Experiment No.: 1

AIM: - Write a Python program to compute different computation on matrix

TITLE: - To perform following operations on the matrix.

a) Addition of two matrices b) Subtraction of two matrices

c) Multiplication of two matrices d) Transpose of a matrix

OBJECTIVES:-

1. To learn how to read and display matrix in python.
2. To implement addition and subtraction on matrix.
3. To understand how nested for loops works in python.
4. To implement multiplication and transpose operation on matrix.

THEORY:-

Matrix is nothing but a rectangular arrangement of data or numbers. In other words, it is a rectangular array of data or numbers. The horizontal entries in a matrix are called as „rows" while the vertical entries are called as „columns". If a matrix has r number of rows and c number of columns then the order of matrix is given by r x c. Each entries in a matrix can be integer values, or floating values, or even it can be complex numbers.

Examples:

```
// 3 x 4 matrix
    1 2 3 4
M = 4 5 6 7
    6 7 8 9

// 2 x 3 matrix in Python
A = ( [ 2, 5, 7 ],
     [ 4, 7, 9 ] )
```

Following code is used to read & display a matrix:

```
# A basic code for matrix input from user

R = int(input("Enter the number of rows:"))
C = int(input("Enter the number of columns:"))
```

# Initialize matrix

```
matrix = []
print("Enter the entries rowwise:")
 # For user input
```

```
for i in range(R):          # A for loop for row entries
   a =[]
   for j in range(C):      # A for loop for column entries
       a.append(int(input()))
   matrix.append(a)

# For printing the matrix
for i in range(R):
   for j in range(C):
      print(matrix[i][j], end = " ")
   print()
```

Matrix operations:

Matrix Addition

mat1 = {{1, 2}, {3, 4}}
mat2 = {{1, 2}, {3, 4}}
mat1 + mat2 = {{2, 4}, {6, 8}}

Matrix Subtraction

mat1 = {{1, 2}, {3, 4}}
mat2 = {{1, 2}, {3, 4}}
mat1 - mat2 = {{0, 0}, {0, 0}}

Matrix Multiplication

mat1 = {{1, 2}, {3, 4}}
mat2 = {{1, 2}, {3, 4}}
mat1 * mat2 = {{7, 10}, {15, 22}}

Algorithm to perform matrix addition, matrix subtraction, matrix multiplication

Matrix addition:

1. Input the order of the matrix.
2. Input the matrix 1 elements.
3. Input the matrix 2 elements.
4. Repeat from i = 0 to m
5. Repeat from j = 0 to n
6. mat3[i][j] = mat1[i][j] + mat2[i][j]
7. Print mat3.

Matrix subtraction:

1. Input the order of the matrix.
2. Input the matrix 1 elements.
3. Input the matrix 2 elements.

4. Repeat from i = 0 to m
5. Repeat from j = 0 to n
6. mat3[i][j] = mat1[i][j] - mat2[i][j]
7. Print mat3.

Matrix multiplication:

1. Input the order of the matrix1 (m * n).
2. Input the order of matrix2 (p * q).
3. Input the matrix 1 elements.
4. Input the matrix 2 elements.
5. Repeat from i = 0 to m
6. Repeat from j = 0 to q
7. repeat from k = 0 to p
8. sum=sum+ mat1[c][k] * mat2[k][d];
9. mat3[c][d]=sum
10. Print mat3.

Matrix transpose:

1. Enter the order of matrix
2. Enter the elements of matrix row-wise using loop
3. Display the entered matrix in standard format (it is not a compulsory step)
4. Assign number of rows with number of column
5. Swap (i, j)th element with (j, i)th
6. Store the new elements as element of transposed matrix
7. Print the elements of transpose matrix in format using loop

CONCLUSION:-

____

____

____

QUESTIONS:-

1. **What is a matrix?**

____

____

____

____

____

2. **How to represent matrix?**

3. **How to read matrix in python?**

4. **Explain in detail how to access contents within matrix to perform various operations?**

5. **What is indexing and how it is done to perform various matrix operations?**

Faculty Signature & Date

Experiment No.: 2

AIM:- To implement the various types of operation on string using python programming.

TITLE:- Write Python program to compute following operation on string:

a) To display word with the longest length.
b) To determine the frequency of occurrence of particular character in the string.
c) To check whether given string is palindrome or not.
d) To display index of first appearance of the substring.
e) To count the occurrences of each word in given string.

OBJECTIVES:-

1) To learn the concept of string.
2) To learn and implement basic operation of string.

THEORY:-

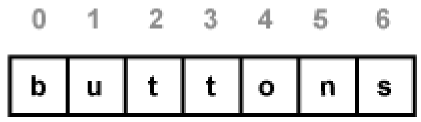
A string object is one of the sequence data types in Python. It is an immutable sequence of Unicode characters. Strings are objects of Python's built-in class 'str'. String literals are written by enclosing a sequence of characters in single quotes ('hello'), double quotes ("hello") or triple quotes ('''hello''' or """hello""").

Example :- str1=”I am the student of AVCOE”
str2=”Hello, everyone”

Various Types of operation performing on strings :-

1) **Indexing To string :-**
The beginning character of a string corresponds to index 0 and the last character corresponds to the index

Index start with “0”



In the string there are two ways to represent the index of string. Firstly index start with “0” and increasing right to left one by one . Another one is from left to right the last element have index by default “-1”.

Palindrome string:-

A palindrome is a string that is the same read forward or backward. For example, "dad" is the same in forward or reverse direction. Another example is "aibohphobia", which literally means, an irritable fear of palindromes. Simply the word read from right to left or left to right is also same.

The operation on string as follows:

Operation	Description
Concatenation	The Addition operator, "+", can be used to concatenate strings together. See String Concatenation
Formatting Data	The STRING function is used to format data into a string. The READS procedure can be used to read values from a string into IDL variables. See Using STRING to Format Data.
Case Folding	The STRLOWCASE function returns a copy of its string argument converted to lowercase. Similarly, the STRUPCASE function converts its argument to uppercase. See Case Folding.
White Space Removal	The STRCOMPRESS and STRTRIM functions can be used to eliminate unwanted white space (blanks or tabs) from their string arguments. See Whitespace.
Length	The STRLEN function returns the length of its string argument. See Finding the Length of a String.
Substrings	The STRPOS, STRPUT, and STRMID routines locate, insert, and extract substrings from their string arguments. See Substrings.
Splitting and Joining Strings	The STRSPLIT function is used to break strings apart, and the STRJOIN function can be used to and glue strings together. See Splitting and Joining Strings
Comparing Strings	The STRCMP, STRMATCH, and STREGEX functions perform string comparisons. See Comparing Strings.

Algorithm:

1) Display word with the longest length :-

```
#To display word with the longest length
str1=input("Enter any string:")
list1=str1.split()
m=0
word=0
print(list1)
```

```
for i in range(len(list1)):
    if m<len(list1[i]):
        m=len(list1[i])
        word=i
print("Word with longest length is:",len(list1[word]))
```

2) Determine the frequency of occurrence of particular character in the string :-

```
#To count of Occurrence of word in the string:
str1=input("Enter the  string")
char=input("Enter character")
counter=0
for i in range(len(str1)):
    if(char==str1[i]):
        counter+=1
print("Character",char,"is  present",counter,"times  in  string")
```

3) **Check whether given string is palindrome:**

```
string1=input("Enter the string")
string2=string1[::-1]
if(string1==string2):
      print("Wao, Given string is palindrome")
else:
     print("Sorry, Given string is not palindrome")
```

4) **Display index of first appearance of the substring :-**

```
string=input("Enter the string:")
substring=input("Enter a perticular part of the string:")
index=0
m=0
for i in range(len(string)):
    if(substring[m]==string[i]):
        m=m+1
        if(m==len(substring)):
            index=i-(len(substring)-1)
            break
    else:
            m=0
            print(substring,"is at index",index+1,"in the given string")
```

5) **Count occurrence of each word in a given string:**-

```
 str=input("Enter the given string:")
```

```
list1=str.split()
list2=set(list1)
list3=[]
list3=list(list2)
print(list1)
print(list3)
list4=[]
list5=[]
for i in range(len(list3)):
      count=0
           for j in range(len(list1)):
                 if(list3[i]==list1[j]):
                       count+=1
              list4=list3[i],count
              list5.append(list4)
 print(list5)
```

CONCLUSION:-

QUESTIONS:-

1. **What is a string?**

2. **What are various operations which can be performed on string?**

3. **How to reverse a string?**

4. **What is difference between string, list and dictionary?**

5. **What is substring?**

Faculty Signature & Date

Experiment No.: 3

AIM:- To perform operations on set

TITLE:- In second year computer engineering class, group A student‟s play cricket, Group B students play badminton and group C students play football. Write a Python program using functions to compute following: -

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.

OBJECTIVES:-

1) To know the basics of set.
2) To perform operation on array.
3) To implement set operation using array

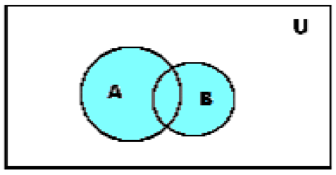
THEORY:-

Set Theory:
No restriction is placed on the nature of the objects in a set. They can be anything: points, lines, numbers, people, countries, etc. Thus the mathematical meaning of the word set is the same as the regular, nontechnical meaning of the word.

Examples of sets.

- All points in a given line segment
- Lines through a given point in space
- The set of all rational numbers
- Solutions of the equation 3x2 + 2y2 - 1 = 0
- Citizens of England
- Rivers of Mexico

Union of sets: The union of two sets A and B is the set consisting of all elements in A plus all elements in B and is denoted by AUB or A + B.

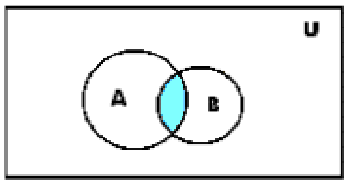
Fig. 1


A U B

Example: If A = {a, b, c, d} and B = {b, c, e, f, g} then AUB = {a, b, c, d, e, f, g}.

Intersection of sets: The intersection of two sets A and B is the set consisting of all elements that occur in both A and B (i.e. all elements common to both) and is denoted by A∩B, A · B or AB.

Fig. 2

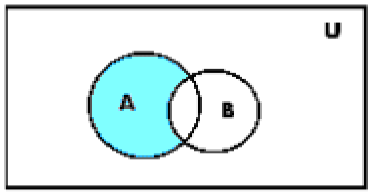


A ∩ B

Example: If A = {a, b, c, d} and B = {b, c, e, f, g} then A∩B = {b, c}.

Difference of two sets: The set consisting of all elements of a set A that do not belong to a set B is called the difference of A and B and denoted by A - B.

Fig. 3



A - B

Example: If A = {a, b, c, d} and B = {b, c, e, f, g} then A - B = {a, d}.

Universal set U: Often a discussion involves subsets of some particular set called the universe of discourse (or briefly universe), universal set or space. The elements of a space are often called the points of the space. We denote the universal set by U.
Example: The set of all even integers could be considered a subset of a universal set consisting of all the integers. Or they could be considered a subset of a universal set consisting of all the rational numbers. Or of all the real numbers.

ALGORITHM:-

1) Start
2) Declare variable and string
3) Accept input from user
4) Define all functions and perform all operations
5) Display the output
6) Stop

CONCLUSION:-

QUESTIONS:-

1. **What a set and how to represent it?**

2. **What is Universal set?**

3. **Explain various ways of representing set?**

4. **Explain various operations which can be performed on set?**

5. **Explain 2D Character Array?**

Faculty Signature & Date

Experiment No.: 4

AIM:- Implement python program for searching data using various searching algorithm.

TITLE:- 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

OBJECTIVES:-

1. To learn basic concepts of different searching techniques.
2. To learn Linear search, Sentinel search, Binary search and Fibonacci search concept.
3. To get result of operations.

THEORY:-

Search techniques

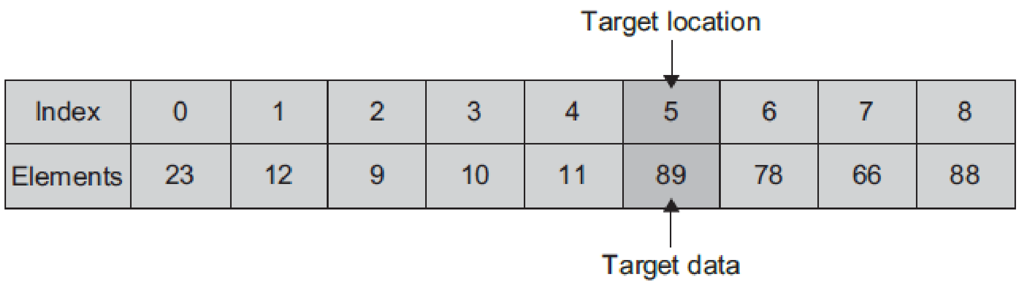
Depending on the way data is scanned for searching a particular record, the search techniques are categorized as follows:

1. Sequential (Linear) search
2. Binary search
3. Fibonacci search
4. Index sequential search
5. Hashed search

The performance of a searching algorithm can be computed by counting the number of comparisons to find a given value. We shall study these algorithms with respect to arrays. For sequential search, the same concept applies for searching data in linked lists as well as files.

1. Linear search-

Linear search is rarely used practically because other algorithms such as the binary search and hash tables allow significantly faster searching, compared to linear search.
The time complexity of linear search is O(n).
Figure shows a sample linear unordered data and traces the search for the target data of 89.



Index	0	1	2	3	4	5	6	7	8
Elements	23	12	9	10	11	89	78	66	88

ALGORITHM:

1. Initialize the integer variables.
2. Get the target number from user.
3. Get the list of numbers to be searched.
4. If the counter is equal to target the print the location.
5. If it is not equal to then print that the location is not found end.
6. Exit.

Average complexity is the sum of comparisons for each position of the target data divided by n. Hence, average number of comparisons is (1 + 2 + 3 + … + n)/n= (n + 1)/2
The worst-case complexity = n
The best-case complexity = 1 if element to be found is at first position.

Variations of Sequential Search is
2. Sentinel Search-
To reduce overhead of checking the list‟s length, the *value* to be searched can be appended to the list at the end (or beginning in case of Reverse Search) as a "sentinel value". A sentinel value is one whose presence guarantees the termination of a loop that processes structured (or sequential) data. Thus on encountering a matching value, its index is returned. The calling function can then determine if the returned index is a valid one or not. Though the optimization resulted in isn‟t much, it reduces the overhead of checking if the index is within limit in each step.
ALGORITHM:

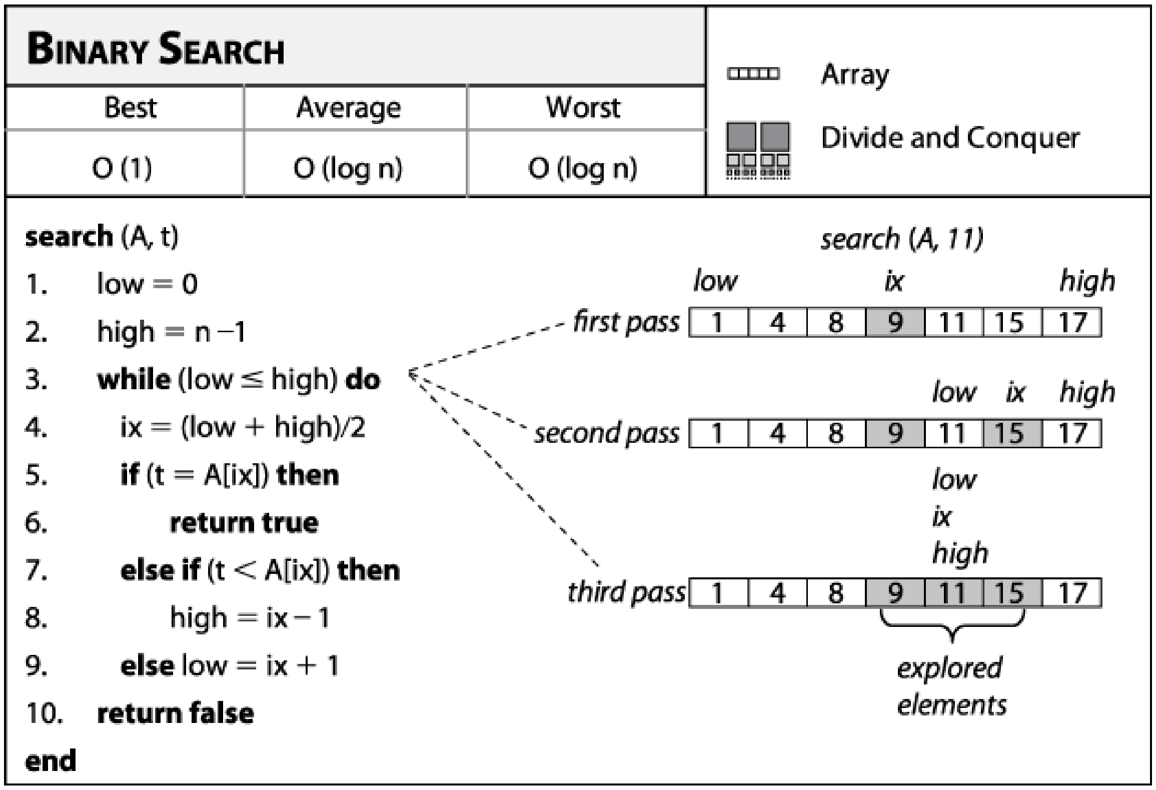
```
1. Set i = 0
2. list[n] = target {add sentinel}
3. Compare key[i] and target
     Set location = i and goto step 6
4. Move to next data element
5. goto step 3
6. if(location < n) then return location as position of target
7. else report as „Target not found‟ and return −1
8. stop
```

3. Binary search-
Binary search is a very fast and efficient searching technique. It requires the list to be in sorted order. In this method, to search an element there is need to compare it with the present element at the center of the list. If it matches, then the search is successful otherwise the list is divided into two halves: one from the 0thelement till middle element which is the center element (first half) another from the center element to the last element (which is the 2nd half) where all values are greater than the center element.

The searching mechanism proceeds from either of the two halves depending upon whether the target element is greater or smaller than the central element. If the element is smaller than the central element, then searching is done in the first half, otherwise searching is done in the second half.

ALGORITHM:

1. Initialize the integer variables.
2. Get the values from the list.
3. Sorted List of numbers is got as input.
4. Get the target number from the user.
5. Initialize first value as zero and last as n-1;
6. The mid value is found.
7. If the target is greater than mid then first is mid + 1 if not last is mid -1.
8. First is last +1.
9. The target is found, if its equal to A(mid) otherwise target is not found.



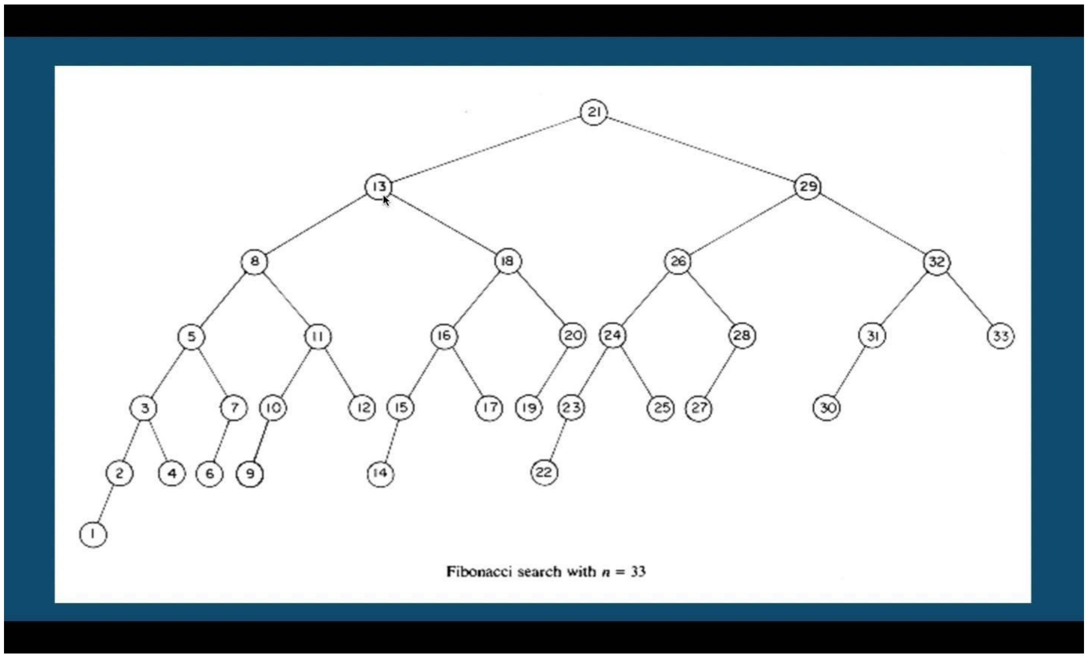
BINARY SEARCH

Best	Average	Worst
O (1)	O (log n)	O (log n)

```
search (A, t)
1.   low = 0
2.   high = n −1
3.   while (low ≤ high) do
4.     ix = (low + high)/2
5.     if (t = A[ix]) then
6.         return true
7.     else if (t < A[ix]) then
8.         high = ix − 1
9.     else low = ix + 1
10.  return false
end
```

4. Fibonacci search-

The Fibonacci search also searches from a sorted array. The Fibonacci search technique uses a divide-and-conquer mechanism that helps decrease the possible locations by using Fibonacci numbers.



Fibonacci search with $n = 33$

The Fibonacci search works like the binary search but with a few modifications. In binary search, we have low, high, and mid positions for the sub-list. Here, we have mid = n - $Fk{-}1$ + 1, $F1$ = $Fk{-}2$, and $F2$ = $Fk{-}3$. The target to be searched is compared with A[mid], mid is computed as follows:

- ***Case 1*** if equal the search terminates;
- ***Case 2*** if the target is greater and $F1$ is 1, then the search terminates with an unsuccessful search; else the search continues at the right of the list with new values of low, high, and mid as mid = mid + $F2$, $F1$ = $Fk{-}4$, and $F2$ = $Fk{-}5$
- ***Case 3*** if the target is smaller and $F2$ is 0, then the search terminates with an unsuccessful search; else the search continues at the left of the list with new values of low, high, and mid as mid = mid - $F2$, $F1$ = $Fk{-}3$ and $F2$ = $Fk{-}4$

The search continues by either searching at the left of mid or at the right of mid in the list. Fibonacci search is more efficient than binary search for large- sized lists. However, it is inefficient in case of small lists. The number of comparisons is of the order of n, and the time complexity is O(log(n)).

CONCLUSION:-

QUESTIONS:-

1. What is need of searching technique?

2. What is difference between linear search and sentinel search?

3. What is difference between binary search and Fibonacci search?

4. What is divide and conquer strategy?

5. Why best case complexity of searching algorithm is 1, explain with an example?

Faculty Signature & Date

Experiment No.: 5

AIM:- To implement various sorting algorithm .

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

OBJECTIVES:-
1. To learn basic concepts of Sorting.
2. To learn Selection sort and Bubble sort concept.
3. To get result of operations.

THEORY:-
Sorting
One of the fundamental problems in computer science is ordering a list of items. There are plenty of solutions to this problem, commonly known as *sorting algorithms*. Some sorting algorithms are simple and iterative, such as the bubble sort. Others such as the quick sort are extremely complicated but produce lightning-fast results. *Sorting* is the operation of arranging the records of a table according to the key value of each record, or it can be defined as the process of converting an unordered set of elements to an ordered set.

Types of Sorting
Sorting algorithms are divided into two categories: internal and external sorts. If all the records to be sorted are kept internally in the main memory, they can be sorted using an internal sort. However, if there are a large number of records to be sorted, they must be kept in external files on auxiliary storage. They have to be sorted using external sort.
The various internal sorting techniques are the following:

1. Bubble sort
2. Insertion sort
3. Selection sort
4. Quick sort
5. Heap sort
6. Shell sort
7. Bucket sort
8. Radix sort
9. File sort
10. Merge sort

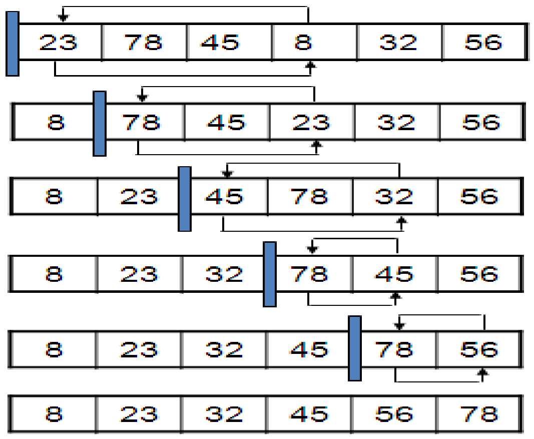
1. Selection sort-

In selection sort the list is divided into two sub-lists sorted and unsorted. These two lists are divided by imaginary wall. We find a smallest element from unsorted sub-list and swap it to the beginning. And the wall moves one element ahead, as the sorted list is increases and unsorted list is decreases.

Assume that we have a list on n elements. By applying selection sort, the first element is compared with all remaining (n-1) elements. The smallest element is placed at the first location. Again, the second element is compared with remaining (n-1) elements. At the time of comparison, the smaller element is swapped with larger element. Similarly, entire array is checked for smallest element and then swapping is done accordingly. Here we need n-1 passes or iterations to completely rearrange the data.

Example

Ex:- A list of unsorted elements are: 23 78 45 8 32 56



A list of sorted elements now : 8 23 32 45 56 78

Algorithm for Selection sort: Selection_Sort (A [ ] , N)

```
1. Repeat For K = 0 to N – 2 Begin
2. Set POS = K
3. Repeat for J = K + 1 to N – 1 Begin
       If A[ J ] < A [ POS ]
       Set POS = J
       End For
4. Swap A [ K ] with A [ POS ]
       End For
5. Exit
```

The total number of comparisons is as follows:

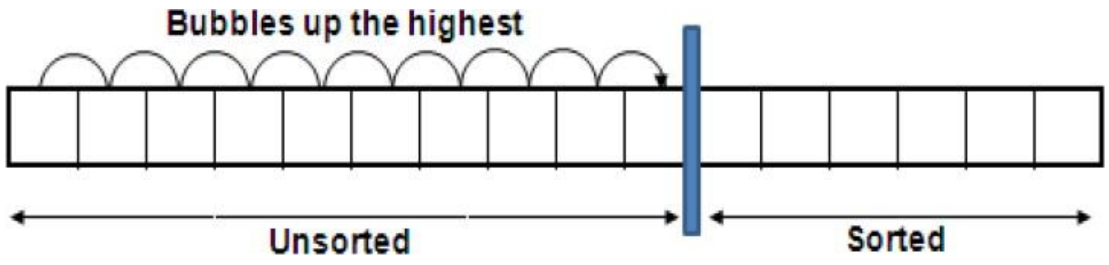
(n - 1) + (n - 2) + ... + 1 = n(n -1)/2

Therefore, the number of comparisons for the selection sort is proportional to n2, which means that it is $O(n^2)$. The different cases are as follows:

Average case: $O(n^2)$ Best case: $O(n^2)$ Worst case: $O(n^2)$

2. Bubble sort-

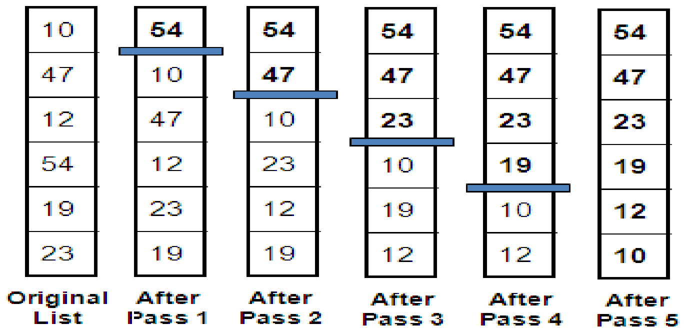
In bubble sort method the list is divided into two sub-lists sorted and unsorted. The smallest element is bubbled from unsorted sub-list. After moving the smallest element the imaginary wall moves one element ahead. The bubble sort was originally written to bubble up the highest element in the list. But there is no difference whether highest / lowest element is bubbled. This method is easy to understand but time consuming. In this type, two successive elements are compared and swapping is done. Thus, step-by-step entire array elements are checked. Given a list of „n" elements the bubble sort requires up to n-1 passes to sort the data.



Example:

Ex:- A list of unsorted elements are: 10 47 12 54 19 23

(Bubble up for highest value shown here)



A list of sorted elements now : 54 47 23 19 12 10

Algorithm for Bubble Sort: Bubble_Sort (A [ ] , N)

```
1. Repeat For P = 1 to N – 1 Begin
2. Repeat For J = 1 to N – P Begin
3. If ( A [ J ] < A [ J – 1 ] )
        Swap ( A [ J ] , A [ J – 1 ] ) End For
        End For
```

The time complexity of bubble sort is same as selection sort that is:

1. Average case complexity = O(n^2)
2. Best case complexity = O(n^2)
3. Worst case complexity = O(n^2)

CONCLUSION:-

QUESTIONS:-

1. What is need of sorting technique?

2. Explain different types of sorting techniques?

3. What is difference between selection sort and bubble sort?

4. Which strategy is used in selection and bubble sort algorithm?

5. Analyze the complexity of selection sort with an example?

Faculty Signature & Date

Experiment No.: 6

AIM:- To implement quick sort algorithm .

TITLE:- 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.

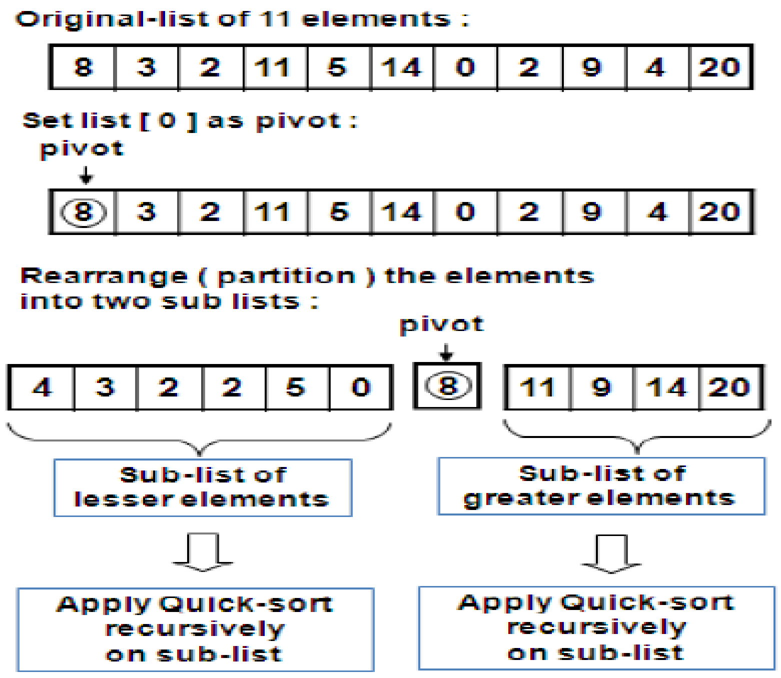
OBJECTIVES:-

1. To learn basic concepts of Sorting.
2. To learn Quick sort concept.
3. To get result of operations.

THEORY:-

Quick sort is based on divide and conquer strategy and uses partition and is known as partition exchange sorting. The basic concept of quick sort process is pick one element from an array and rearranges the remaining elements around it. This element divides the main list into two sub lists. This chosen element is called pivot. Once pivot is chosen, then it shifts all the elements less than pivot to left of it and all the elements greater than pivot are shifted to the right side. This procedure of choosing pivot and partition the list is applied recursively until sub-lists consisting of only one element.

Ex:- A list of unsorted elements are: 8 3 2 11 5 14 0 2 9 4 20


Original-list of 11 elements :
8 3 2 11 5 14 0 2 9 4 20
Set list [0] as pivot :
pivot
8 3 2 11 5 14 0 2 9 4 20
Rearrange (partition) the elements into two sub lists :
pivot
4 3 2 2 5 0
8
11 9 14 20
Sub-list of lesser elements
Sub-list of greater elements
Apply Quick-sort recursively on sub-list
Apply Quick-sort recursively on sub-list

Algorithm for quick sort:

```
Repeat process till low < high
1. Select pivot = A[Low], pivot location P = low
2. i = low and j = high;
3. Increment index i till A[i] >= pivot
4. Decrement index j till A[i] <= pivot
5. Swap A[i] with A[j]
6. Repeat steps 4, 5, 6 till i < j
7. if i < j
        Swap a[P] with a[j]
8. call Quicksort(low, j – 1)
9. call Quicksort(j + 1, high)
10. Stop
```

Time Complexity of Quick sort:

Best case : O (n log n)

Average case : O (n log n)

Worst case : O (n^2) when the given list is already sorted.

Advantages of quick sort:

1. This is faster sorting method among all.
2. Its efficiency is also relatively good.
3. It requires relatively small amount of memory

Disadvantages of quick sort:

It is complex method of sorting so, it is little hard to implement than other sorting methods.

CONCLUSION:-

____

____

____

QUESTIONS:-

1. What is pivot element, how it can be selected?

2. Which strategy is used in quick sort explain with an example?

3. Which are other algorithms which also uses the strategy of quick sort?

4. What is the time complexity of quick sort algorithm?

5. Analyze the worst case complexity of quick sort with an example?

Faculty Signature & Date

Experiment No.: 7

AIM: To implement linked list.

TITLE: 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

OBJECTIVES:-

1. To learn basic concepts of linked list.
2. To learn different types of linked list.
3. To get result of operations.

THEORY:-

LINKED LIST:

It is a dynamic data structure and linear collection of data items. Direction is associated with it i.e. Logical link exits between items. Pointers acts as the logical link. Consists of nodes that has two fields.

Data field: info of the element.
Next field: next pointer containing the address of next node.

TYPES OF LINKED LIST:

i. Singly or chain: Single link between items.
ii. Doubly: There are two links, forward and backward link.
iii. Circular: The last node is again linked to the first node. These can be singly circular & doubly circular list.

ADVANTAGES:

1. Linked list use dynamic memory allocation thus allocating memory when program is initialised. List can grow and shrink as needed. Arrays follow static memory allocation. Hence there is wastage of space when fewer elements are declared. There is possibility of overflow too because of fixed amount of storage.
2. Nodes are stored in contiguously thus insertion and deletion operations are easily implemented.
3. Linear data structures like stack and queues are easily implemented using linked list.

DISADVANTAGES:

1. Wastage of memory as pointers requires extra storage.
2. Nodes are in contiguously stored thereby increasing time required to access individual elements. To access nth item arrays need a single operation while linked list need to pass through (n-1) items.
3. Nodes must be read in order from beginning as they have inherent sequential access.
4. Reverse traversing is difficult especially in singly linked list. Memory is wasted for allocating space for back pointers in doubly linked list.

DEFINING LINKED LIST:

```
struct node {
int info;
struct node *next; \\next field. An eg of self referencetial structure.(#)
} *ptr;
```

(#)Self Referencetial structure: A structure that is referencing to another structure of same type. Here "next" is pointing to structure of type "node". ptr is a pointer of type node. To access info n next the syntax is: ptr->info; ptr->next;

OPERATIONS ON SINGLY LINKED LIST:

i. Searching
ii. Insertion
iii. Deletion
iv. Traversal
v. Reversal
vi. Splitting
vii. Concatenation

Some operations:

a: Insertion :

```
void push(struct node** headref, int data) -------- (1)
{
struct node* newnode = malloc(sizeof(struct node));
newnode->data= data;
newnode->next= *headref;
*headref = newnode;}
```

(1) : headref is a pointer to a pointer of type struct node. Such passing of pointer to pointer is called Reference pointer. Such declarations are similar to declarations of call by reference. When pointers are passed to functions ,the function works with the original copy of the

variable.

i. Insertion at head:

```
struct node* head=NULL;
for(int i=1; i<6;i++)
{ push(&head,i); \\ push is called here. Data pushed is 1."&" used coz references are
passed in function arguments.
}
return(head);
}
# :o\p: 5 4 3 2 1
# :Items appear in reverse order.(demerit)
```

ii. Insertion at tail:

```
struct node* add1()
{ struct node* head=NULL;
struct node* tail;
push(&head,1);
tail = head;
for(int i=2 ;i<6; i++)
{ push(&(tail->next),i);
tail= tail->next;
}
return(head);
}
# : o\p: 1 2 3 4 5
```

b. Traversal:

```
int count( struct node* p)
{int count =0;
struct node* q;
current = q;
while(q->next != NULL)
{ q=q->next;
count++; }
return(count);
}
```

c. Searching:

```
struct node* search( struct node* list, int x)
{ struct node* p;
for(p= list; p ->next != NULL; p= p->next )
{
if(p->data==x)
return(p);
return(NULL);
}}
```

ALGORITHM:-

1. Start
2. Declare linked list
3. Accept student liking vanilla and butter scotch ice-cream.
4. Display Menu.
5. Accept choice from user if no then go to step
6. Call one by one class member function to perform operations.
7. Stop.

CONCLUSION:-

QUESTIONS:-

1. **What is Linked List?**

2. **What are types of link list?**

3. **How to define a linked list using class?**

4. **What are the operations that can be performed on linked list?**

5. **What are the advantages and disadvantages of linked list?**

Faculty Signature & Date

Experiment No.: 8

AIM:- To implement doubly linked list.

TITLE: - Write C++ program for storing binary number using doubly linked lists. Write functions-

a) To compute 1„s and 2„s complement

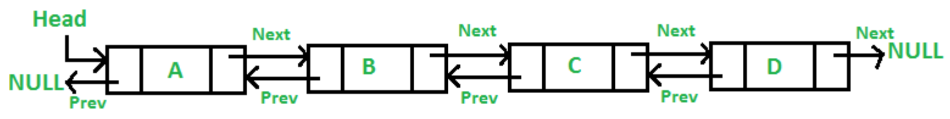
b) Add two binary numbers

OBJECTIVES:-

1. To learn basic concepts of Linked List.
2. To learn Doubly Linked List.
3. To get result of operations.

THEORY:-

A Doubly Linked List (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.



Representing DLL in CPP:

```
/* Node of a doubly linked list */
        struct node{
           int data;
           struct Node* next; // Pointer to next node in DLL
           struct Node* prev; // Pointer to previous node in DLL
        }
```

Advantages of DLL over Singly Linked List :

1) A DLL an be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to node to be deleted is given

3) We can quickly insert a new node before a given node.

In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages of DLL over Singly Linked List :

1) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though.
2) All operations require an extra pointer previous to be maintained. In insertion, we need to modify previous pointers together with next pointers.

ALGORITHM:-

1. Start
2. Declare variables
3. Accept the number from the user.
4. Display Menu of 1‟s complement, 2‟s complement and Binary addition.
5. Accept choice from user if no then go to step 7
6. Call one by one class member function to perform operations.
7. Stop.

CONCLUSION:-

QUESTIONS:-

1. **What is doubly Linked List?**

2. **Why to use doubly link list?**

3. **How to define a doubly linked list using class?**

4. **What are the operations that can be even performed on doubly linked list?**

5. **What are the advantages and disadvantages of doubly linked list?**

Signature & Date

Experiment No.: 9

AIM:- Implement C++ program to determine whether string is palindrome or not using stack.

TITLE: - 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 dan is in a 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

OBJECTIVES:-

Real time applications of stack.

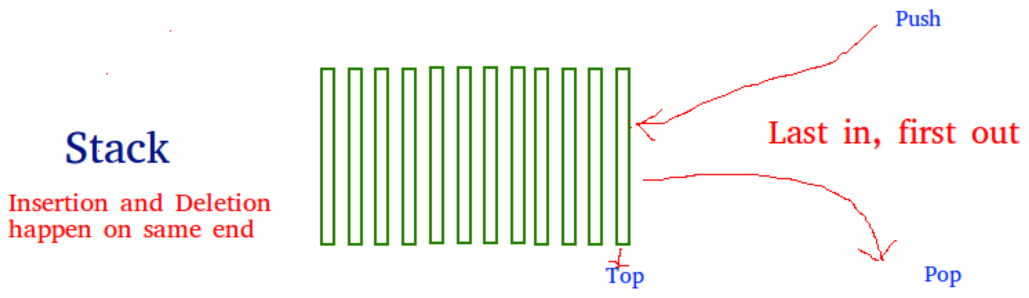
THEORY:-

Stack Data Structure

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO (First In Last Out).

Mainly the following three basic operations are performed in the stack:

- Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- Peek or Top: Returns top element of stack.
- isEmpty: Returns true if stack is empty, else false.



Implementation:

There are two ways to implement a stack:

- Using array
- Using linked list

Palindrome: 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 dan is in a 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.

ALGORITHM:-

1. Start
2. Take a string as input and store it in the array s[].
3. Load each character of the array s[] into the array stack[].
4. Use variables front and top to represent the last and top element of the array stack[].
5. Using for loop compare the top and last element of the array stack[]. If they are equal, then delete the top element, increment the variable front by 1 and compare again.
6. If they are not equal, then print the output as "It is not a palindrome".
7. Compare the elements in the steps 4-5 upto the middle element of the array stack[].
8. If every characters of the array is equal, then it is a palindrome.
9. Stop.

CONCLUSION:-

QUESTIONS:-

1. What is stack?

2. Explain push(), pop(), isEmpty(), isFull() operations?

3. How to convert all characters of string into lower case?

4. How to remove punctuation marks from a string?

5. How to check a string is palindrome or not?

Faculty Signature & Date

Experiment No.: 10

AIM:- To implement stack operations to check parenthesized.

TITLE:- In any language program mostly syntax error occurs due to unbalancing delimiter such as (),{},[]. Write C++ program using stack to check whether given expression is well parenthesized or not.

OBJECTIVES:-

1. To learn basic concepts of Stack.
2. Various operations that can be performed on stack.
3. To get result of operations.

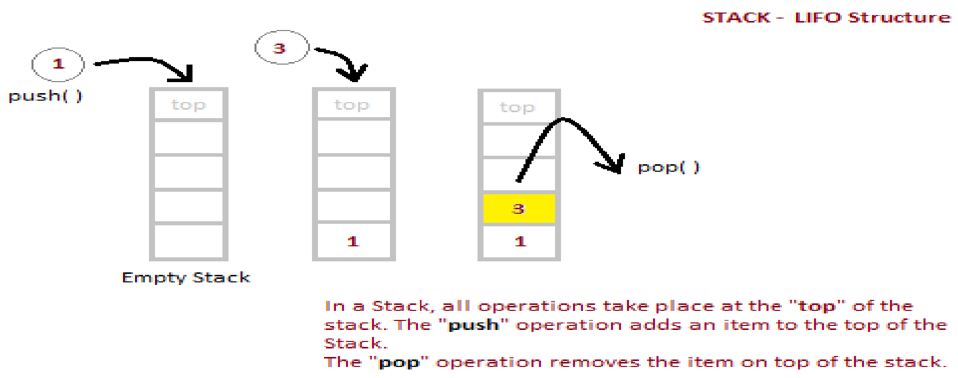
THEORY:-

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).
Mainly the following three basic operations are performed in the stack:

1) **Push**: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
2) **Pop**: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
3) **Peek or Top**: Returns top element of stack.
4) **isEmpty**: Returns true if stack is empty, else false.

Implementation of Stack Data Structure:-

Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size. Here we will implement Stack using array.


STACK - LIFO Structure
1
push( )
3
top
top
top
1
3
1
pop( )
Empty Stack
In a Stack, all operations take place at the "top" of the stack. The "push" operation adds an item to the top of the Stack.
The "pop" operation removes the item on top of the stack.

Implementation of Stack:-

Algorithm for PUSH operation

1. Check if the stack is full or not.
2. If the stack is full, then print error of overflow and exit the program.
3. If the stack is not full, then increment the top and add the element.

Algorithm for POP operation

1. Check if the stack is empty or not.
2. If the stack is empty, then print error of underflow and exit the program.
3. If the stack is not empty, then print the element at the top and decrement the top.

Applications of stack:

1. Balancing of symbols
2. Infix to Postfix /Prefix conversion
3. Redo-undo features at many places like editors, photoshop.
4. Forward and backward feature in web browsers
5. Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
6. Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver
7. In Graph Algorithms like Topological Sorting and Strongly Connected Components

ALGORITHM:-

1. Start
2. Declare variables
3. Accept the string from the user.
4. Implement stack to check whether given expression is well parenthesized or not..
5. Call function to check parenthesized (i.e {}, (), []).
6. Display result.
7. Stop.

CONCLUSION:-

____

____

____

QUESTIONS:-

1. **What is Stack and various operations which can be performed on it?**

2. **How to define stack using sequential and linked organization?**

3. **How to define a stack using class?**

4. **What are the advantages and disadvantages of stack?**

5. **What are various applications of stack?**

Faculty Signature & Date

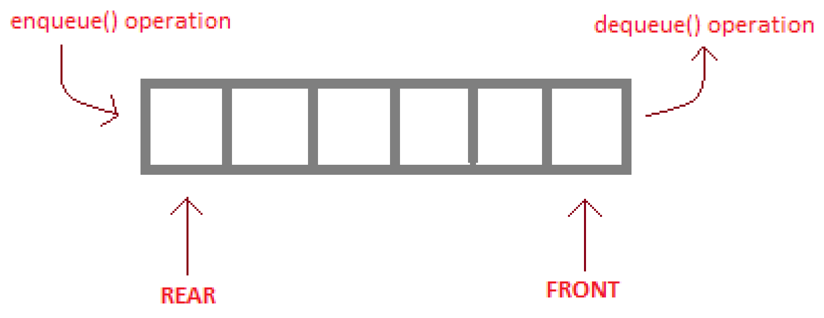
Experiment No.: 11

AIM:-Implement C++ program for queue.

TITLE:- 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.

OBJECTIVES:-
To learn basic concepts of queue.

THEORY:-
Queue Data Structure
Queue is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the REAR(also called tail), and the removal of existing element takes place from the other end called as FRONT(also called head).
This makes queue as FIFO(First in First Out) data structure, which means that element inserted first will be removed first.
This is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Same is the case with Queue data structure. Data inserted first, will leave the queue first.
The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.



enqueue( ) is the operation for adding an element into Queue.
dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

Implementation:

There are two ways to implement a queue:

- Using array
- Using linked list

Algorithm for ENQUEUE operation

1. Check if the queue is full or not.
2. If the queue is full, then prints overflow error and exit the program.
3. If the queue is not full, then increment the tail and add the element.

Algorithm for DEQUEUE operation

1. Check if the queue is empty or not.
2. If the queue is empty, then print underflow error and exit the program.

If the queue is not empty, then print the element at the head and increment the head.

CONCLUSION:-

QUESTIONS:-

1. What is queue?

2. What is front and rear?

3. Explain various operations which can be performed on queue?

4. How to represent a queue?

5. What are real time applications of queue?

Faculty Signature & Date

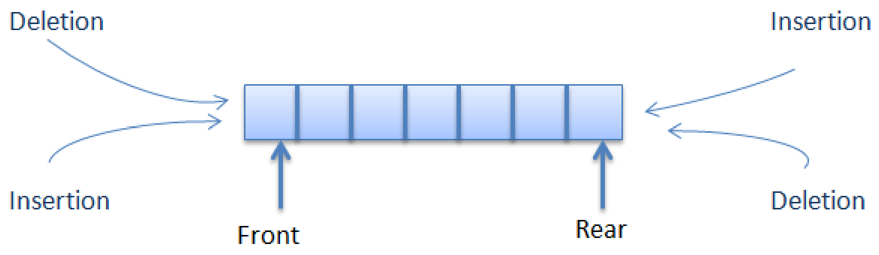
Experiment No.: 12

AIM:-Implement C++ program for deque.

TITLE:- 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.

OBJECTIVES:-
To learn basic concepts of deque.

THEORY:-
Deque
A **deque**, also known as a double-ended queue, is an ordered collection of items similar to the queue. It has two ends, a front and a rear, and the items remain positioned in the collection. What makes a deque different is the unrestrictive nature of adding and removing items. New items can be added at either the front or the rear. Likewise, existing items can be removed from either end. In a sense, this hybrid linear structure provides all the capabilities of stacks and queues in a single data structure.



Operations on Deque:

Mainly the following four basic operations are performed on queue:
insertFront(): Adds an item at the front of Deque.
insertLast(): Adds an item at the rear of Deque.
deleteFront(): Deletes an item from front of Deque.
deleteLast(): Deletes an item from rear of Deque.

In addition to above operations, following operations are also supported
getFront(): Gets the front item from queue.
getRear(): Gets the last item from queue.

isEmpty(): Checks whether Deque is empty or not.
isFull(): Checks whether Deque is full or not.

Applications of Deque:
Since Deque supports both stack and queue operations, it can be used as both. The Deque data structure supports clockwise and anticlockwise rotations in O(1) time which can be useful in certain applications.
Also, the problems where elements need to be removed and or added both ends can be efficiently solved using Deque.

Implementation:
There are two ways to implement a deque:

- Using array
- Using linked list

Algorithm for ENQUEUE operation

1. Check if the queue is full or not.
2. If the queue is full, then prints overflow error and exit the program.
3. If the queue is not full, then increment the tail and add the element.

Algorithm for DEQUEUE operation

1. Check if the queue is empty or not.
2. If the queue is empty, then print underflow error and exit the program.
3. If the queue is not empty, then print the element at the head and increment the head.

CONCLUSION:-

QUESTIONS:-
1. What is deque?

2. Why deque data structure is similar to both stack and queue?

3. Explain various operations which can be performed on deque?

4. How to represent a deque using link list?

5. What are various applications of deque?

Faculty Signature & Date

Experiment No.: 13

AIM:-Implement C++ program for circular queue.

TITLE:- 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.

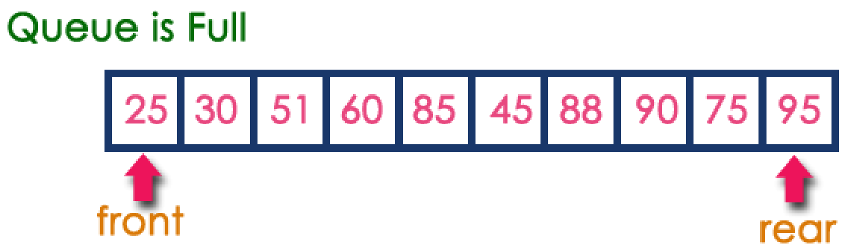
OBJECTIVES:-

a. To study Circular Queue as data structure.
b. To understand implementation of Circular Queue and perform various operations on it.

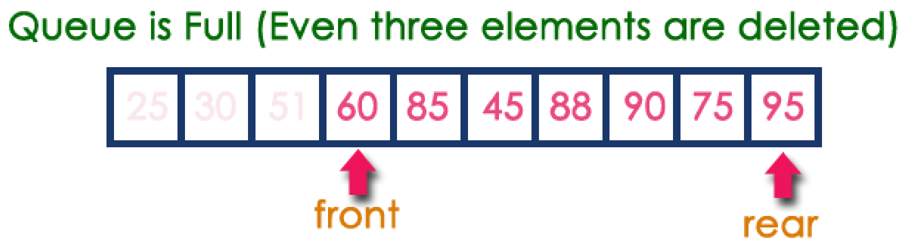
THEORY:

In a normal Queue Data Structure, we can insert elements until queue becomes full. But once if queue becomes full, we can‟t insert the next element until all the elements are deleted from the queue. For example consider the queue below...

After inserting all the elements into the queue...



Now consider the following situation after deleting three elements from the queue...

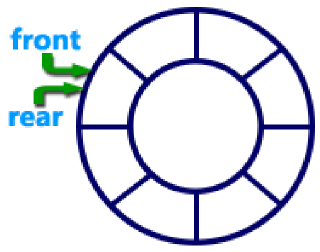


This situation also says that Queue is Full and we can‟t insert the new element because, „rear‟ is still at last position. In above situation, even though we have empty positions in the queue we can‟t make use of them to insert new element. This is the major problem in normal queue data structure. To overcome this problem we use circular queue data structure.

Circular 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.

Graphical representation of a circular queue is as follows...



Implementation of Circular Queue:

To implement a circular queue data structure using 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: Increment rear value by one (rear++), 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 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 increment the front value by one (front ++). Then check whether front == SIZE, if it is TRUE, then set front = 0. Then check whether both front – 1 and rear are equal (front -1 == rear), if it TRUE, then set both front and rear to „-1‟ (front = rear = -1).

Display () – Displays the elements of a Circular Queue:
We can use the following steps to display the elements of a circular queue…
Step 1: Check whether queue is EMPTY. (front == -1)
Step 2: If it is EMPTY, then display “Queue is EMPTY!!!” and terminate the function.
Step 3: If it is NOT EMPTY, then define an integer variable „i‟ and set „i = front‟.
Step 4: Check whether „front <= rear‟, if it is TRUE, then display „queue[i]‟ value and increment „i‟ value by one (i++). Repeat the same until „i <= rear‟ becomes FALSE.
Step 5: If „front <= rear‟ is FALSE, then display „queue[i]‟ value and increment „i‟ value by one (i++). Repeat the same until‟i <= SIZE – 1′ becomes FALSE.
Step 6: Set i to 0.
Step 7: Again display „cQueue[i]‟ value and increment i value by one (i++). Repeat the same until „i <= rear‟ becomes FALSE.

CONCLUSION:-

____

____

____

QUESTIONS:-
1. What is circular queue?

2. Why circular queue is preferred?

3. Explain various operations which can be performed on circular queue?

4. How to represent a circular queue using link list?

5. What are various applications of circular queue?

Faculty Signature & Date