

SINHGAD ACADEMY OF ENGINEERING PUNE



**DATA STRUCTURES
LABORATORY**

LAB MANUAL

Department of Computer Engineering

Prepared by:

Prof. M. A. Giri

Academic Year 2022-23

ASSIGNMENT LIST

Assignment No.	Name of the Assignment /Experiment
1	<p>In second year computer engineering class, group A students play cricket, group B students play badminton and group C students play football.</p> <p>Write a Python program using functions to compute following: -</p> <ul style="list-style-type: none"> a) List of students who play both cricket and badminton b) List of students who play either cricket or badminton but not both c) Number of students who play neither cricket nor badminton d) Number of students who play cricket and football but not badminton. <p>(Note- While realizing the group, duplicate entries should be avoided, Do not use SET built-in functions)</p>
2	<p>Write a Python program to store marks scored in subject “Fundamental of Data Structure” by N students in the class. Write functions to compute following:</p> <ul style="list-style-type: none"> a) The average score of class b) Highest score and lowest score of class c) Count of students who were absent for the test d) Display mark with highest frequency
3	<p>Write a python program to compute following computation on matrix:</p> <ul style="list-style-type: none"> a) Addition of two matrices b) Subtraction of two matrices c) Multiplication of two matrices d) Transpose of a matrix
4	<ul style="list-style-type: none"> a) Write a python program to store roll numbers of student in array who attended training program in random order. Write function for searching whether particular student attended training program or not, using Linear search and Sentinel search. b) Write a python program to store roll numbers of student array who attended training program in sorted order. Write function for searching whether particular student attended training program or not, using Binary search and Fibonacci search
5	<p>Write a python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using</p> <ul style="list-style-type: none"> a) Selection Sort b) Bubble sort and display top five scores.
6	<p>Write a python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores.</p>
7	<p>Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of second, third and final year of department can be granted membership on request.</p>

	<p>Similarly one may cancel the membership of club. First node is reserved for president of Club and last node is reserved for secretary of club. Write C++ program to maintain club member's information using singly linked list. Store student PRN and Name. Write functions to:</p> <ol style="list-style-type: none"> Add and delete the members as well as president or even secretary. Compute total number of members of club Display members Two linked lists exists for two divisions. Concatenate two lists.
8	<p>Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C++ program to store two sets using linked list. compute and display-</p> <ol style="list-style-type: none"> Set of students who like both vanilla and butterscotch Set of students who like either vanilla or butterscotch or not both Number of students who like neither vanilla nor butterscotch
9	<p>A palindrome is a string of character that's the same forward and backward. Typically, Punctuation, capitalization, and spaces are ignored. For example, "Poor Dan is in a droop" is a palindrome, as can be seen by examining the characters "poor danisina droop" and observing that they are the same forward and backward. One way to check for a palindrome is to reverse the characters in the string and then compare with them the original-in a palindrome, the sequence will be identical. Write C++ program with functions-</p> <ol style="list-style-type: none"> To print original string followed by reversed string using stack To check whether given string is palindrome or not
10	<p>Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions:</p> <ol style="list-style-type: none"> Operands and operator, both must be single character. Input Postfix expression must be in a desired format. Only '+', '-', '*' and '/' operators are expected.
11	<p>Queues are frequently used in computer programming, and a typical example is the Creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.</p>
12	<p>A double-ended queue (deque) is a linear list in which additions and deletions may be made at either end. Obtain a data representation mapping a deque into a one-dimensional array. Write C++ program to simulate deque with functions to add and delete elements from either end of the deque</p>
13	<p>Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using circular queue using array.</p>

Course Objectives

- 1. Effective use of Object Oriented Programming concepts in data structures and files in programming.**
- 2. Effective use of multi-core programming architecture in programming.**
- 3. Effective use of latest programming tools like Eclipse Programming Framework, Microsoft Visual Studio, TC++, QT/ Java/Python etc**

Mapping Of Subjects With Cos and Pos

Course Name	Course Outcome	a	b	c	d	e	f	g	h	i	j	k	l
SE- 2019 Data Structure Lab	1. Effective use of Object Oriented Programming concepts in data structures and files in programming.	2	2	2	1							1	1
	2. Effective use of multi-core programming architecture in programming.	2	1	2	1		2					1	1
	3. Effective use of latest programming tools like Eclipse Programming Framework, Microsoft Visual Studio, TC++, QT/ Java/Python etc	2	2	2	2	2	1				1	2	2

Mapping Of Assignments to Cos and Pos

Sr No	Name of the Assignment /Experiment	Course Outcomes achieved	PEO,s achieved
1	<p>In second year computer engineering class, group A student's play cricket, group B students play badminton and group C students play football.</p> <p>Write a Python program using functions to compute following: -</p> <p>a) List of students who play both cricket and badminton</p> <p>b) List of students who play either cricket or badminton but not both</p> <p>c) Number of students who play neither cricket nor badminton</p> <p>d) Number of students who play cricket and football but not badminton.</p> <p>(Note- While realizing the group, duplicate entries should be avoided, Do not use SET built-in functions)</p>	1,2,3	a,b,c,d,e
2	<p>Write a Python program to store marks scored in subject "Fundamental of Data Structure" by N students in the class. Write functions to compute following:</p> <p>a) The average score of class</p> <p>b) Highest score and lowest score of class</p> <p>c) Count of students who were absent for the test</p> <p>d) Display mark with highest frequency</p>	1,3	b,c,d,e
3	<p>Write a python program to compute following computation on matrix:</p> <p>a) Addition of two matrices</p> <p>b) Subtraction of two matrices</p> <p>c) Multiplication of two matrices</p> <p>d) Transpose of a matrix</p>	1,2,3	a,b,c,d,e
4	<p>a) Write a python program to store roll numbers of student in array who attended training program in random order. Write function for searching whether particular student attended training program or not, using Linear search and Sentinel search.</p> <p>b) Write a python program to store roll numbers of student array who attended training program in sorted order. Write function for searching whether particular student attended training program or not, using Binary search and Fibonacci search</p>	1,3	b,c,d,e,j
5	<p>Write a python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using</p> <p>a) Selection Sort</p> <p>b) Bubble sort and display top five scores.</p>	1,3	b,c,d,e,j

6	Write a python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores.	1,2,3	b,c,d,e
7	Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member's information using singly linked list. Store student PRN and Name. Write functions to: a) Add and delete the members as well as president or even secretary. b) Compute total number of members of club c) Display members d) Two linked lists exists for two divisions. Concatenate two lists.	1,3	a,b,c,d,e
8	Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C++ program to store two sets using linked list. compute and display- a) Set of students who like both vanilla and butterscotch b) Set of students who like either vanilla or butterscotch or not both c) Number of students who like neither vanilla nor butterscotch	1,3	a,b,c,d,e
9	A palindrome is a string of character that's the same forward and backward. Typically, punctuation, capitalization, and spaces are ignored. For example, "Poor Dan is in a droop" is a palindrome, as can be seen by examining the characters "poor danisina droop" and observing that they are the same forward and backward. One way to check for a palindrome is to reverse the characters in the string and then compare with them the original-in a palindrome, the sequence will be identical. Write C++ program with functions- a) To print original string followed by reversed string using stack b) To check whether given string is palindrome or not	1,3	b,c,d,e
10	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/' operators are expected.	1,3	a,b,c,d,e
11	Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.	1,3	a,b,c,d,e
12	A double-ended queue (deque) is a linear list in which additions and deletions may be made at either end. Obtain a data representation	1,2,3	a,b,c,d,e

	mapping a deque into a one-dimensional array. Write C++ program to simulate deque with functions to add and delete elements from either end of the deque		
13	Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using circular queue using array.	1,2,3	a,b,c,d,e,j

Assignment No. 1

Problem Statement:

In Second year Computer Engineering class of M students, set A of students play cricket and set B of students play badminton. Write Python program to find and display-

- i. Set of students who play either cricket or badminton or both
- ii. Set of students who play both cricket and badminton
- iii. Set of students who play only cricket
- iv. Set of students who play only badminton
- v. Number of students who play neither cricket nor badminton

Theory

Set-:

Set is a collection of elements.

The term class is used to denote set.

A set may contain finite or infinite no. of elements.

Empty set-: a set is called as empty set if it contains no element denoted by ϕ .

Universal set-: if all the set, subset of given set then this set is called as universal set denoted by U.

Subset-: if every element of set A is an element of set B, then we say that A subset of B.

Equality-> Two sets are said to be equal if they are subset of each other.

Operations of set-:

1. Union
2. Intersection ϕ
3. Difference
4. Complement
5. Cartesian product

1. Union- union of 2 sets A&B is the set containing all elements which are in A or B or in both sets.

e.g. $A=\{1,2,3\}$ and $B=\{4,2,5\}$ $A \cup B=\{1,2,3,4,5\}$

2. Intersection-> it is the common elements between set A and Set B.

e.g. $A \cap B=\{2\}$

3. Difference ->

Let A and B be 2 sets then $A-B$ is defined as

$$A-B=\{x/x \in A \text{ and } x \notin B\}$$

E.g. $A-B=\{1,3\}$

4. Complement- set containing all elements of universal set excepting its own elements.

e.g. $A=\{1,2,3\}$

$$A_c = U - \{1,2,3\}$$

5. Cartesian product- it is the set whose members are all +ve ordered pair(a,b), where a is member and b is member of B.

e.g.- $a=\{a,b,c\}$

$$b=\{a,b\}$$

$$a \times b = \{(a,a), (a,b), (b,a), (b,b), (c,a), (c,b)\}$$

Algebra of sets:-

1. Associative law-

$$(a) A \cup B = A \cup (B \cap C)$$

$$(b) A \cap B \cap C = A \cap (B \cap C)$$

2. Commutative laws:-

$$(a) A \cup B = B \cup A$$

$$(b) A \cap B = B \cap A$$

3. Distributive law

$$(a) A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

4. Independent laws:-

(a) $A \cup B = A$

(b) $A \cap A = A$

5. Identity laws:-

(a) $A \cup \emptyset = A$

(b) $A \cap U = A$

(c) $A \cup U = U$

(d) $A \cap \emptyset = \emptyset$

6. Complement laws:-

(a) $A \cup A_c = U$

(b) $A \cap A_c = \emptyset$

7. Absorption laws:-

(a) $A \cup (A \cap B) = A$

(b) $A \cap (A \cup \emptyset) = A$

8. Involution law:-

$$A_c = A$$

9. Demorgans law:-

(a) $(A \cup B)_c = A_c \cap B_c$

(b) $(A \cap B)_c = A_c \cup B_c$

Conclusion: we implemented set assignment successfully.

Assignment No. 2

Problem Statement:

Write Python program to store marks scored for first test of subject 'Data Structures and Algorithms' for N students. Compute

- i. The average score of class
- ii. Highest score and lowest score of class
- iii. Marks scored by most of the students
- iv. List of students who were absent for the test

Introduction

Algorithm is a sequence of unambiguous instruction for solving a problem.

It is procedure of formula for solving a problem.

It is written in simple English.

It should be unique and self-explanatory.

Characteristic

- Precision: Steps are precisely defined.
- Uniqueness: Results of each folder are uniquely defined.
- Finiteness: Algorithm stops after finite number of steps.
- Input: Algorithm receives input.
- Output: Algorithm produces proper output.
- Generality: Algorithm applies is set input.

Complexity of an Algorithm

Complexity of an algorithm is a measure of amount of time and space required by it. For input of given size „n” we define complexity as a numerical function. T(n)- Time vs. input size.

Time Complexities

It is total time required by algorithm to run for completion. It is expressed in bigO notation.

Space Complexities

Amount of computer memory required during program execution as a function of input size. There are 3 scenarios:

- i. Best case: Least amount of time.
- ii. Worst Case: Maximum amount of time.
- iii. Average Case: Average amount of time.

Introduction to Data Structure

Data structure is a specified format for organizing and storing data. It includes array, files, records, tables, trees and so on.

Types of Data Structure

1. Linear
Elements are organized sequentially.
2. Non Linear
Data Elements can be attached to several other elements. Example- Tree, graph.
3. Static
The size of structure is fixed. Storing and performing operations is easy.
4. Dynamic
Size of structure is not defined. Difficult for storing or performing operation. Memory is allocated at execution time.

Conclusion

Hence, we have studied the concept of data structure and algorithm.

Assignment No. 3

Problem Statement:

Write a **python** program to compute following computation on matrix:

- a) Addition of two matrices
- b) Subtraction of two matrices
- c) Multiplication of two matrices
- d) Transpose of a matrix

Theory

Introduction to Array

Array is a data structure that can store fixed size sequential collection of element of same type. An array is used for storing a collection of data. But it is often more useful to think of an array as collection of variables of same types.

Instead of declaring individual variables you declare one array variable and use number [0]. A specific element in array is accessed by index.

All arrays consists of contiguous memory locations the lower address corresponds to first element and highest address to last element.

num[0]	num[1]	num[2]	last
element			
First element	Second Element	Third Element	

Single dimensional array

The one dimensional array in C + + is simplest type of array that contains only one row has single set of square „[]“ bracket.

eg: `int age[]=New int[6]`

The array of age is one dimensional containing only 6 elements in a single row

Two dimensional array

An array track of multiple pieces of information in linear order is a 1D array. A 2D array is nothing more than an array of arrays.

A local 1D array looks like

```
New int [6]
```

And 2D array looks like

```
int[ ][ ] my array={ (0, 1, 2, 3) (3, 2, 1, 0) (3, 5, 6, 1) (3, 8, 3, 4) }
```

For our purpose it is better to think of 2D array as a matrix.

Illustration

```
int[ ][ ] my array={ (0, 1, 2, 3)
                    (3, 2, 1, 0)
                    (3, 5, 6, 1)
                    (3, 8, 3, 4) }
```

Multidimensional Array

Array having more than subscript variable is called multidimensional array.

Array with more than 2 square brackets is also called a matrix.

```
int[ ][ ][ ] my array is an example of multidimensional array
```

Memory Representation and Address Calculation

1. 2D arrays are stored in continuous memory location row wise. stored in continuous
2. 3x3 array is shown below in first diagram.
3. Consider 3x3 array is stored which starts from 4000.
4. Array element [0] will be stored at 4000 again a[0][1] will be stored in next memory.
5. After element of first row and stored appropriate memory location element of next row gets their corresponding memory locations.

	col 0	col 1	col 2
Row 0	1	2	3
Row 1	4	5	6
Row 2	7	8	9

Element	Mem location
a[0][0]	4000
a[0][1]	4002
a[0][2]	4004
a[1][0]	4006
a[1][1]	4008
a[1][2]	4010
a[2][0]	4012
a[2][1]	4014
a[2][2]	4016

1	2	3
4000	4002	4004

4	5	6
4006	4008	4010

7	8	9
4012	4014	4016

Matrix Operations

1. Addition

2	3		1	4		3	7
2	4	+	5	6	=	7	10

2. Multiplication

1	2		1	2		7	10
3	4	X	3	4	=	15	22

3. Transpose

A=	1	3	A^T=	1	2
	2	4		3	4

4. Subtraction

$$\begin{array}{r} 2 \\ 4 \end{array} \begin{array}{r} 3 \\ 5 \end{array} - \begin{array}{r} 1 \\ 3 \end{array} \begin{array}{r} 2 \\ 3 \end{array} = \begin{array}{r} 1 \\ 1 \end{array} \begin{array}{r} 1 \\ 2 \end{array}$$

Conclusion

Hence, we have learned to use array.

Assignment No. 4

Problem Statement:

- a) Write a **python** program to store roll numbers of student in array who attended training program in random order. Write function for searching whether particular student attended training program or not, using Linear search and Sentinel search.
- b) Write a **python** program to store roll numbers of student array who attended training program in sorted order. Write function for searching whether particular student attended training program or not, using Binary search and Fibonacci search.

Theory

Linear Search

A linear search is the basic and simple search algorithm. A linear search searches an element or value from an array till the desired element or value is not found and it searches in a sequence order. It compares the element with all the other elements given in the list and if the element is matched it returns the value index else it return -1. Linear Search is applied on the unsorted or unordered list when there are fewer elements in a list.

Example with Implementation

To search the element 5 it will go step by step in a sequence order.

8	2	6	3	5
---	---	---	---	---

```
function findIndex(values, target)
{
    for(var i = 0; i < values.length; ++i)
```

```
{  
    if (values[i] == target)  
    {  
        return i;  
    }  
}  
return -1;  
}
```

//call the function findIndex with array and number to be searched

```
findIndex([ 8 , 2 , 6 , 3 , 5 ] , 5) ;
```

Binary Search

Binary Search is applied on the sorted array or list. In binary search, we first compare the value with the elements in the middle position of the array. If the value is matched, then we return the value. If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We repeat this procedure on the lower (or upper) half of the array. Binary Search is useful when there are large numbers of elements in an array.

Example with Implementation

To search an element 13 from the sorted array or list.

2	4	7	9	13	15
---	---	---	---	----	----

- As we can see the above array is sorted in ascending order.
- Binary Search is applied on sorted lists only, so that we can make the search fast, by breaking the list everytime.
- Start with middle element,
- if its EQUAL to the number we are searching, then RETURN
- if its less than it, then move to the RIGHT.
- if its more than it, then move to the LEFT.
- And then, REPEAT, till you find the number.

```
function findIndex(values, target)
{
  return binarySearch(values, target, 0, values.length - 1);
};

function binarySearch(values, target, start, end) {
  if (start > end) { return -1; } //does not exist

  var middle = Math.floor((start + end) / 2);
  var value = values[middle];

  if (value > target) { return binarySearch(values, target, start, middle-1); }
  if (value < target) { return binarySearch(values, target, middle+1, end); }
  return middle; //found!
}

findIndex([2, 4, 7, 9, 13, 15], 13);
```

In the above program logic, we are first comparing the middle number of the list, with the target, if it matches we return. If it doesn't, we see whether the middle number is greater than or smaller than the target.

If the Middle number is greater than the Target, we start the binary search again, but this time on the left half of the list, that is from the start of the list to the middle, not beyond that.

If the Middle number is smaller than the Target, we start the binary search again, but on the right half of the list, that is from the middle of the list to the end of the list.

CONCLUSION-: Hence we studied searching & sorting.

Assignment No. 5

Problem Statement:

Write C++ program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using

- a) Selection Sort
- b) Bubble sort and display top five scores.

Theory:

Selection Sort

Selection sort is a simple sorting algorithm. This sorting algorithm is a in-place comparison based algorithm in which the list is divided into two parts, sorted part at left end and unsorted part at right end. Initially sorted part is empty and unsorted part is entire list.

Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n are no. of items.

Working of selection Sort:



For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.



So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of sorted list.



For the second position, where 33 is residing, we start scanning the rest of the list in linear manner.



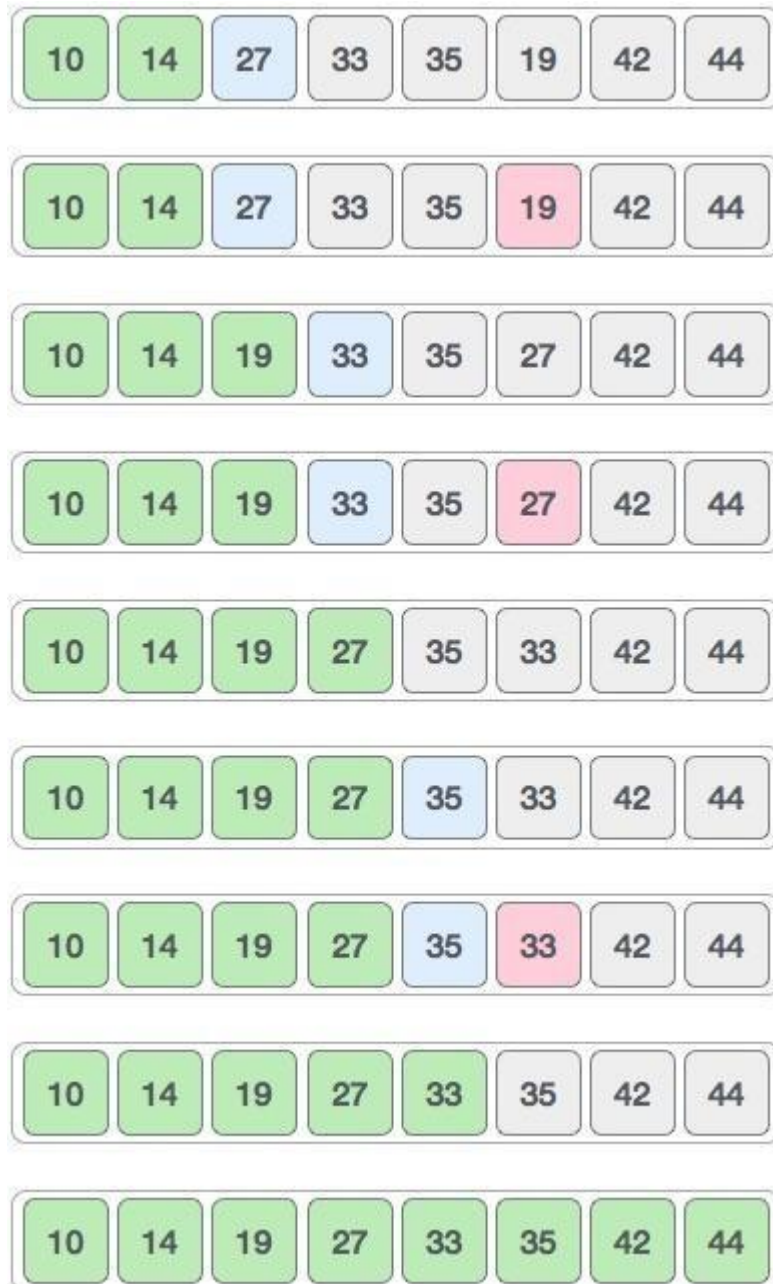
We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.



After two iterations, two least values are positioned at the beginning in the sorted manner.



The same process is applied on the rest of the items in the array. We shall see an pictorial depiction of entire sorting process –



Bubble Sort:

Bubble sort, sometimes incorrectly referred to as **sinking sort**, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort. Although the algorithm is simple, most of the other sorting algorithms are more efficient for large lists.

Working of Bubble Sort:

We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping short and precise.



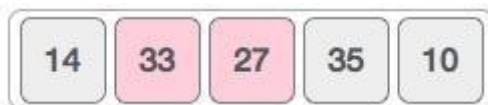
Bubble sort starts with the very first two elements, comparing them to check which one is greater.



In this case, the value 33 is greater than 14, so it is already in a sorted location. Next, we compare 33 with 27.



We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this –



Next we compare 33 and 35. We find that both are already in sorted positions.



Then we move to the next two values, 35 and 10.



We know that 10 is smaller than 35. Hence they are not sorted.



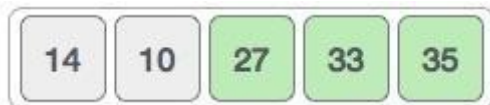
We swap these values. We find that we reach at the end of the array. After one iteration the array should look like this –



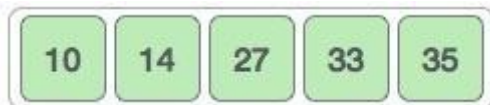
To be precise, we are now showing that how array should look like after each iteration. After second iteration, it should look like this –



Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sort learns that array is completely sorted.



Conclusion

Hence, we learned to use selection sort and bubble sort.

Assignment No. 6

Problem Statement:

Write Python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores.

Quick Sort:

The quick sort is an in-place, divide-and-conquer, massively recursive sort. As a normal person would say, it's essentially a faster in-place version of the merge sort. The quick sort algorithm is simple in theory, but very difficult to put into code.

The recursive algorithm consists of four steps:

1. If there is one or less elements in the array to be sorted, return immediately.
2. Pick an element in the array to serve as a "pivot" point. (Usually the left-most element in the array is used.)
3. Split the array into two parts - one with elements larger than the pivot and the other with elements smaller than the pivot.
4. Recursively repeat the algorithm for both halves of the original array.

The quick sort is by far the fastest of the common sorting algorithms.

Algorithm:

1. Get N elements which are to be sorted, and store it in the array A.
2. Select the element from A[0] to A[N-1] for middle. This element is the pivot.
3. Partition the remaining elements into the segments left and right so that no elements in left has key larger than that of the pivot and no elements in right has a key smaller than that of the pivot.
4. Sort left using quick sort recursively.
5. Sort right using quick sort recursively.
6. Display the sorted array A.

Example:

p										r
0	1	2	3	4	5	6	7	8	9	
9	7	5	11	12	2	14	3	10	6	



p			q							r
0	1	2	3	4	5	6	7	8	9	
5	2	3	6	12	7	14	9	10	11	



p	q	r		p			q		r	
0	1	2		4	5	6	7	8	9	
2	3	5	6	7	9	10	11	14	12	



p,r		p,r		p		q,r		p,q	r	
0		2		4	5	6		8	9	
2	3	5	6	7	9	10	11	12	14	



				p	q,r				p,r	
				4	5				9	
2	3	5	6	7	9	10	11	12	14	



				p,r						
				4						
2	3	5	6	7	9	10	11	12	14	

Assignment No. 7

PROBLEM STATEMENT:-

Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member's information using singly linked list. Store student PRN and Name. Write functions to

- a) Add and delete the members as well as president or even secretary.
- b) Compute total number of members of club
- c) Display members
- d) Display list in reverse order using recursion
- e) Two linked lists exists for two divisions. Concatenate two lists.

Theory

Linked List

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.

Arrays can be used to store linear data of similar types, but arrays have following limitations.

- 1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
- 2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to be shifted.

For example, in a system if we maintain a sorted list of IDs in an array `id[]`.

`id[] = [1000, 1010, 1050, 2000, 2040]`.

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in `id[]`, everything after 1010 has to be moved.

Advantages over arrays

- 1) Dynamic size
- 2) Ease of insertion/deletion

Drawbacks:

- 1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.
- 2) Extra memory space for a pointer is required with each element of the list.

Representation in C:

A linked list is represented by a pointer to the first node of the linked list. The first node is called head. If the linked list is empty, then value of head is NULL.

Each node in a list consists of at least two parts:

- 1) data
- 2) pointer to the next node

In C, we can represent a node using structures. Below is an example of a linked list node with an integer data.

In Java, Linked List can be represented as a class and a Node as a separate class. The Linked List class contains a reference of Node class type.

Linked List vs Array

Both Arrays and Linked List can be used to store linear data of similar types, but they both have some advantages and disadvantages over each other.

Following are the points in favour of Linked Lists.

- (1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, upper limit is rarely reached.
- (2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to be shifted.

For example, suppose we maintain a sorted list of IDs in an array `id[]`.

`id[] = [1000, 1010, 1050, 2000, 2040,]`.

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in `id[]`, everything after 1010 has to be moved.

So Linked list provides following two advantages over arrays

- 1) Dynamic size
- 2) Ease of insertion/deletion

Linked lists have following drawbacks:

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.

2) Extra memory space for a pointer is required with each element of the list.

3) Arrays have better cache locality that can make a pretty big difference in performance.

Conclusion

Hence, we learnt different operations on linked list.

Assignment No. 8

PROBLEM STATEMENT:-

Second year Computer Engineering class, set A of students like Vanilla ice-cream and set B of students like butterscotch ice-cream. Write a C++ program to store two sets using linked list. Compute & display –

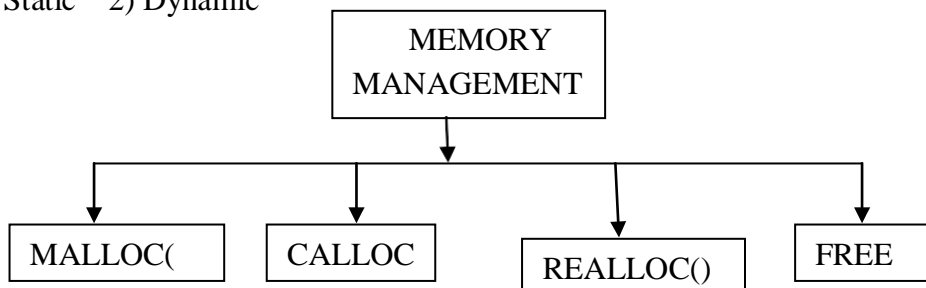
- i. Set of students who like either vanilla or butterscotch or both
- ii. Set of students who like both vanilla and butterscotch
- iii. Set of students who like only vanilla and not butterscotch
- iv. Set of students who like not vanilla and only butterscotch
- v. No of students who like neither vanilla nor butterscotch

Theory

DYNAMIC MEMORY MANAGEMENT:-

In dynamic data structure, the memory space required by variables is calculated & allocated during execution. C gives us two choices for reserving memory locations for a variable-

- 1) Static 2) Dynamic



1.] Block Memory Allocation (malloc()):

The malloc() function allocates a block of memory that contains the no. of bytes specified in its parameter. This returns a pointer to an area of memory with size, byte-size.

2.] Contiguous Memory Allocation (calloc()):

This function can be used to allocate a contiguous block of memory, primarily for arrays. It differs from malloc() only to the extent that it sets memory to null characters. This function returns a pointer to the allocated memory.

3.] Reallocation Of Memory (realloc()):

This function changes the size of the previously allocated block of memory. This is done by either deleting or extending the memory at the end of the block. If memory cannot be extended, realloc() allocates a completely new block of memory.

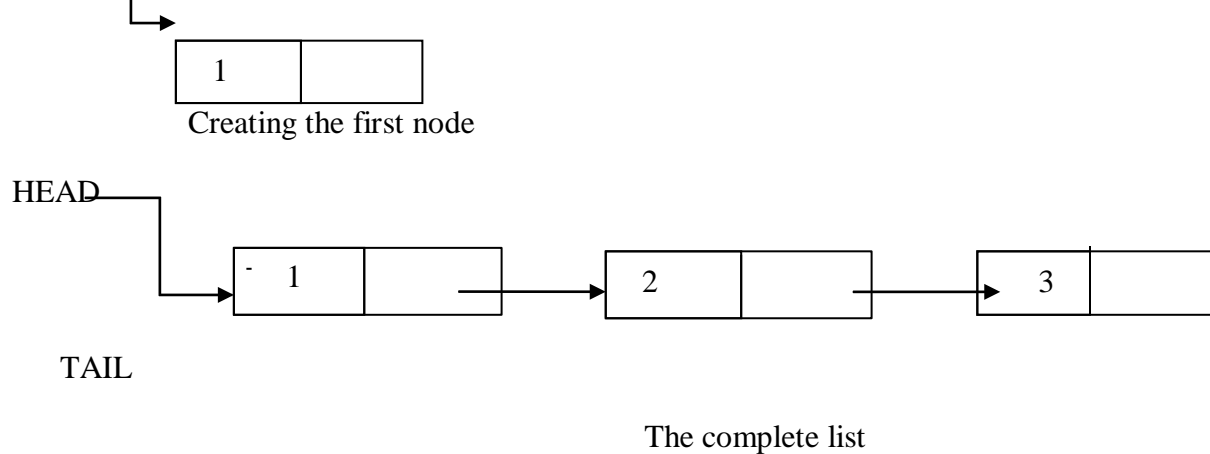
4.] Releasing Memory (free()):

When the allocated memory is no longer needed, it should be returned back by using the function free().

OPERATIONS ON LINK LIST:

1.CREATION:-

This operation is used to create constituent node as & when required. The structure of a list changes when nodes are inserted & deleted. Normally, creation of a list does not require alteration of the list structure except the addition of new node at the end or the beginning of the list.



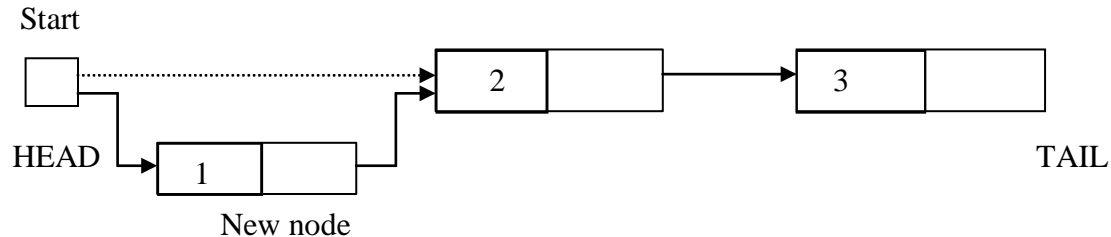
2.INSERTION :

This operation is used to insert a new node in the link list. This can be done by three ways as:-

- (i) At the beginning of the list
- (ii) At a certain position
- (iii) At the end

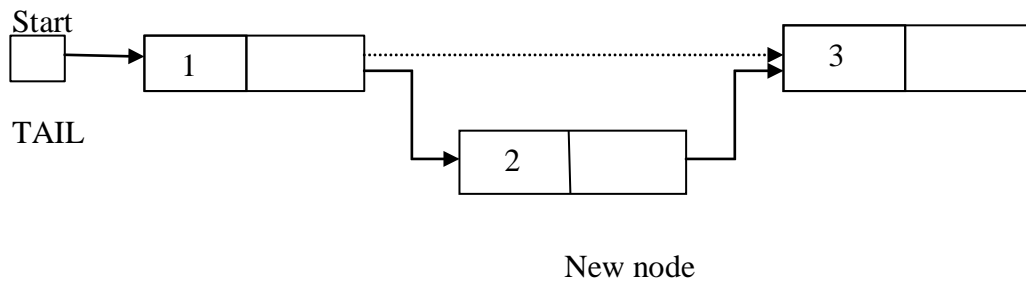
(i.) Insertion at the beginning of list:-

The insertion of a new node at the head of list is easier. The new node becomes the first node in the list i.e the address of the pointer head is changed to the address of new node.



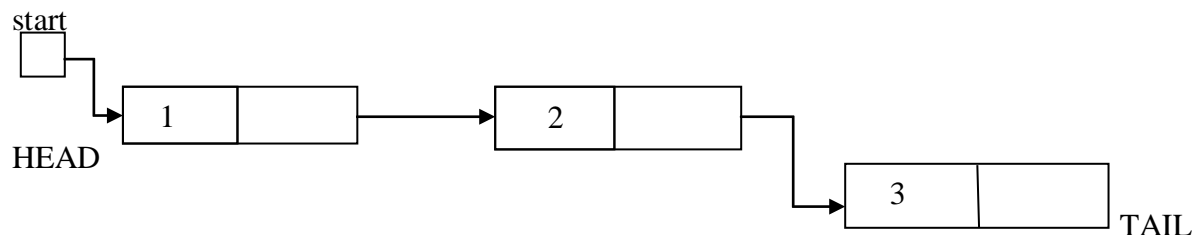
(ii.) Insertion at a certain position:-

The insertion of a data element at any point in data structure should be done in constant time. It requires two pointers to be changed. The node with data element 2 is to be inserted between data element 1 & 3. The link of new node is assigned the address of the node with data 3 & the link of the first node is changed to point to the new node.



(iii.) Insertion at the end of list:-

In the link list, only the head of the list is known to us. Therefore, to insert a node at the tail of the list, entire list has to be traversed. Once the last node is reached, the new node can be attached to the list.



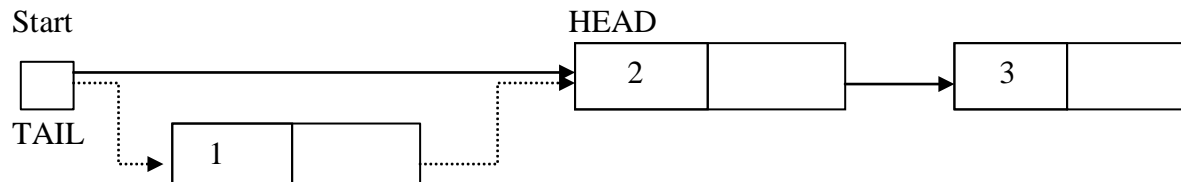
3.DELETION:-

This operation is used to delete node from the list. This can be done by three ways:-

- (i)At the beginning of the list
- (ii)At a certain position
- (iii)At the end of list

(i.)Deletion at the beginning of the list:-

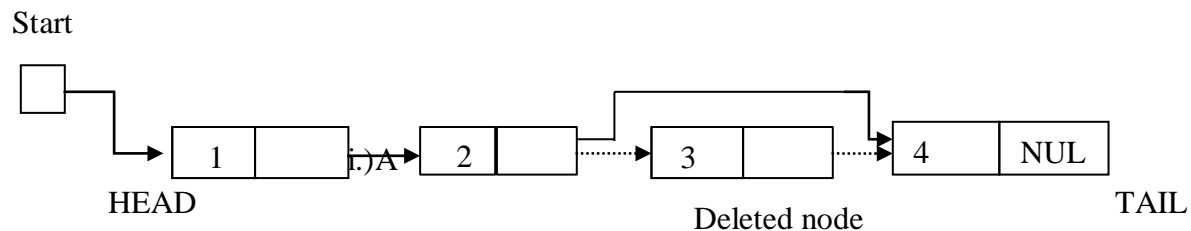
If the element to be deleted is located at the first position, the deletion procedure only consists of advancing the HEAD pointer to the second element in the list.



Delete node

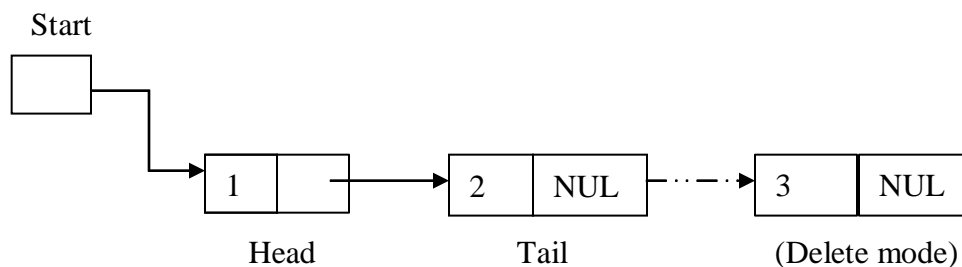
(ii.)At a certain position of the list:-

If an element in the middle of the list has to be deleted, only one pointer value has to be changed. The value in the link field of the preceding node is changed to the address of succeeding node.



If the element to be deleted is the last node in the list, the link of the previous is made NULL

Thereby excluding the last element from the list.



4. SEARCHING:-

Searching refers to the finding of an item or data from a given list of elements .So in case of link list searching means finding a specific code from a specific list. In this operation we compare the node value to be searched with each node value in the list.

CONCLUSION:

Hence we have studied all the operations on link list successfully.

Assignment No. 9

Problem Statement

Write C++ program to store set of negative and positive numbers using linked list. Write functions

- a) Insert numbers
- b) Delete nodes with negative numbers
- c) To create two more linked lists using this list, one containing all positive numbers and other containing negative numbers
- d) For two lists that are sorted; Merge these two lists into third resultant list that is sorted

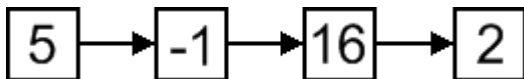
Types of linked list

Singly-linked list

Linked list is a very important dynamic data structure. Basically, there are two types of linked list, singly-linked list and doubly-linked list. In a singly-linked list every element contains some data and a link to the next element, which allows to keep the structure. On the other hand, every node in a doubly-linked list also contains a link to the previous node. Linked list can be an underlying data structure to implement stack, queue or sorted list.

Example

Sketchy, singly-linked list can be shown like this:



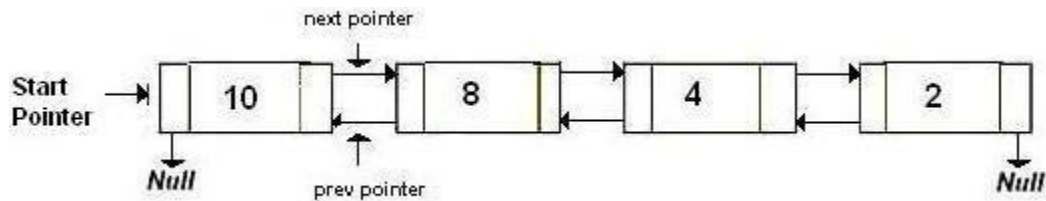
Each cell is called a node of a singly-linked list. First node is called head and it's a dedicated node. By knowing it, we can access every other node in the list. Sometimes, last node, called tail, is also stored in order to speed up add operation

Performance

- 1. The advantage of a singly linked list is its ability to expand to accept virtually unlimited number of nodes in a fragmented memory environment.
- 2. The disadvantage is its speed. Operations in a singly-linked list are slow as it uses sequential search to locate a node.

Doubly Linked List

Doubly-linked list is a more sophisticated form of linked list data structure. Each node of the list contain two references (or links) – one to the previous node and other to the next node. The previous link of the first node and the next link of the last node points to NULL. In comparison to singly-linked list, doubly-linked list requires handling of more pointers but less information is required as one can use the previous links to observe the preceding element. It has a dynamic size, which can be determined only at run time.



Performance

1. The advantage of a doubly linked list is that we don't need to keep track of the previous node for traversal or no need of traversing the whole list for finding the previous node.
2. The disadvantage is that more pointers need to be handled and more links need to be updated.

Circular Linked List

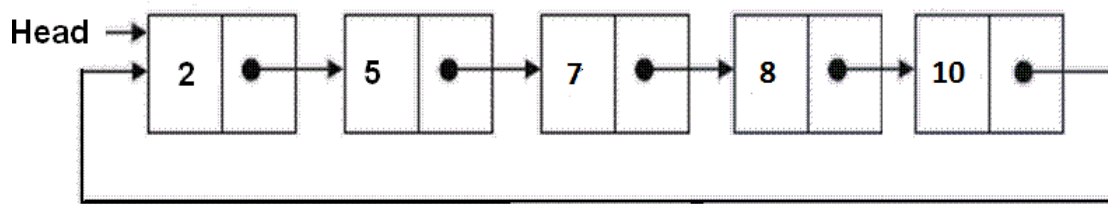
Circular linked list is a more complicated linked data structure. In this the elements can be placed anywhere in the heap memory unlike array which uses contiguous locations. Nodes in a linked list are linked together using a next field, which stores the address of the next node in the next field of the previous node i.e. each node of the list refers to its successor and the last node points back to the first node unlike singly linked list. It has a dynamic size, which can be determined only at run time.

Performance

1. The advantage is that we no longer need both a head and tail variable to keep track of the list. Even if only a single variable is used, both the first and the last list elements can be found in constant time. Also, for implementing queues we will only need one pointer namely tail, to locate both head and tail.

Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

2. The disadvantage is that the algorithms have become more complicated.



Basic Operations on a Circular Linked List

Insert – Inserts a new element at the end of the list.

Delete – Deletes any node from the list.

Find – Finds any node in the list.

Print – Prints the list

Insertion at the Beginning

Steps to insert a Node at beginning :

Step1. Create the new node

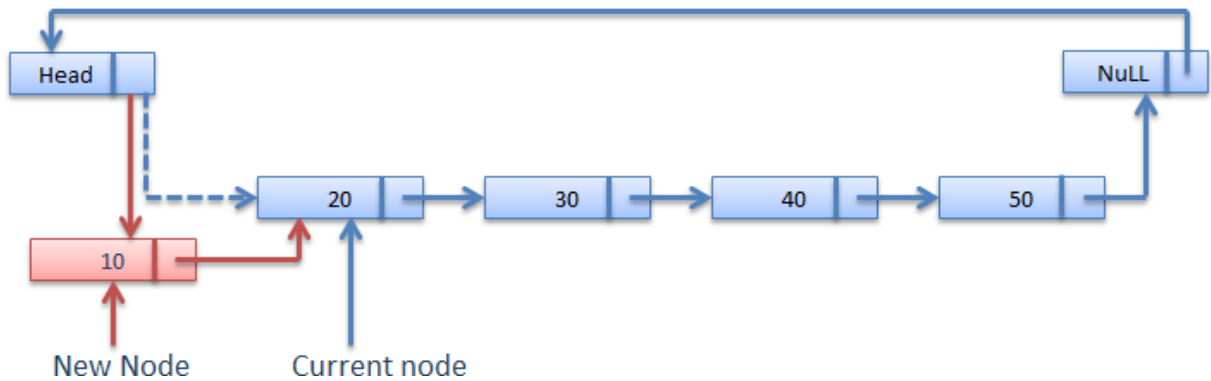
Step2. Set the new node's next to itself (circular!)

Step3. If the list is empty, return new node.

Step4. Set our new node's next to the front.

Step5. Set tail's next to our new node.

Step6. Return the end of the list.



Insertion at the End

Steps to insert a Node at the end :

Step1. Create the new node

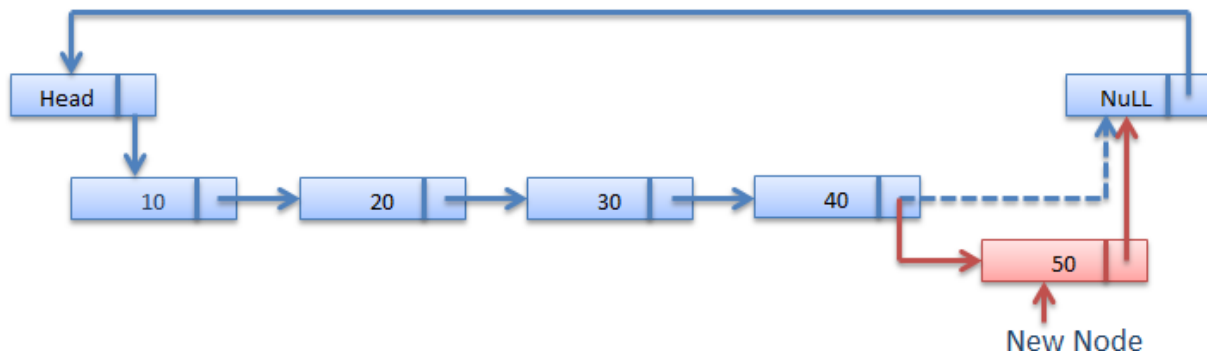
Step2. Set the new node's next to itself (circular!)

Step3. If the list is empty, return new node.

Step4. Set our new node's next to the front.

Step5. Set tail's next to our new node.

Step6. Return the end of the list.



Searching for an Element in the List

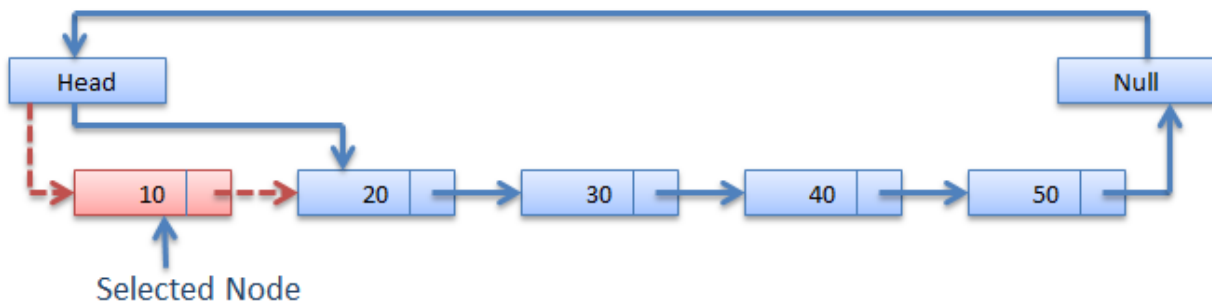
In searching we do not have to do much, we just need to traverse like we did while getting the last node, in this case we will also compare the **data** of the Node. If we get the Node with the same data, we will return it, otherwise we will make our pointer point the next Node, and so on.

Deleting a Node from the List

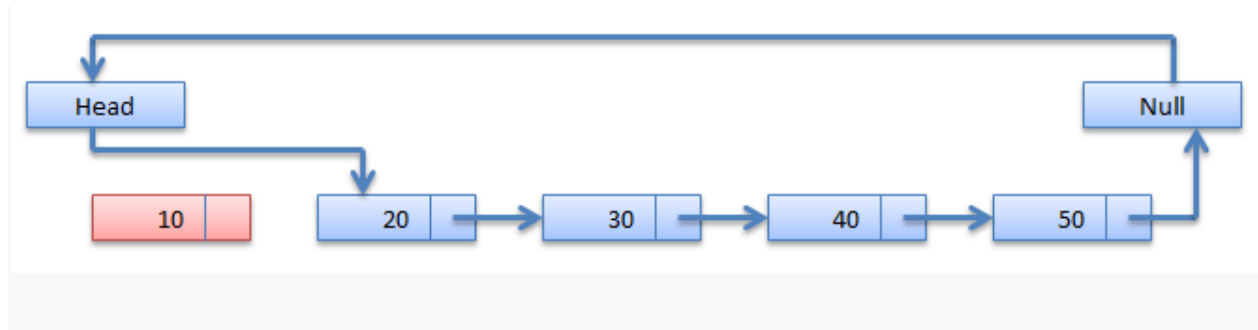
Deleting a node can be done in many ways, like we first search the Node with **data** which we want to delete and then we delete it. In our approach, we will define a method which will take the **data** to be deleted as argument, will use the search method to locate it and will then remove the Node from the List.

To remove any Node from the list, we need to do the following :

- If the Node to be deleted is the first node, then simply set the Next pointer of the Head to point to the next element from the Node to be deleted. And update the next pointer of the Last Node as well.
- If the Node is in the middle somewhere, then find the Node before it, and make the Node before it point to the Node next to it.
- If the Node is at the end, then remove it and make the new last node point to the head.



After Deletion



Applications of linked list

- 1) Multiple Precision Arithmetic
- 2) To implement associative arrays
- 3) Stack and Queue can be represented dynamically using linked list
- 4) Representation and manipulation of polynomial expressions

Conclusion

Hence, we learnt insertion, deletion, sorting and merging linked list.

ASSIGNMENT NO. 10

Problem Statement:

Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions:

1. Operands and operator, both must be single character.
2. Input Postfix expression must be in a desired format.
3. Only '+', '-', '*' and '/' operators are expected.

Theory:

When you write an arithmetic expression such as $B * C$, the form of the expression provides you with information so that you can interpret it correctly. In this case we know that the variable B is being multiplied by the variable C since the multiplication operator $*$ appears between them in the expression. This type of notation is referred to as **infix** since the operator is *in between* the two operands that it is working on.

Consider another infix example, $A + B * C$. The operators $+$ and $*$ still appear between the operands, but there is a problem. Which operands do they work on? Does the $+$ work on A and B or does the $*$ take B and C ? The expression seems ambiguous.

In fact, you have been reading and writing these types of expressions for a long time and they do not cause you any problem. The reason for this is that you know something about the operators $+$ and $*$. Each operator has a **precedence** level. Operators of higher precedence are used before operators of lower precedence. The only thing that can change that order is the presence of parentheses. The precedence order for arithmetic operators places multiplication and division above addition and subtraction. If two operators of equal precedence appear, then a left-to-right ordering or associativity is used.

Let's interpret the troublesome expression $A + B * C$ using operator precedence. B and C are multiplied first, and A is then added to that result. $(A + B) * C$ would force the addition of A and B to be done first before the multiplication. In expression $A + B + C$, by precedence (via associativity), the leftmost $+$ would be done first.

Although all this may be obvious to you, remember that computers need to know exactly what operators to perform and in what order. One way to write an expression that guarantees there will be no confusion with respect to the order of operations is to create what is called a **fully parenthesized** expression. This type of expression uses one pair of parentheses for each operator. The parentheses dictate the order of operations; there is no ambiguity. There is also no need to remember any precedence rules.

The expression $A + B * C + D$ can be rewritten as $((A + (B * C)) + D)$ to show that the multiplication happens first, followed by the leftmost addition. $A + B + C + D$ can be written as $((((A + B) + C) + D)$ since the addition operations associate from left to right.

There are two other very important expression formats that may not seem obvious to you at first. Consider the infix expression $A + B$. What would happen if we moved the operator before

the two operands? The resulting expression would be $+ A B$. Likewise, we could move the operator to the end. We would get $A B +$. These look a bit strange.

These changes to the position of the operator with respect to the operands create two new expression formats, **prefix** and **postfix**. Prefix expression notation requires that all operators precede the two operands that they work on. Postfix, on the other hand, requires that its operators come after the corresponding operands. A few more examples should help to make this a bit clearer.

$A + B * C$ would be written as $+ A * B C$ in prefix. The multiplication operator comes immediately before the operands B and C , denoting that $*$ has precedence over $+$. The addition operator then appears before the A and the result of the multiplication.

In postfix, the expression would be $A B C * +$. Again, the order of operations is preserved since the $*$ appears immediately after the B and the C , denoting that $*$ has precedence, with $+$ coming after. Although the operators moved and now appear either before or after their respective operands, the order of the operands stayed exactly the same relative to one another.

Table 2: Examples of Infix, Prefix, and Postfix

Infix Expression	Prefix Expression	Postfix Expression
$A + B$	$+ A B$	$A B +$
$A + B * C$	$+ A * B C$	$A B C * +$

Now consider the infix expression $(A + B) * C$. Recall that in this case, infix requires the parentheses to force the performance of the addition before the multiplication. However, when $A + B$ was written in prefix, the addition operator was simply moved before the operands, $+ A B$. The result of this operation becomes the first operand for the multiplication. The multiplication operator is moved in front of the entire expression, giving us $* + A B C$. Likewise, in postfix $A B +$ forces the addition to happen first. The multiplication can be done to that result and the remaining operand C . The proper postfix expression is then $A B + C *$.

Consider these three expressions again (see [Table 3](#)). Something very important has happened. Where did the parentheses go? Why don't we need them in prefix and postfix? The answer is that the operators are no longer ambiguous with respect to the operands that they work on. Only infix notation requires the additional symbols. The order of operations within prefix and postfix expressions is completely determined by the position of the operator and nothing else. In many ways, this makes infix the least desirable notation to use.

Table 3: An Expression with Parentheses

Infix Expression	Prefix Expression	Postfix Expression
$(A + B) * C$	$* + A B C$	$A B + C *$

[Table 4](#) shows some additional examples of infix expressions and the equivalent prefix and postfix expressions. Be sure that you understand how they are equivalent in terms of the order of the operations being performed.

Table 4: Additional Examples of Infix, Prefix, and Postfix

Infix Expression	Prefix Expression	Postfix Expression
$A + B * C + D$	$++ A * B C D$	$A B C * + D +$
$(A + B) * (C + D)$	$* + A B + C D$	$A B + C D + *$
$A * B + C * D$	$+ * A B * C D$	$A B * C D * +$
$A + B + C + D$	$+++ A B C D$	$A B + C + D +$

Infix Notation

We write expression in **infix** notation, e.g. $a-b+c$, where operators are used **in**-between operands. It is easy for us humans to read, write and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix Notation

In this notation, operator is **prefixed** to operands, i.e. operator is written ahead of operands. For example $+ab$. This is equivalent to its infix notation $a+b$. Prefix notation is also known as **Polish Notation**.

Postfix Notation

This notation style is known as **Reversed Polish Notation**. In this notation style, operator is **postfixed** to the operands i.e., operator is written after the operands. For example $ab+$. This is equivalent to its infix notation $a+b$.

The below table briefly tries to show difference in all three notations –

S.n.	Infix Notation	Prefix Notation	Postfix Notation
------	----------------	-----------------	------------------

1	$a + b$	$+ a b$	$a b +$
---	---------	---------	---------

2	$(a + b) * c$	$* + a b c$	$a b + c *$
3	$a * (b + c)$	$* a + b c$	$a b c + *$
4	$a / b + c / d$	$+ / a b / c d$	$a b / c d / +$
5	$(a + b) * (c + d)$	$* + a b + c d$	$a b + c d + *$
6	$((a + b) * c) - d$	$- * + a b c d$	$a b + c * d -$

Parsing Expressions

As we have discussed, it is not very efficient way to design an algorithm or program to parse infix notations. Instead, these infix notations are first converted into either postfix or prefix notations and then computed.

To parse any arithmetic expression, we need to take care of operator precedence and associativity also.

Precedence

When an operand is in between two different operator, which operator will take the operand first, is decided by the precedence of an operator over others. For example –

As multiplication operation has precedence over addition, $b * c$ will be evaluated first. A table of operator precedence is provided later.

Associativity

Associativity describes the rule where operators with same precedence appear in an expression. For example, in expression $a+b-c$, both $+$ and $-$ has same precedence, then which part of expression will be evaluated first, is determined by associativity of those operators. Here, both $+$ and $-$ are left associative, so the expression will be evaluated as $(a+b)-c$.

Precedence and associativity, determines the order of evaluation of an expression. An operator precedence and associativity table is given below (highest to lowest) –

S.n.	Operator	Precedence	Associativity
1	Esponentiation $^$	Highest	Right Associative
2	Multiplication ($*$) & Division ($/$)	Second Highest	Left Associative

3 Addition (+) & Subtraction (-) Lowest Left Associative

The above table shows the default behavior of operators. At any point of time in expression evaluation, the order can be altered by using parenthesis. For example –

In $a+b*c$, the expression part $b*c$ will be evaluated first, as multiplication has precedence over addition. We here use parenthesis to make $a+b$ be evaluated first, like $(a+b)*c$.

Postfix Evaluation Algorithm

We shall now look at the algorithm on how to evaluate postfix notation –

- Step 1 – scan the expression from left to right
- Step 2 – if it is an operand push it to stack
- Step 3 – if it is an operator pull operand from stack and perform operation
- Step 4 – store the output of step 3, back to stack
- Step 5 – scan the expression until all operands are consumed
- Step 6 – pop the stack and perform operation

Conclusion: We have implemented C++ program for expression conversion.

Assignment No. 11

Problem Statement:

Write program to implement a priority queue in C++ using an inorder List to store the items in the queue. Create a class that includes the data items(which should be template) and the priority (which should be int)The inorder list should contain these objects ,with operator <= overloaded so that the items with highest priority appear at the beginning of the list (which will make it relatively easy to retrieve the highest item.)

Theory

Priority Queue

Priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

Priority Queue is more specialized data structure than Queue. Like ordinary queue, priority queue has same method but with a major difference. In Priority queue items are ordered by key value so that item with the lowest value of key is at front and item with the highest value of key is at rear or vice versa. So we're assigned priority to item based on its key value. Lower the value, higher the priority. Following are the principal methods of a Priority Queue.

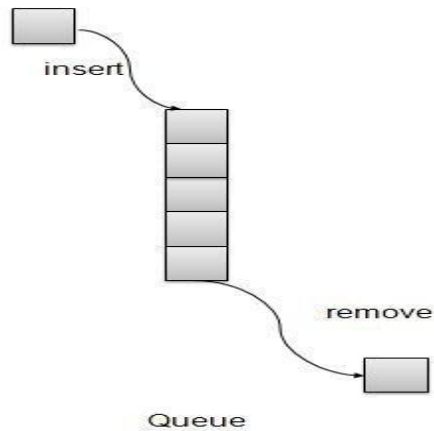
An important application of a priority queue is sorting

- PriorityQueueSort (collection S of n elements)
- put the elements in S in an initially empty priority queue by means of a series of n insert() operations on the pqueue, one for each element
- extract the elements from the pqueue by means of a series of n removeMin() operations

Basic Operations

- insert / enqueue – add an item to the rear of the queue.
- remove / dequeue – remove an item from the front of the queue.

Priority Queue Representation



There are few more operations supported by queue which are following.

- `getHighestPriority()`: Returns the highest priority item.
- `Peek` – get the element at front of the queue.
- `isFull` – check if queue is full.
- `isEmpty` – check if queue is empty.

Using Array: We're going to implement Queue using array. A simple implementation is to use array of following structure.

```
Struct item{
```

```
    Int item;
```

```
    Int priority;
```

```
}
```

`insert()` operation can be implemented by adding an item at end of array in $O(1)$ time.

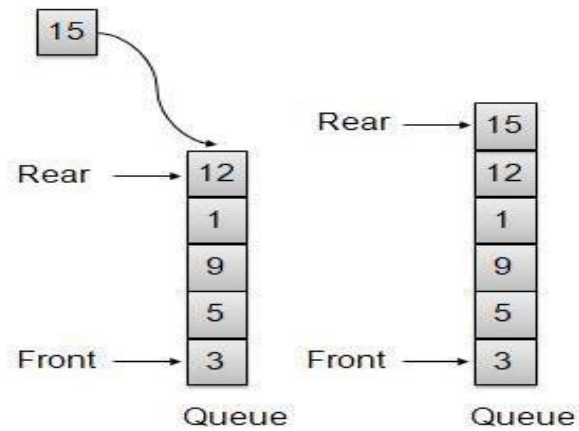
`getHighestPriority()` operation can be implemented by linearly searching the highest priority item in array. This operation takes $O(n)$ time.

`deleteHighestPriority()` operation can be implemented by first linearly searching an item, then removing the item by moving all subsequent items one position back.

We can also use Linked List, time complexity of all operations with linked list remains same as array. The advantage with linked list is `deleteHighestPriority()` can be more efficient as we don't have to move items.

Insert/Enqueue Operation

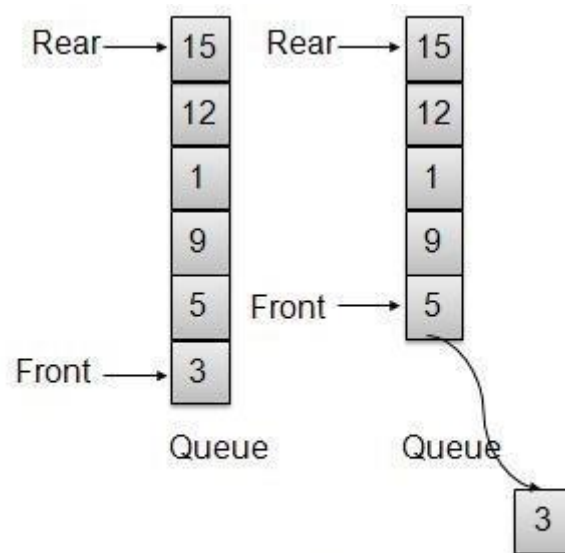
Whenever an element is inserted into queue, priority queue inserts the item according to its order. Here we're assuming that data with high value has low priority.



One item inserted at rear end

Remove/Dequeue Operation

Whenever an element is to be removed from queue, queue get the element using item count. Once element is removed. Item count is reduced by one.



One Item removed from front

Conclusion

Hence, we learnt to implement a priority queue in C++ using an inorder.

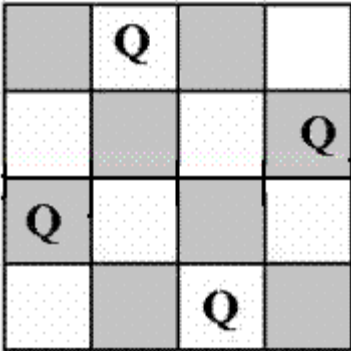
Assignment No. 12

Problem Statement:

A classic problem that can be solved by backtracking is called the Eight Queens problem, which comes from the game of chess. The chess board consist of 64 square arranged in an 8 by 8 grid. The board normally alternates between black and white square, but this is not relevant for the present problem. The queen can move as far as she wants in any direction, as long as she follows a straight line, vertically, horizontally, or diagonally. Write C++ program with recursive function for generating all possible configurations for 4-queen's problem.

Theory:

The N Queen is the problem of placing N chess queens on an $N \times N$ chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



	Q		
			Q
Q			
		Q	

The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example following is the output matrix for above 4 queen solution.

```
{ 0, 1, 0, 0}  
{ 0, 0, 0, 1}  
{ 1, 0, 0, 0}  
{ 0, 0, 1, 0}
```

NaïveAlgorithm

Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints.

while there are untried configurations

```
{  
    generate the next configuration  
    if queens don't attack in this configuration then  
    {  
        print this configuration;  
    }  
}
```

BacktrackingAlgorithm

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

- 1) Start in the leftmost column
- 2) If all queens are placed
 return true
- 3) Try all rows in the current column. Do following for every tried row.
 - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
 - b) If placing queen in [row, column] leads to a solution then
 return true.
 - c) If placing queen doesn't lead to a solution then unmark this [row,column] (Backtrack) and go to step (a) to try other rows.
- 3) If all rows have been tried and nothing worked, return false to trigger
 backtracking.

The n -queens problem is to find *all* ways to place n queens on an $n \times n$ checkerboard so that no queen may take another. As an example, the eight-queens problem is to find *all* ways to place eight queens on the chessboard so that no queen may take any other that is, move into a position on its next move that is occupied by any other queen. The question is how to solve the n -queens problem. At first glance this problem has little relation to trees, but let us see. It is not even clear that a solution exists for all n .

Since a queen can be moved along one of its rows, columns, or diagonals, a solution clearly is achieved by specifying how to place the n queens so that *exactly* one queen appears in each row and column, and no more than one queen appears on any diagonal of the board. One could attempt to find a solution by searching through all possible placements of the n queens satisfying these restrictions. There are, however, $n!$ such placements. Even for a moderate n , this is obviously not a practical solution. Instead we must construct a solution. The idea behind the construction is to determine if the placements obtained for the first i queens can lead to a solution. If they cannot, abandon that construction and try another, since placing the remaining $n - i$ queens will not be fruitful. This *may* avoid the need to consider all possible constructions.

To construct a solution involves testing a partial construction to determine whether or not it can lead to a solution or must be abandoned as a dead end. This testing is the next question.

Consider a general tree, with each of its nodes representing a sequence of decisions on the placement of queens. Take the root to represent the initial situation, in which no decisions have yet been made. The root will have n successors. Each successor corresponds to a choice of row for the placement of the first queen. Each of these successor nodes, in turn, has n successors, corresponding to a choice of row for the placement of the second queen, and so on. The tree for the four-queens problem is shown in [Figure 7.17](#).

Since exactly one queen must appear in each column in *any* solution, assume the i th queen is placed in column i . Hence, each path, from the root to a node at depth i in the tree, specifies a partial construction in which the first i queens have been placed on the board. The path indicated in the four-queens tree specifies a partial construction in which the first, second, and third queens have been placed in rows 2, 4, and 1 (and columns 1, 2, and 3), respectively. The tree helps us visualize all possible solutions and provides a framework in which to view what we are doing.

Suppose a path has been found to a node at depth k . The path is *feasible* if none of the k queens whose positions are specified by the path can take any other. If the node at depth k is a terminal node, and the path is feasible, then a solution has been found and may be printed. If the node at depth k is not terminal, and the path is feasible, we want to extend the path to a node at depth $k + 1$. Let p point to such a node. Then node. p must be a successor of the node on the path at depth k .

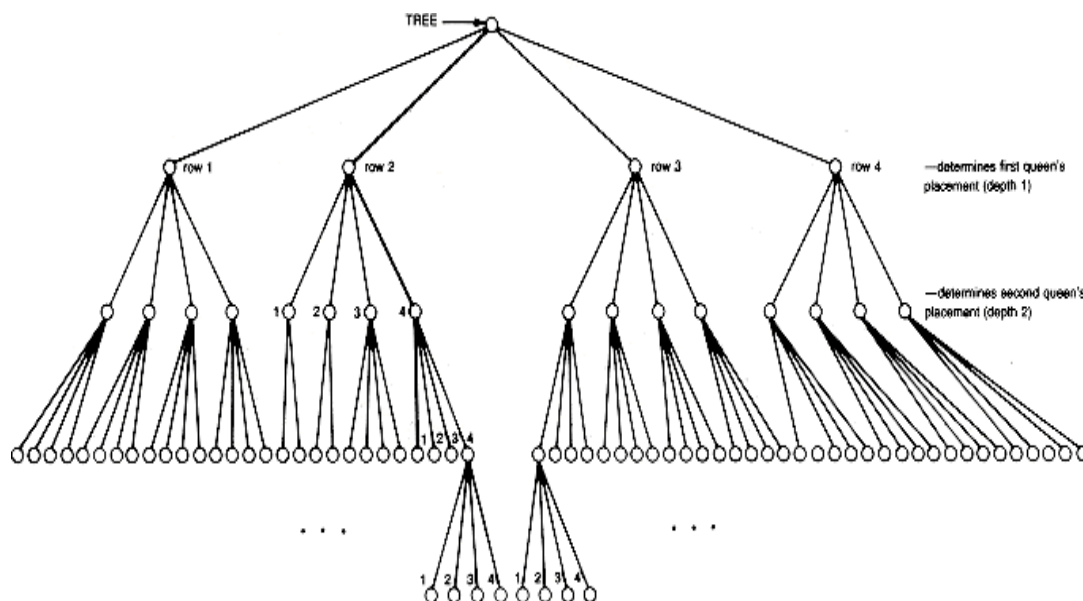


Figure 7.17 Tree for the Four-queens Problem

P is *feasible* if the position that node p specifies for the $(k + 1)$ th queen does not allow it to take any of the other k queens. If p is feasible, the path can be extended downward in the tree to include node p , so there is now a feasible path of depth $k + 1$. If p is *not* feasible, then another successor of the node at depth k must be tried until a feasible one is found to extend the path, or until all have been tried and none is feasible. In the latter case, the choice of node at depth k cannot be extended to a solution, and the computer backs up in the tree to a node representing a shorter path that has not yet been explored. This procedure to extend the new path is repeated until a solution is found or until all possibilities have been tried.

The procedure described amounts to a modified general tree traversal called ***backtracking***. A different approach would be to create an algorithm for the problem by applying a general tree preorder traversal that uses a process routine. The process routine would check each node to be processed to see whether the node is terminal and represents a feasible path. If so, it prints the solution. This approach amounts to an exhaustive search of all $n!$ possibilities, since the preorder traversal backs up only when the preorder traversal of a subtree has been completed that is, after a *terminal* node has been processed. The backtracking traversal need not access and process all nodes. It backs up when the traversal of a subtree has been completed or when it becomes known that no path involving the root of a subtree can be extended to a solution. The backtracking procedure generates only *feasible* paths not *all* paths. In effect, it prunes the tree by ignoring all nodes that cannot lead to a solution.

Assignment No. 13

Problem Statement:

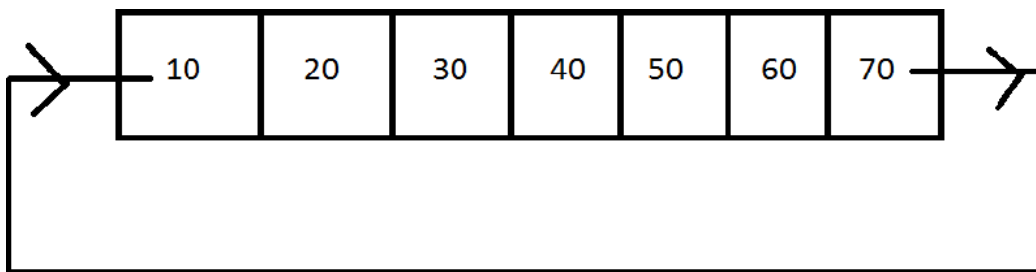
Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using circular queue using array.

Theory

CIRCULAR QUEUE:

A circular queue is an abstract data type that contains a collection of data which allows addition of data at the end of the queue and removal of data at the beginning of the queue. Circular queues have a fixed size. In a standard queue data structure re-buffering problem occurs for each dequeue operation. To solve this problem by joining the front and rear ends of a queue to make the queue as a circular queue Circular queue is a linear data structure. It follows FIFO principle.

- In circular queue the last node is connected back to the first node to make a circle.
- Circular linked list follow the First In First Out principle
- Elements are added at the rear end and the elements are deleted at front end of the queue
- Both the front and the rear pointers points to the beginning of the array.
- It is also called as “Ring buffer”.
- Items can inserted and deleted from a queue in $O(1)$ time.



Circular Queue

Circular Queue can be created in three ways they are

- Using single linked list

- Using double linked list
- Using arrays

Using array

In arrays the range of a subscript is 0 to n-1 where n is the maximum size. To make the array as a circular array by making the subscript 0 as the next address of the subscript n-1 by using the formula $\text{subscript} = (\text{subscript} + 1) \% \text{maximum size}$. In circular queue the front and rear pointer are updated by using the above formula.

To implement a circular queue data structure using array, we first perform the following steps before we implement the actual operation:

Step 1: include all the header files which are used in program and define the constant SIZE with specific value.

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

Step 3: Create a one dimension array with above defined 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 operation list and make suitable fuction calls to perform operation selected by the user on the circular queue.

Algorithm for Enqueue operation using array

Step 1. start

Step 2. if (front == (rear+1)%max)
 Print error “circular queue overflow “

Step 3. else
 { rear = (rear+1)%max
 Q[rear] = element;
 If (front == -1) f = 0;
 }

Step 4. stop

Algorithm for Dequeue operation using array

Step 1. start

Step 2. if ((front == rear) && (rear == -1))
 Print error “circular queue underflow “

Step 3. else
 { element = Q[front]
 If (front == rear) front=rear = -1

```
        Else
            Front = (front + 1) % max
    }
    Step 4. Stop
```

Algorithm for create () function:-

```
Step 1) allocate the memory for newnode
        newnode = new (node )
Step 2) newnode->next=newnode. // circular
Step 3) Repeat the steps from 4 to 5 until choice = „n“
Step 4) if (root=NULL)
        root = prev=newnode // prev is a running pointer which points last node of a list
    else
        newnode->next = root
        prev->next = newnode
        prev = newnode
Step 5) Read the choice
Step 6) return
```

Algorithm for display () function :-

```
Step 1) start
Step 2) declare a variable of pointer to structure node called temp, assign root to temp
        temp = root
Step 3) display temp->info
Step 4) temp = temp->next
Step 5) repeat the steps 6 until temp = root
Step 6) display temp info
Step 7) temp=temp->next
Step 8) return
```

Conclusion

Hence, we learnt to implement circular queue using array in C++.

Oral Questions

GROUP A

ASSIGNMENT 1

1. What is disjoint set?
2. What are the different operations on sets?
3. What are the different properties of sets?
4. What is proper subset?
5. What is singleton set or unit set?
6. What is algebra of set of laws?
7. What do you mean by multiset?
8. Explain the principle of exclusion and inclusion?
9. What is Cartesian product?
10. Define De-Morgan's Law.

ASSIGNMENT 2

- 1) When is the worst case occurred in linear search algorithm?
- 2) What does the complexity of sorting algorithm measures as a function of the number n of items to be sorted?
- 3) When does Average case occurs in linear search algorithm?
- 4) The complexity of bubble sort algorithm is
- 5) Order is the best possible for array sorting algorithm which sorts n item
- 6) Which order is the best possible for array sorting algorithm which sorts n item?
- 7) The time complexity of a quick sort algorithm which makes use of median, found by an $O(n)$ algorithm, as pivot element is?
- 8) Which are the notations of time complexity?
- 9) Time complexity of bubble sort in best case is?
10. Counting sort performsNumbers of comparisons between input elements.

ASSIGNMENT 3

- Q1. What do you mean by permutation and combination?
- Q2. Give the formula for permutation and combination?
- Q3. State the difference between permutation and combination?
- Q4. What are the different fundamental principles of counting?
- Q5. What are the types of permutations?
- Q6. Explain the password code?
- Q7. Give some examples on permutations and combinations?
- Q8. What are the applications of permutations and combinations?
- Q9. Describe some basic operations of permutation?
- Q10. What is the use of keyword „rand“ in the password program?

ASSIGNMENT 4

- Q.1 Define row matrix?
- Q.2 If all the element of two equal square matrices i.e. all $a[i][j]=b[i][j]$ are equal ,can we say matrices equal or not?
- Q.3 if all the elements of two equal size square matrices i.e. all $a[i][j]=b[j][i]$ are equal , will b matrix be transpose of matrix „a“?
- Q.4 Transpose can be only done if matrix is square matrix , is it true or false?
- Q.5 What is square matrix?
- Q.6 Representation of size of a matrix is.....*.....?
- Q.7 How addition is performed in matrix?
- Q.8 What will $(A^T)^T=.....?$
- Q.9 What is a diagonal matrix?
- Q.10 Define identity matrix?

ASSIGNMENT 5

1. How does transpose of a square matrix found?
2. What is symmetry?
3. In order to multiplication of 2 matrix no. of columns of a matrix is equal to _____?
4. What is an array?
5. How many types of array?
6. Define 1D array?
7. Define multi dimensional array?
8. 2D array are generally used in?
9. Two matrices to be added must be _____?
10. Explain the mapping of a 2D array?

ASSIGNMENT 6

- Q.1 What does the string(string1, string2) library function mean ?
- Q.2 Give any 2 operations that can be performed using string.
- Q.3 What is the library function to concatenate two strings, string2 is appended to string 1 ?
- Q.4 Under which header file string handling functions are defined ?
- Q.5 Which two functions are handled in string ?
- Q.6 What is the output of the following program ?

```
#include<stdio.h>
int main()
{
char string[20] = "ABCD";
printf("The string is : %s \n", string);
return 0;
}
```

- Q.7 Which library function gives length of string ?
- Q.8 What is the loop used for scanning string in an array?
- Q.9 Explain the code ?
- Q.10 What is the logic for palindrome?

ASSIGNMENT 7

- Q1. How to find the first repeating element in an array of integer?
- Q2. Why do we need algorithm analysis?
- Q3. What is linear data structure?
- Q4. What are asymptotic notations?
- Q5. What is the worst case run time complexity of binary search?
- Q6. Is dynamic allocation of array possible in c?

Q7. Which is the algorithm design technique used in the quick sort algorithm?

Q8. Define time complexity?

Q9. What are the two main measures for the efficiency of algorithm?

Q10. How to display multidimensional array ?

GROUP B

Q 1] What are the merits of linked list over array?

Q2] What are the memory management functions during runtime?

Q3] Explain in brief what is the structure of a singly linked list and also explain what Does each of its part consist of?

Q4] Give the importance of a head pointer?

Q5] State any 5 basic operations performed on singly linked list?

Q 6] What is the basic concept of Linked list?

Q 7] What is the main difference between Array and Linked list?

Q 8] Describe the different types of Linked lists in short?

Q 9] What is the main advantage of Linked list?

Q 10] Which basic operations can be performed on the Linked list?

Q 10] Why is insertion or deletion in a Linked List a constant time operation?

Q 11] What is the difference between algorithms of Insertion and Deletion of any nodes?

Q12] Explain the situation of underflow in link list ?

Q13] How many null pointer are there in a doubly circular link list ?

Q14] What is the time required to delete a node with given address between singly and doubly link list?

Q15] What is the difference between singly and doubly link list?

Q16] Explain the term of insertion in doubly link list?

Q17]explain the three term of deletion in link list?

Q18]what are the application of circular link list?

Q19]what are the operation perform on doubly link list?

Q20]explain the term of garbage collection?

Q21]A linear collection of data elements where the linear node is given by means of pointer is called?

Q22]Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?

Q23]What would be the asymptotic time complexity to add an element in the linked list?

Q24]Consider the following definition in c programming language

```
struct node
{
int data;
struct node * next;
}
typedef struct node NODE;
NODE *ptr;
```

1. Which of the following c code is used to create new node?

ANS: ptr=(NODE*)malloc(sizeof(NODE));

Linked lists are not suitable to for the implementation of?

In worst case, the number of comparison need to search a singly linked list of length n for a given element is?

The concatenation of two list can performed in O(1) time. Which of the following variation of linked list can be used?

Q26] What are the applications of circular link list..? Explain any one application..?

Q27 What is the logic behind the insertion and deletion of a node into a circular link list..?

Q28] Explain the code for insertion of a node at a specified position into a circular link list..?

Q29] Explain the code for traversing a circular linked list..?

Q30] What is the use of 'flags' in circular link list..?

Q31]Consider an implementation of unsorted singly linked list. Suppose it has its representation with a head and tail pointer. Given the representation, which of the following operation can be implemented in $O(1)$ time?

ANS=Deletion of the front node of linked list.

Q32]Each node in singly linked list has fields.?

ANS = 2

Q33]Each node in a linked list must contain at least ?

ANS=Two fields

Q34]To insert a new node in linked list free node will be available in ?

ANS Avail list

Q35]Header linked lists are frequently used for maintaining in memory?

ANS= Polynomials

Q36]Linked lists are not suitable to for the implementation of?

Q37] In circular linked list, insertion of node requires modification of?

Q38] A variant of the linked list in which none of the node contains NULL pointer is?

Q39] What kind of linked list is best to answer question like “What is the item at position n?

Ans) Array implementation of linked list.

Q40] The concatenation of two list can performed in $O(1)$ time. Which of the following variation of linked list can be used?

Q41]Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?

Q42] A linear collection of data elements where the linear node is given by means of pointer is called?

Q43]How is circular link list different from singly link list

Q44]Stack and Queue are dynamically represented using

Q45]Which is the more sophisticated linked list among all &why?

Q46]What kind of operations are performed on circular linked list?

Q47]The extra element at the head of the list is called as ?

Q48]A node consist of ?

GROUP C-STACKS

1. What are primitive operations in stack?
2. Explain the stack concept?
3. Explain stack as an ADT?
4. What are Multiple stacks?
5. How to represent stack using sequential Organization?
6. Define linked stacks?
7. Why do we need postfix and prefix operator?
8. Explain the evaluation of an infix expression?
9. Explain the evaluation of an prefix expression?
10. Explain the evaluation of an postfix expression?
11. What is backtracking algorithm?
12. Explain LIFO structure.
13. What are the operations of stack?
14. What are the functions for stack operations?
15. Explain Evaluation of a postfix expression using a stack with an algorithm?
16. Explain one application of stack with example.
17. Which data structures is applied when dealing with a recursive function?
18. Is backtracking an algorithm or is it more of a concept?
19. When do we use backtracking?
20. What are the advantages and disadvantages of a backtracking algorithm?
21. Explain Linked representation of stacks along with structure of node?
22. Explain the conversion of an Expression from infix to postfix?
23. Explain the conversion of an Expression from infix to prefix?
24. Explain the conversion of an Expression from postfix to infix?

25. Explain the conversion of an Expression from postfix to prefix?
26. Explain the conversion of an Expression from prefix to infix?
27. Explain fully parenthesizing an infix expression?
28. What is well formedness of parenthesis?
29. How can two stacks be represented in an Array?
30. How can multiple stacks be represented in an Array?
31. Explain the logic for finding the length of a string?
32. Explain the logic behind reversing a string?
33. Explain the logic behind searching a number in an Array?
34. Explain the logic behind finding largest element in an array?
35. Explain the logic for binary search?
36. What is use of stack in back tracking?
37. Explain the concept of removal of recursion?
38. What are rules for converting a recursive algorithm to non recursive one?
39. Simply explain the 8 queen problem with solutions and the logic behind it?
40. Explain the Use of stack in Backtracking?
41. What is direct recursion?
42. What is indirect recursion?
43. What is tail recursion?
44. What is tree recursion?
45. Explain the conversion of a recursive function to an equivalent C-function?

GROUP D-QUEUES

Q1.)A queue is which type of collection?

Q2.)In the standard library of class, the data type queue comes in which class?

Q3.)what type of operations are allowed in queue?

Q4.)Explain briefly about breadth first search with queue.

Q5.)Difference between stack and queue?

Q6)What are the different applications of queues?

Q7)How can we represent a queue using array?explain in brief?

Q8)How can we insert and delete elements in queue? Also give queue data tp?

Q9)What is disadvantage of linear queue?suggest suitable method to overcome.?

Q10)Explain queue as ADT.?

Q11)What is a queue?

Q12)What are the applications of queue?

Q13)What are the prototype of function used for various operations on queue?

Q14)What is a dequeue?

Q15) What is the datatype for queue in an array?

Q.16) which entities are required for array representation of queue?

Q17)What are the steps for insertion of an element in queue?

Q18) write steps for inserting element in circular queue?

Q19) List the method to implement dequeue?

Q20) Which operation associated with dequeue?

Q21)A queue is which type of collection?

Q22.)In the standard library of class, the data type queue comes in which class?

Q23)what type of operations are allowed in queue?

Q24) Explain briefly about breadth first search with queue.

Q25) Difference between stack and queue?

Q26) Describe key operations?

Q27) How many stacks are needed to implement a queue?

Q28) How many queues are needed to implement a stack?

Q29) Give the advantage of the circular queue using array?

Q30) Difference between Queue and dequeue?

Q31) What is priority Queue?

Q32) What is round robin technique...explain?

Q33) Give some applications of queue?

Q34) What is Josephus problem?

Q35) What are the different types of operations which we can do in priority queue?

Q36) What are the operations performed by the queue?

Q37) What care should be taken while initializing and deletion of the queue?

Q38). What do you mean by circular queue? What are its advantages?

Q39) What are the operations possible by the

Q40) State the application of the queue?

Q41) What are the steps for insertion and deletion of the queue?

Q42) How many pointers required to be maintained in case of

GROUP E : SEARCHING & SORTING

1. How many are the Case Study of the Fibonacci Search and Name them.
2. Explain about the Non-Access Memory Storage in short.
3. Explain the terms:
 - Internal Sorting
 - External Sorting
4. Difference between the Internal and External Sorting?
5. Explain about the Passes. Write the example on the Passes?
6. What are the correct intermediate steps of the following data set when it is being sorted with the Shell sort? 15,20,10,18?
7. Shell sort is a generalization form of which sorting?
8. What is the main difference between a shell sort and the sort that is similar to shell sort?
9. True or False: When the shell sort is complete, the table has exactly three columns left? justify your answer?
10. "Each time the process repeats itself, there is a smaller number of the longer columns." check whether the given statement for shell sort algorithm is true or false? justify your answer?
11. Why does Fibonacci search requires the elements in non-decreasing order?
12. What are the conditions in comparison of the middle element and the key to be searched?
13. Binary Search Vs Fibonacci search. How is Fibonacci search better than Binary search?
14. How can a sorting algorithm which is not stable in nature be modified to stable?
15. Why worst case and average case performance is significant and not best case?
16. What is time complexity of bubble sort algorithm?
17. Is bubble sort a stable algorithm?
18. What would happen if bubble sort didn't keep track of the number of swaps made on each pass through the list?
19. What is the worst case scenario for bubble sort, and why?
20. What simple modification could be made to the bubble sort algorithm that would make it perform more efficiently ?

21. What is Binary Insertion Sort?
22. How to implement Insertion Sort for Linked List?
23. What is Time Complexity of insertion sort?
24. Where is insertion sort used?
25. What is the worst case time complexity of insertion sort where position of the data to be inserted is calculated using binary search?
26. Explain the algorithm of linear search?
27. explain the algorithm of binary search?
28. Explain the algorithm of sentinel search?
29. explain the efficiency of binary search?
30. Explain the algorithm fibonacci series?
31. what do you mean by sort order? explain in brief.
32. explain about stability efficiency
33. what do you mean by fibonacci search?
34. explain the term no of passes in sorting .
35. how will you generate fibonacci series?
36. What is prerequisite for binary search?
37. Which sentinel is available to users?
38. Explain sentinel search algorithm?
39. Explain fibonacci search with example.
40. Distinguish between linear and binary search.
41. When computing Fibonacci (10) how many times would Fibonacci (5) be called?
42. how long does it take for Fibonacci (45) to execute?
43. write down the efficiency of binary search?
44. what is the linear search algorithm?
45. give an estimate of an asymptotic characterization of the number of times the function is called when Fibonacci (n) is computed?
46. Why external sort is more efficient than internal sort?

47. What are the advantages of internal and external sorting?
48. why is it necessary for a sorting algorithm to be stable?
49. What is the logic behind fibonacci series?
50. Explain with an example Fibonacci search?
51. Fibonacci logic ??
52. Difference between linear and binary search?
53. Explain linear search.?
54. Explain binary search.?
55. What is sentinel search?
56. What is the worst case performance for bucket sort?
57. What is a bucket in data structure ?
58. What does it mean for an algorithm to be stable?
59. What is the process for Bucket sort ??
60. What are applications of Bucket sort??
61. What is Fibonacci sequence?
62. What is the 15th term of the Fibonacci sequence ?
63. What is the fibonacci sequence in nature?
64. Which fibonacci numbers can be written as half the difference or sum of two cubes ?
65. Are there infinitely many prime fibonacci numbers ?
66. What is the average case complexity for quick sort algorithm ?
67. Why quick sort uses Divide and Conquer technique ?
68. What pattern of splits gives the worst-case performance for quick sort?
69. Why Quick sort is also known as pointer sort and external sorting?
70. In which cases are the time complexities same in quick sort?
71. What is FIFO?
72. What is an ordered list?
73. How do you insert a new item in a binary search tree?
74. What are multidimensional arrays?

75. Difference between internal and external sorting?
76. When is a binary search best applied?
77. Briefly explain recursive algorithm.
78. Which sorting algorithms can be used to sort a random linked list with minimum time complexity? Support your answer.
79. Which of the following is not a limitation of binary search algorithm
80. Where the binary search algorithm cannot be applied?
81. What are the correct intermediate steps of the following data set when it is being sorted with the Selection sort? 15,20,10,18
82. In a selection sort structure, how are loops used?
83. What is the first step in a selection sort algorithm?
84. How many passes/scans will go through a list of 10 elements?
85. How many passes (or "scans") will there be through a list being sorted using a selection sort?.