# Experiment No 1

**AIM: Implementation of Set Theory Problem (Cricket, Badminton and Football) OBJECTIVES:**

* To understand the concept of lists in python.
* To explore the concept of set theory.

PROBLEM STATEMENT:

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: -

1. List of students who play both cricket and badminton
2. List of students who play either cricket or badminton but not both
3. Number of students who play neither cricket nor badminton
4. Number of students who play cricket and football but not badminton.

(Note-While realizing the group, duplicate entries should be avoided, Do not use SET built-injunctions

OUTCOMES:

* Use algorithms on various linear data structure using sequential organization to solve real life problems.

SOFTWARE & HARDWARE REQUIREMENTS:

Software’s : Open Source Python, Programming tool like Google Colab, Jupyter Notebook, Pycharm, Spyder, G++/GCC

Hardware: Pentium Dual Core (3.00GHz)

4 GB RAM

THEORY:

**Set theory** is the branch of mathematical logic that studies sets ,which informally are collections of objects .Although any type of object can be collected into a set, set theory is applied most often to objects that are relevant to mathematics. The language of set theory can be used in the definitions of nearly all mathematical objects.

Definitions:

* 1. **Equal sets:** We define A=B if A and B have the same elements.
  2. **Subset :**We say that A is a subset of B and we write A⊂B or B⊃A if every element of A is also an element of B.(We also say that A is included in B or B includes A or B is a superset of A.)
  3. **Proper subset:** We say that A is a proper subset of B and we write A⊂B strictly if A⊂B and A≠B. (There exists at least one element b∈ B such that b∉A.)
  4. **The empty set:** The set which has no element is called the empty set and is denoted by∅.(That is∅={ x∈A :x∉A },where A is any set.)
  5. **Power set of a set:** Let X be any set. The set of all subsets of X is called the power set of X and is denoted by P(X).(That is we define P(X):={A:A⊂X})

Remarks:

Let A and B be any sets. Then the following propositions can be proved easily:

1. A=B if and only if A⊂B and B⊂A,
2. A⊂A and ∅⊂A,
3. A∈P(A)and∅∈P(A),
4. P(∅) ={∅}.(P(∅)is not empty ,it has exactly one element ,the ∅.)

OPERATIONS BETWEEN SETS

Let H be a set including all sets A, B, C,… which occur in the following .Let us call H the basic set.

**Union of sets:** (denoted by ∪, called” union” or” cup”)

1. The union of sets A and B is defined by A∪B :={x∈H: x∈A or x∈B}.
2. The union of a set A of sets is defined by SA:={x∈H:∃A∈A x∈A}.(x belongs to atleast one element of A)

**Intersection of sets:** (denoted by ∩, called “intersection” or “cap”)

1. The intersection of sets A and B is defined by A∩B: ={x∈H :x∈A and x∈B }.
2. The intersection of a set A6=∅is defined by TA: ={x∈H:∀A∈Ax∈A}.(x belongs to all elements of A)

**Definition** (Disjoint sets.):

A and B are called disjoint sets if A ∩ B = ∅ (they have no elements in common).

**Difference of sets**: (denoted by \ )

1. The difference of sets A and B is defined by A\B: ={x∈H: x∈A and x/∈B}.(We also say that A\B is the complement of B with respect to A.)
2. H\B is called the complement of B and is denoted by Bc, that is Bc:={x∈H: x/∈

B}.

**Ordered pairs:** Let x and y be any objects(e.g. any elements of the basic set H).The ordered pair (x,y) is defined by(x,y):={{x},{x,y}}.We call x and y the first and the second components of the ordered pair(x,y), respectively. In case x=y we have(x,x)={{x}

} .

**Cartesian product of sets :**Let A and B be sets. The Cartesian product of A and B is defined by A×B:={(a, b):a∈A and b∈B},i.e. the Cartesian product A×B is the set of all ordered pairs(a,b ) with a∈A, b∈B.

Examples:

1. A ∪ B = { 1, 2, 3}



2. A ∩ B = { 2 }



3. Ac or ~A= {3, 4}



4. A – B= {1 }



5. ~(A U B )= {4}



6. ~(A ^ B)or~( }= {1, 3,4}



**CONCLUSION:** Thus we have implemented python program for set operations

# Experiment No 2

**Title:** Write a Python program to compute different operations on String.

**Objectives:**

**•** To understand the use standard library functions for string operations

• To accepts string/statements from user

• To perform the string operations.

**Problem Statement: -** Write a Python program to compute following operations on String:

a) To display word with the longest length

b) To determines 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 a given string.

**Outcome:**

**•** Display string/statements

• Find and display longest length of word, palindrome of string, occurrences of character, find 1stindex position of substring.

### How to create a string in Python

* Strings can be created by enclosing characters inside **single quotes** or **double-quotes**.
* Triple quotes can also be used in Python, but are generally used to represent multi-line strings and docstrings.
* *# string with single quotes*
* my\_string = 'Welcome'
* print(my\_string)
* *# string with double quotes*
* my\_string = "Welcome I’m in Strings"
* print(my\_string)
* *# string with triple quotes*
* my\_string = '''Welcome'''
* print(my\_string)

### Indexing in Strings

* We can access individual characters using indexing or a range of characters using slicing.
* Index will always start from 0.
* Trying to access a character out of index range will raise an IndexError.
* The index must be an integer.

### Accessing Values in String

To access each value or sub-string, use the square brackets to slice along the index or indices to obtain your sub-string.

*#Accessing string characters in Python*

str1 = 'Computer'

print('str1 = ', str1)

*#string are immutable*

*# str1[0] ='c'*

*#first character*

print('str1[0] = ', str1[0])

*#last character*

print('str1[-1] = ', str1[-1])

*#index Error*

*#print('str1[-1] =', str1[9])*

*#slicing 2nd to 5th character*

print('str1[3:5] = ', str1[3:5])

*#slicing can be done by slice function*

x=slice(3,5)

print('str1[3,5]= ', str1[x])

*#slicing 6th to 2nd last character*

print('str1[5:-2] = ', str1[5:-2])

* Joining two or more strings into a single string is called concatenation.
* The + operator will be used to concatenate in Python.
* The \* operator can be used to repeat the string for a given number of times.

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""").

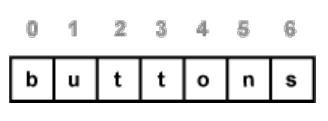
Example :- str1==I am the student of AVCOE=

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=.

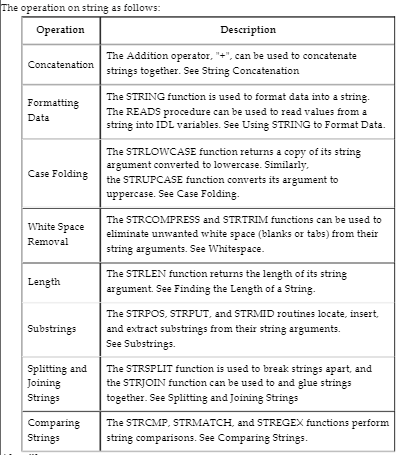
Palindrome string:-

**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.



**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)

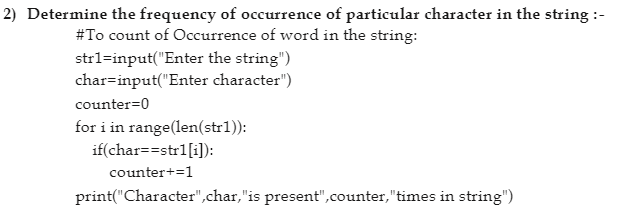
for i in range(len(list1)):

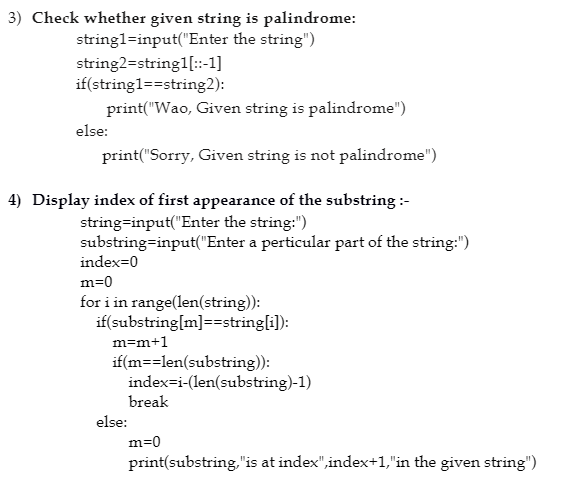
if m<len(list1[i]):

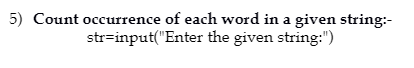
m=len(list1[i])

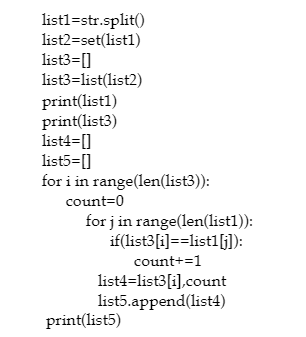
word=i

print("Word with longest length is:",len(list1[word]))









# Experiment No. 3

**Title**: Matrix Operations.

## Objectives:

* To understand implementation of two-dimensional array.
* Understand the implementation of Matrix and perform various operations on it using two dimensional arrays.

## Problem Statement:

Write C/C++program for storing matrix. Write functions for

1. Check whether given matrix is upper triangular or not
2. Compute summation of diagonal elements
3. Compute transpose of matrix
4. Add, subtract and multiply two matrices
5. Determines the location of a saddle point if one exists (An m x n matrix is said to have a saddle point if some entry a[i][j]is the smallest value in row i and the largest value inj.)

## Outcomes:

On completion of this assignment students will be able to-

* + Implement the two-dimensional arrays.
  + Solve real world problem of matrix and perform various operations on it logically using two dimensional arrays.

## Software &Hardware requirements:

* + Open Source C Programming tools like G++/G CC or Eclipse.
  + 64-bitOpensourceLinuxoritsderivative.

## Theory-Concept:

**2DimensionalArrays:**

* Two-dimensional array are those type of array, which has finite number of rows and finite number of columns. The declaration formof2-dimensionalarray is

# Data\_type Array\_name[rowsize][columnsize];

* The type maybe any valid type supported by C.
* The rule for giving the array name is same as the ordinary variable.
* The row size and column size should be an individual constant.

The following declares a two-dimensional 3 by 3 array of integers and sets the first and last elementstobe10.

# int matrix [3][3];

**matrix [0][0]=10;**

# matrix [2][2]=10;

The following Figure illustrates a two dimensional array, matrix. The array contains three rows and tree columns, so it is said to be a 3-by-3 array. In general, an array with m rows and n columns is called an m-by-n array.

|  |  |  |  |
| --- | --- | --- | --- |
|  | [0] | [1] | [2] |
| [0] | 10 |  |  |
| [1] |  |  |  |
| [2] |  |  | 10 |

Every element in array matrix is identified by an element name of the form matrix[ i ][ j ]; matrix is the name of the array, and i and j are the subscripts that uniquely identify each element in matrix .Notice that the names of the elements in the first row all have a first subscript of 0; the names of the elements in the third column all have a second subscript of2.

In the case of Two-dimensional array, during declaration the maximum number of rows and maximum number of column should be specified for processing all array elements.

The implementation of the array stores all the elements in a single contiguous block of memory. The other possible implementation would be a combination of several distinct one-dimensional arrays. That’s not how C does it. In memory, the array is arranged with the elements of the rightmost index next to each other. In other words ,matrix[1][1] comes right before matrix[1][2]in memory.

The following array:

|  |  |  |  |
| --- | --- | --- | --- |
|  | [0] | [1] | [2] |
| [0] | 1 | 2 | 3 |
| [1] | 4 | 5 | 6 |
| [2] | 7 | 8 | 9 |

## Would be stored:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**SAMPLE PROGRAM for Two Dimensional Array to store and display values:#include<conio.h>**

## #include<stdio.h>

## intmain()

**{**

## int matrix [3][3],i,j,r,c;

## clrscr();

**printf(“Enter the order of matrix\n”);**

**scanf(“%d%d”,&r,&c);**

## printf(“Enter elements of %d \* %d matrix \n”,r,c);

## for(i=0;i<r;i++)

**for(j=0;j<c;j++)**

## scanf(“%d”,&matrix[i][j]);

## printf(“Givenmatrix:\n”);

**for(i=0;i<r;i++)**

## {

**for(j=0;j<c;j++)**

## printf(“%d\t”,matrix[i][j]);

**printf(“\n”);**

## }

**}**

## Output;

**printf(“%d\t”,matrix[2][2]);getch();**

## return0;

**Enter the order of matrix2**

## 2

**Enter elements of 2\*2 matrix1**

## 2

**3**

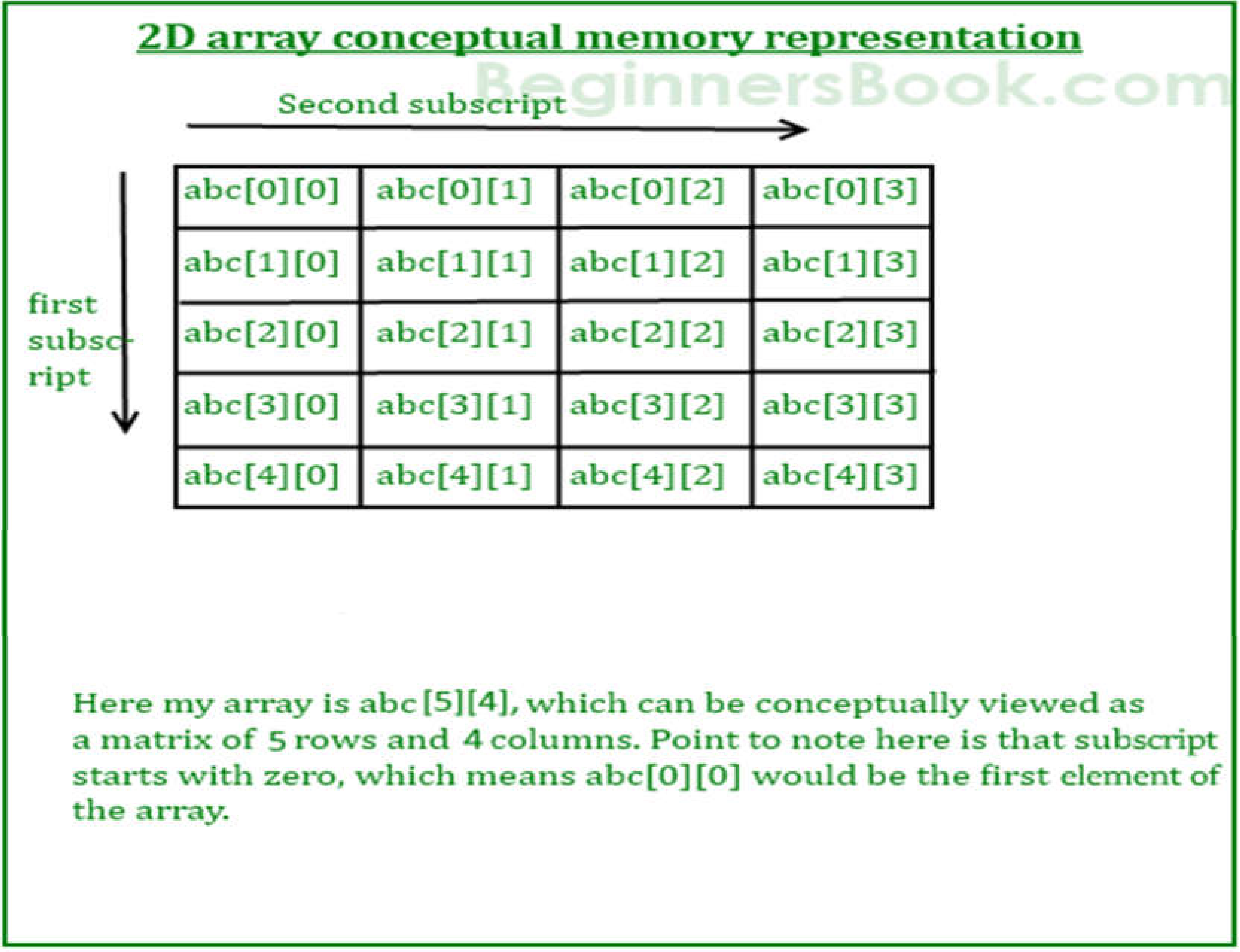
## 4

**Given matrix:**

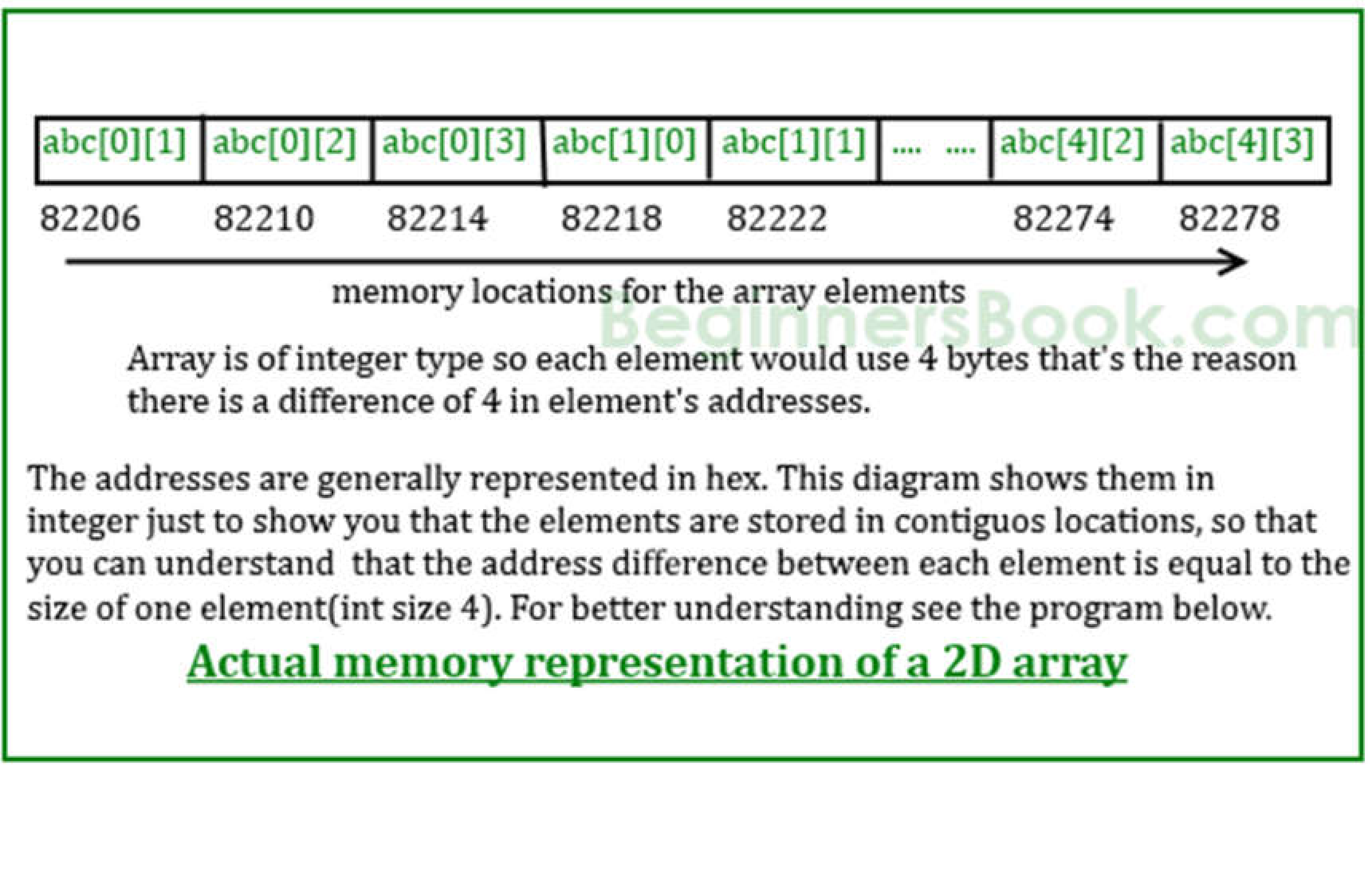
## 1 2

**3 4**

In below example, I have a 2D array abc of integer type. Conceptually you can visualize the above array like this:



How ever the actual representation of this array in memory would be something like this:



## Algorithm:

1. **To perform the addition of two matrices**

## Description: The program takes the two matrixes of same size and performs the addition Algorithm:

Step1: start

Step 2: read the size of matrices A,B – m,n

Step3: read the elements of matrix A

Step4: read the elements of matrix B

Step 5: perform the addition operation

Step6: print sum of matrices A and B

Step7: Stop

## Calculating transpose of a matrix in- place manner.

**DESCRIPTION: The transpose of a matrix is obtained by interchanging the r hence the order of the resultant matrix changes.**

## ALGORITHM:

Step1: start

Step2: read the size of matrix A

Step3: read the elements of matrix A

Step4: perform the transpose operation by interchanging the row and column values, the order of the resultant matrix number of rows in transpose matrix=number of columns in the given matrix number of columns in transpose matrix=number of rows in the given matrix

Step5: Transpose is obtained through

at[i][j]=a[i][j]

Step 6: print the resultant transpose matrix at.

Step7: stop

## Matrix multiplication by checking compatibility

**DESCRIPTION: Takes the two matrixes of different sizes and checks for possibility of multiplication nand perform multiplication if possible. ALGORITHM:**

Step1: start

Step2: read the size of matrices A, B

Step3: check compatibility of matrices for multiplication i.e., number of columns in the first matrix should be equal to number of rows in the second matrix.

Step 4: read the elements of matrix A

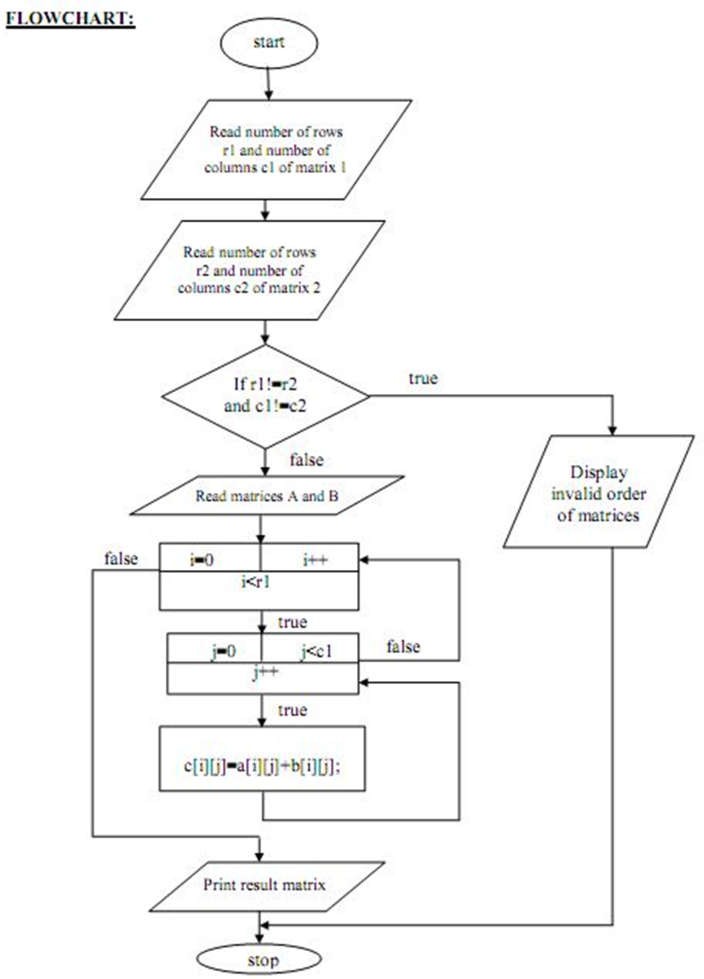
Step5: read the elements of matrix B

Step 6: perform the multiplication operation by storing the resulting values into matrix C.

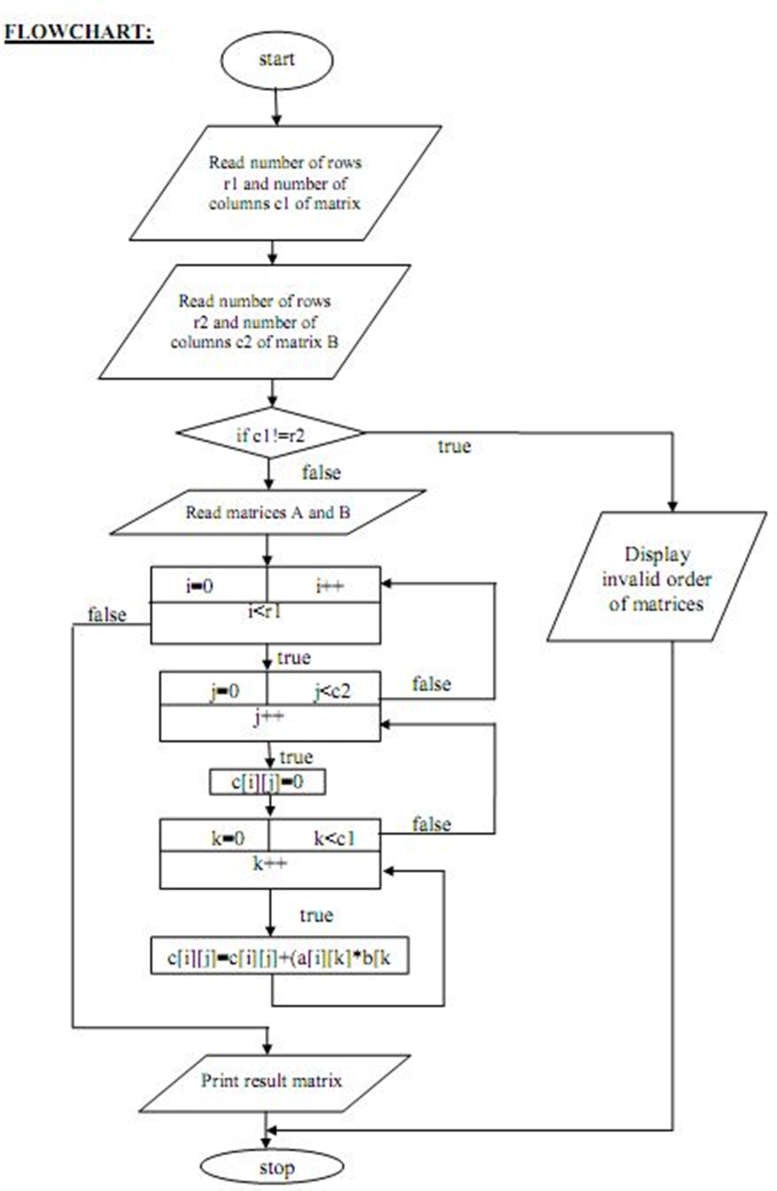
Step7: print the resultant matrix C.

Step8: Stop

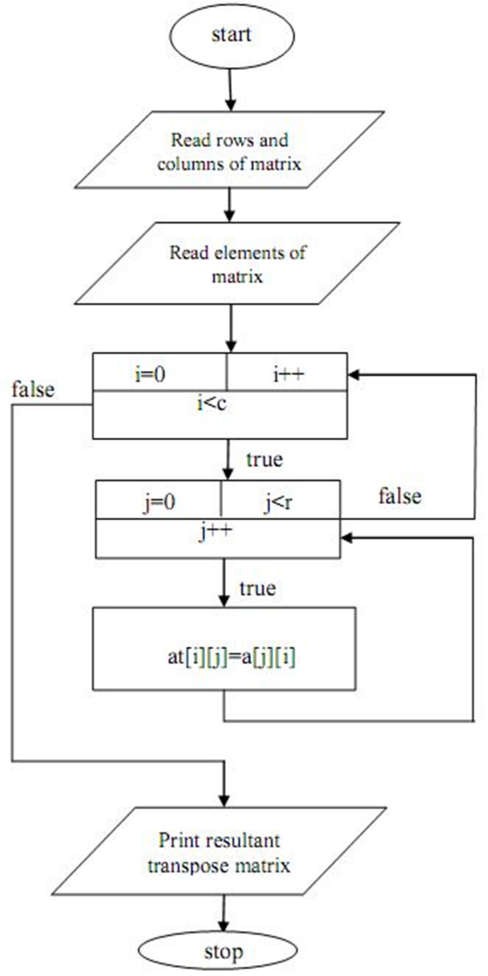
## Flowchart:



**Fig.1Flowchart for Addition of Two Matrices**



## Fig.2Flowchart for Multiplication of Two Matrices



**Fig.3Flowchart for Transpose of Matrix**

## Conclusion:

Hence we have learned Two Dimensional Array as a Data structure and how to use it to implement matrix.

# Experiment No 4

AIM: Implementation of sorting methods 1. Selection sort 2. Bubblesort

OBJECTIVES:

* To understand the concept of sorting mechanism.
* To explore the deep concept of time complexities of different sorting techniques.

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

1. Selection Sort
2. Bubble sort and display top five scores.

OUTCOMES:

 To analyze & understand various searching &sorting algorithm.

SOFTWARE & HARDWARE REQUIREMENTS:

Softwares: Open-Source Python, Programming tool like Google Colab, Jupyter Notebook, Pycharm, Spyder, G++/GCC

Hardware: Pentium Dual Core (3.00GHz) 4 GB RAM

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 left most 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(n2) where n are no. of items.

**Example**:

We take the below depicted array for our example.



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 were place 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 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−



Now, we should learn some programming aspects of selection sort.

procedure selection sort list : array of items

n :sizeoflistfori=1ton-1

/\* set current element as minimum\*/

min = i

**Pseudocode**

/\*checktheelementtobeminimum\*/ forj=i+1ton

iflist[j]<list[min]then min =j;

end if end for

/\*swaptheminimumelementwiththecurrentelement\*/ if indexMin != ithen

swaplist[min]andlist[i] end if

endfor

end procedure

**Bubble sort:**

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison based algorithm in which each pair of adjacent elements is compared and elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average e and worst case complexity are of O(n2) where n are no. of items.

How bubble sort works?

We take an unsorted array for our example. Bubble sort take Ο(n2)times we're keeping short and precise.



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

In this case,value33isgreaterthan14,so it is already insorted locations. Next, we compare 33 with

27.



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



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



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



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



We swap these values. We find that were a chat 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 sorts learns that array is completely sorted.



Now we should look into some practical aspects of bubble sort.

**Pseudocode**

We assume **list** is an array of **n** elements. We further assume that **swap** function, swaps the values of given array elements.

begin BubbleSort(list) forallelementsoflist

if list[i] >list[i+1] swap(list[i], list[i+1])

end if end for return list

end BubbleSort

We observe in algorithm that Bubble Sort compares each pair of array element unless the whole array is completely sorted ascending. This may cause few complexity issues like what if the array needs no more swapping as all the elements are already ascending.

To ease-out the issue, we use one flag variable **swapped** which will help us to see if any swap is happened or not. If no swap is occurred, i.e. the array requires no more processing to be sorted, it will come out of the loop.

**CONCLUSION:** Thus we have implemented python program for sorting elements using Selection sort &Bubble sort methods.

# Experiment No. 5

**Title :Sorting of an array using** insertion and shell sort.

* To store N number elements in array/list
* To apply specific strategy for sorting
* To perform Insertion and Shell sort on array

**Problem Statement:** - 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

1. Insertion Sort
2. Shell sort and display top five scores

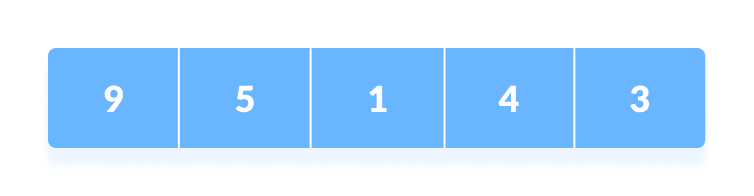
**Outcome:**

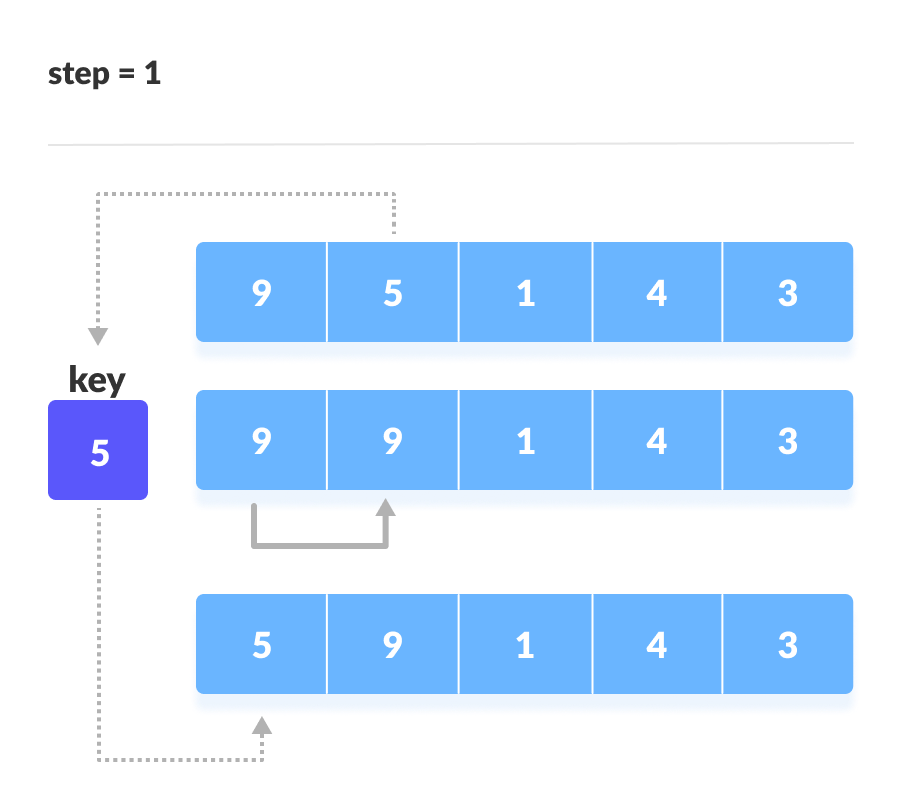
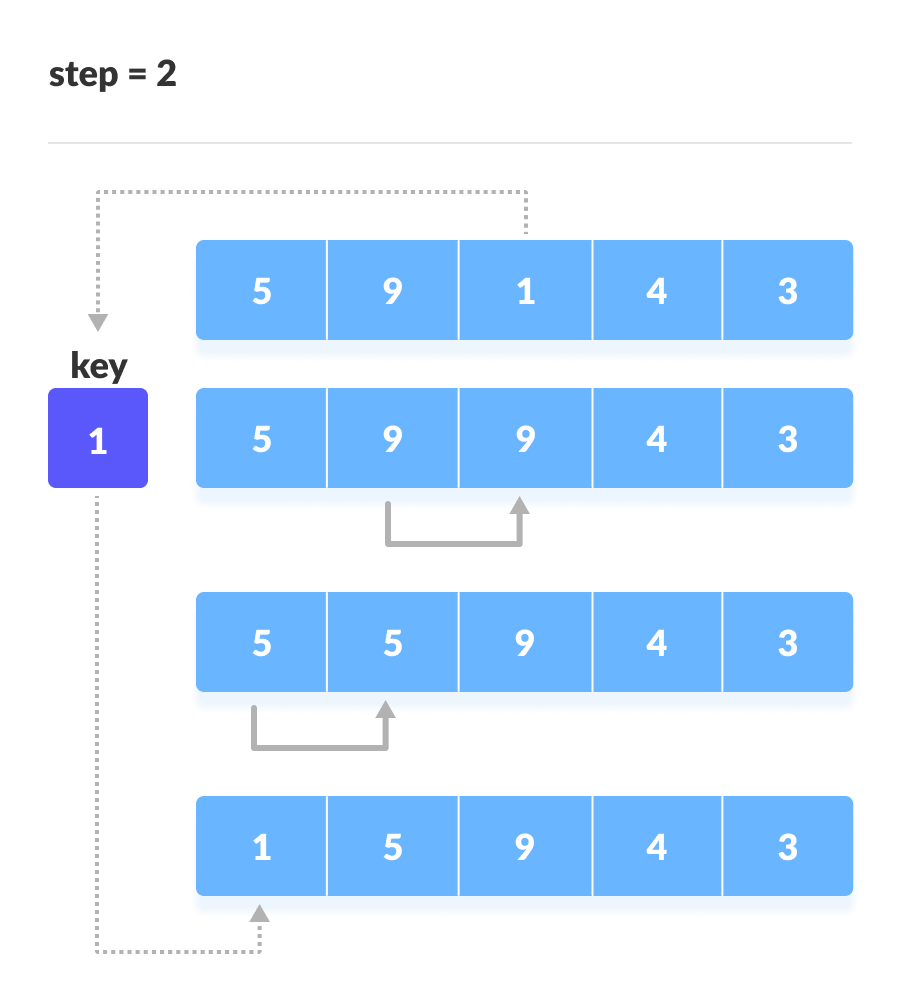
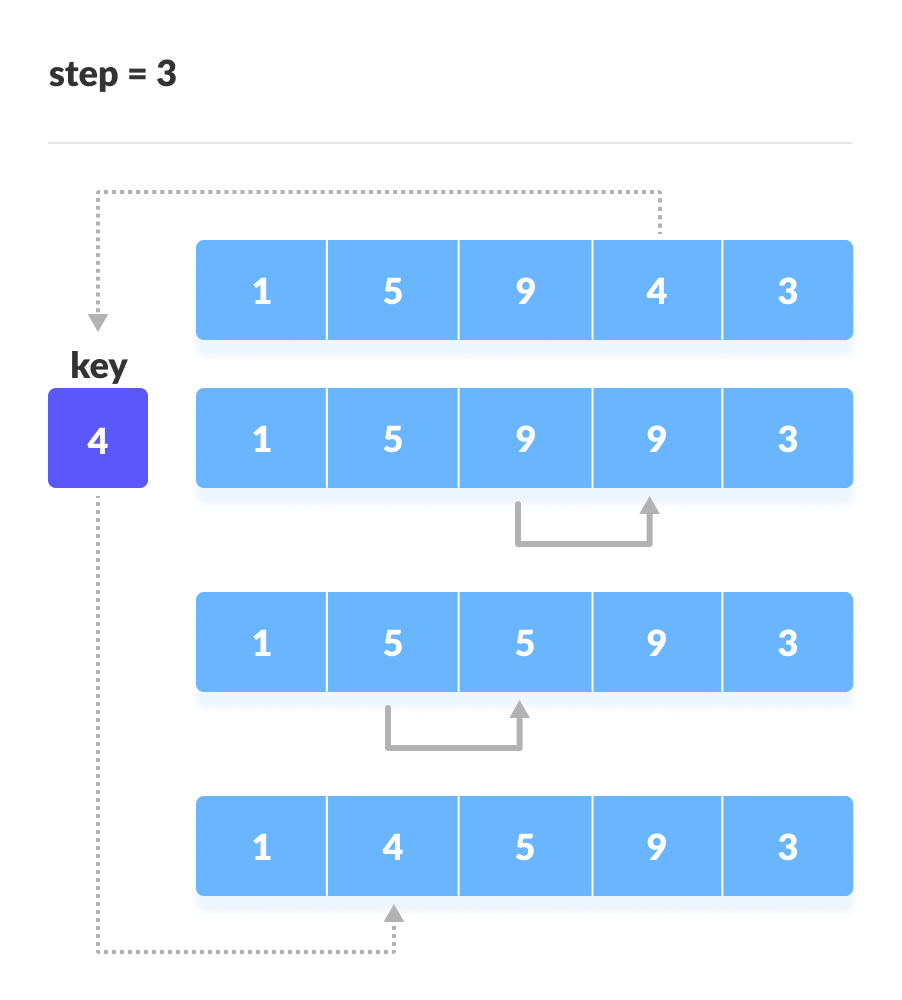
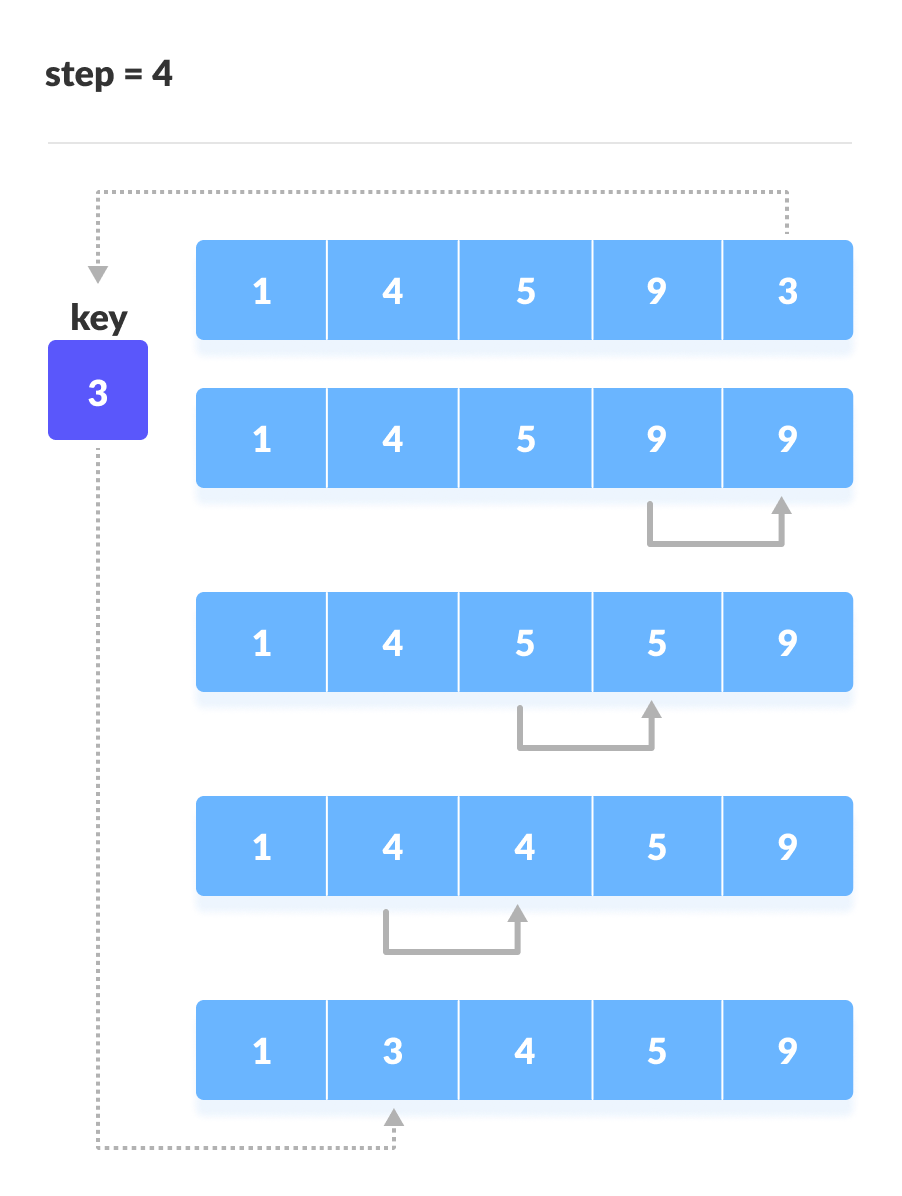
* Display Unsorted array/elements accepted by user
* PerformInsertion&Shellsortanddisplaysortedelementsandtopfivescores

a**) Insertion Sort Algorithm:** Insertion sort is [a sorting algorithm](https://www.programiz.com/dsa/sorting-algorithm) that places an unsorted element at its suitable place in each iteration. Insertion sort works similarly as we sort cards in our hand in a card game.

## Working of Insertion Sort

Suppose we need to sort the following array.

Initial array

1. The first element in the array is assumed to be sorted. Take the second element and store it separately in key.  
     
   Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.If the first element is greater than key, then key is placed in front of the first element.
2. Now, the first two elements are sorted.  
     
   Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.Place 1 at the beginning
3. Similarly, place every unsorted element at its correct position.Place 4 behind 1Place 3 behind 1 and the array is sorted

## Insertion Sort Algorithm

insertionSort(array)

markfirst element as sorted

foreach unsorted element X

'extract'the element X

forj <- lastSortedIndex down to 0

ifcurrent element j > X

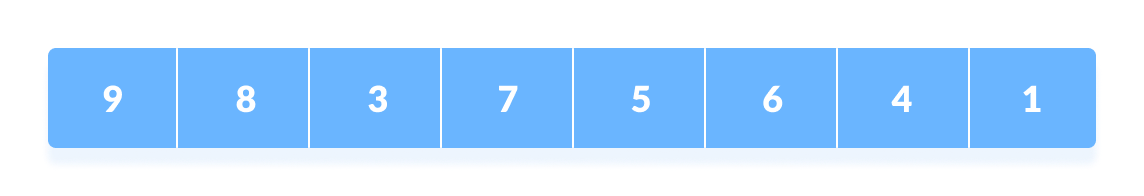
movesorted element to the right by 1

breakloop and insert X here

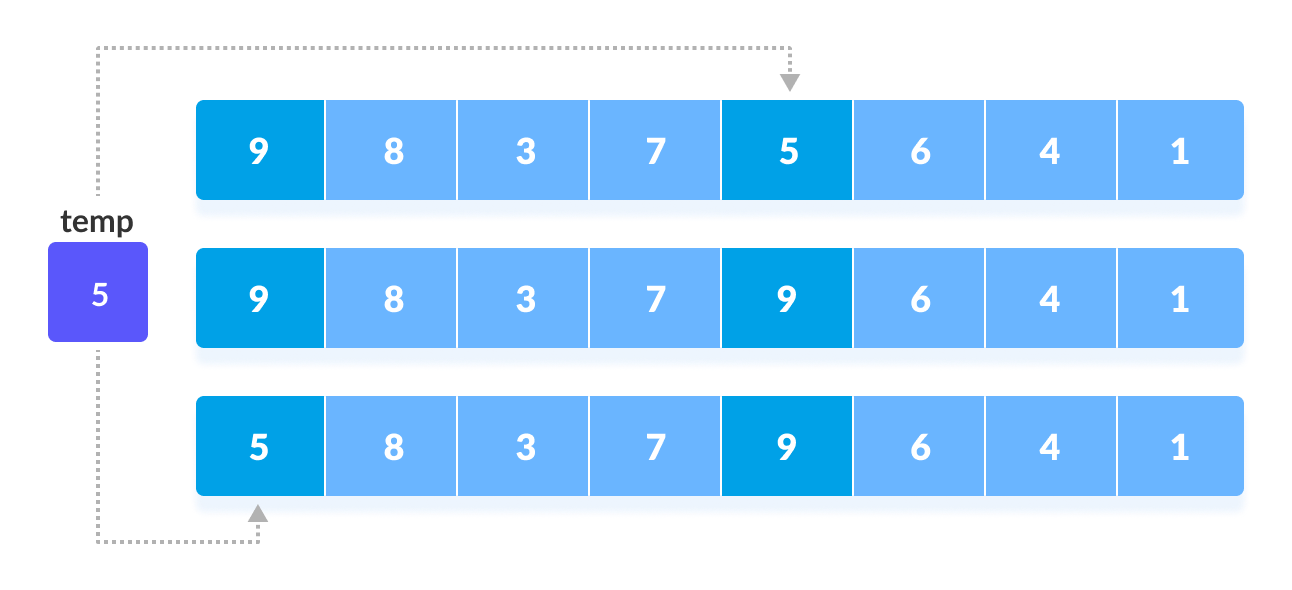
endinsertionSort

# b) Shell Sort Algorithm: Shell sort is a generalized version of the [insertion sort algorithm](https://www.programiz.com/dsa/insertion-sort). It first sorts elements that are far apart from each other and successively reduces the interval between the elements to be sorted.

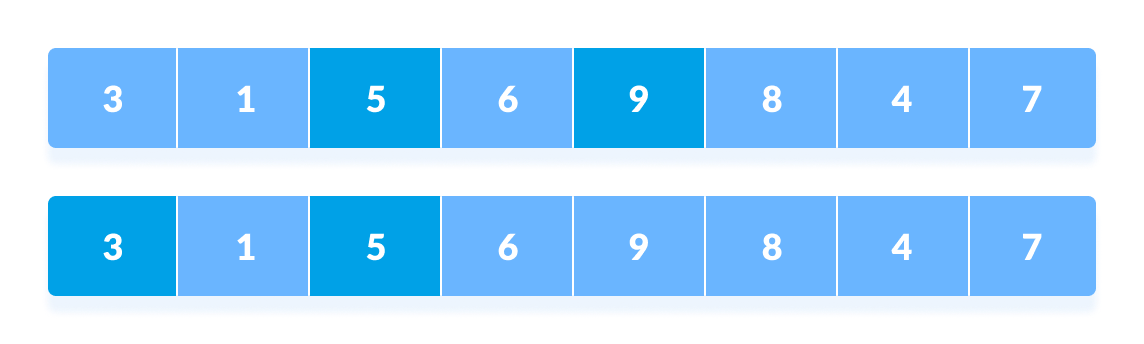
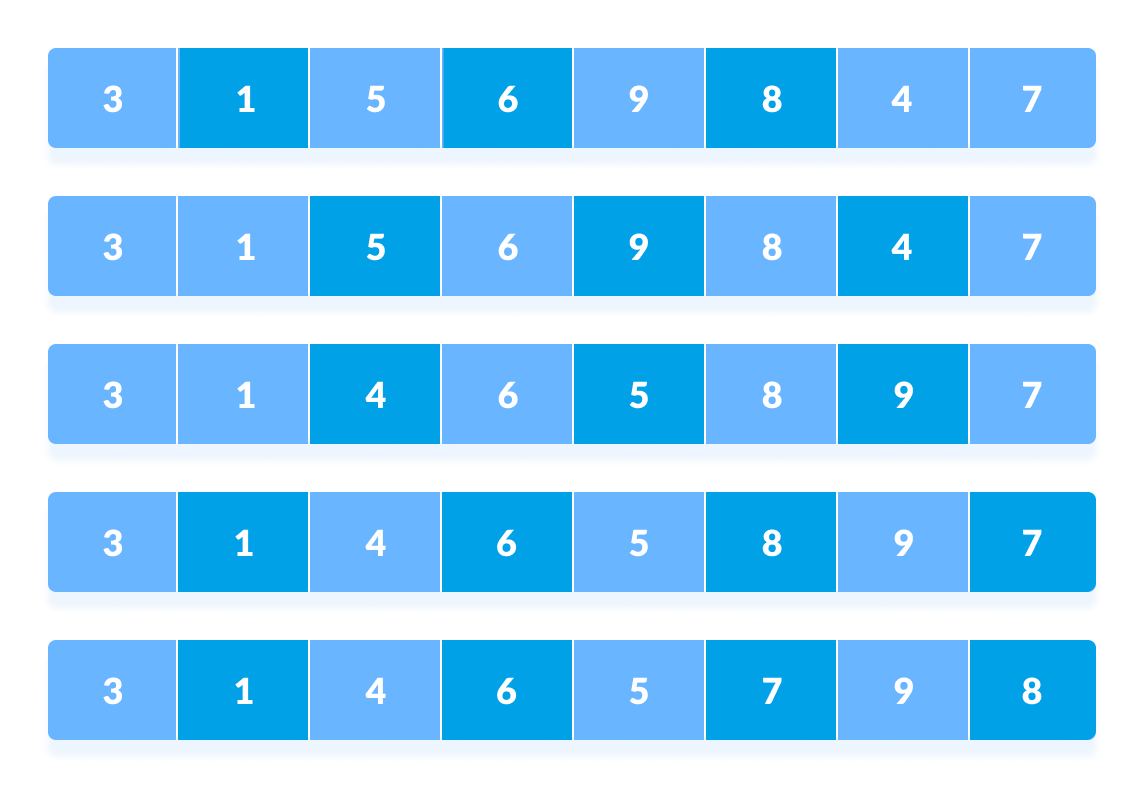
## Working of Shell Sort:

1. Suppose, we need to sort the following array.

Initial array

1. We are using the shell's original sequence (N/2, N/4, ...1) as intervals in our algorithm.  
     
   In the first loop, if the array size is N = 8 then, the elements lying at the interval of N/2 = 4 are compared and swapped if they are not in order.
   1. The 0th element is compared with the 4th element.
   2. If the 0th element is greater than the 4th one then, the 4th element is first stored in temp variable and the 0th element (ie. greater element) is stored in the 4th position and the element stored in temp is stored in the 0th position.Rearrange the elements at n/2 interval  
      This process goes on for all the remaining elements.

Rearrange all the elements at n/2 interval

1. In the second loop, an interval of N/4 = 8/4 = 2 is taken and again the elements lying at these intervals are sorted.Rearrange the elements at n/4 interval  
   You might get confused at this point.All the elements in the array lying at the current interval are compared.  
   The elements at 4th and 2nd position are compared. The elements at 2nd and 0th position are also compared. All the elements in the array lying at the current interval are compared.
2. The same process goes on for remaining elements.Rearrange all the elements at n/4 interval
3. Finally, when the interval is N/8 = 8/8 =1 then the array elements lying at the interval of 1 are sorted. The array is now completely sorted.
4. Rearrange the elements at n/8 interval

## Shell Sort Algorithm:

shellSort(array, size)

for interval i <- size/2n down to 1

for each interval "i" in array

sort all the elements at interval "i"

end shellSort

**Write algorithm/pseudocode foreach function:**

* 1. To accept percentage from user for N number of students.
  2. To perform Insertion sort and print sorted elements
  3. To perform Shell sort and print sorted elements
  4. To display top five scores of Insertion sort
  5. To display top fivescoresof Shellsort

# Experiment No 6

AIM: Implementation of Quick Sort. OBJECTIVES:

* To understand the concept of sorting mechanism.
* To explore the deep concept of time complexities of different sorting techniques.

PROBLEM STATEMENT:

Write C++program to store first year percentage of students in array. Sort array of floating point numbers in ascending order using quick sort and display top five scores.

OUTCOMES:

 To analyze & understand various searching & sorting algorithm.

THEORY:

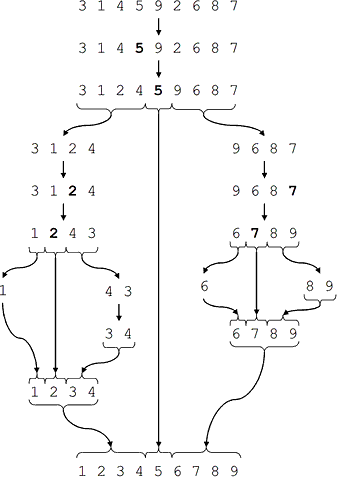
**Quick Sort:**

As one of the more advanced sorting algorithms, you might think that the Quicksort Algorithm is steeped in complicated theoretical background, but this is not so. Like Insertion Sort, this algorithm has a fairly simple concept at the core, but is made complicated by the constraints of the array structure.

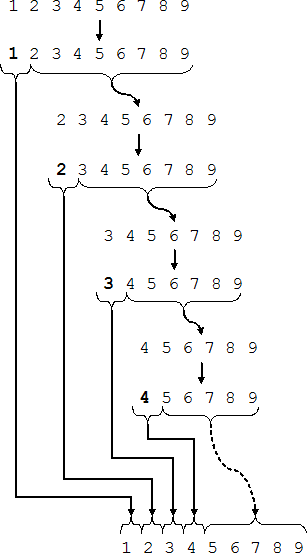
The basic concept is to pick one of the elements in the array as a pivot value around which the other elements will be rearranged. Everything less than the pivot is moved left of the pivot(into the left partition). Similarly, everything greater than the pivot goes into the right partition. At this point each partition is recursively quick sorted.

The Quick sort algorithm is fastest when the median of the array is chosen as the pivot value. That is because the resulting partitions are of very similar size. Each partition splits itself in two and thus the base case is reached very quickly.

In practice, the Quick sort algorithm becomes very slow when the array passed to it is already close to being sorted. Because there is no efficient way for the computer to find the median element to use as the pivot, the first element of the array is used as the pivot. So when the array is almost sorted, Quicksort doesn't partition it equally. Instead, the partitions are lopsided like in Figure2. This means that one of the recursion branches is much deeper than the other, and causes execution time to go up. Thus, it is said that the more random the arrangement of the array, the faster the Quick sort Algorithm finishes.



**Figure :** The ideal Quicksort on a random array.



**Figure :** Quicksort on an already sorted array.

These are the steps taken to sort an array using Quick Sort. The pseudo code for the above algorithm can be derived as−

function partitionFunc(left, right, pivot) leftPointer = left -1

rightPointer = right while Truedo

while A[++leftPointer] <pivotdo

//do-nothing end while

while rightPointer >0 &&A[--rightPointer] >pivot do

//do-nothing end while

if leftPointer >= rightPointer

break

else

swap leftPointer,rightPointer end if

end while

swap leftPointer,right return leftPointer

end function

To get more into it, let see the pseudocode for quick sort algorithm −

procedure quickSort(left, right) if right-left <=0

return else

pivot = A[right]

partition=partitionFunc(left,right,pivot) quickSort(left,partition-1) quickSort(partition+1,right)

end if

end procedure

**CONCLUSION:** Thus we have implemented python program to implement Quick sort to display top 5scores

# Experiment No: 7

**Title:**

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

a) to compute 1‘s and 2‘s complement

1. b) add two binary numbers

**Index terms:** Binary Numbers, Doubly linked list

**Theory:**

* **Doubly Linked list:**

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list.



**Node structure of DLL in C :**

/\* Node of a doubly linked list \*/

structnode

{

   intdata;

   structnode \*next; // Pointer to next node in DLL

   structnode \*prev; // Pointer to previous node in DLL

};

# 1’s and 2’s complement of a Binary Number

**1’s complement** of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.

Examples:

1's complement of "0111" is "1000"

1's complement of "1100" is "0011"

**2’s complement** of a binary number is 1 added to the 1’s complement of the binary number.  
Examples:

2's complement of "0111" is "1001"

2's complement of "1100" is "0100"

**Algorithm:**

1. Read decimal value from user.
2. Convert it into binary format.
3. Store all digits in Doubly linked list.
4. Finds the 1's complement of the binary number

for(i=0; i<SIZE; i++)

{

if(binary[i]=='1')

{ onesComp[i] = '0';

} else if(binary[i]=='0')

{ onesComp[i] = '1';

}

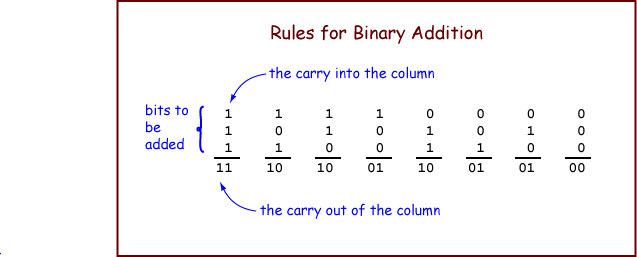
}

onesComp[SIZE] = '\0';

1. Find 2’ s complement by adding one to 1's complement.
2. Print 1's complement and 2s complement.

* **Addition of binary numbers:**

**Rules for binary addition:**



**Conclusion** Thus I have studied 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

# Experiment No 8

**Title:** set operations using linked list.

**Objectives:** To understand set operation. Representation and implementation of operation of sets using linked list

## 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 C++ program to store two sets using linked list. Compute and display-

1. Set of students who like both vanilla and butter scotch
2. Set of students who like either vanilla or butterscotch or not both
3. Number of students who like neither vanilla nor butterscotch

## Outcomes:

1. List of students who like either vanilla or butterscotch or both
2. List of students who like both vanilla and butterscotch
3. List of students who like only vanilla not butterscotch
4. List of students who like only butterscotch not vanilla
5. Number of students who like neither vanilla nor butterscotch

## Software &Hardware requirements:

Linux operating system Eclipse IDE with g++compiler

## Theory-Concept in brief:

Set:

a set is a collection of objects which are called the members or elements of that set. If we have a set we say that some objects belong(or do not belong)to this set, are (or are not) in the set.

Examples: the set of students in this room; the English alphabet maybe viewed as the set of letters of the English language; the set of natural numbers Operations on sets:

1. Union: The union of A and B, written A 𝖴 B , is the set whose elements are just the elements of A or B or of both.
2. Intersection: The intersection of A and B , written A ∩ B , is the set whose elements are just the elements of both A and B.
3. Difference: Another binary operation on arbitrary sets is the difference― A minus B‖, written A–B, which‗subtracts from A all elements which are in

B.[Also called relative complement :the complement of B relative to A.]

1. Complement: This operation is creating a set A, which is the set consisting of everything not in A

## Algorithm:

1. **Algorithm difference (struct node \*head1, struct node \*head2) Precondition:** Accept the two sets of name in the form of linked list. **Post condition**: Difference of two sets

**Return:** head node of the resultant linked list(Difference)

* 1. Setp=head1,i.e.headof1stlinkedlist
  2. Initialize the 3rdlinked listasemptyi.e.head3=NULL
  3. while(p!=NULL)
     1. Set flag=0
     2. Set q=head2 ,i.e.headof2ndlinkedlist
     3. while(q!=NULL)
        1. if( strcmp (p->name, q->name)==0)
           1. Setflagto1andgotostepd
        2. else
           1. Move q to the next node, q=q->next
     4. if(flag!=1)
        1. if(head3==NULL)
           1. Allocate memory for New node
           2. Strcpy (New->name, p->name)
           3. Set r=New and r->next=NULL
           4. Make Newnode as head node of resultant linked list
        2. else
           1. Allocate memory for Newnode
           2. r->next=New
           3. strcpy(New->name, p->name)
           4. Set r=r->next and r->next=NULL
     5. Move p to the next node, p=p->next
  4. Return head 3i.e. head of resultant linked list
  5. Stop

**Algorithm Intersection (struct node \*head1, struct node \*head2) Precondition:** Accept the two sets of name in the form of linked list.

**Post condition**: Intersection of two sets

**Return:** Nil

1. Set p=head1,i.e.headof1stlinkedlist
2. Initialize the 3rdlinked listasemptyi.e.head3=NULL
3. while(p!=NULL)
   1. Set q=head2,i.e.headof2ndlinkedlist
   2. while(q!=NULL)
   3. f(strcmp(p->name,q->name)==0)
      1. if(head3==NULL)
         1. Allocate memory for Newnode
         2. Strcpy (New->name ,p->name)
         3. Set r=New and r->next=NULL;
         4. Make new node as head of linked list
      2. else
         1. Allocate memory for Newnode
         2. r->next=New
         3. strcpy (New->name, p->name)
         4. Set r=r->next and r->next=NULL
      3. Go to step c

**ii.** Move q to the next node, q=q->next

* 1. Move p to the next node, p=p->next

1. Call display(head3) function to display resultant linked list containing intersection
2. Stop
3. **Algorithm Union(struct node \*head1,struct node \*head2) Precondition:** Accept the two sets of name in the form of linked list. **Post condition**: Union of two sets

**Return:** head node of the resultant linked list(Union)

* 1. Set q=head2,i.e.headof2ndlinkedlist
  2. Copy all contents of first linked list in third, head3=head1
  3. Set r=head3,i.e.headof3rdlinkedlist
  4. while(r->next!=NULL)Move r to the next node ,r=r->next
  5. while(q!=NULL)
     1. Set p=head1,i.e.headof1stlinkedlist
     2. Initializeflagto0
     3. while(p!=NULL)
        1. if(strcmp(p->name, q->name)==0)Set flag to 1and go to step d
        2. else Move p to the next node ,p=p->next
     4. if(flag==0)
        1. Allocate memory for Newnode
        2. r->next=New
        3. strcpy(New->name ,q->name);
        4. Set r=r->next and r->next=NULL
     5. Move q to the next node, q=q->next
  6. Return head 3,i.e.head of resultant linked list containing union
  7. Stop

**Conclusion:**

Thus I have studied concept of set and its representation using array. I have also implemented all the operations of set.

# Experiment No. 9

**AIM:** Write a C++ program using stack to check whether given expression is well parenthesized or not.

AIM: To check the given expression is well parenthesized or not. OBJECTIVES:

* To understand the concept of stack.
* To explore the deep concept of pointers and parenthesis.
* To understand the different operations on stack.
* To understand the memory requirement for stack data structures.

PROBLEM STATEMENT:

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.

OUTCOMES:

 To design all the aspects of stack &its variants.

SOFTWARE & HARDWARE REQUIREMENTS:

Softwares: Eclipse/Geany Editor/gedit

G++ compiler

Fedora20OperatingSystem Hardware: Pentium Dual Core(3.00GHz)

2 GB RAM

THEORY:

*Basic operations onStack*

1. **Void initialize(stack\*P)**:It initializes a stack as an empty stack. Initial value of stack is set to-1.

void initialize (stack \*P)

{

P -> top = -1;

}

1. **intempty(stack\*P)**: Function checks whether the stack isempty. It returns 1or 0 depending on whether the stack is empty or not.

int empty ( stack \*P)

{

if(P->top==-1)

return(1);

return (0);

}

1. **intfull(stack\*P)**: Function checks whether the stack is full. Whenever the stack is full, top points to the last element (ie. MAX-1) of the array. It returns 1 or 0 depending on whether the stack is full or not.

int empty ( stack \*P)

{

if (P -> top ==)

return (1);

return (0);

}

1. int push(stack\*P,intx):The function inserts the element x onto the stack pointed by

P. Insertion will cause an overflow if the stack is full. Void push(stack\*P, intx)

{

P->top=P->top+1; P->data[P->top]=x;

}

1. int pop(stack\*P):The function deletes top most element from the stack and also returnsittothecallingprogram.Deletionfromanemptystackwillcauseunderflow.

int pop ( stack \*P)

{

int x;

x=P->data[P->top]; P->top=P->top-1; return(x);

}

*Applications ofStack*

1. Expression Conversion
2. Expression Evaluation
3. Parsing well-formed parenthesis
4. Recursion
5. FunctionCall
6. Decimal to binary conversion
7. Reversing astring.

##### Well formedness ofparenthesis

An expression is said to be well formed if every opening bracket has a corresponding closing bracket and there is no extra bracket.

Examples (a + b- (c +d))

(p–(q/(r+s)^(t–u))\*v) (a–[b/c\*{d+e/f}^g])

Algorithm:

* Declare a character stackS.
* Now traverse the expression stringexp.
  1. If the current character is a starting bracket(**‘(‘or‘{‘or‘[‘**)then push it to stack.
  2. If the current character is a closing bracket(**‘)’or‘}’or‘]’**)then pop from stack and if the popped character is the matching starting bracket then fine else brackets are not balanced.
* After complete traversal ,if there is some starting bracket left in stack then “not balanced”

**CONCLUSION:** Thus we implemented C++program to check whether the given expression is well parenthesized or not.

# Assignment No 10

**Title:** Infix to Postfix Conversion and valuation

**Objective:**

1. Understand the concept to how to convert in fix to postfix

expression.

1. Understand how to evaluate the expression using stack**.**

## Problem Statement:

Implement C++program for expression conversion as in fix to post fix 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.

**Outcome:**

1. Will be able to understand the concept of how to convert infix to postfix expression.
2. Will be able to understand how to evaluate the expression using stack.

**Software & Hardware Requirements:**

1. 64-bitOpensourceLinuxoritsderivative
2. Open Source C++Programming tool like G++/GCC

## Theory:

**Infix expression:** The expression of the form a opb. When an operator is in- between every pair of operands.

**Post fix expression:** The expression of the form a bop. When an operator is followed for every pair of operands.

## Need of postfix representation of the expression

The compiler scans the expression either from left to right or from right to left. Consider the below expression :a op1bop2cop3dIfop1=+,op2=\*,op3=+

The compiler first scans the expression to evaluate the expression b\*c, then again scan the expression to add at o it. The result is then added to d after another scan. The repeated scanning make sit very in- efficient. It is better to convert the expression to postfix (or prefix) form before evaluation.

The corresponding expression in postfix form is: abc\*+d+. The postfix expressions can be evaluated easily using a stackToimplementconversionofaninfixexpressionintopostfixexpression.

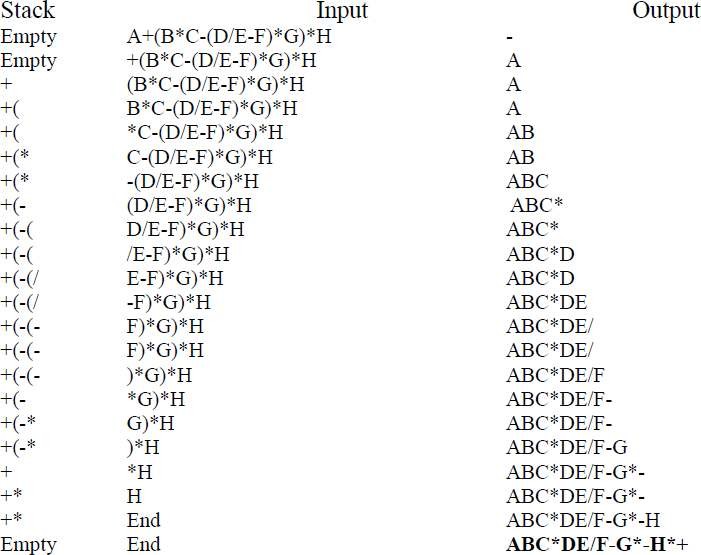
Algorithm:

1. Define a stack of to hold the characters.
2. Initialize stack top=-1.
3. Read the infix expression.
4. Add„)‟to the end of the infix expression.
5. Push„(„to the stack.
6. Scan the infix expression from left to right and repeat the step7 for all the characters in the infix expression.
7. If the character is an operand Add it to the postfix expression If the character is „(„ Push it to the stack If the character is „)‟ Repeatedly Pop the characters from the stack and add it to the postfix expression until „)‟ is encountered. Pop „)‟ If the character is an operator If the precedence of the character is lesser than or equal to the precedence of the operator in the top of the stack repeatedly pop the characters from the stack and add it to the post fix expression till an operator of higher precedence is encountered. Push the operator to the top of the stack.
8. Print the postfix expression.
9. Stop.

Conversion To Postfix

EXAMPLE:

A+(B\*C-(D/E-F)\*G)\*H



To implement evaluation of a postfix expression. Algorithm for Postfix Evaluation:

1. Declare the structure for the stack.
2. Read the postfix expression.
3. Repeatedly execute the following for all the characters in the expression from left to right If the character is a number Then convert the character to integer by subtracting „0‟ from the character. Push the integer value into the stack. If the character is a operator Pop two values from the stack and perform the operation and store the result to stack.
4. The stack top has the result. Pop it and Print.

1. Stop

## Flowchart:

**Draw Flowchart**

## Test Cases:

Input:

Infix Expression:(3\*2)/(5-3)Output:

PostfixExpression:32\*53-/

Evaluationresult:3

## Conclusion/Analysis:

Understood the concept of how to convert infix to postfix expression and how to evaluate the expression using stack**.**

Experiment No11 **AIM: Implementation of job queue for operating system**

**OBJECTIVES:**

* + To understand the concept of queue.
  + To explore the deep concept insertion & deletion of elements in/from queue.

PROBLEM STATEMENT:

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.

OUTCOMES:

 To realize different types of queue & its implementation.

Software & Hardware requirements:

Softwares: Eclipse/Geany Editor/Gedit

GCC & G++ compiler Fedora20OperatingSystem

Hardware: Pentium Dual Core (3.00GHz)

2 GB RAM

THEORY:

Queue is an abstract data structure, somewhat similar to Stack. In contrast to Queue, queue is opened at both end. One end is always used to insert data(enqueue)and the other is used to remove data(dequeue).Queue follows First-In-First-Out methodology ,i.e., the data item stored first will be accessed first.

A real world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world example can be seen as queues at ticket windows &bus- stops.

**Queue Representation**

As we now understand that in queue, we access both ends for different reasons, a diagram given below tries to explain queue representation as data structure−



Same as Queue, queue can also be implemented using Array, Linked-list, Pointer and Structures .For the sake of simplicity we shall implement queue using one-dimensional array.

**Basic Operations**

Queue operations may involve initializing or defining the queue, utilizing it and then completing erasing it from memory. Here we shall try to understand basic operations associated with queues−

* **enqueue()** − add (store) an item to the queue.
* **dequeue()**−remove(access)an item from the queue.

Fewmorefunctionsarerequiredtomakeabovementionedqueueoperationefficient.These are−

* **isfull()**−checks if queue is full.
* **isempty()** − checks if queue is empty.

In queue, we always **delete**(or access) data ,pointed by **front** pointer and while inserting(or storing)data inqueue we take help of **rear** pointer.

Let's first learn about supportive functions of a queue –

Is full()

As we are using single dimension array to implement queue, we just check for the rear pointer to reach at MAXSIZE to determine that queue is full. Incase we maintain queue in a circular linked-list, the algorithm will differ. Algorithm of is full()function−

begin procedure isfull

if**rear**equalstoMAXSIZE returntrue

else

return false endif

end procedure

Implementation of isfull() function in C programming language −

bool isfull() {

if(rear == MAXSIZE - 1) return true;

else

return false;

}

Is empty()

Algorithm of is empty()function−

begin procedure isempty

if **front** is less than MIN OR **front** is greater than **rear**

return true else

return false endif

end procedure

If value of **front** is less than MIN or0, it tells that queue is not yet initialized, hence empty. Here's the C programming code−

bool isempty() {

if(front <0 || front >rear)

return true; else

return false;

}

**Insertion Operation:**

As queue maintains two data pointers ,**front** and **rear**, its operations are comparatively more difficult to implement than Queue.

The following steps should be taken to insert data into a queue −

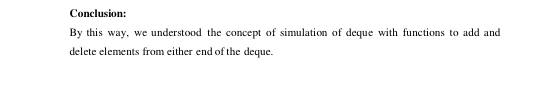
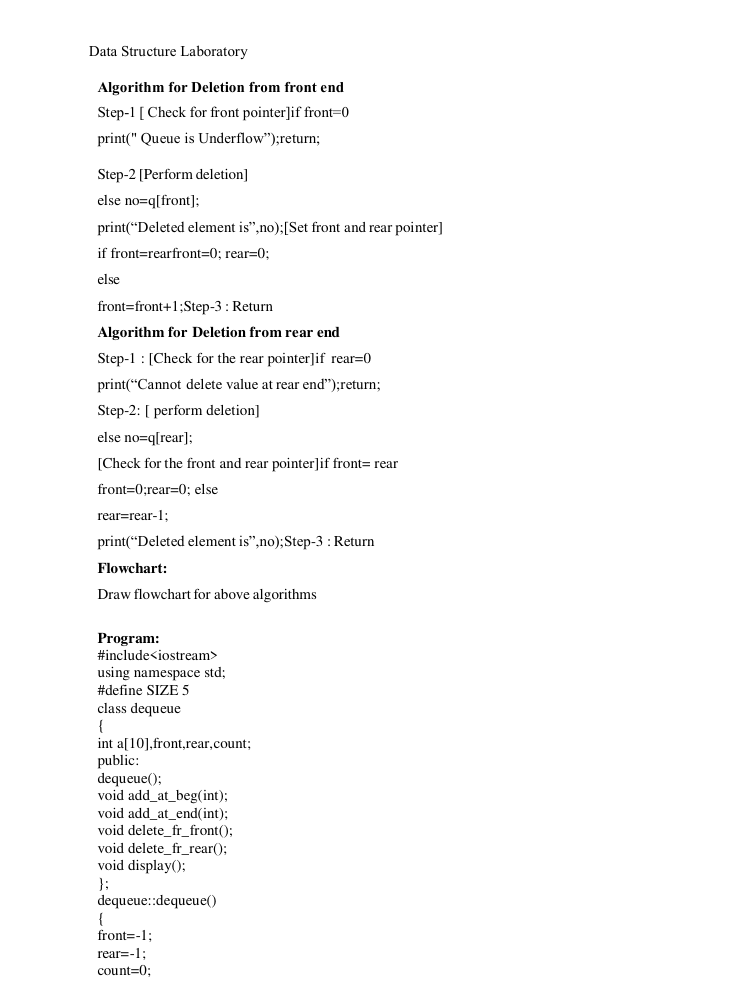
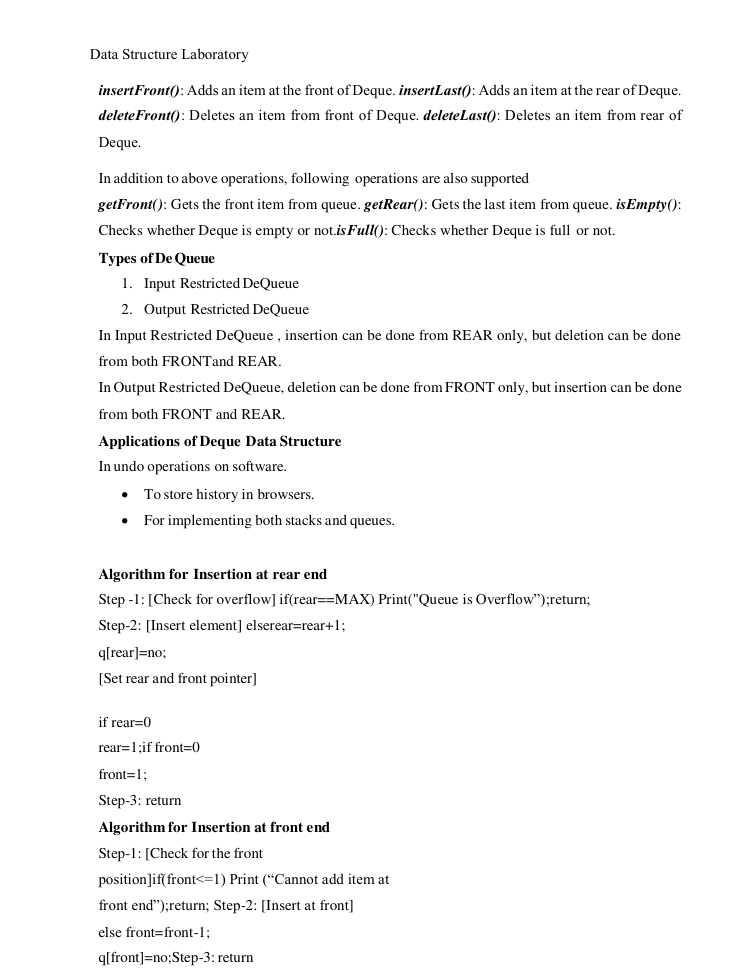
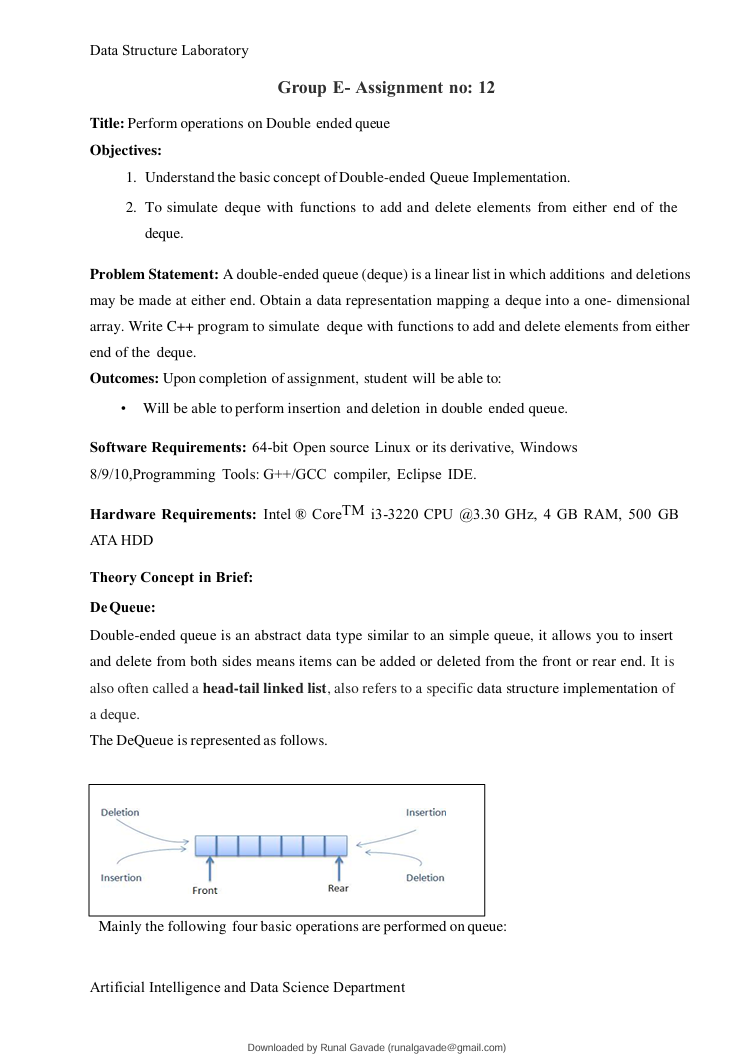
* **Step1**−Check if queue is full.
* **Step2**−If queue is full,produce overflow error and exit.
* **Step3**−If queue is not full, increment **rear** pointer to point next empty space.
* **Step4**−Add data element to the queue location, where rear is pointing.
* **Step 5** – return success.

Sometimes, we also check that if queue is initialized or not to handle any unforeseen situations.

**Deletion Operation:**

Accessingdatafromqueueisaprocessoftwotasks−accessthedatawhere**front**is pointing and remove the data after access. The following steps are taken to perform **delete** operation −

* **Step1**−Checkifqueueisempty.
* **Step2**−If queue is empty, produce underflow error and exit.
* **Step3**−If queue is not empty ,access data where **front** is pointing.
* **Step4**−Increment **front** pointer to point next available data element.
* **Step 5** – return success.

**CONCLUSION:** Thus we have implemented C++program for job queue for operating system. 

# Experiment No: 13

**Title:** Perform the different operations on Circular Queue.

## Objectives:

1. Understand the basic concept of Circular Queue Implementation.
2. Understand the concept of insertion ,deletion in a circular queue.

**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.

**Outcomes:** Upon completion of assignment ,student will be able to:

* Will be able to understand the concept of insertion and deletion in circular queue
* **Software Requirements:**64-bitOpen source Linuxor its derivative ,Windows 8/9/10,ProgrammingTools: G++/G CCcompiler, EclipseIDE.

**Hardware Requirements:** Intel ®CoreTMi3-3220CPU@3.30GHz,4GBRAM,500GB ATA HDD

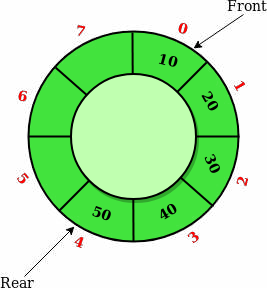
**Date of Completion:**

**Assessment Grade/Marks:**

**Assessor’s Sign:**

**Theory Concept in Brief:**

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **‘Ring Buffer’**.



## Basic Operations

Some of the basic operations of the circular queue areas follows:

1. **Front:** Returns the front position in the circular queue.
2. **Rear:** Returns the rear position in the circular queue.
3. **Enqueue:** Enqueue (value) is used to insert an element in the circular queue. The element is always inserted at the rearend of the queue.

Wefollowthefollowingsequenceofstepstoinsertanewelementinthecircular queue.

#1) Check if the circular queue is full: test ((rear== SIZE-1 &&front ==0) ||(rear== front- 1)), where „SIZE ‟is the size of the circular queue.

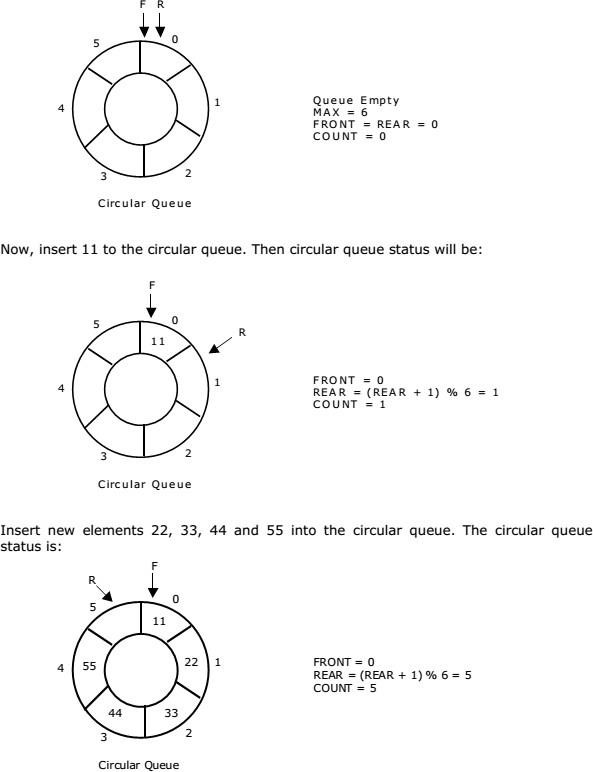
#2) If the circular queue is full then it displays a message as “Queue is full”. If queue is not full then, check if(rear == SIZE – 1 && front != 0). If it is true then set rear=0 and insert element.

1. **Dequeue:** Dequeue function is used to delete an element from the queue. In the circular queue, the element is always deleted from the front end. Given below is the sequence for dequeue operation in a circular queue.

## Steps:

1. Check if the circular queue is Empty: check if(front==-1).
2. If it is empty then display the message “Queue is empty”. If queue is not empty then perform step 3.
3. Check if(front==rear).If it is true then set front=rear=-1elsecheckif(front==size-1), if it is

True then set front=0 and return the element.



## RepresentationofCircularQueue:Letusconsideracircularqueue,whichcanhold maximum (MAX) of six elements. Initially queue is empty.

## Application of Circular Queue:

* 1. **CPU Scheduling:** Operating system process that requires some event to occur or for some other processes to complete for execution is often maintained in a circular queue so that they execute one after the other when all the conditions are met or when all events occur.
  2. **Memory Management:** Use of ordinary queues wastes memory space as already mentioned in our above discussion. Using a circular queue for memory management is beneficial for optimum memory usage.
  3. **Computer Controlled Traffic Signal System:** Computerized traffic signals are often added to a circular queue so that they repeat themselves after the specified time interval has elapsed.

## Drawback of Circular Queue

The drawback of circular queue is , difficult to distinguished the full and empty cases. It is also known as

## Boundary case problem.

1. In circular queue it is necessary that:
2. Before insertion, fullness of Queue must be checked(for overflow).
3. Before deletion ,emptiness of Queue must be checked(for underflow).

## Circular Queue Complexity Analysis:

The complexity of the enqueue and dequeue operations of a circular queue is O(1) for(array implementations).

## Algorithm of a Circular queue:

1. Initialize the queue, with size of the queue defined(maxSize),and head and tail pointers.
2. enqueue: Check if the number of elements is equal to max Size - 1:If Yes, then return Queue is full.

If No, then add the new data element to the location of tail pointer and increment the tail pointer.

1. dequeue: Check if the number of elements in the queue is zero: If Yes, then return Queue is empty.

If No, then increment the head pointer.

1. Finding the size:

If, tail>=head, size=(tail-head) +1

But if, head>tail, then size=max Size-(head-tail)+1

## Flowchart:

Draw Flow chart for above operations.

## Conclusion/Analysis

In this way, we understood the concept of insertion and deletion in circular queue