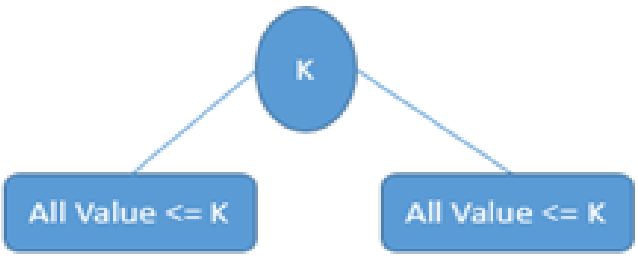
Heap Data Structure :

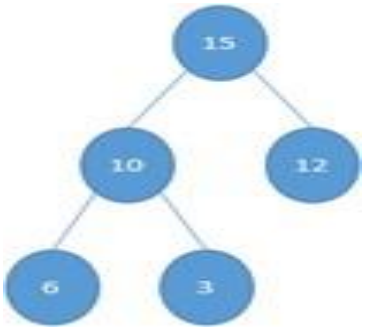
a) Concept

i) Heap Tree

Heap is a binary tree structure with the following properties:1.The tree is complete or almost complete

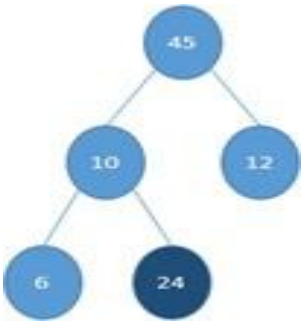
2.The key value of each node is greater than or equal to the key value of itsdescendants.





(A)

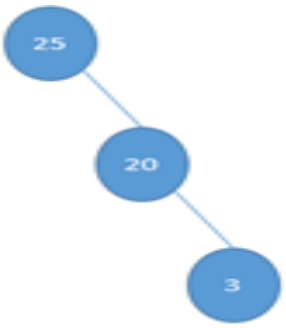
Valid Heap Tree



(B)

Invalid Heap Tree

due to sub-tree 10



(C)

Invalid Heap Tree

not nearly complete

ii) Max-Heap

If the key value of each node is greater than or equal to the key value of itsdescendants, this heap structure is called Max-Heap.

iii) Min-Heap

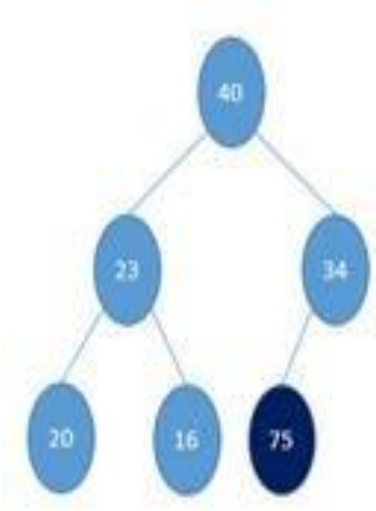
If the key value of each node is less than or equal to the key value of itsdescendants, this heap structure is called Min-Heap.

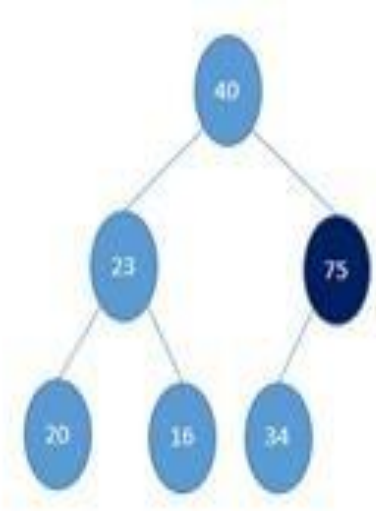
b) Maintenance Operations on Heap

To perform insert and delete a node in heap structure, it needs two basic algorithms :

1. Reheap up

This operation reorders a “broken” heap by floating the last element up thetree until it is at its correct position in the heap.





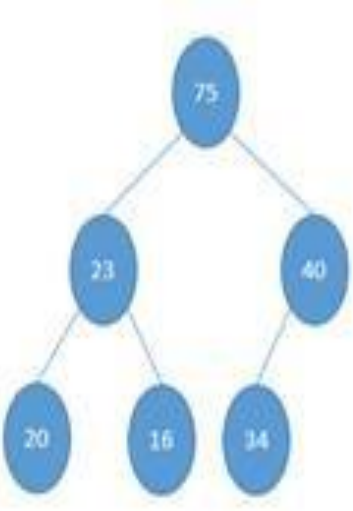
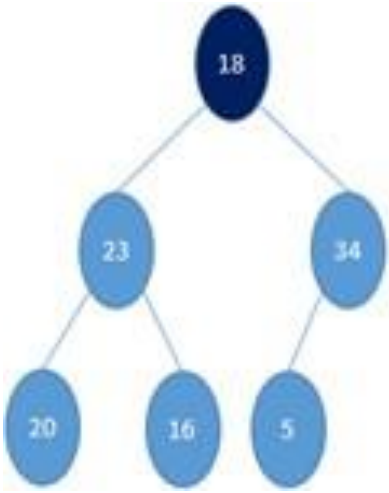


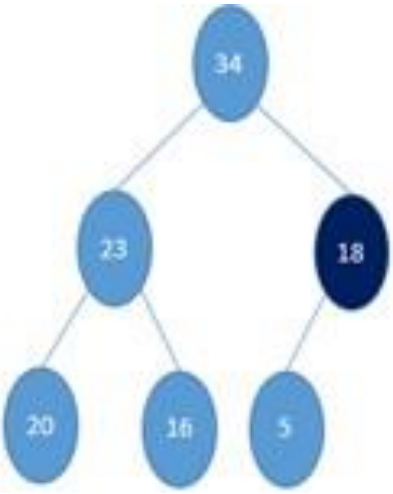
Fig. a Fig. b Fig. c

Not a heap tree node 75 moved up node 75 movedat correct position, it is heap tree

2. Reheap down

This operation reorders a “broken” heap by pushing the root down the treeuntil it is at its correct position in the heap.





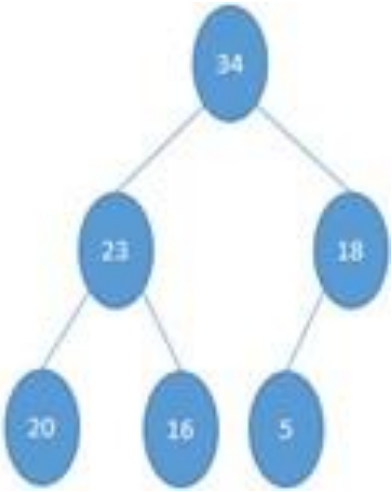


Fig. a Fig. b Fig. c

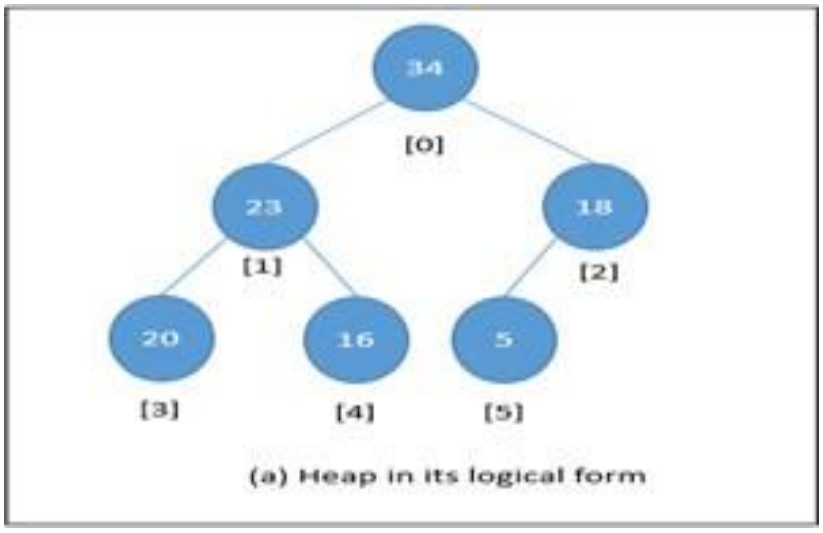
Not a heap tree Moved down node It is heap tree
at its correct place (Max heap)

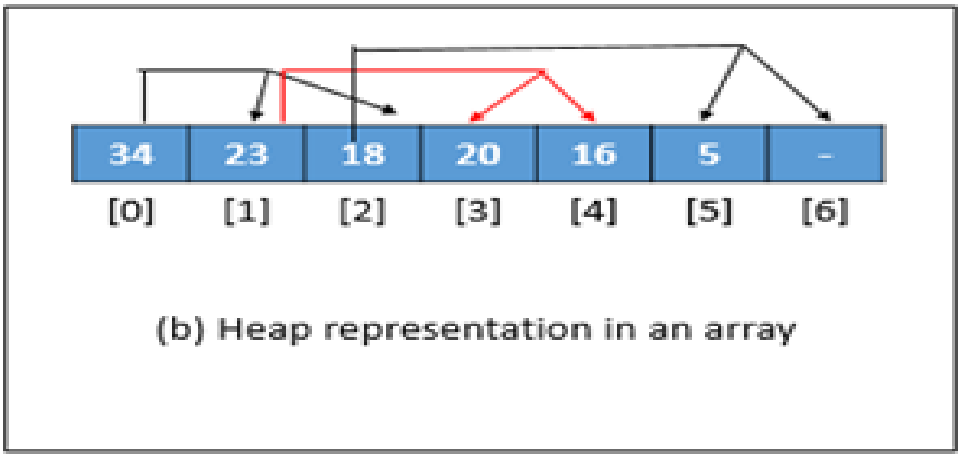
2. Heap Implementation

Heap implementation is possible using an array because it must be a complete or almost complete binary tree, which allows a fixed relationship between each node andits children and it can be calculated as :

For node located at ith index, its children are :Left
child : 2i + 1
Right child : 2i + 2

Parent of node located at ith index : (i-1)/2
Given the index for a left child, j,its right sibling at : j+1
Given the index for a right child, k, its left sibling at :k-1





3. Heap ADT

4. Applications of Heap

Selection algorithm

Priority Queue

Sorting (Heap Sort)

5. Heap Sort Complexity

The Heap sort efficiency is O(n log n)

ALGORITHM:

Heapify_asc(data,i) Step 1 : Set Lt = Left(i) Step 2 : Set Rt = Right(i)Step 3 :

if Lt <= heap_size[data] - 1 and data[Lt] > data[I]

then Set Max = Lt;

else

Set Max = i

Step 4 : if Rt <= heap_size[data] - 1 and data[Rt] > data[Max]

then Set Max = Rt

Step 5 : if Max != i

then Swap(data[i], data[Max])

Step 6 : Heapify(data, Max)

BuildHeap(data)

Step 1 : Set heap_size[data] = length[data]Step 2 : for i= (length[data]-1)/2 to 0

Heapify(data,i)

Heapsort(data)

Step 1 : BuildHeap(data)

Step 2 : for i = length[data] - 1 to 1

Swap(data[0],data[i])

Set heap_size[data] = heap_size[data] - 1

Heapify(data,0)

INPUT:

45, 78, 24, 36, 12

OUTPUT:

12,24,36,45,78

Conclusion : Thus we have implemented Heap sort to sort given set of values using max or min heap.

FAQS:

1. How do you sort an array using heap sort?
2. Explain max heap and min heap.
3. What is time complexity of heap sort

Assignment No	**10**
Title	**Implementation of sequential file**
Roll No	
Date of performance	
Date of completion	
Marks out of 10	
Signature of staff	

10. Implementation of sequential file

AIM: Department maintains student's database. The file contains roll number, name, division and address. Write a program to create a sequential file to store and maintain student data. It should allow the user to add, delete information of student. Display information of particular student. If record of student does not existan appropriate message is displayed. If student record is found it should display the student details. Implement sequential file for student Database and perform following operations point:

i) Create Database
ii) Display Database
iii) Add a record
iv) Delete a record

Objective:

To Study:

1. The local and physical organization of files.
2. To understand the concept of sequential files.
3. To know about File Operations
4. To understand the use of sequential files.
5. Sequential file handling methods.
6. Creating, reading, writing and deleting records from a variety of file structures.
7. Creating code to carry out the above operations.

Theory:

File : A file is a collection on information, usually stored on a computer's disk. Information can be saved to files and then later reused.

Type of File

- Binary File
  - The binary file consists of binary data
  - It can store text, graphics, sound data in binary format
  - The binary files cannot be read directly

- Numbers stored efficiently

- Text File
  - The text file contains the plain ASCII characters
  - It contains text data which is marked by _end of line‘ at the end of eachrecord
  - This end of record marks help easily to perform operations such asread and write
  - Text file cannot store graphical data.

File Organization:

The proper arrangement of records within a file is called as file organization. Thefactors that affect file organization are mainly the following:

- Storage device
- Type of query
  Number of keys
- Mode of retrieval/updateof

record Different types of File

Organizations are as :

- Sequential file
- Direct or random access file
- Indexed sequential file

1. **Sequential file :** In sequential file, records are stored in the sequential order oftheir entry. This is the simplest kind of data organization. The order of the records is fixed. Within each block, the records are in sequence . A sequential file stores records in the order they are entered. New records always appear atthe end of the file.

   Features of Sequential files :

   - Records stored in pre-defined order.
   - Sequential access to successive records.
   - Suited to magnetic tape.
   - To maintain the sequential order updating becomes a more complicated anddifficult task. Records will usually need to be moved by one place in order to add (slot in) a record in the proper sequential order. Deleting records willusually require that records be shifted back one place to avoid gaps in the sequence.
   - Access times are still too slow (no better on average than serial) to be useful in on

line applications.

Drawbacks of Sequential File Organization

- Insertion and deletion of records in in-between positions huge data movement
- Accessing any record requires a pass through all the preceding records, which is time consuming. Therefore, searching a record also takes more time.
- Needs reorganization of file from time to time. If too many records are deleted logically, then the file must be reorganized to free the space occupied by unwanted records

Primitive Operations on Sequential files

Open—This opens the file and sets the file pointer to immediately before the first record

Read-next—This returns the next record to the user. If no record is present, then EOF condition will be set.

Close—This closes the file and terminates the access to the file

Write-next—File pointers are set to next of last record and write the record to the file ***EOF***—If EOF condition occurs, it returns true,otherwise it returns false

Search—Search for the record with a given key

Update—Current record is written at the same position with updated values

2. **Direct or random-access file:** Files that have been designed to make direct record retrieval as easy and efficiently as possible is known as directly organized files. Though we search records using key, we still need to know theaddress of the record to retrieve it directly. The file organization that supports Files such access is called as direct or random file organization. Direct access files are of great use for immediate access to large amounts of information. They are often used in accessing large databases.

Advantages of Direct Access Files:

- Rapid access to records in a direct fashion.
- It doesn't make use of large index tables and dictionaries and therefore response times are very fast.

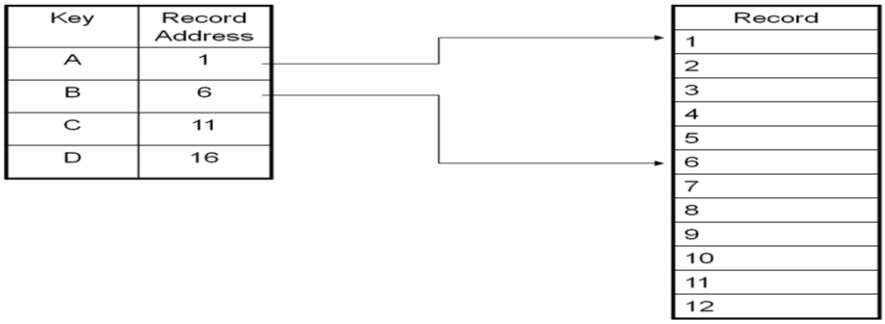
3. **Indexed sequential file:** Records are stored sequentially but the index file is prepared for accessing the record directly. An index file contains records ordered by a record key. The record key uniquely identifies the record and determines the sequence in which it is accessed with respect to other records.
    - A file that is loaded in key sequence but can be accessed directly by use ofone or more indices is known as an indexed sequential file. A sequential data file that is indexed is called as indexed sequential file. A solution to improve speed of retrieving target is index sequential file. An indexed file contains records ordered by a record key. Each record contains a field thatcontains the record key.

    - This system organizes the file into sequential order, usually based on a keyfield, similar in principle to the sequential access file. However, it is also possible to directly access records by using a separate index file. An indexed file system consists of a pair of files: one holding the data and one storing an index to that data. The index file will store the addresses of the records stored on the main file. There may be more than one index createdfor a data file e.g. a library may have its books stored on computer with indices on author, subject and class mark.

Characteristics of Indexed Sequential File

- Records are stored sequentially but the index file is prepared for accessing the record directly
- Records can be accessed randomly
- File has records and also the index
- Magnetic tape is not suitable for index sequential storage
- Index is the address of physical storage of a record
- When randomly very few are required/accessed, then index sequential isbetter
- Faster access method
- Addition overhead is to maintain index
- Index sequential files are popularly used in many applications like digital library

.



This is basically a mixture of sequential and indexed file organisation techniques. Records are held in sequential order and can be accessed randomly through an index. Thus, these files share the merits of both systems enabling sequential or direct access to the data. The index to these files operates by storing the highest record key in given cylinders and tracks. Note how this organisation gives the index a tree structure. Obviously this type of file organisation will require a direct accessdevice, such as a hard disk. Indexed sequential file organisation is very useful where records are often retrieved randomly and are also processed in (sequential) key order. Banks may use this organisation for their auto-bank machines i.e. customers randomly access their accounts throughout the day and at the end of the day the banks can update the whole file sequentially.

Advantages of Indexed Sequential Files

1. Allows records to be accessed directly or sequentially.
2. Direct access ability provides vastly superior (average) access times.

Disadvantages of Indexed Sequential Files

1. The fact that several tables must be stored for the index makes for a considerable storage overhead.
2. As the items are stored in a sequential fashion this adds complexity to the addition/deletion of records. Because frequent updating can be very inefficient, especially for large files, batch updates are often performed.

File classes :

Class	Functions	Meaning
ifstream	open()	Open the file in default input mode(read from file) ios::in
	get()	Read one character from the file Read
	getline()	one complete line from the file Read
	read()	any type of data from the file
	tellg(), seekg()	Direct access file functions(not required for this assignment)
ofstream	open()	Open the file in default output mode(write into file)ios::out
	put()	Write one character into the file
	write()	Write any type of data into the file
	tellp(), seekp()	Direct access file functions(not required for this assignment)
fstream	Inherits all above functions through iostream	Provides support for simultaneous input and output operations. Ios::in|ios::out

Algorithms :

1. **Algorithm for main function**

MAIN FUNCTION()

S1: Read the two filenames from user master and temporary.

S2: Read the operations to be performed from thekeyboard

S3: If the operation specified is create go to the create function, if the operation specified is display go to the display function, if the operation specified is add go to the add function , if the operation specified is delete go to delete function, if the operation specified is display particular record go to the search function, if the operation specified is exit go to step 4.

S4: Stop

Algorithm for create function

S1: Open the file in the write mode, if the file specified is not found or able to open then display error message and go to step5, else go to step2.

S2: Read the no: of records N to be inserted to the file. S3:

Repeat the step4 N number of times.

S4: Read the details of each student from the keyboard and write the same tothe file.

S5: Close the file.

S6: Return to the main function

Algorithm for displaying all records

S1: Open the specified file in read mode.

S2: If the file is unable to open or not found then display error message and go to step 4 else go to Step 3

S3: Scan all the student details one by one from file and display thesame at the console until end of file is reached.

S4: Close the file

S5: Return to the main function

Algorithm for add a record

S1: Open the file in the append mode, if the file specified is not found orunable to open then display error message and go to step5, else go to step2

S2: Scan all the student details one by one from file until end of file is reached. S3: Read the details of the from the keyboard and write the same to the file S4: Close the file.

S5: Return to the main function

Algorithm for deleting a record

S1: Open the file in the append mode, if the file specified is not found orunable to open then display error message and go to step5 , else go to step2

S2: Accept the roll no from the user to delete the record

S3: Search for the roll no in file.If roll no. exits, copy all the records in the fileexcept the one to be deleted in another temporary file.

S4:Close both files

S5:Now, remove the old file & name the temporary file with name same asthat of old file name.

Algorithm for displaying particular record(search)

S1: Open the file in the read mode ,if the file specified is not found or unableto open then display error message and go to step6 , else go to step2.

S2: Read the roll number of the student whose details need to bedisplayed.

S3: Read each student record from the file.

S4: Compare the students roll number scanned from file with roll number specified by the user.

S5: If they are equal then display the details of that record else displayrequired roll number not found message and go to step6.

S6: Close the file.

S7: Return to the main function.

Test Conditions:

1. Input valid filename.
2. Input valid record.
3. Check for opening / closing / reading / writing file errors

FAQ:

1. Define File? What are the factors affecting the file organization?
2. Compare Text and Binary File?
3. Explain the different File opening modes in C++?
4. What is indexed sequential file ? Explain the primitive operations on

indexed sequential file.

5. Write a note on Direct Access File?
6. What are the advantages and disadvantages of indexed-sequentialfile organization?
7. What are the advantages of Sequential File Organization?
8. What are serial and sequential files and how are they used in organization?
9. Distinguish between logical and physical deletion of records and illustrateit with suitable examples.
10. Compare and contrast sequential file and random access file organization?
11. Explain the different types of external storage devices?
12. With the prototype and example, explain following functions:
    i. i) seekg()ii)tellp() iii) seekp() iv) tellp()

Conclusion:

Thus we have implemented sequential file and performed all the primitive operations on it.