Marathwada Mitra Mandal's

College of Engineering, Pune

Karve Nagar,Pune-411 052

Accredited with 'A' Grade by NAAC, Recipient of "Best College Award 2019" by SPPU

Department of Computer Engineering

Lab Manual

Data Structure Laboratory (FDS)

(210243)

Prepared by,

Prof. Shubhada P. Mone

SE COMP

Semester I Academic Year 2020-21

PROGRAM OUTCOMES:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: The problems that cannot be solved by straightforward application of knowledge, theories and techniques applicable to the engineering discipline.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of t h e engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

INDEX

Sr. No.	Group	Title of Assignment	CO	PO
1	A	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. (Note- While realizing the group, duplicate entries should be avoided, Do not use SET built-in functions)	CO1	PO1, PO2, PO3, PO4, PO6
2	A	Write a Python program to store marks scored in subject "Fundamental of Data Structure" by N students in the class. Write functions to compute following: a) The average score of class b) Highest score and lowest score of class c) Count of students who were absent for the test d) Display mark with highest frequency	CO1	PO1, PO2, PO3, PO5, PO6
3(A)	A	Write a python Program for magic square. A magic square is an n * n matrix of the integers 1 to n2 such that the sum of each row, column, and diagonal is the same. (Please refer the example from syllabus copy).	CO1	PO1, PO2, PO5, PO4, PO6
3(B)	A	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	CO1	PO1, PO2, PO5, PO6
3(C)	A	Write a python program to compute following computation on matrix: a) Addition of two matrices b) Subtraction of two matrices c) Multiplication of two matrices d) Transpose of a matrix	CO1	PO1, PO4, PO5, PO6
4	B	Write a python program to store second year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using a) Insertion sort b) Shell Sort and display top five scores.	CO2	PO1, PO2, PO4, PO6
5	B	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.	CO1	PO1, PO2, PO3, PO4

6	**B**	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.	CO1	PO1, PO2, PO3, PO4, PO6
7	**C**	Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member‘s information using singly linked list. Store student PRN and Name. Write functions to: a) Add and delete the members as well as president or even secretary. b) Compute total number of members of club c) Display members d) Two linked lists exists for two divisions. Concatenate two lists.	CO1, CO2	PO2, PO3, PO4, PO5, PO6
8	**C**	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	CO1	PO2, PO3, PO4, PO5, PO6
9	**D**	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.	CO1	PO2, PO3, PO6
10	**D**	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.	CO1	PO2, PO4, PO5
11	**E**	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.	CO1	PO2, PO3, PO4, PO5, PO6

12	**E**	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.	CO1	PO2, PO3, PO4, PO5, PO6
13	**E**	Write C++ program to store set of negative and positive numbers using linked list. Write functions a) Insert numbers b) Delete nodes with negative numbers c) To create two more linked lists using this list, one containing all positive numbers and other containing negative numbers For two lists that are sorted; Merge these two lists into third resultant list that is sorted	CO1, CO2	PO2, PO3, PO4, PO5, PO6
14	**Content Beyond Syllabus**	Implement single source shortest path algorithm -Dijkstra's shortest path algorithm.	CO1	PO2, PO3, PO4, PO5, PO6, PO12
15	**Virtual Labs**	Generate expression tree and evaluate expression using tree.	CO1	PO1 PO5

Software Required:

1. 64 bit open source operating system
2. Eclipse version 3.8

Write-ups must include:

- **Group:**
- **Assignment No.**
- **Title**
- **Objectives**
- **Problem Statement**
- **Outcomes**
- **Theory(in brief)**
- **Algorithm**
- **Flowchart**
- **Test Cases**
- **Conclusion:**
- **FAQs:**

GROUP: A

ASSIGNMENT NO: 1

TITLE: Program for Set operations.

OBJECTIVE: To study & implement set operations.

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: -

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.

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

OUTCOME: Student will be able to use array concepts for implementing set operations.

THEORY: A set is an unordered collection of different elements. A set can be written explicitly by listing its elements using set bracket. If the order of the elements is changed or any element of a set is repeated, it does not make any changes in the set.

Set Union

The union of sets A and B (denoted by A ∪ B) is the set of elements which are in A, in B, or in both A and B. Hence, A ∪ B = {x | x ∈ A OR x ∈ B}.

Example − If A = {10, 11, 12, 13} and B = {13, 14, 15}, then A ∪ B = {10, 11, 12, 13, 14, 15}. (The common element occurs only once)

Set Intersection

The intersection of sets A and B (denoted by A ∩ B) is the set of elements which are in both A and B. Hence, A ∩ B = {x | x ∈ A AND x ∈ B}.

Example − If A = {11, 12, 13} and B = {13, 14, 15}, then A ∩ B = {13}.

Set Difference

The set difference of sets A and B (denoted by A − B) is the set of elements which are only in A but not in B. Hence, A − B = {x | x ∈ A AND x ∉ B}.

Example − If A = {10, 11, 12, 13} and B = {13, 14, 15}, then (A− B) = {10, 11, 12} and (B − A) = {14, 15}. Here, we can see (A − B) ≠ (B − A)

ALGORITHM:

Algorithm (a, b, n1, n2)

Where a,b are two arrays of Set A & set B ,n1& n2 are the elements of two respective sets.

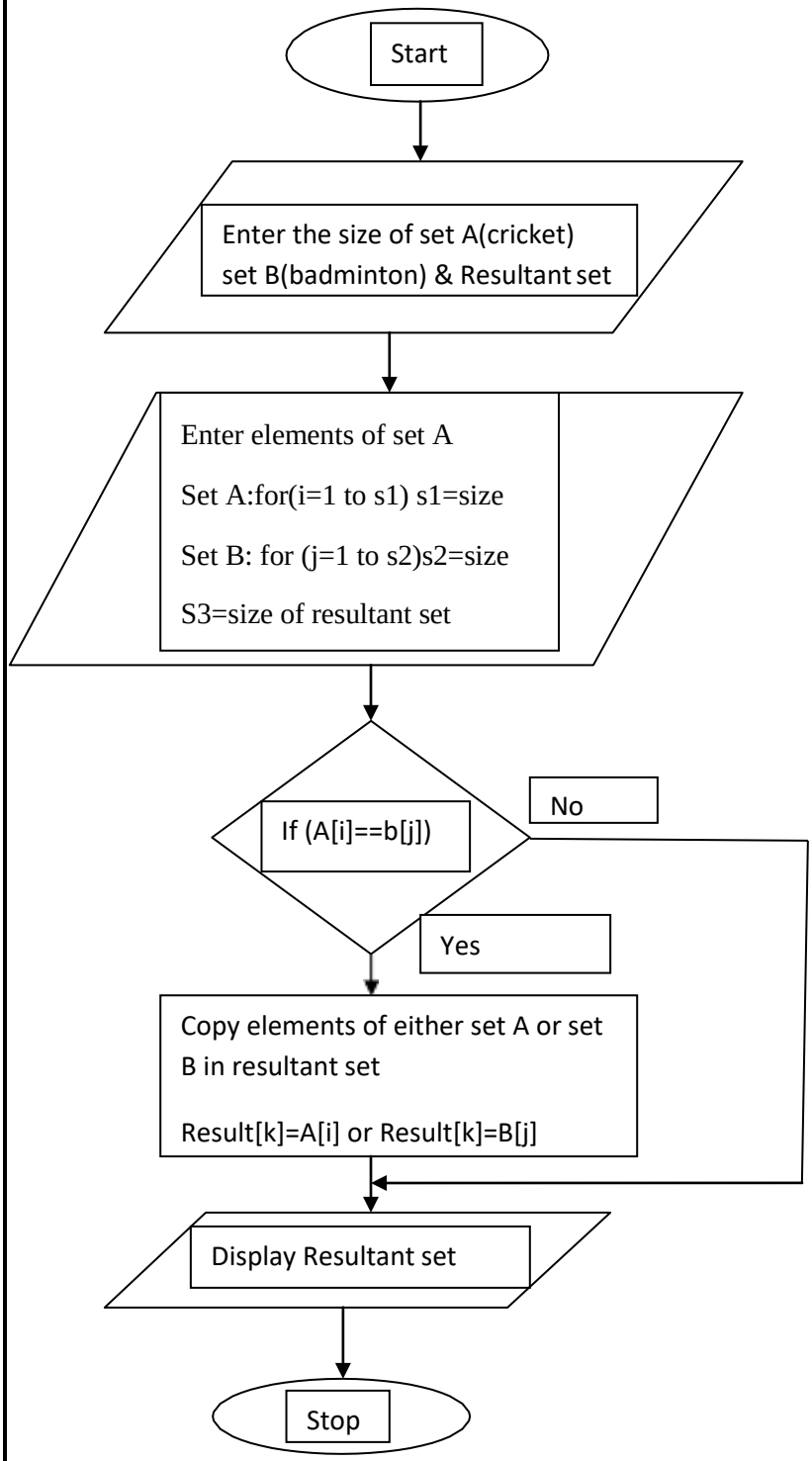
```
 for i=1 to n1 do
{
c[k]=a[i];
k++;
}
For i=1 to n2 do
{
Flag=1;
For j=1 to n1 do
{
 If(b[i]==a[j]) then
Flag=0;
Break;
}
If (flag==1) then
{
c[k]=b[i];
k++;
}
Write “union of two sets is:{”;
For i=0 to k do
Write c[i];
Write “}”;
```

```
For i=1 to n1 do
For j=1 to n2 do
 If(a[i]==b[j]) then do
{
C[k]=a[i];
K++;
}
Write “intersection of set A & B is{”
For i=0 to k do
Write c[i];
Write”}”;
For i=0 to n1 do
Flag=1
For j=1 to n2 do
{
If(a[i]==b[j])
 Flag=0;
}
If (flag==1)
{
C[k]=a[i];
K++;
}
Write “Difference A-B is{“
For i=0 to k do
Write c[i];
```

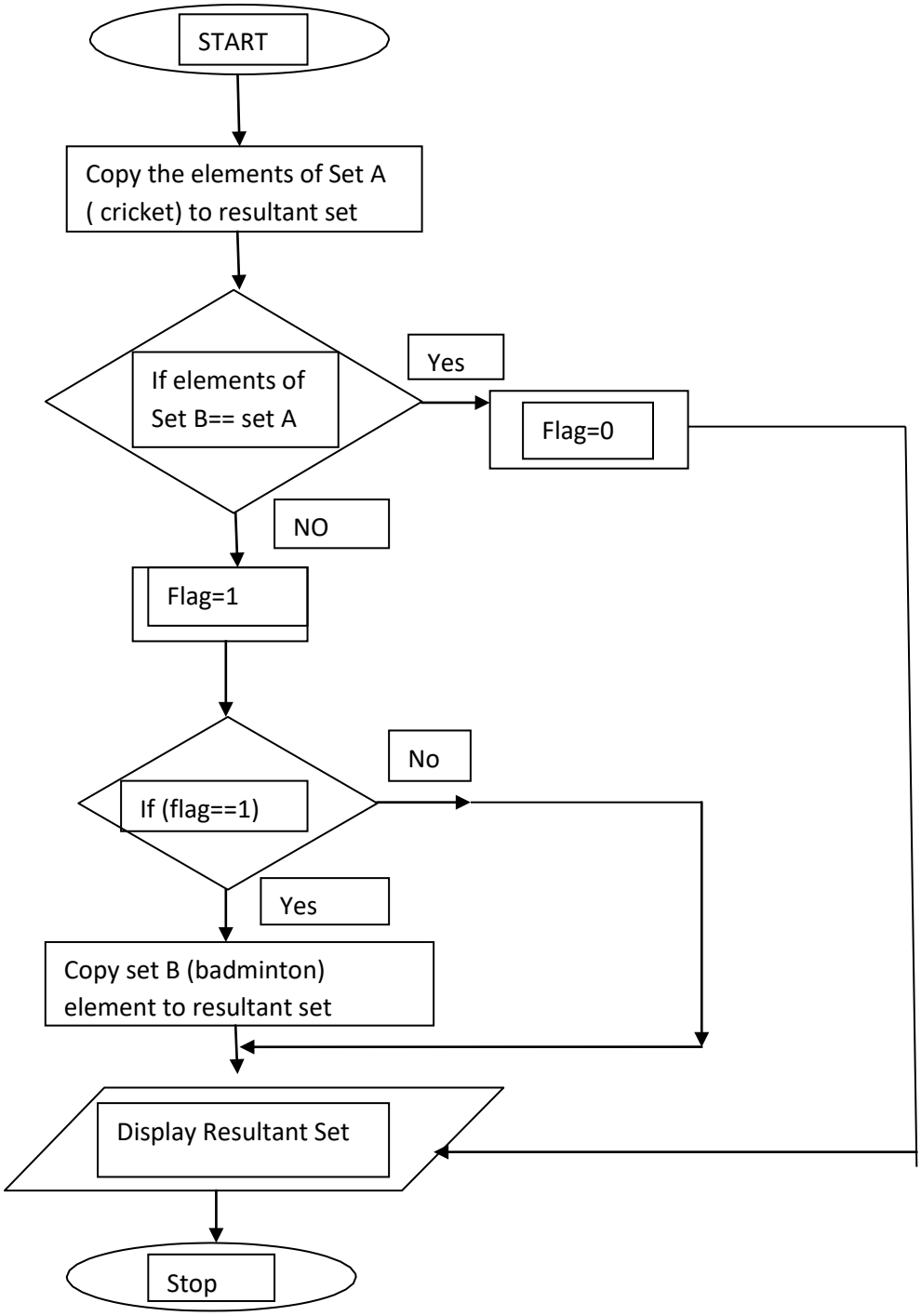
Write "}"

FLOW CHART:

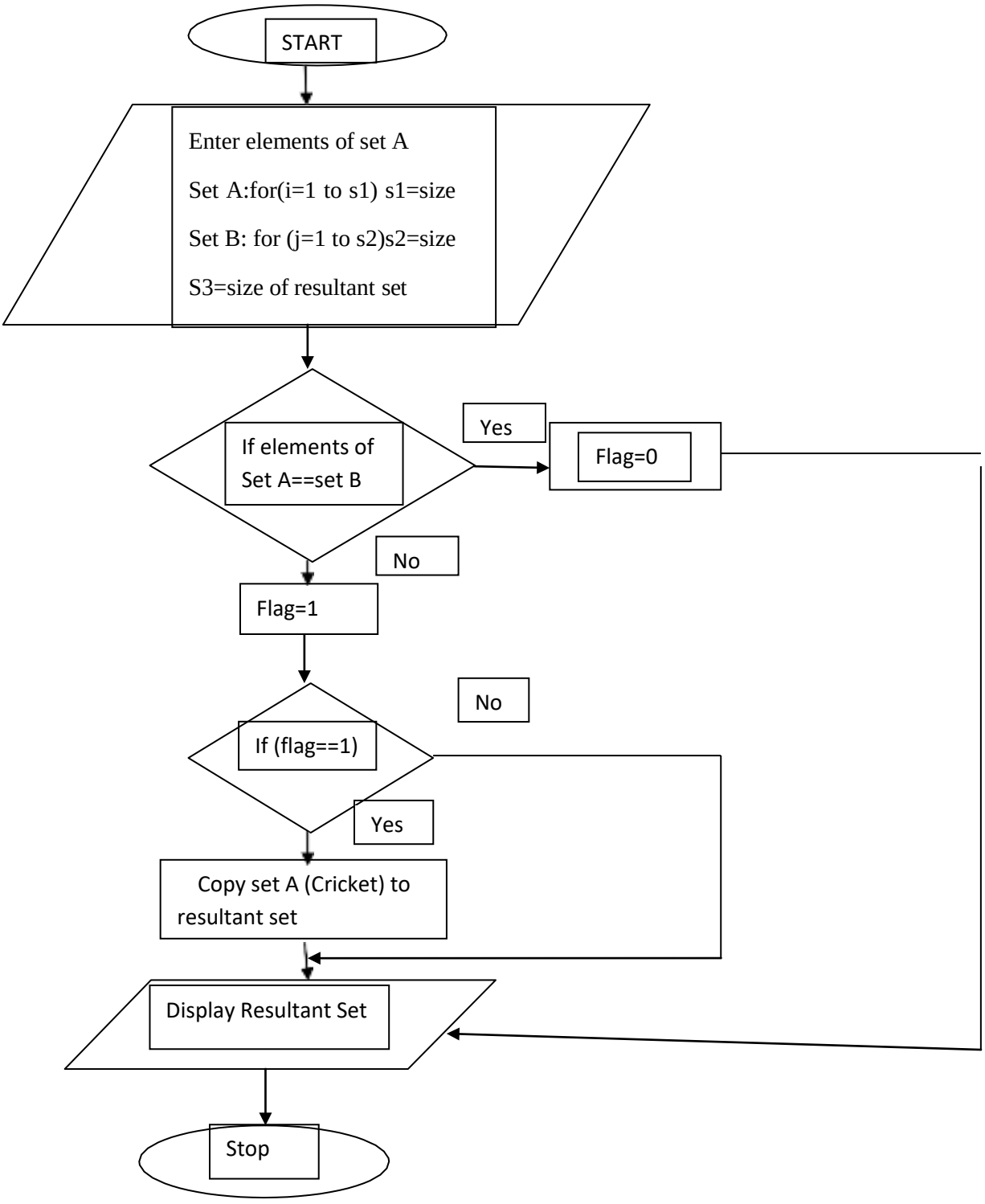
Intersection:



Union:


START
Copy the elements of Set A (cricket) to resultant set
If elements of Set B== set A
Yes
Flag=0
NO
Flag=1
If (flag==1)
No
Yes
Copy set B (badminton) element to resultant set
Display Resultant Set
Stop

Difference:


START
Enter elements of set A
Set A:for(i=1 to s1) s1=size
Set B: for (j=1 to s2)s2=size
S3=size of resultant set
If elements of Set A==set B
Yes
Flag=0
No
Flag=1
No
If (flag==1)
Yes
Copy set A (Cricket) to resultant set
Display Resultant Set
Stop

TEST CASES:

Sr.No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Enter the duplicate number	{1,2,2,3}	Message to be displayed duplicate elements.	Message displayed	Pass
2	ID2	Enter the negative roll number	{-1,2,3}	Message to be displayed negative elements.	Message displayed	Fail
3	ID3	Enter two sets	{1,2,3} {1,4,5}	{1,2,3,4,5} Union	{1,2,3,1,4,5}	Fail
4	ID4	Enter two sets	{1,2,3} {1,4,5}	{2,3} Difference A-B	{1,2,3}	Fail

CONCLUSION: Successfully implemented all set operation.

FAQ:

1. Write some real life applications of an array.
2. What is the need to use an array?
3. Analyze the necessity to use loops in the programming.
4. Elaborate the properties of Python language.
5. Justify the use of flag in your program?
6. Analyze time complexity of any one algorithm using step count method.
7. Discuss asymptotic notations with at least one example.
8. Elaborate the memory locations of an array elements with one example.
9. Justify statement: At run time, I cannot add any element in an array.
10. Compare between static memory allocation and dynamic memory allocation.

ASSIGNMENT NO: 2

TITLE: Implement array of structure for handling student record

OBJECTIVE: To study & implement array of structure for handling student record

PROBLEM STATEMENT:

Write a Python program to store marks scored in subject "Fundamental of Data Structure" by N students in the class. Write functions to compute following:
a) The average score of class
b) Highest score and lowest score of class
c) Count of students who were absent for the test
d) Display mark with highest frequency

OUTCOME: Student will be able to use array of structure for handling student record

THEORY: Structure is a collection of different data types which are grouped together and each element in a structure is called member.

- If you want to access structure members in C, structure variable should be declared.
- Many structure variables can be declared for same structure and memory will be allocated for each separately.
- It is a best practice to initialize a structure to null while declaring, if we don't assign any values to structure members.

Defining a Structure

To define a structure, you must use the **struct** statement. The **struct** statement defines a new data type, with more than one member. The format of the **struct** statement is as follows −

Syntax : struct structure_name

{

//Statements

};

Example of Structure: struct Student

{ char[20] name;

int age , rollno; } ;

Here the struct Book declares a structure to hold the details of book which consists of three data fields, namely name, price and pages. These fields are called structure elements or members. Each member can have different data type, like in this case, name is of char type and price is of int type etc. Book is the name of the structure and is called structure tag.

Declaring Structure Variables

It is possible to declare variables of a structure, after the structure is defined. Structure variables can be declared in following two ways.

1) Declaring Structure variables separately

```
struct Student

{ char[20] name;

 int age;

 int rollno; } ;

struct Student S1 , S2; //declaring variables of Student
```

2) Declaring Structure Variables with Structure definition

```
struct Student

{ char[20] name;

 int age;

 int rollno;

} S1, S2 ;
```

Here S1 and S2 are variables of structure Student. However this approach is not much recommended.

Accessing Structure Members

Structure member has no meaning independently. In order to assign a value to a structure member, the member name must be linked with the structure variable using dot . operator also called period or member access operator.

```
S1.rollno=101;      //S1 is variable of student type and rollno is member of student
```

Structure Initialization

Like any other data type, structure variable can also be initialized at compile time.

struct Student S1 = { “abcd”,20,101 }; //initialization

or

struct Student S1;

S1.name=”abcd”; //initialization of each member separately

S1.age = 20;

S1.rollno=101;

Array of Structure

We can also declare an array of structure. Each element of the array representing a structure variable. Example : struct Student S[5];

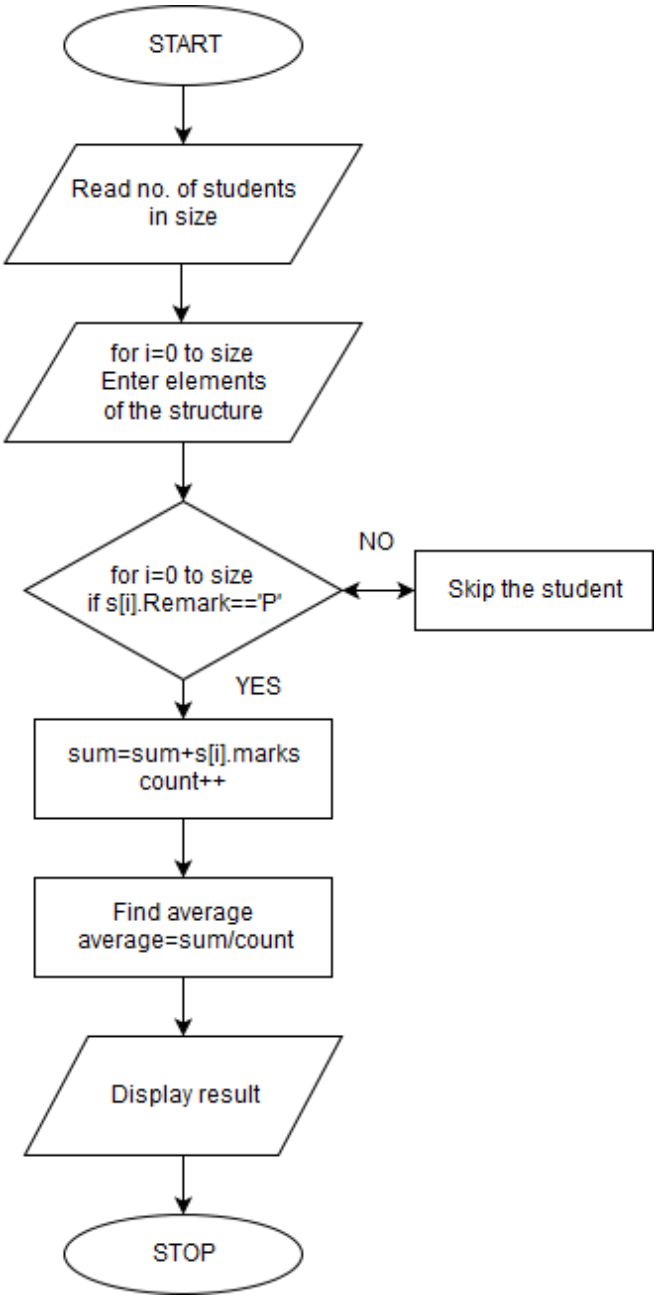
Algorithm:

1. Declare a structure with members : roll_no,marks & remark
2. Let s[20] be the array of structure,n be the no. of students in class
3. Read size in n
4. for i=0 to n do
   Read roll_no,marks and remark
5. int sum=0;
6. for i=0 to n do //average score of students
   sum=sum+s[i].marks;
7. float avg=sum/n;
8. min=max=s[0].marks; // highest and lowest score
9. for i=0 to n do
   if(max<s[i].marks) then do
   max=st[i].marks;
   if(min>st[i].marks) then do
   min=st[i].marks;
   Write min and max;
10. for i=0 to n do //students who were absent
    if(s[i].remark=='A'|| s[i].remark=='a') then do
    Write s[i].roll_no;
11. for i=0 to n do //marks scored by most of the students
    count=0;
    for j=0 to n do
    if(s[i].marks==s[j].marks) then do
    {
    count++;
    result[i]=count;

}

12. find maximum out of result[] //step 9
13. for i=0 to n do
    if(max==result[i]) then do
    Write s[i].roll_no;

FLOW CHART:



TEST CASES:

Sr.No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Enter the negative size of structure array	-5	Message to be displayed negative elements.	Message displayed	Fail
2	ID2	If all remarks are absent	For i=0 to size S[i].remark='a'	Message to be displayed all students absent for test	Message to be displayed all students absent for test	Pass

CONCLUSION: Successfully implemented array of structures.

FAQ:

1. Compare array with structures.
2. Elaborate different ways to declare a structure variable.
3. Design one application using structures.
4. Interpret the advantages of structures over arrays.
5. Elaborate at least 5 advance symbols used in the flowchart.
6. Analyze the time complexity of your program.
7. Justify to need of using array of structures in your program.
8. Apply the use of pointer operator instead of using dot operator in your program and write at least one function with the same.
9. Test the possibility to display marks with lowest priority. Mark the changes in your program.
10. Design the function to display the count of students those were present for the exam.

GROUP A

ASSIGNMENT NO: 3(A)

TITLE: Program for magic square.

OBJECTIVE: To study Multidimensional Array & implement magic square.

PROBLEM STATEMENT: Write a python program for magic square. A magic square is an n * n Matrix of the integers 1 to n2 such that the sum of each row, column, and diagonal is the same.

OUTCOME: Student will be able to use array concepts for implementing magic square.

THEORY: Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. The simplest form of multidimensional array is the two-dimensional array. 2D arrays are generally known as matrix. A two dimensional array is a grid of rows and columns.

Syntax: To declare a two-dimensional integer array of size [x][y]

```
type arrayName [ x ][ y ];
```

A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array a, which contains three rows and four columns can be shown as follows −

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array a is identified by an element name of the forma[i][j], where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns. The nested braces, which indicate the intended row, are optional.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
```

```
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
OR

int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Magic Square

A magic square of order n is an arrangement of n^2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. A magic square contains the integers from 1 to n^2.

The constant sum in every row, column and diagonal is called the magic constant or magic sum, M. The magic constant of a normal magic square depends only on n and has the following value:
$M = n(n^2+1)/2$

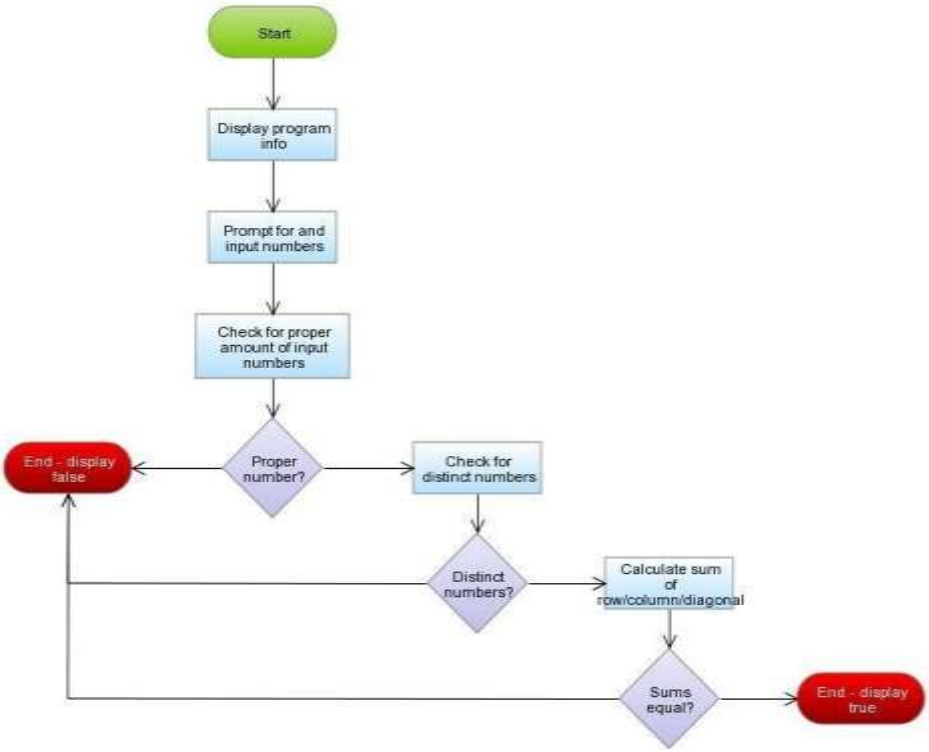
```
Magic Square of size 3
----------------------
  2   7   6
  9   5   1
  4   3   8
Sum in each row & each column = 3*(3^2+1)/2 = 15
```

ALGORITHM:

```
1.  Input   size of matrix and values in [rows][columns]
2.  Output  Message to display if there exist magic square
3.  //For diagonal elements
4.  for i =1 to rows [size]
5.      for j = 1 to columns [size]
6.        If(row==column)
7.              sum = sum + matrix[row][column];
8.   //For Rows
9.   for i =1 to rows [size]
10.     for j = 1 to columns [size]
11.             sum_rows = sum_rows + matrix[row][column];
12.     if sum_diagonal ==sum_rows
13.             Set flag to 1
14. // For Columns
15. for i =1 to rows [size]
16.     for j = 1 to columns [size]
17.             sum_columns =sum_columns + matrix[column][row];
18.     if sum_diagonal==sum_columns
19.             Set flag to 1
20. //For Displaying message
21.  if flag==1
```

```
22.      Display message “Magic square”
23. else
24.      Display message “Not Magic square”
```

FLOW CHART:



TEST CASES:

Sr.No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Enter matrix size and positive values in rows	{ 2 7 6 9 5 1 4 3 8}	Message to be displayed “is a Magic Square”.	Message displayed	Pass
2	ID1	Enter matrix size and positive values in rows	{ 2 3 6 9 1 1 4 3 8}	Message to be displayed “is not a Magic Square”.	Message displayed	Pass
3	ID2	Enter the negative roll number	{ 2 7 6 9 5 1 4 3 -8}	Message to be displayed negative elements.	Message displayed	Fail

CONCLUSION: Successfully implemented magic square matrix.

FAQ:

1. Elaborate the concept of Multidimensional Array?
2. Discuss the syntax for declaration and initialization of Two-Dimensional Arrays
3. Elaborate dynamic allocation in a multidimensional array?
4. What will happen if in a python program you assign a value to an array element whose subscript exceeds the size of array?
5. Find the output
6.
```
int main()
{
        int a[5] = {5, 1, 15, 20, 25};
        int i, j, m;
        i = ++a[1];
        j = a[1]++;
        m = a[i++];
        printf("%d, %d, %d", i, j, m); return 0;
}
```
7. Analyze the time complexity of your program.
8. Analyze the space complexity of your program.
9. Illustrate the memory allocation using row major of your example.
10. Illustrate the memory allocation using column major of your example.

GROUP A

ASSIGNMENT NO: 3(B)

TITLE: Program for String operation.

OBJECTIVE: To study & implement string operation.

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: Student will be able to use array concepts for implementing string operations.

THEORY: Strings are nothing but array of characters ended with null character ('\0').This null character indicates the end of the string. Strings are always enclosed by double quotes. Whereas, character is enclosed by single quotes.

Eg: char string[20] = { 'f' , 'r' , 'e' , 's' , 'h' , '2' , 'r' , 'e' , 'f' , 'r' , 'e' , 's' , 'h' , '\0'}; (or)

char string[20] = "Hello World"; (or)

char string [] = "Hello World ";

String functions:

String.h header file supports all the string functions. All the string functions are given below.

1. strcat( ) function is concatenates two given strings. It concatenates source string at the end of destination string. Syntax for strcat( ) function is given below.

   strcat (destination string , source string);

As you know, each string in C is ended up with null character ('\0').In strcat( ) operation, null character of destination string is overwritten by source string's first character and null character is added at the end of new destination string which is created after strcat( ) operation.

2. strcpy( ) function copies contents of one string into another string. Syntax for strcpy function is given below.

   strcpy (destination string, source string);

Example: strcpy (str1, str2) – It copies contents of str2 into str1.

strcpy (str2, str1) – It copies contents of str1 into str2.

If destination string length is less than source string, entire source string value won't be copied into destination string. For example, consider destination string length is 20 and source string

length is 30. Then, only 20 characters from source string will be copied into destination string and remaining 10 characters won't be copied and will be truncated.

3. strstr( ) function returns pointer to the first occurrence of the string in a given string. Syntax for strstr( ) function is given below.

   strstr(str1,str2);

strstr( ) function is used to locate first occurrence of the string "test" in the string "This is a test string for testing". Pointer is returned at first occurrence of the string "test".

4. strcmp( ) function in C compares two given strings and returns zero if they are same. If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value. Syntax for strcmp( ) function is given below.

   strcmp (str1, str2);

   strcmp( ) function is case sensitive. i.e, "A" and "a" are treated as different characters.

5. strlen( ) function in C gives the length of the given string. Syntax for strlen( ) function is given below.

   strlen (str1);

strlen( ) function counts the number of characters in a given string and returns the integer value.It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

ALGORITHM: Algorithm (str1, str2)

Input: Two strings

Begin

consider str1 & str2 are the two strings.

//string length

for i=0 to str1[i]!='\0' ;

print length of string(i)

//string copy

for i=0 to str1[i]!='\0'

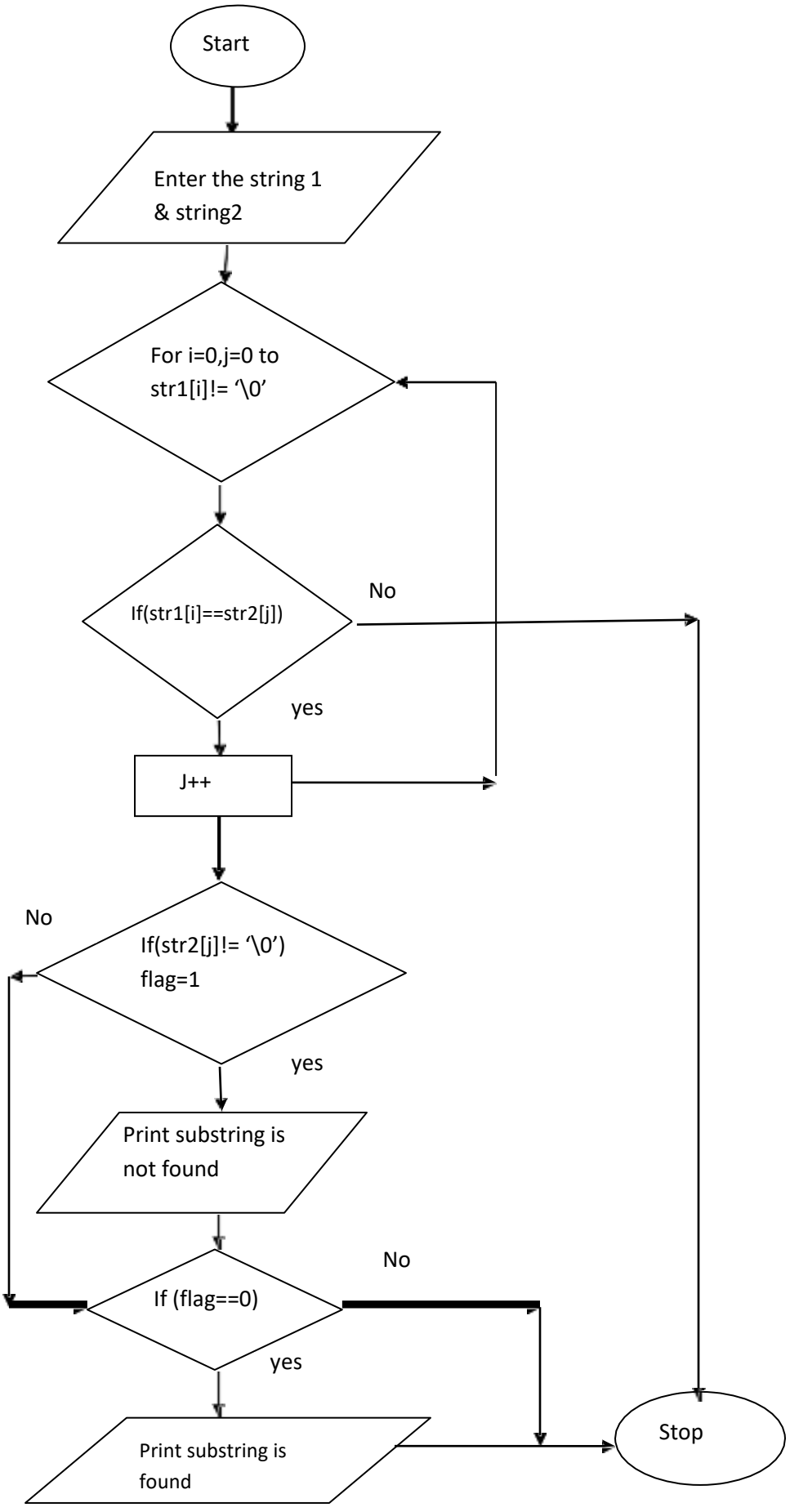
str2=str1

for i=0 to str2[i]!='\0'

print string (str2)

```
//string concatenation
for i=0 to str1[i]! =‘\0’
        for j=0 to str2[j]!= ‘\0’
        str1[i]=str2[j]
str2[j]=‘\0’
print string (str1)
//string equal
j=0
For i=0 to str1[i]!=‘\0’
        If(str1[i]==str2[j])
                j++
        else
                Strings are not equal
//reverse of string
For i=0 to str1[i]! =‘\0’
        l= --i
for j=l to j>=0
        print string (str1)
//substring of string
j=0
For i=0 to str1[i]!=‘\0’
        If (str2[j]==str1[i]) j++
        if(str2[j]!=‘\0’)
        substring not found
        else
        substring is found
        }
End
```

FLOW CHART:


Start
Enter the string 1
& string2
For i=0,j=0 to
str1[i]!= '\0'
If(str1[i]==str2[j])
No
yes
J++
No
If(str2[j]!= '\0')
flag=1
yes
Print substring is
not found
No
If (flag==0)
yes
Print substring is
found
Stop

TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	If character array not there	Char c;	It must return the complete string	But it return only first character from string	Fail
2	ID2	Length is in the format of integer	Int l; L=strlen(str 1)	It must return the length in the form of integer	It is not printing in integer format	Fail
3	ID3	At the time of concatenation after completion of first string it must append next string	Strcat(str1,st r2)	First string1will read then string2 & then concat the string.Display the string	Yes it displayed concatenated string(str1 then str2)	Pass

CONCLUSION: Successfully implemented all string operation.

FAQ:

1. Compare character array & string?
2. What will be output when you will execute following code

```
#include<stdio.h>
#include<string.h>
void main()
{ char arr[11]="The African Queen";
cout<<”The String is=%s<<arr; }
```

3. Elaborate the syntax of string copy.
4. Analyze the time complexity of your program.
5. Analyze the space complexity of your program.
6. What will be the output of

```
int main()
{
printf(“Hello”,”World\n”);
return 0;
}
```

GROUP: A
ASSIGNMENT NO: 3(C)

TITLE: To study two dimensional arrays as a data structure and be able to perform different operations on 2D arrays.

OBJECTIVE: To study and implement array operation.

OUTCOME: Student will be able to use array data structure & its operations.

PROBLEM STATEMENT:
Write a python program to compute following computation on matrix:
a) Addition of two matrices
b) Subtraction of two matrices
c) Multiplication of two matrices
d) Transpose of a matrix

THEORY:
What are Multi-dimensional arrays?
C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration −
type name[size1][size2]...[sizeN];
For example, the following declaration creates a three dimensional integer array −
int threedim[5][10][4];
Two-dimensional Arrays
The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows −
type arrayName [x][y];
Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows −

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array **a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.
Initializing Two-Dimensional Arrays
Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}  /* initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is

equivalent to the previous example −
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

Accessing Two-Dimensional Array Elements

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example −
int val = a[2][3];
The above statement will take the 4th element from the 3rd row of the array.

```
#include <stdio.h>
int main () {
   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;
   /* output each array element's value */
   for ( i = 0; i < 5; i++ )
   {
      for ( j = 0; j < 2; j++ )
      {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

Implementation:

Inputs:

1. Elements of First 3×3 Matrix
2. Elements of Second 3×3 Matrix

Algorithms:

Addition of two matrices

1. Let A[3][3] and B[3][3] be input matrices and C[3][3] be resultant Matrix
2. for i:=0 to 3
   begin
   for j:=0 to 3
   begin
   C[i][j]←A[i][j]+B[i][j]
   end
   end
3. Display Resultant Matrix
   for i:=0 to 3
   begin

   for j:=0 to 3

   begin

   print C[i][j]

```
    end
  end
```

Subtraction of two matrices

1. Let A[3][3] and B[3][3] be input matrices and C[3][3] be resultant Matrix
2. for i:=0 to 3

```
begin
    for j:=0 to 3
    begin
        C[i][j]←A[i][j]-B[i][j]
    end
end
```

3. Display Resultant Matrix

```
for i:=0 to 3
begin
    for j:=0 to 3
    begin
        print C[i][j]
    end
 end
```

Mutltiplication of two matrices

1. Let A[3][3] and B[3][3] be input matrices and C[3][3] be resultant Matrix
2. for i:=0 to 3

```
begin
    for j:=0 to 3
    begin
        C[i][j]=0
        for k:=0 to 3
        begin
            C[i][j]←C[i][j]+A[i][k]*B[k][j]
        end
    end
 end
```

3. Display Resultant Matrix

```
for i:=0 to 3

begin

    for j:=0 to 3

    begin

        print C[i][j]

    end
 end
```

Transpose of a Matrix

1. Let A[3][3] be a input matrix and C[3][3] be resultant Matrix
2. for i:=0 to 3

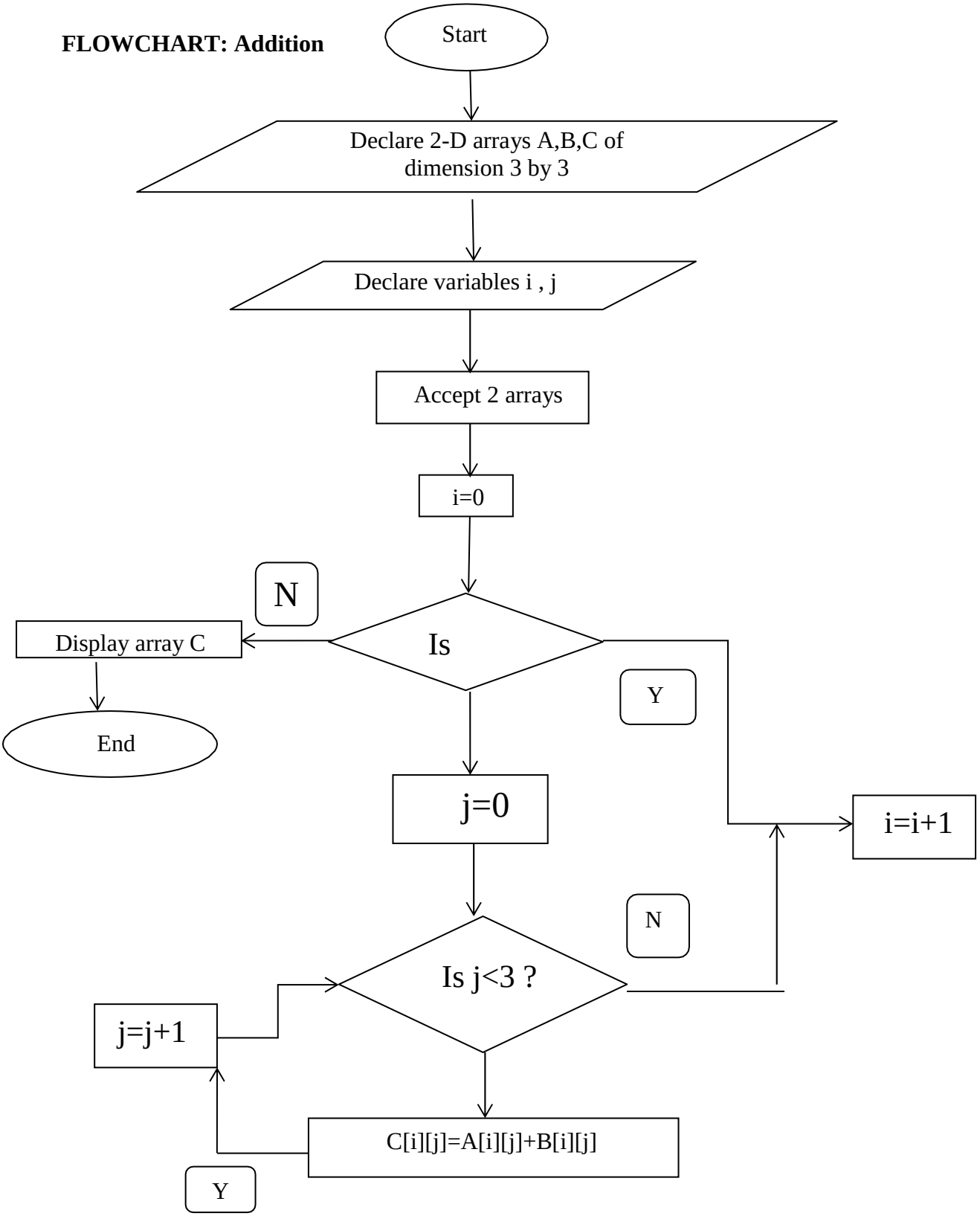
```
begin

    for j:=0 to 3
```

```
        begin
                C[i][j]←A[j][i]
        end
    end
```

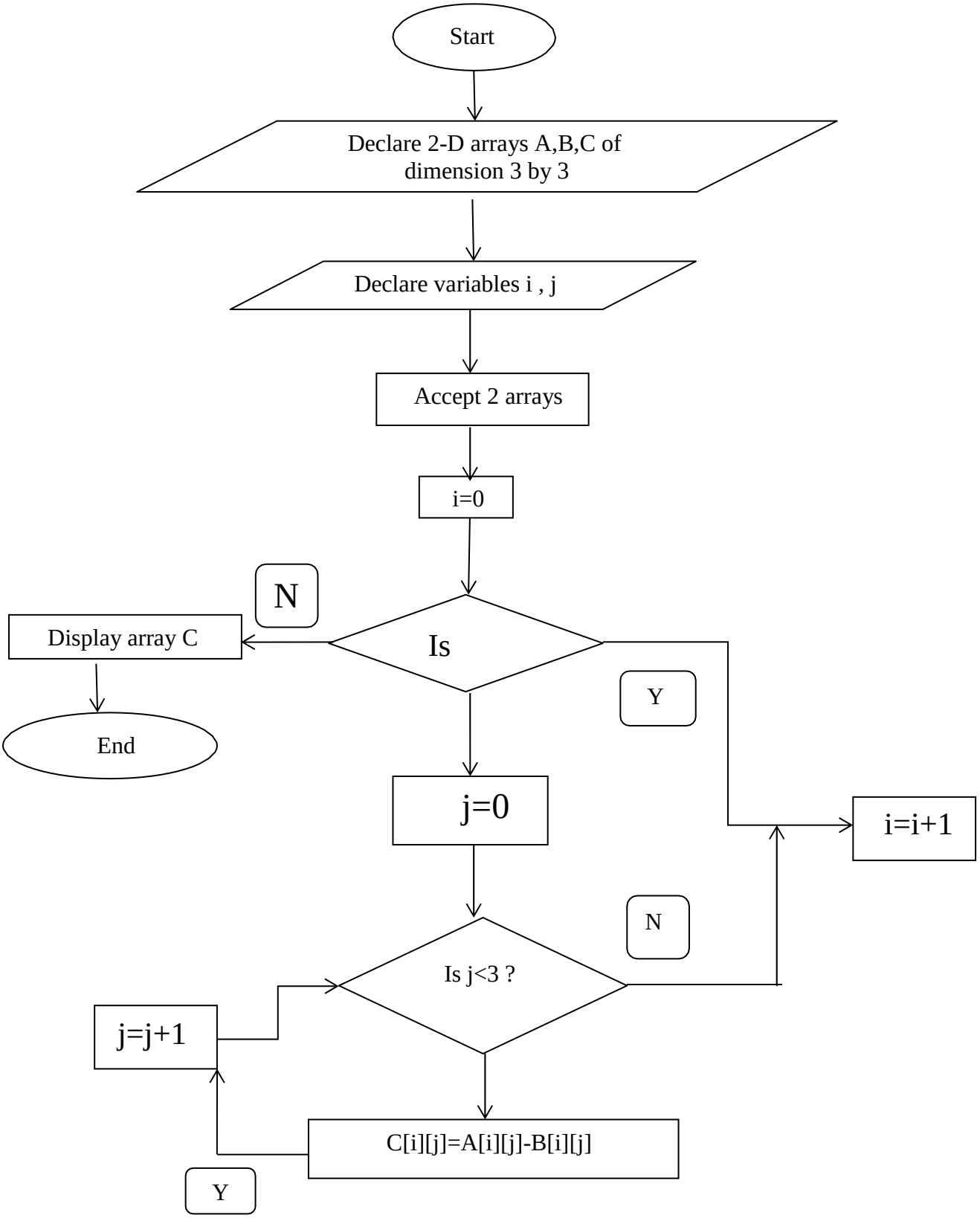
Display Resultant Matrix

```
    for i:=0 to 3
    begin
        for j:=0 to 3
        begin
                print C[i][j]
        end
     end
```

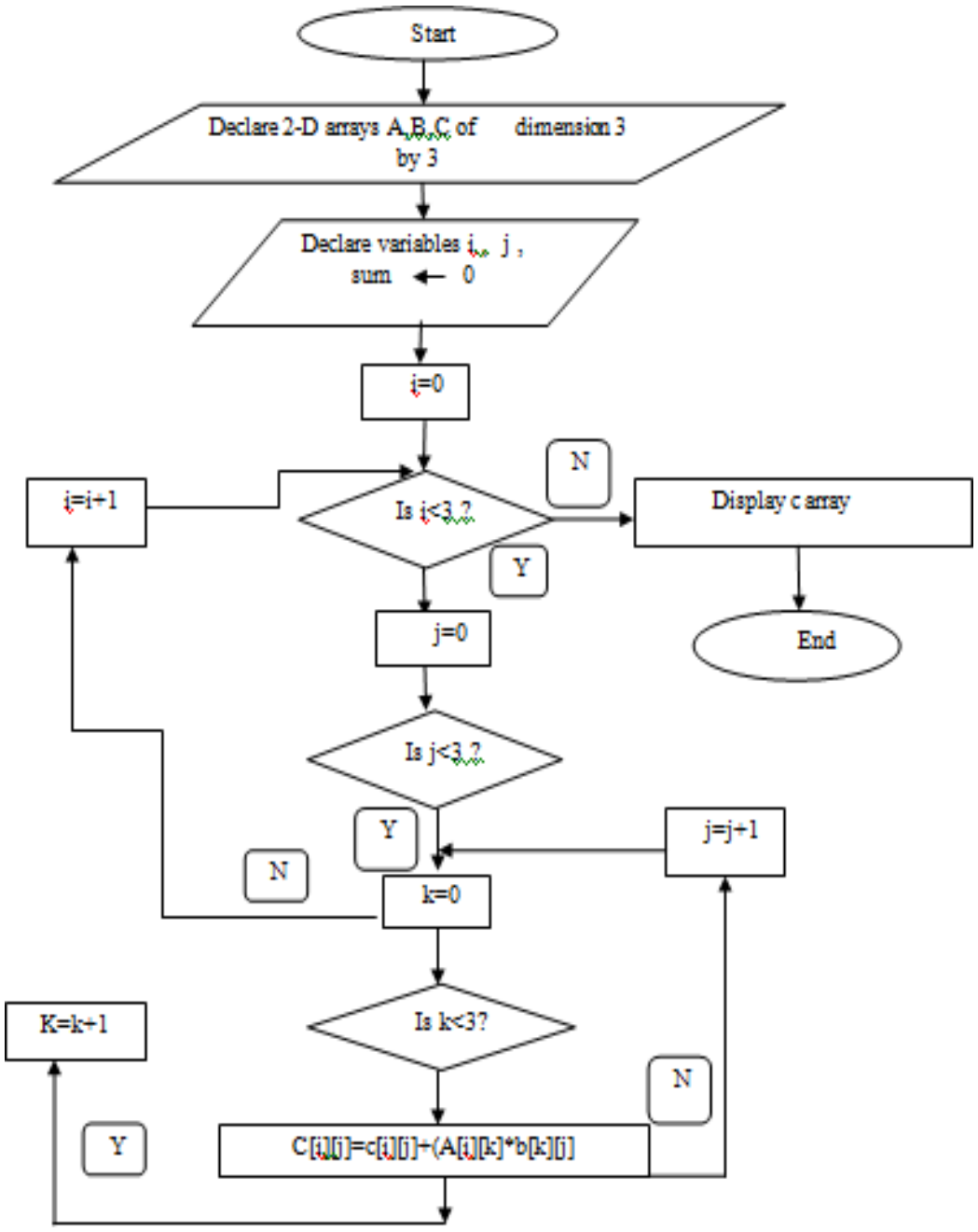
FLOWCHART: Addition



Flowchart: Subtraction



Flowchart: Multiplication



Test Cases:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Input Arrays	Matrices A and B	Addition of A and B	Addition of A and B	Pass
2	ID2	Input Arrays	Matrices A and B	Subtraction A from B	Subtraction A from B	Pass
3	ID3	Input Arrays	Matrices A and B	Multiplication of A and B	Multiplication of A and B	Pass

Conclusion: Thus, we have studied two dimensional arrays and implemented different operations on matrices successfully.

FAQ:

1. Analyze the time complexity of your program.
2. Elaborate the matrix multiplication logic with example.
3. Analyze space complexity of your program.
4. Compare the time complexity required by addition, subtraction and multiplication.
5. Compare the time complexity required if the matrix is stored in row major fashion with matrix is stored in column major fashion.
6. Elaborate three dimensional matrix with one example.
7. Design any real life application of two/three dimensional matrix.
8. Compare the space complexity required by addition, subtraction and multiplication.
9. Consider an integer array int arr[4][5].If base address is 1020 find the address of the element arr[3][4] with row major representation.
10. Consider an integer array int arr[4][5].If base address is 1020 find the address of the element arr[3][4] with column major representation.

GROUP: B

ASSIGNMENT NO: 4

TITLE: Implementing Insertion sort and Shell Sort

OBJECTIVE: To study sorting algorithms and its complexity

PROBLEM STATEMENT: Write a python program to store second year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using
a) Insertion sort b) Shell Sort and display top five scores.

OUTCOME: Student will understand and implement sorting of data

THEORY:

Insertion sort:

- This is a in-place comparison based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted.
- A element which is to be inserted in this sorted sub-list, has to find its appropriate place and insert it there. Hence the name insertion sort.
- The array is searched sequentially and unsorted items are moved and inserted into sorted sub-list (in the same array).
- 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.

Complexity:

Worst case performance	O(n2) comparisons, swaps
Best case performance	O(n) comparisons, O(1) swaps
Average case performance	O(n2) comparisons, swaps
Worst case space complexity	O(n) total, O(1) auxiliary

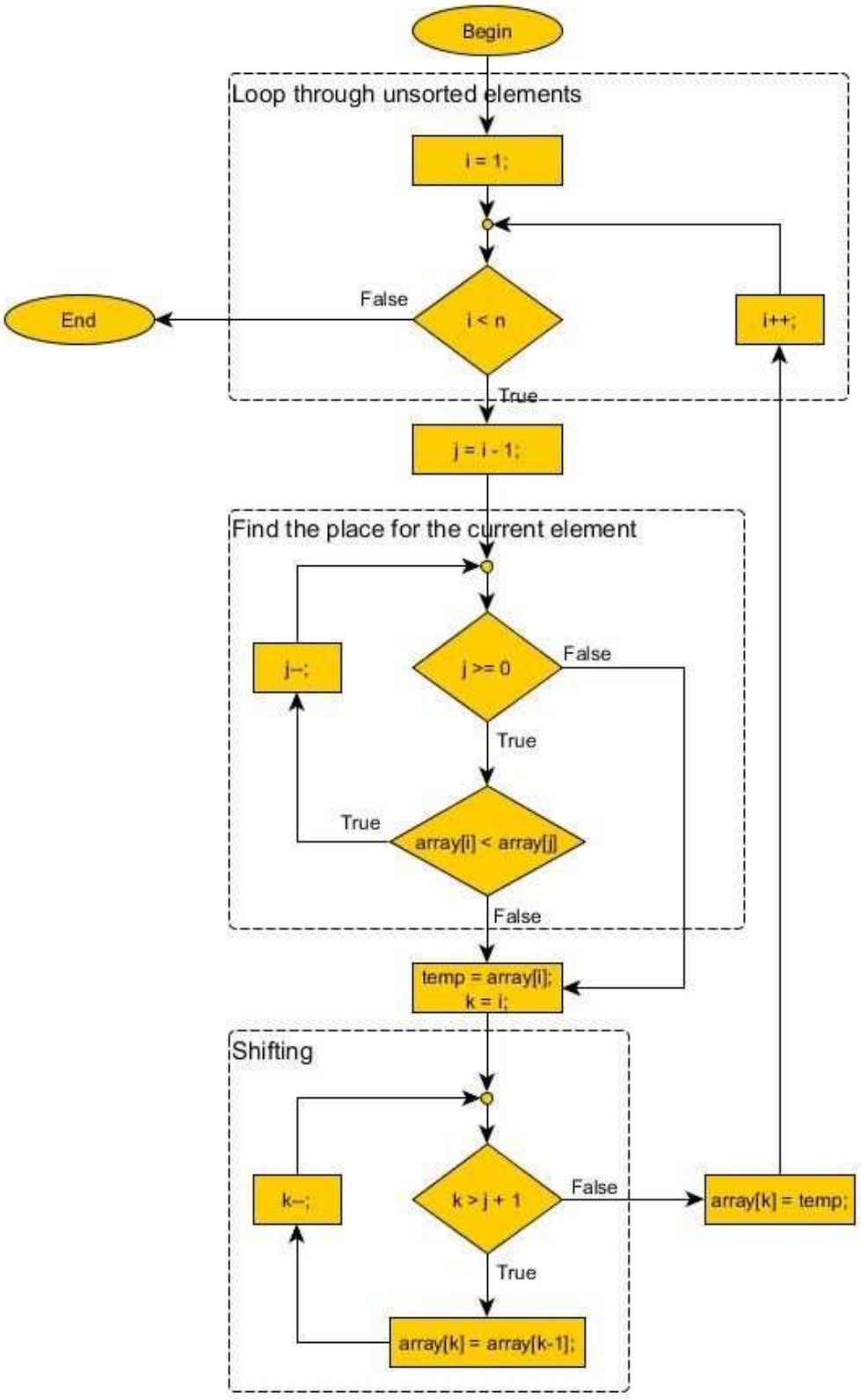
Algorithm

For the implementation of insertion sort we need two loops. The first will iterate through the unsorted part of the array. For each unsorted element the second loop will go through the sorted part and search for its correct location. This means that we need to nest the second loop into the first. After we find the correct place to insert the current element, we need to shift all remaining elements. For that purpose we need one more loop.

Input: Unsorted elements
Output: Sorted elements

Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted

FLOW CHART: Insertion Sort



Shell sort:

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort if smaller value is very far right and have to move to far left.

This algorithm uses insertion sort on widely spread elements first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval. This interval is calculated based on, Knuth's formula as − h = h * 3 + 1

where − h is interval with initial value 1

This algorithm is quite efficient for medium sized data sets as its average and worst case complexity are of O(n) where n are no. of items.

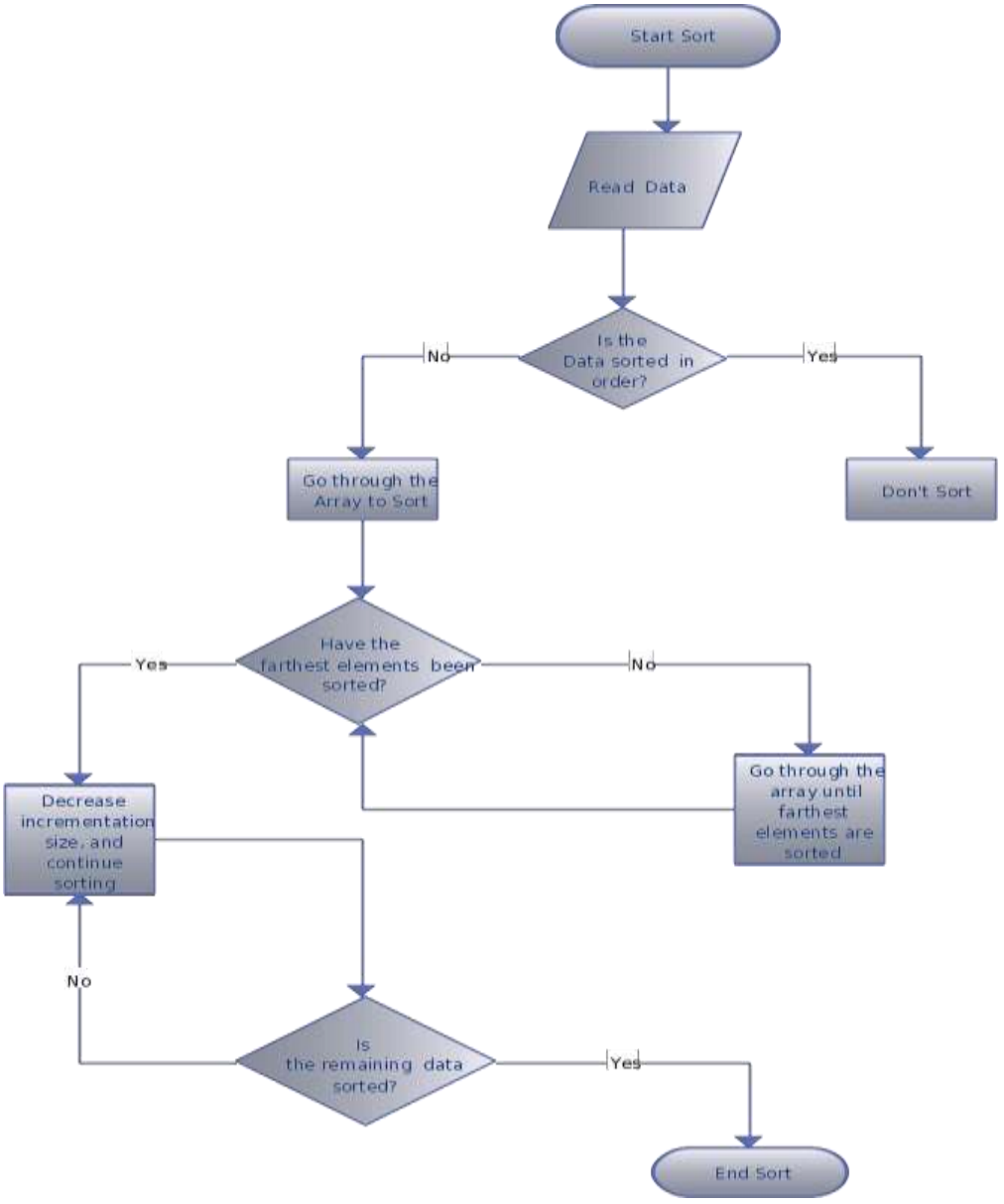
Complexity

Worst case performance	O(nlog2 2n) [1]
Best case performance	O(n log n)[2]
Worst case space complexity	O(n) total, O(1) auxiliary

Algorithm

Step 1 − Initialize the value of h
Step 2 − Divide the list into smaller sub-list of equal interval h
Step 3 − Sort these sub-lists using insertion sort
Step 3 − Repeat until complete list is sorted

FLOW CHART: Shell Sort



TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Apply Insertion Sort	{5, 1, 6, 2, 4, 3}	{1, 2, 3, 4, 5, 6}	{1, 2, 3, 4, 5, 6}	Pass
2	ID2	Apply Insertion Sort	{5, 1, 6, 2, 4, 3}	{1, 2, 3, 4, 5, 6}	{1, 6, 3, 2, 5, 4}	Fail
3	ID3	Apply Shell Sort	{5, 1, 6, 2, 4, 3}	{1, 2, 3, 4, 5, 6}	{1, 2, 3, 4, 5, 6}	Pass
4	ID4	Apply Shell Sort	{5, 1, 6, 2, 4, 3}	{1, 2, 3, 4, 5, 6}	{1, 6, 3, 2, 5, 4}	Fail

CONCLUSION: Successfully implemented shell and insertion Sort

FAQ:

1. Justify that Insertion sort better than Quick sort for small list of elements?
2. Analyze the time complexity of both functions.
3. Which of the following is not a non-comparison sort?
    a) Counting sort
    b) Bucket sort
    c) Radix sort
    d) Shell sort
4. If the given input array is sorted or nearly sorted, which of the following algorithm gives the best performance?
    a) Insertion sort
    b) Selection sort
    c) Quick sort
    d) Merge sort
5. Elaborate stable sorting algorithm with one example.
6. Analyze the space complexity of both functions.
7. Compare the time complexity of both sorts. Justify the best one among these two sorts.
8. Compare Shell sort with insertion sort.
9. Among all sorting techniques which sorts are not stable sorts? Justify your answer.
10. Elaborate the advantages of shell sort.

GROUP B

ASSIGNMENT NO: 05

TITLE: Implementing Selection Sort and Bubble sort

OBJECTIVE: To study sorting algorithms and its complexity

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

OUTCOME: Student will understand and implement sorting of data

THEORY:

Sorting:

- Refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order.
- Most common orders are numerical or lexicographical (also known as lexical order, dictionary order, alphabetical) order.

Bubble sort:

- 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 and worst case complexity are of $O(n^2)$ where n are no. of items.

Complexity

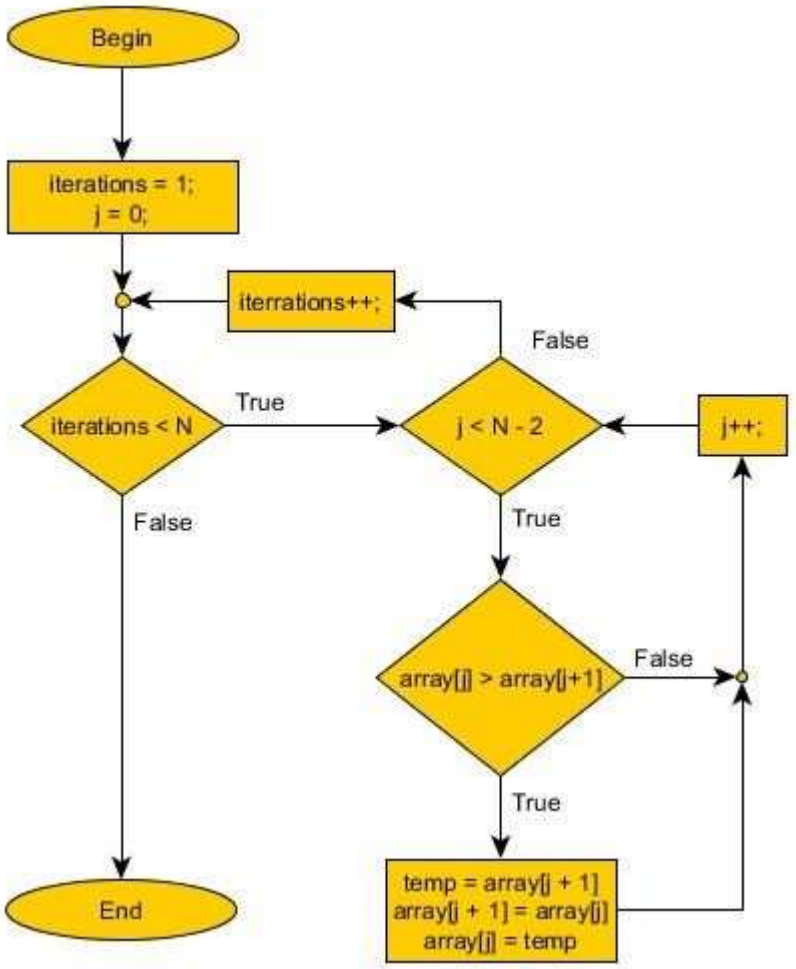
Worst case performance	O (n ^2)
Best case performance	O (n)
Average case performance	O (n^2)

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

Input: Unsorted elements
Output: Sorted elements

```
begin BubbleSort(list)

    for all elements of list if
        list[i] > list[i+1]
            swap(list[i], list[i+1])
        end if
    end for

    return list

end BubbleSort
```

FLOW CHART: Bubble Sort



Selection sort:

- This sorting algorithm is a in-place comparison based algorithm in which the list is divided into two parts, sorted part at left end and unsorted part at right end.
- Initially sorted part is empty and unsorted part is entire list.
- Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array.
- This process continues moving unsorted array boundary by one element to the right.
- This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n are no. of items.

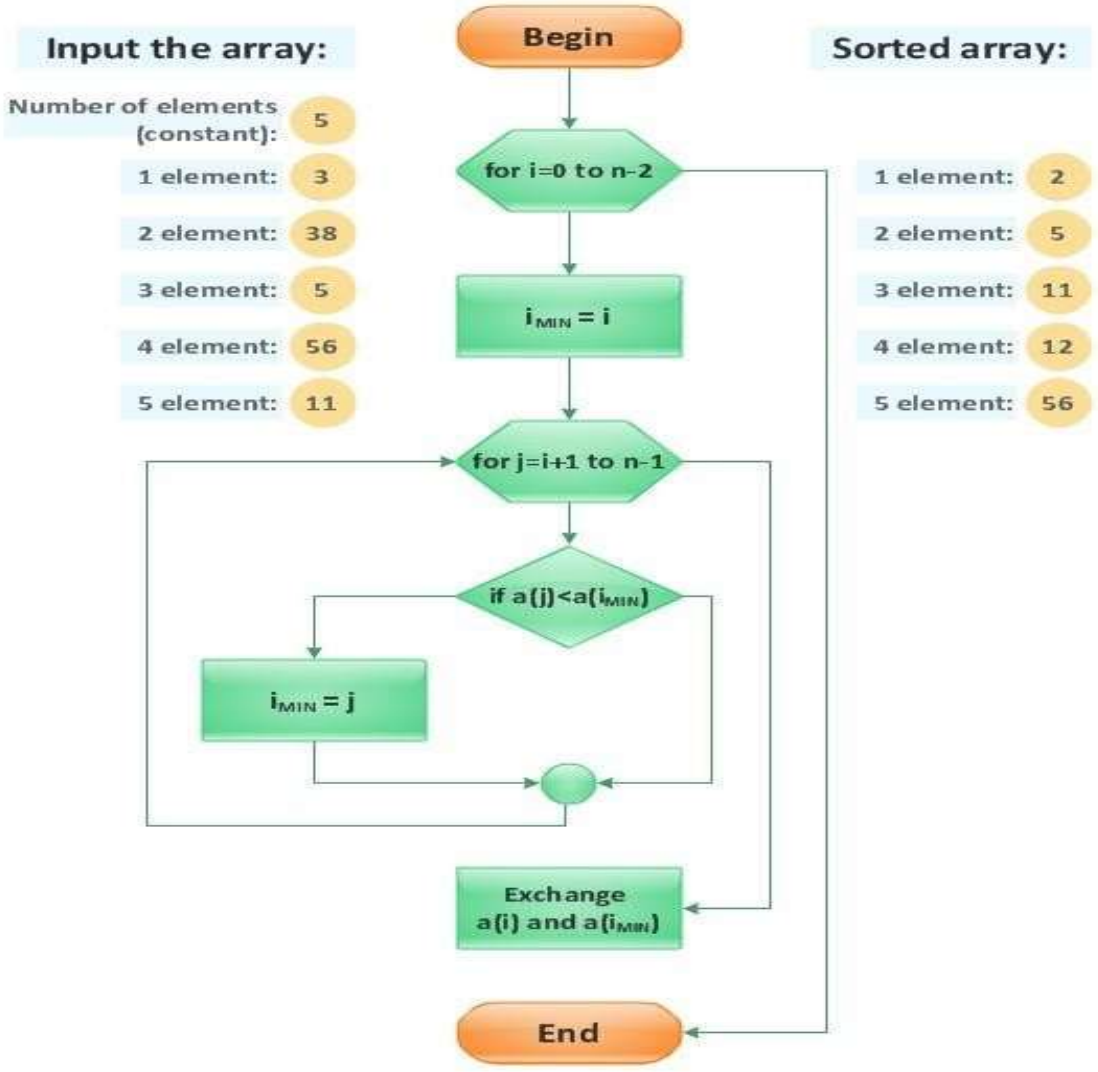
Complexity

Worst case performance O(n2)
Best case performance O(n2)
Average case performance O(n2)

Algorithm

Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list
Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted

FLOW CHART: Selection Sort



TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Apply Bubble sort	{5, 1, 6, 2, 4, 3}	{1, 2, 3, 4, 5, 6}	{1, 2, 3, 4, 5, 6}	Pass
2	ID2	Apply Bubble sort	{5, 1, 6, 2, 4, 3}	{1, 2, 3, 4, 5, 6}	{1, 6, 3, 2, 5, 4}	Fail
3	ID3	Apply Selection sort	{3, 6, 1, 8, 4, 5}	{1, 3, 4, 5, 6, 8}	{1, 3, 4, 5, 6, 8}	Pass

CONCLUSION: Successfully implemented Selection Sort and Bubble sort.

FAQ:

1. What is the max. number of comparisons that can take place when a bubble sort is implemented? Assume there are n elements in the array?
    a) (1/2)(n-1)
    b) (1/2)n(n-1)
    c) (1/4)n(n-1)
2. Analyze the worst case and best case time complexity of bubble sort consequently?
3. Analyze the space complexity of both programs.
4. Justify the statement: selection sort is more superior to bubble sort.
5. Compare these two sorts with respect to advantages and disadvantages.
6. How will you increase the efficiency of bubble sort?
7. Inspect the application where we cannot apply bubble sort in any real life example.
8. Analyze worst case time complexity for selection sort algorithm.
9. A sorting technique in which successive elements are selected in order and placed into their proper sorted positions is called _____
10. Which of the following sorting algorithms has the lowest worst-case complexity?
    a) Merge
    b) Quick
    c) Selection
    d) Bubble

GROUP B
ASSIGNMENT NO. 06

TITLE: Program for Quick Sort.

OBJECTIVE: To study & implement Quick Sort.

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 quick sort and display top five scores.

OUTCOME: Student will be able to use array concept for sorting.

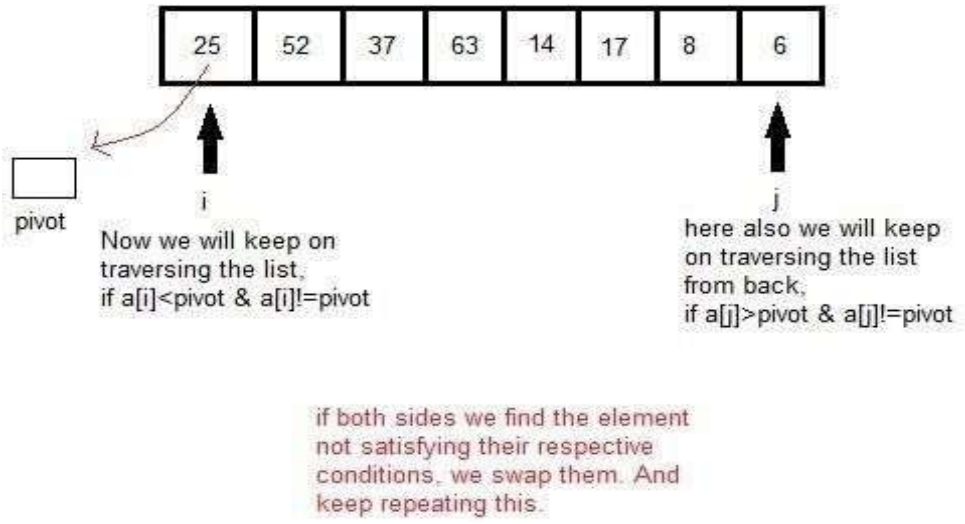
THEORY: Quick Sort, as the name suggests, sorts any list very quickly. Quick sort is not stable search, but it is very fast and requires very less additional space. It is based on the rule of **Divide and Conquer**(also called *partition-exchange sort*). This algorithm divides the list into three main parts :

1. Elements less than the Pivot element
2. Pivot element
3. Elements greater than the pivot element

In the list of elements, mentioned in below example, we have taken **25** as pivot. So after the first pass, the list will be changed like this.

6 8 17 14 **25** 63 37 52

Hence after the first pass, pivot will be set at its position, with all the elements smaller to it on its left and all the elements larger than it on the right. Now 6 8 17 14 and 63 37 52 are considered as two separate lists, and same logic is applied on them, and we keep doing this until the complete list is sorted.



Time Complexity:

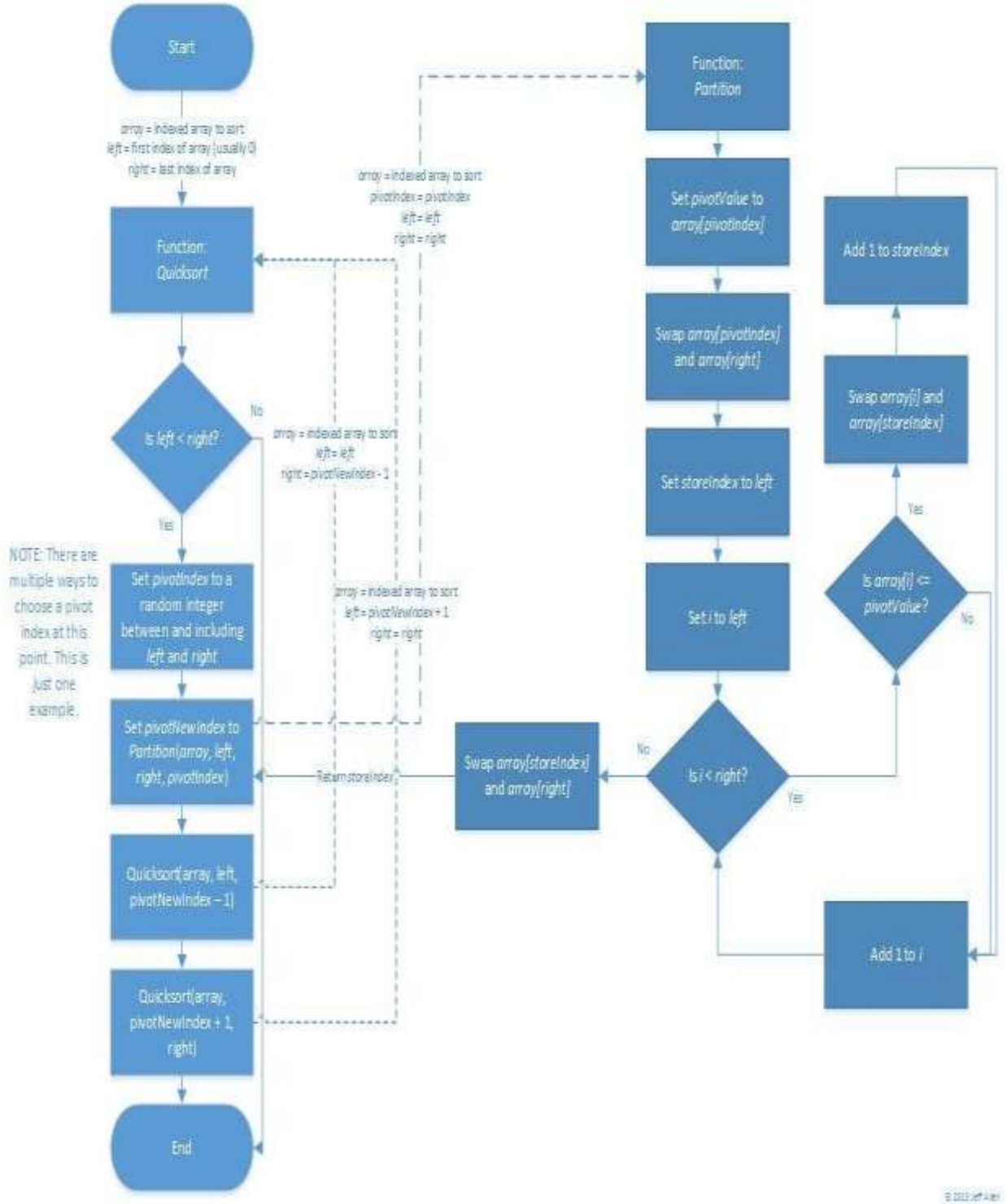
Worst Case Time Complexity: $O(n^2)$ Best Case Time Complexity: O(n log n) Average Time Complexity: O(n log n) Space Complexity: O(n log n)

ALGORITHM:

/* a[] is the array, p is starting index, that is 0, and r is the last index of array. */

```
void quicksort(int a[], int p, int r)
{
 if(p < r)
 {
  int q;
  q = partition(a, p, r); quicksort(a, p, q); quicksort(a, q+1, r);
 }
}

int partition(int a[], int p, int r)
{
 int i, j, pivot, temp; pivot =
 a[p];
 i=p; j=r; while(1)
 {
  while(a[i] < pivot && a[i] != pivot) i++;
  while(a[j] > pivot && a[j] != pivot) j--;
  if(i < j)
  {
   temp = a[i]; a[i] = a[j];
   a[j] = temp;
  }
  else
  {
   return j;
  }
 }
  }
```

FLOW CHART:


Start
array = indexed array to sort
left = first index of array (usually 0)
right = last index of array
Function: Quicksort
Is left < right?
No
Yes
NOTE: There are multiple ways to choose a pivot index at this point. This is just one example.
Set pivotIndex to a random integer between and including left and right
Set pivotNewIndex to Partition(array, left, right, pivotIndex)
Quicksort(array, left, pivotNewIndex – 1)
Quicksort(array, pivotNewIndex + 1, right)
End
array = indexed array to sort
left = left
right = pivotNewIndex - 1
array = indexed array to sort
left = pivotNewIndex + 1
right = right
array = indexed array to sort
pivotIndex = pivotIndex
left = left
right = right
Function: Partition
Set pivotValue to array[pivotIndex]
Swap array[pivotIndex] and array[right]
Set storeIndex to left
Set i to left
Is i < right?
No
Yes
Swap array[storeIndex] and array[right]
Return storeIndex
Is array[i] <= pivotValue?
Yes
No
Swap array[i] and array[storeIndex]
Add 1 to storeIndex
Add 1 to i

TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Enter the sorted list	{11,23,34,45}	Message to be displayed already sorted list.	Message not displayed	Fail
2	ID2	Entered table is	Name / Mark abc / 22 xyz / 11 pqr / 12 mnl / 05 def / 11	Name / Mark mnl / 05 xyz / 11 def / 11 pqr / 12 abc / 22	Name / Mark mnl / 05 def / 11 xyz / 11 pqr / 12 abc / 22	Fail (not stable sort)

CONCLUSION: Successfully implemented Quick Sort.

FAQ:

1. Explain the functioning of Quick sort
2. What is recurrence for worst case of Quick Sort and what is the time complexity in Worst case?
3. Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this:
   2 5 1 7 9 12 11 10
   Which statement is correct?
   a) The pivot could be either the 7 or the 9
   b) The pivot could be the 7, but it is not the 9
   c) The pivot is not the 7, but it could be the 9
   d) Neither the 7 nor the 9 is the pivot.
4. Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).
   a) Quick Sort
   b) Heap Sort
   c) Merge Sort
   d) Insertion Sort

GROUP C

ASSIGNMENT NO: 7

TITLE: Implement singly linked list & its operations

OBJECTIVE: To study & implement singly linked list

PROBLEM STATEMENT:
Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member‘s information using singly linked list. Store student PRN and Name. Write functions to:
a) Add and delete the members as well as president or even secretary.
b) Compute total number of members of club
c) Display members
d) Two linked lists exists for two divisions. Concatenate two lists.

OUTCOME: Student will be able to use linked list data structure & its operations.

THEORY:
Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.

Basic operations of a singly-linked list are:

Insert – Inserts a new element at the end of the list.
Delete – Deletes any node from the list.
Find – Finds any node in the list.
Print – Prints the list.

1. Insert – This function takes the start node and data to be inserted as arguments. New node is inserted at the end so, iterate through the list till we encounter the last node. Then, allocate memory for the new node and put data in it. Lastly, store the address in the next field of the new node as NULL.

2. Delete - This function takes the start node (as pointer) and data to be deleted as arguments. Firstly, go to the node for which the node next to it has to be deleted, If that

node points to NULL (i.e. pointer->next=NULL) then the element to be deleted is not present in the list. Else, now pointer points to a node and the node next to it has to be removed, declare a temporary node (temp) which points to the node which has to be removed. Store the address of the node next to the temporary node in the next field of the node pointer (pointer->next = temp->next). Thus, by breaking the link we removed the node, which is next to the pointer (which is also temp). Because we deleted the node, we no longer require the memory used for it, free() will deallocate the memory.

3. Find - This function takes the start node (as pointer) and data value of the node (key) to be found as arguments. First node is dummy node so, start with the second node. Iterate through the entire linked list and search for the key. Until next field of the pointer is equal to NULL, check if pointer->data = key. If it is then the key is found else, move to the next node and search (pointer = pointer -> next). If key is not found return 0, else return 1.

ALGORITHM:

1. Start
2. Define node structure by using class or structure
3. Define head & tail pointer assign to null inside the constructor.
4. Define methods for create(),getnode(),insert_node(),delete_node(),display();
5. Create()
   While(1)
   Begin
   Enter any more node to be added(y/n)
   If(ans==n) then break
   Else
   NewNode=GetNode();
   Insert_node(NewNode);
   End

Algorithm for creation of node()

Input: Pointer to the linked list
Output: Newly created node newnode
Begin
Newnode=new Node;
Enter data for new node
Assign Link field to NULL;
Return that node to create function
End

Algorithm for Insertion of node()

Input: Pointer & position for inserting a node
Output: Pointer to linked list.
Create one pointer which will start from head
Node *temp=head
Initialize count=1
Enter the position where you want insert the node;
if(pos==1) then
Assign the newnode at head position
Else
while(count!=pos-1)
Begin
temp=temp->link ; // increment the temp pointer
if(temp==NULL) then
Begin
Flag=0;break; Count++;
End
If(flag==1) then
Begin
NewNode->link=temp->link;
temp->link=NewNode; End
Else
Print position is not found
End
End

Algorithm for Display of node()

Input: pointer to the linked list
Output: Elements of the list
Node *temp=head;
if(temp==NULL) then
print list is empty
Else Begin
While(temp!=NULL)
Begin

Display node data
Increment the pointer i.e.
temp=temp->link;
End

End

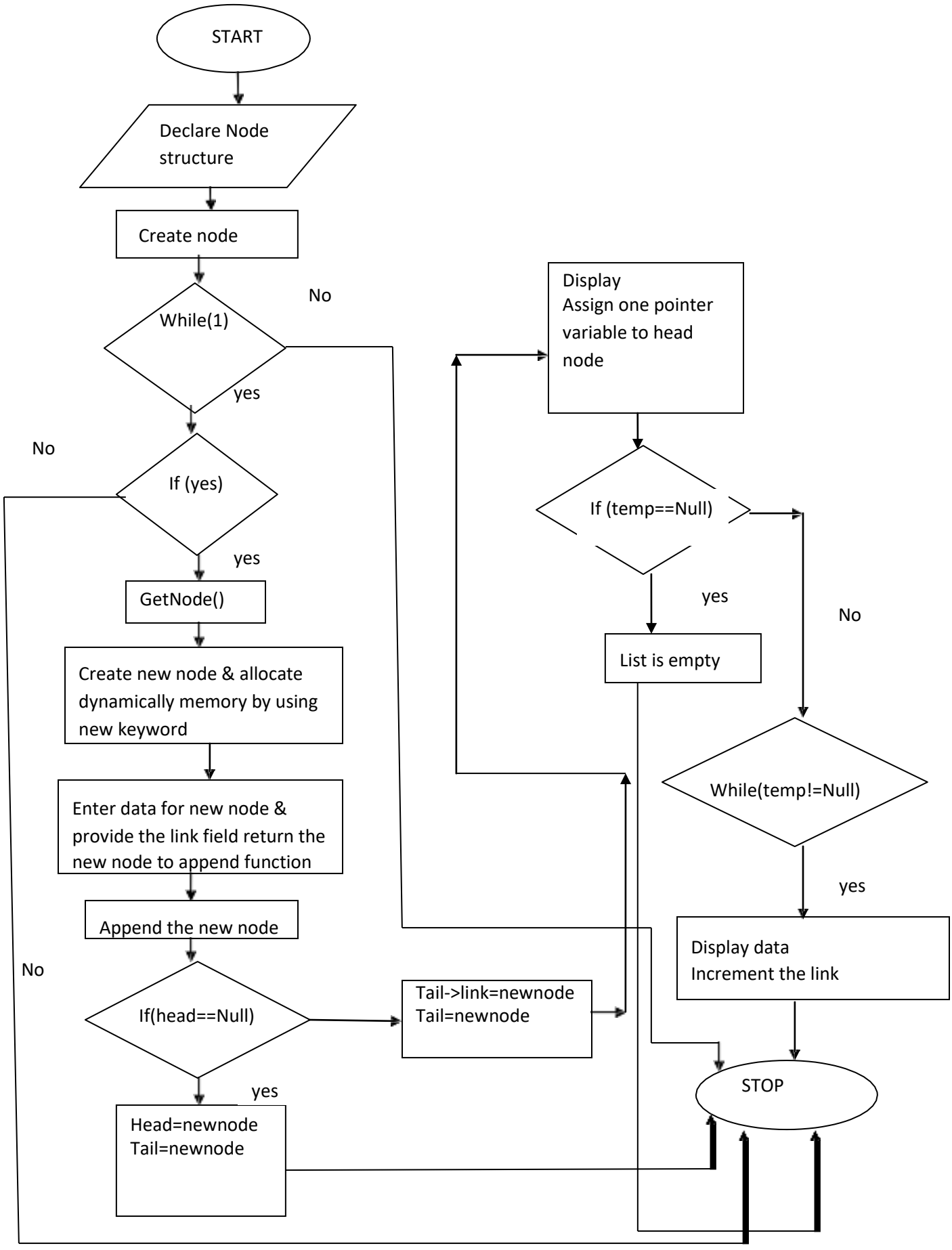
Algorithm for Deletion of node()

Input: Position in the linked list
Output: Linked list with desired node deleted
Initialize
int count=1,flag=1;
Node *curr,*temp;
temp=head; if(pos==1)
then Begin
head=head->link;
Delete temp;
End
Else Begin
While (count!=pos-1)
Begin
Increment the pointer till that
position temp=temp->link;
temp->link=curr->link;
delete curr;
end
Else
Print position is not found
End

Algorithm for Displaying linked list in reverse order

Input: Pointer to the linked list
Output: Elements of linked list in reverse order
Node *temp=head;
If(temp==NULL) then
    Return ;
Reverse_Display(temp->link) //recursively call the function then
Print the data

FLOWCHART:


START
Declare Node structure
Create node
While(1)
No
yes
If (yes)
No
yes
GetNode()
Create new node & allocate dynamically memory by using new keyword
Enter data for new node & provide the link field return the new node to append function
Append the new node
No
If(head==Null)
yes
Head=newnode
Tail=newnode
Tail->link=newnode
Tail=newnode
Display
Assign one pointer variable to head node
If (temp==Null)
yes
List is empty
No
While(temp!=Null)
yes
Display data
Increment the link
STOP

TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	If inserted position not present in list	List size 5 & entered position is 6	Position not found	Message displayed	Pass
2	ID2	Traverse the list till last node	Node->link!=Null	Last node in list	Last node data displayed & when link is Null	Pass

CONCLUSION: Successfully implemented singly linked list.

FAQ:

1. Elaborate the advantages of Linked List over sequential organization.
2. Discuss the applications of pointers with syntax.
3. Analyze your program to find time complexity.
4. Design any other real life application of linked list.
5. Compare array with linked list.
6. Analyze the space complexity of your program.
7. Justify that you always need to use structure while using linked list.
8. Elaborate the applications of linked list.
9. A linked list is dynamic data structure. The size of a linked list can be changed dynamically (during execution). How does this feature benefit a programmer?
10. Elaborate the different ways to allocate memory dynamically with syntax.

GROUP C

ASSIGNMENT NO: 8

TITLE: Implement singly linked list & its operations

OBJECTIVE: To study & implement singly 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-

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

OUTCOME: Student will be able to use linked list data structure & its operations.

THEORY:

A linked list is a linear collection of data elements, called nodes, each pointing to the next node by means of a pointer. It is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of data and a reference (in other words, a *link*) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing efficient insertion or removal from arbitrary element references.

Terminologies of linked list:

1. Header Node: Header node is option node in the list. The node contains the information about the node structure of node & number of nodes present in list.
2. Head Node: This is the first node in linked list.
3. Tail Node: This is the last node in linked list.
4. Head Pointer: This pointer holds the address of first node.

Advantages of Linked Lists

They are a dynamic in nature which allocates the memory when required.
Insertion and deletion operations can be easily implemented.
Stacks and queues can be easily executed.
Linked List reduces the access time.

Disadvantages of Linked Lists

The memory is wasted, as pointers require extra memory for storage.
No element can be accessed randomly; it has to access each node sequentially.
Reverse Traversing is difficult in linked list.

Applications of Linked Lists

Linked lists are used to implement stacks, queues, graphs, etc.
Linked lists let you insert elements at the beginning and end of the list.
In Linked Lists we don't need to know the size in advance.

Following are the basic operations on linked list:

1. **Create**: This function creates the newnode allocate memory dynamically by using new keyword. After creation of node it will add the node at first or at last if list is contains nodes.

2. **Print/Display** - function takes the start node (as pointer) as an argument. If pointer = NULL, then there is no element in the list. Else, print the data value of the node (pointer->data) and move to the next node by recursively calling the print function with pointer->next sent as an argument.

ALGORITHM:

1. Start
2. Define node structure by using class or structure
3. Define head & tail pointer assign to null inside the constructor.
4. Define methods for create(),getnode(),append(),delete_node(),display(),Union(),Intersection(),Difference();
5. Create()
   While(1)
   Begin
   Enter any more node to be added(y/n)
   If(ans==n) then break
   Else
   NewNode=GetNode();
   Append(NewNode);
   Union();
   Intersection();
   Difference();
   End

Algorithm for getting data of node()

Input: pointer to the linked list

Output: Newly created node newnode

```
Begin
        Newnode=new Node;
        Enter data for new node
        Assign Link field to NULL;
        Return that node to create function
End
```

Algorithm for appending a node()

Input: pointer to the linked list
Output: Linked list with a added node

```
        if(head==NULL)
        then
        head=NewNode;
        tail=NewNode;
        Else
        Begin
                tail->link=NewNode;
                tail=NewNode;

        End
```

Algorithm: to find intersection of two linked list

Input: Pointers to the two linked list
Output: New Linked list with common nodes of two linked list

```
Node *temp1=List1.head,*temp2=List2.head;
While(temp1!=Null)
begin
While(temp2!=Null)
begin
If(List[i]==List[j]) then
Then add that rollno into the resultant list
End
Display resultant list
```

Algorithm: to find Union of two linked list

Input: Pointers to the two linked list

Output: New Linked list with all nodes of two linked list

```
Initialization flag=0;
Node *temp1=List1.head,*temp2=List2.head;
Copy the rollno of vanilla list into resultant list
While(temp2!=Null)
begin
        While(temp1!=Null)
         begin
                If (List2[i])==List1[j]) then
                Set flag=1;break;
                If(flag==0) then
                Then add the remaining rollno. to resultant list
        End
End
```

Display resultant list

Algorithm: To find Difference of two linked list

Input: Pointers to the two linked list

Output: New Linked list with nodes of either linked list

```
Initialization flag=1;

Node *temp1=List1.head,*temp2=List2.head;
While(temp1!=Null)
Begin
        While(temp2!=Null)
        Begin
                If (list1[i]==list2[j])
                Flag=0;
                Break;
                If(flag==1)
                Copy list2 rollno to resultant list
        End
 End
Display the resultant list;
```

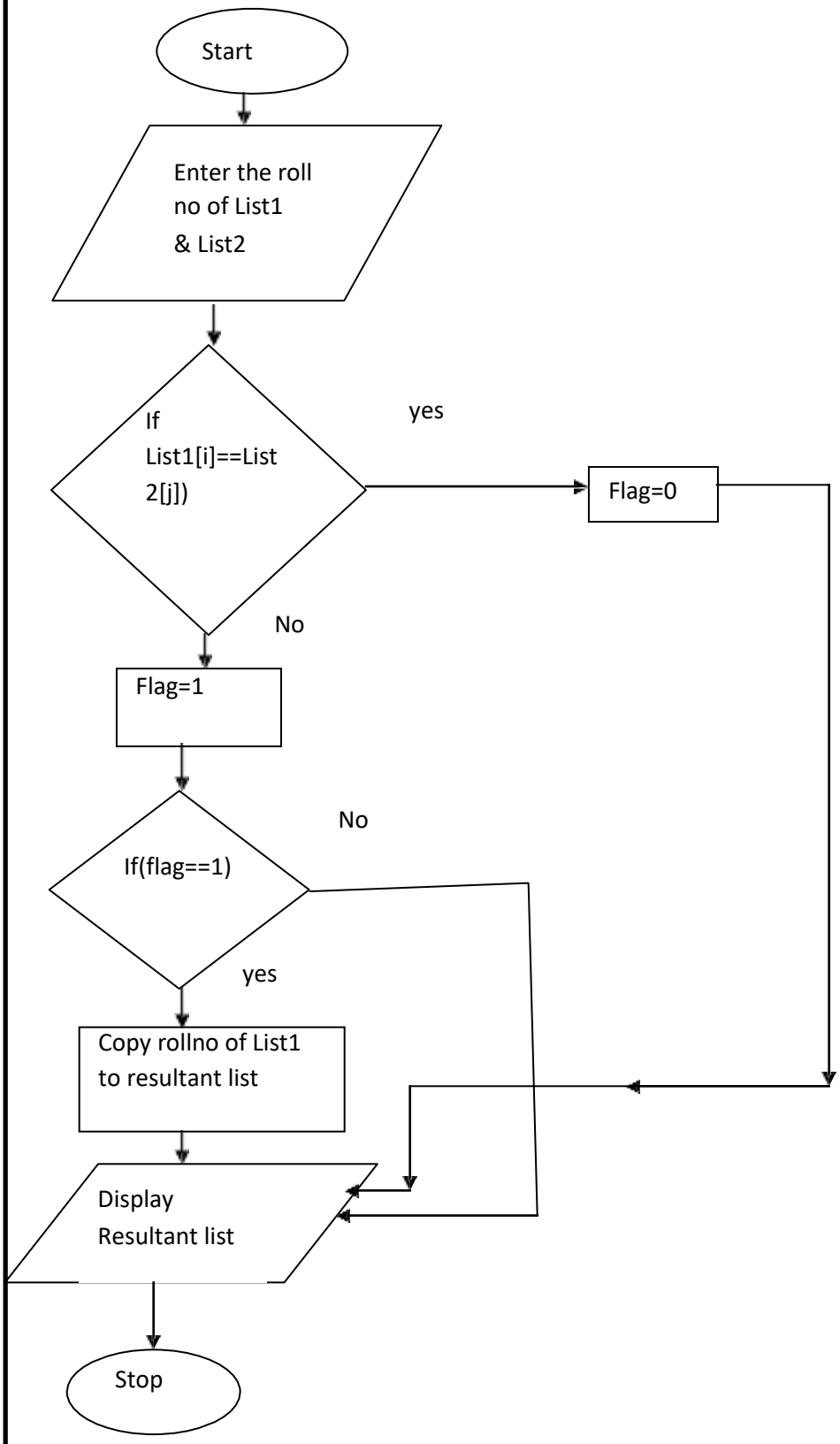
Algorithm for Display of node()

Input: pointer to the linked list
Output: Elements of the list

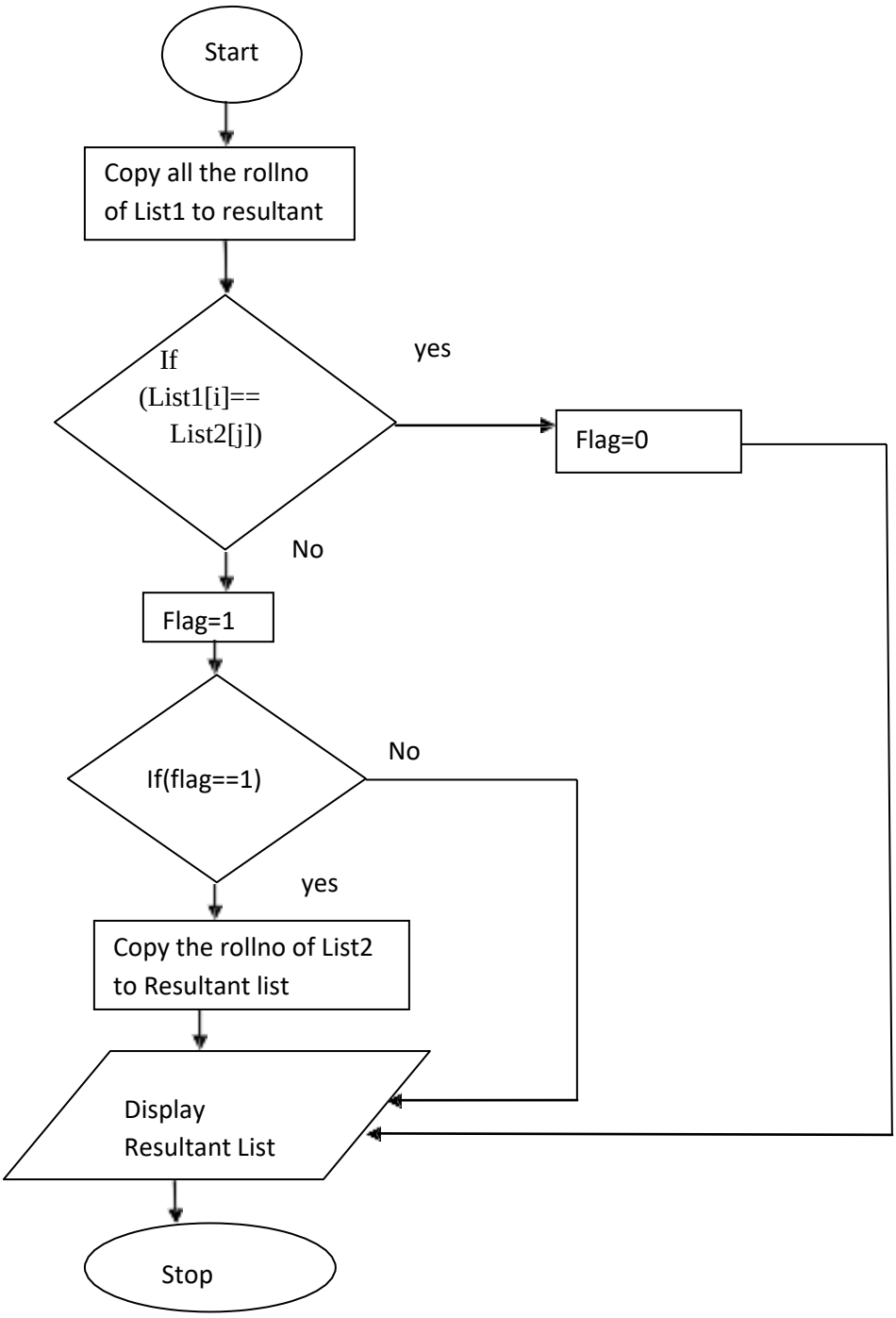
```
Node *temp=head;

if(temp==NULL) then

print list is  empty

Else Begin

While(temp!=NULL)

        Begin

        Display node data

        Increment the pointer i.e.

        temp=temp->link;

        End

        End
```

FLOWCHART:

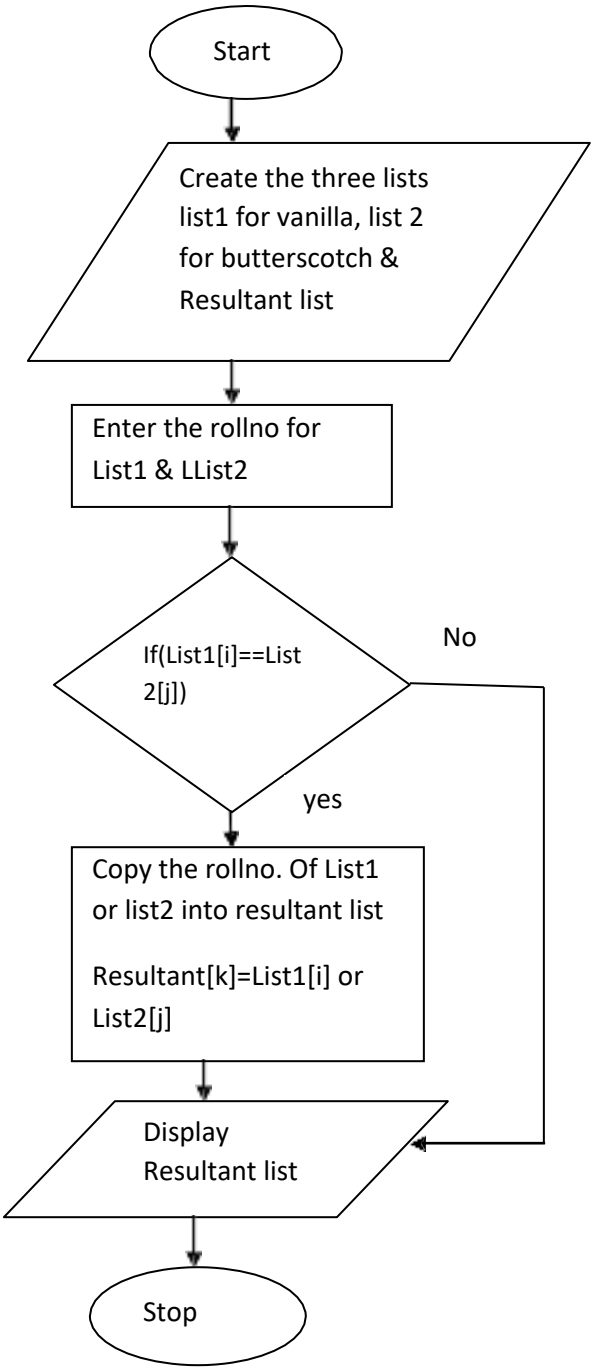
Difference:



Union:


Start
Copy all the rollno of List1 to resultant
If (List1[i]== List2[j])
yes
Flag=0
No
Flag=1
If(flag==1)
No
yes
Copy the rollno of List2 to Resultant list
Display Resultant List
Stop

Intersection:



TEST CASES:

Sr.No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Enter the duplicate number	{1,2,2,3}	Message to be displayed duplicate elements.	Message displayed	Pass
2	ID2	Enter the negative roll number	{-1,2,3}	Message to be displayed negative elements.	Message displayed	Fail
3	ID3	Enter two sets	{1,2,3} {1,4,5}	{1,2,3,4,5} Union	{1,2,3,1,4,5}	Fail
4	ID4	Enter two sets	{1,2,3} {1,4,5}	{2,3} Difference A-B	{1,2,3}	Fail

CONCLUSION: Successfully implemented all singly linked list operation.

FAQ:

1. Elaborate all types of linked list with at least one example.
2. Analyze your program to find tie complexity
3. Elaborate the linked list terminologies 1.Header Node 2.Head Node 3.Tail Node
4. Analyze your program to find space complexity.
5. Elaborate garbage collection.
6. Classify doubly linked list and circular linked list.
7. Experiment any one function of your program with doubly linked list.
8. Compare that function using doubly linked list and same using singly linked list with respect to time complexity and space complexity.
9. Experiment any one function of your program with circular linked list.
10. Design the pseudo-code for traversal in a linked list with its importance.

GROUP D

ASSIGNMENT NO: 9

TITLE: Implement stack

OBJECTIVE: To study & implement stack to check whether given expression is well parenthesized or not.

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.

OUTCOME: Student will be able to check whether given expression is well parenthesized or not, using stack

THEORY:

A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation.

Basic Operations

push() − pushing (storing) an element on the stack.
pop() − removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack through following functionality,

gettop() − get the top data element of the stack, without removing it.
isFull() − check if stack is full.
isEmpty() − check if stack is empty.

The **top** pointer provides top value of the stack without actually removing it.

Algorithm of gettop( ) function –

begin procedure gettop
return stack[top]
end procedure

Algorithm of isfull( ) function −

begin procedure isfull
if **top** equals to MAXSIZE
return true
else
return false
endif
end procedure

Algorithm of isempty( ) function-

begin procedure isempty if **top**
less than 1
return true
else
return false
endif
end procedure

Algorithm for PUSH( ) operation-

begin procedure push: stack, data if stack
is full
return null endif
top ← top + 1 stack[top]
← data
end procedure

Algorithm for POP( ) operation-

begin procedure pop: stack if stack
is empty
return null
endif
data ← stack[top]
top ← top - 1 return
data
end procedure

ALGORITHM:

2) Declare a character stack S.

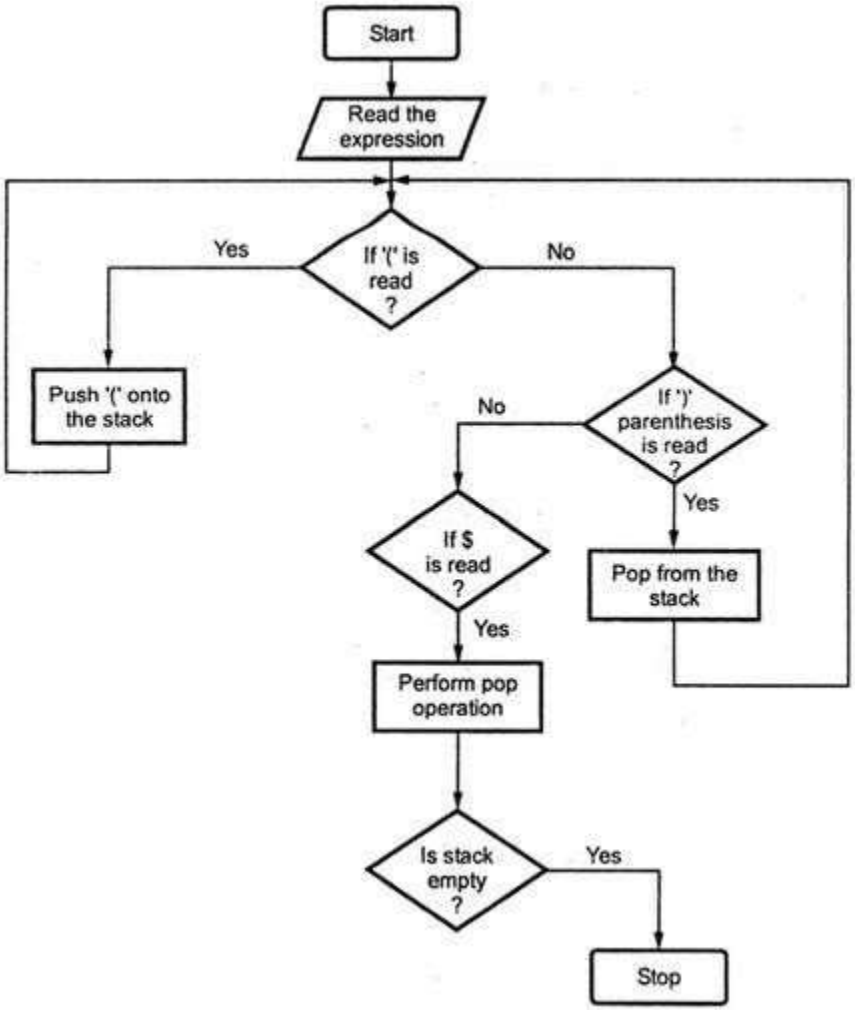
3) Now traverse the expression string exp.

a) If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.

b) If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and ifthe popped character is the matching starting bracket then fine else parenthesis are not balanced.

4) After complete traversal, if there is some starting bracket left in stack then “not balanced”

FLOWCHART:



TEST CASES:

Sr.No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Input a infix expression	{(a+b)/[d-e]}	Well formed parentheses	Well formed parentheses	Pass
2	ID2	Input a infix expression	{(a+b)/[d-e]	Not Well formed parentheses	Not Well formed parentheses	Fail
3	ID3	Input a infix expression	[(a+b])	Not Well formed parentheses	Not Well formed parentheses	Fail

CONCLUSION: Successfully implemented of stack to check the well-formed parentheses.

FAQ:

1. Compare stack with queue.
2. Analyze your program to find time complexity.
3. Design one real life application of stack.
4. Elaborate the stack implementation using array.
5. Analyze your program to find space complexity.
6. Elaborate the stack implementation using linked list.
7. Write the postfix form of the expression (a+(b*(c/d)-e))?
8. Elaborate the operation of stack.
9. Comment on the value of TOP=-1.
10. List the advantages of stack over queue.

GROUP D

ASSIGNMENT NO: 10

TITLE: Implement expression conversion using stack

OBJECTIVE: To study and implement conversion of infix to postfix notation and its evaluation

PROBLEM STATEMENT:
Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions:
1. Operands and operator, both must be single character.
2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.

OUTCOME: Student will be able to convert infix to postfix and evaluate it, using stack

THEORY:

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: **push** the item into the stack, and **pop** the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. **push** adds an item to the top of the stack, **pop** removes the item from the top.

Basic Operations

push() − pushing (storing) an element on the stack.
pop() − removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack through following functionality,

gettop() − get the top data element of the stack, without removing it.
isFull() − check if stack is full.
isEmpty() − check if stack is empty.

The **top** pointer provides top value of the stack without actually removing it.

Infix notation: X + Y
Operators are written in-between their operands. This is the usual way we write expressions.
Postfix notation (also known as "Reverse Polish notation"): X Y +
Operators are written after their operands. The infix expression given above is equivalent to A B C + * D /

Prefix notation (also known as "Polish notation"): + X Y
Operators are written before their operands. The expressions given above are equivalent to / * A + B C D

ALGORITHM:

Algorithm: Conversion INFIX TO POSTFIX

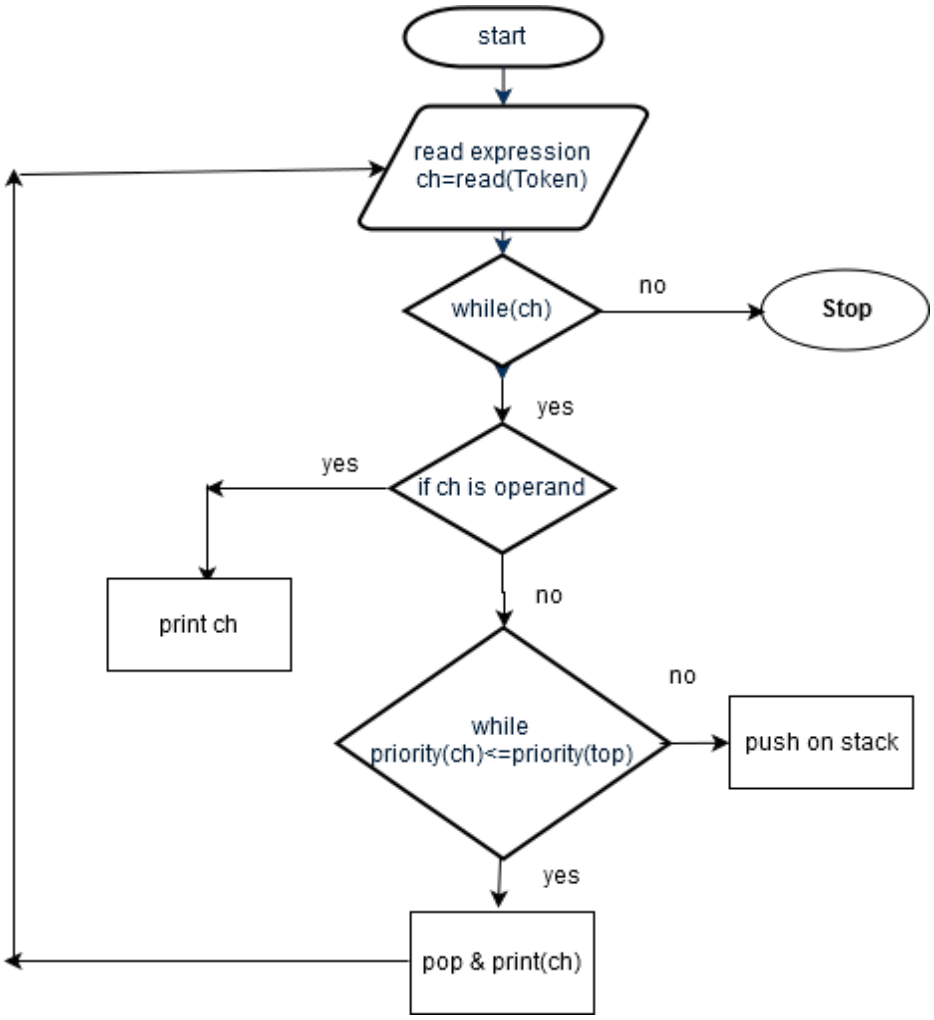
```
   Stack S
   Char ch
   Char element
while(Tokens are Available)
 {
    ch = Read(Token);
    if(ch is Operand)
      {
      Print ch ;
      }
    else
      {
      while(Priority(ch) <= Priority(Top Most Stack))
          {
          element = Pop(S);
          Print(ele);
          }
      Push(S,ch);
      }
}
while(!Empty(S))
{
element = Pop(S);
Print(ele);
}
```

Algorithm : Evaluate POSTFIX Expression

```
Initialize(Stack S)

x = ReadToken(); // Read Token
while(x)
{

if ( x is Operand )
        Push ( x ) Onto Stack S.
    if ( x is Operator )
          {
          Operand1 = Pop(Stack S);
          Operand2 = Pop(Stack S);
          Evaluate (Operand1,Operand2,Operator x);
          }
x = ReadNextToken(); // Read Token

}
```

FLOWCHART:

(a) Infix to Postfix



(b) Evaluate Postfix

TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Input a infix expression	(a+b)/(d-e)	ab+de-/	ab+de-/	Pass
2	ID2	Input a infix expression	(a+b)/(d-e)	ab+de-/	ab+de/-	Fail

CONCLUSION: Successfully implemented of stack to convert Infix to Postfix.

FAQ:

1. Elaborate Polish Notations.
2. Elaborate Pros and Cons of Recursion.
3. Analyze your program to find time complexity.
4. Justify that the priority is necessary to convert infix to postfix expression.
5. Design the function evaluate the infix expression.
6. Can we use queue to convert infix to postfix expression? Justify your answer.
7. Compare this application with the stack implementation using linked list and stack implementation using array. Analyze which is the beneficial.
8. Design one example to evaluate prefix expression.
9. Design one function to evaluate prefix expression.
10. Justify the need to convert infix to prefix in any application.

GROUP: E
ASSIGNMENT NO: 11

TITLE: Implement Queue

OBJECTIVE: To study & implement queue and its operations

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.

OUTCOME: Student will be able to implement queue and its operations

THEORY:

Queue is also an abstract data type or a linear data structure, in which the first element is inserted from one end called **REAR**(also called tail), and the deletion of existing element takes place from the other end called as **FRONT**(also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed 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**.

Basic Operations :

enQueue(value) - Inserting value into the queue

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a 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 queue. We can use the following steps to insert an element into the queue...

Step 1: Check whether **queue** is **FULL**. (**rear == SIZE-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 increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

deQueue() - Deleting a value from the Queue

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

Step 1: Check whether **queue** is **EMPTY**. (**front == rear**)
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 increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front == rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front = rear = -1**).

ALGORITHM

Algorithm : procedure isfull

```
  if rear equals to MAXSIZE
    return true
  else
    return false
  endif
end procedure
```

Algorithm : procedure isempty

```
  if front is less than MIN OR front is greater than rear
    return true
  else
    return false
  endif
end procedure
```

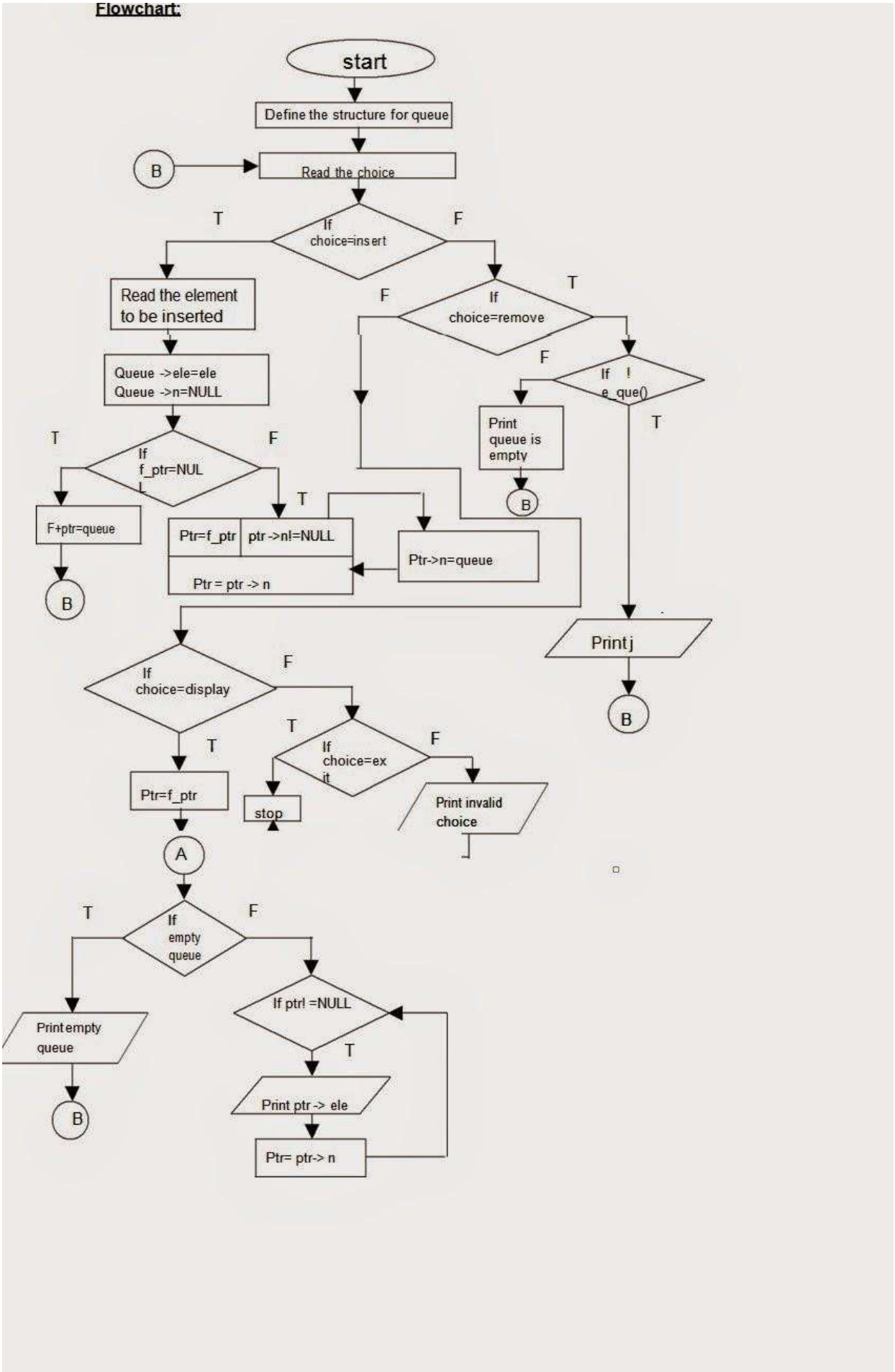
Algorithm : procedure enqueue(data)

```
  if queue is full
    return overflow
  endif
   rear ← rear + 1
  queue[rear] ← data
return true end
   procedure
```

Algorithm : procedure dequeue

```
  if queue is empty
    return underflow
  end if
  data = queue[front]
  front ← front + 1
  return true
end procedure
```

Flowchart:


start
Define the structure for queue
B
Read the choice
If choice=insert
T
F
Read the element to be inserted
Queue ->ele=ele
Queue ->n=NULL
If f_ptr=NULL
F+ptr=queue
B
Ptr=f_ptr
ptr->n!=NULL
Ptr = ptr -> n
Ptr->n=queue
If choice=remove
If ! e_que()
Print queue is empty
Print j
If choice=display
If choice=exit
stop
Print invalid choice
Ptr=f_ptr
A
If empty queue
Print empty queue
If ptr! =NULL
Print ptr -> ele
Ptr= ptr-> n

TEST CASES:

Sr.No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Enqueue(x) If full()	x	Queue is full	Queue is full	Fail
2	ID2	Enqueue(x) If empty()	x	1 element added in queue	1 element added in queue	Pass
3	ID3	dequeue() If empty()	-	error	Queue is empty	Fail
4	ID4	dequeue() If full()	-	1 element deleted from the queue	1 element deleted from the queue	Pass

CONCLUSION: Successfully implemented of queue and its operation.

FAQ:

1. Elaborate FRONT and REAR in a queue.
2. Analyze your program to find time complexity.
3. Elaborate various operations of queue.
4. Elaborate Queue full and Queue empty condition.
5. Discover applications of queue.
6. Compare disadvantages of queue with stack.
7. Elaborate any real life application of queue.
8. Analyze your program to find space complexity.
9. Compare queue implementation using array and queue implementation using linked list.
10. Design any one function using queue implementation using linked list.

GROUP E

ASSIGNMENT NO: 12

TITLE: Implement deque

OBJECTIVE: To study & implement deque and its operations

PROBLEM STATEMENT:

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.

OUTCOME: Student will be able to implement deque and its operations

THEORY:

Deque (usually pronounced like *"deck"*) is an irregular acronym of **d**ouble-**e**nded **que**ue. Double-ended queues are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back).

Operations of deque :

1. Deque() creates a new deque that is empty. It needs no parameters and returns an empty deque.
2. addFront(item) adds a new item to the front of the deque. It needs the item and returns nothing.
3. addRear(item) adds a new item to the rear of the deque. It needs the item and returns nothing.
4. removeFront() removes the front item from the deque. It needs no parameters and returns the item. The deque is modified.
5. removeRear() removes the rear item from the deque. It needs no parameters and returns the item. The deque is modified.
6. isEmpty() tests to see whether the deque is empty. It needs no parameters and returns a boolean value.
7. size() returns the number of items in the deque. It needs no parameters and returns an integer.

ALGORITHM :

Algorithm to add an element into DeQueue :

Assumptions: pointer f,r and initial values are -1,-1
Q[] is an array
max represent the size of a queue

enq_front

step1. Start
step2. Check the queue is full or not as if (f <>
step3. If false update the pointer f as f= f-1

step4. Insert the element at pointer f as Q[f] = element
step5. Stop

enq_back

step1. Start
step2. Check the queue is full or not as if (r == max-1) if yes queue is full
step3. If false update the pointer r as r= r+1
step4. Insert the element at pointer r as Q[r] = element
step5. Stop

Algorithm to delete an element from the DeQueue

deq_front

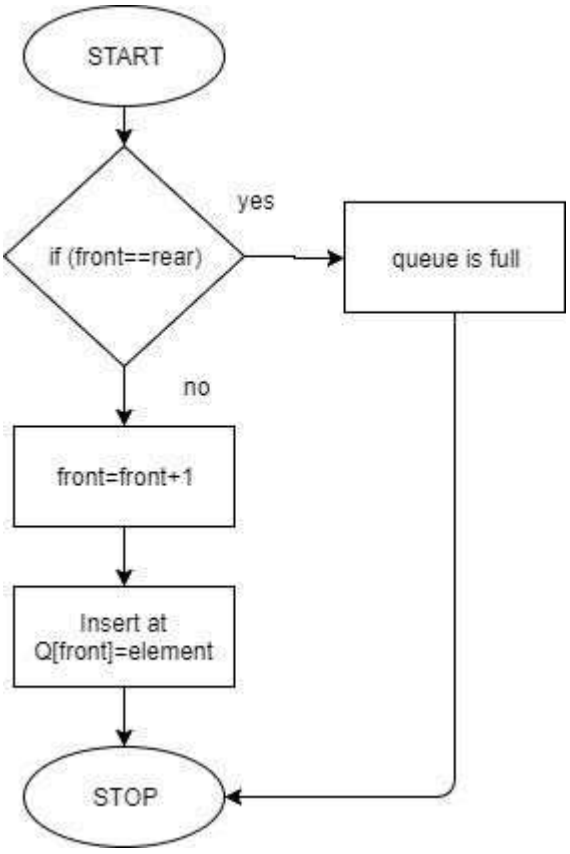
step1. Start
step2. Check the queue is empty or not as if (f == r) if yes queue is empty
step3. If false update pointer f as f = f+1 and delete element at position f as element = Q[f]
step4. If (f== r) reset pointer f and r as f=r=-1
step5. Stop

deq_back

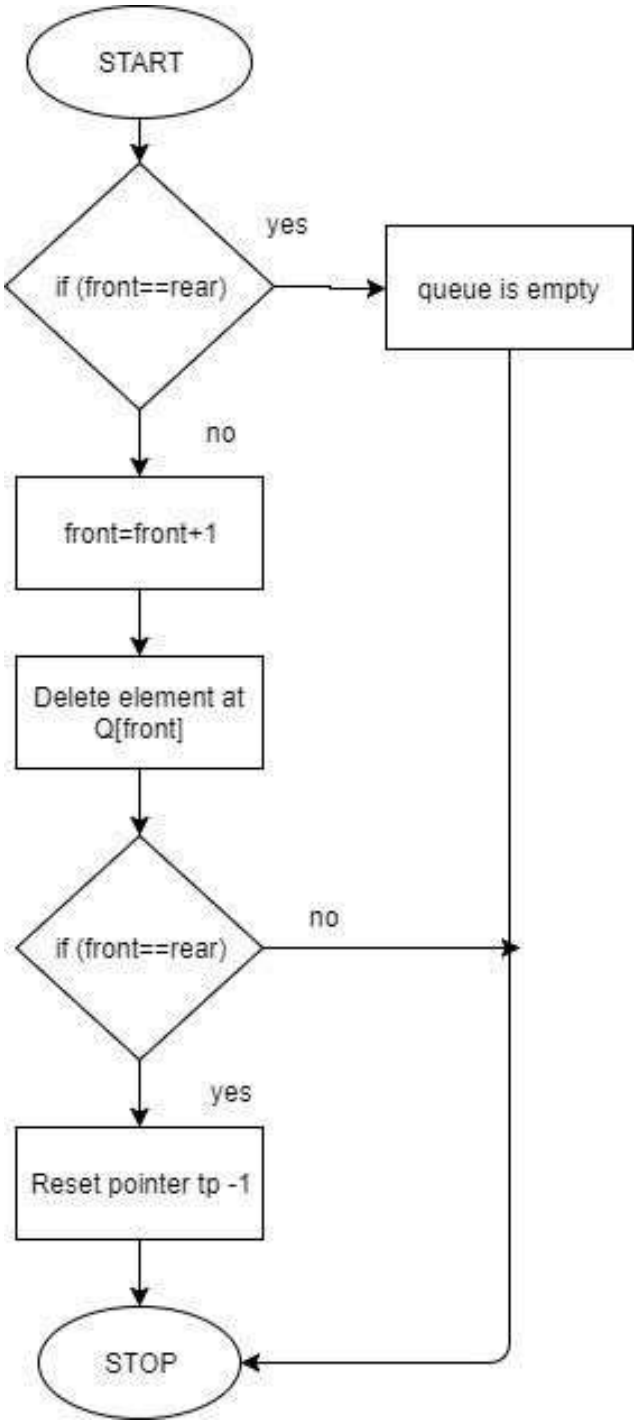
step1. Start
step2. Check the queue is empty or not as if (f == r) if yes queue is empty
step3. If false delete element at position r as element = Q[r]
step4. Update pointer r as r = r-1
step5. If (f == r) reset pointer f and r as f = r= -1
step6. Stop

FLOWCHART:

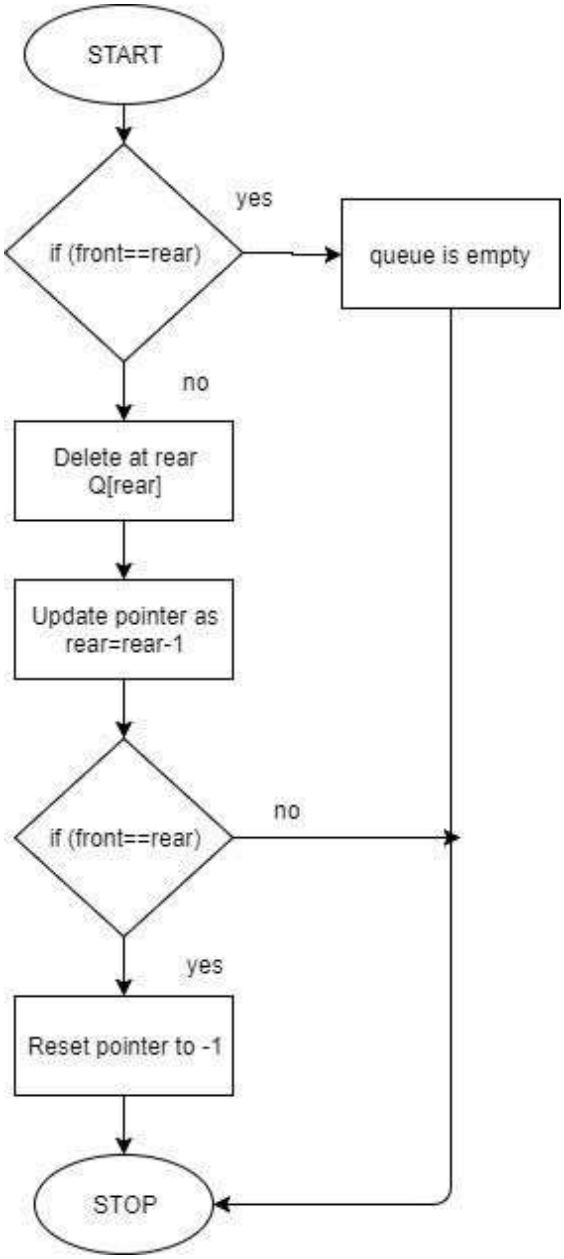
Insert element from front



Delete element from front



Delete element from rear



TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Insert element from front of the deque	10	10,20,30,40	10,20,30,40	Pass
2	ID2	Delete element from front of the deque	20	30,40	30,40	Pass
3	ID3	Delete element from rear of the deque	40	30	30,40	Fail

CONCLUSION: Successfully implemented of deque and its operation.

FAQ:

1. Compare queue with double ended queue data structure.
2. Elaborate various operations of the double-ended queue.
3. Analyze your program to find time complexity.
4. Elaborate the applications of double-ended queue.
5. Compare double-ended queue and circular queue.
6. Analyze your program to find space complexity.
7. Design any one function of double ended queue with the implementation of linked list.
8. Design one real life application of double ended queue.
9. Elaborate the advantages of double ended queue over simple queue and circular queue.
10. Elaborate the advantage of circular queue over linear queue.

GROUP: E

ASSIGNMENT NO: 13

TITLE: To study Linked List as a data structure and be able to perform different operations on it.

OBJECTIVE: To study and implement linked list operation.

OUTCOME: Student will be able to use linked list data structure & its operations.

PROBLEM STATEMENT:

Write C++ program to store set of negative and positive numbers using linked list. Write functions to
a) Insert numbers
b) Delete nodes with negative numbers
c) Create two more linked lists using this list, one containing all positive numbers and other containing negative numbers
d) For two lists that are sorted; Merge these two lists into third resultant list that is sorted

THEORY:

What are Linked Lists?

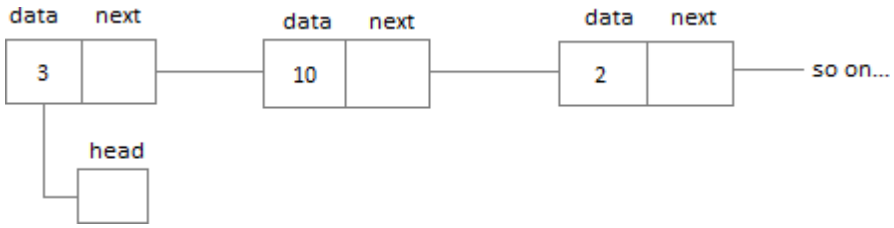
Linked List is a linear data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

Types of Linked Lists

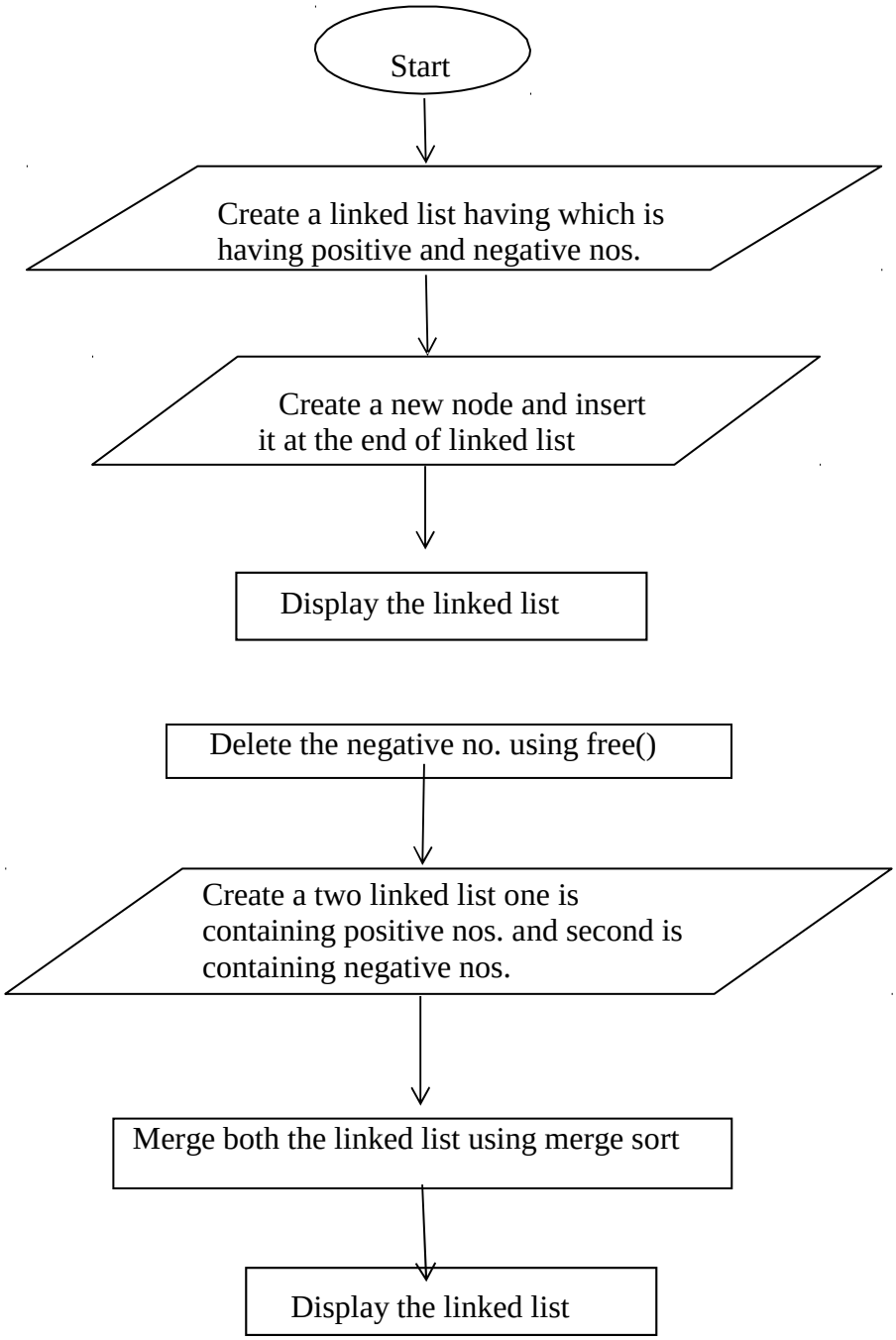
- **Singly Linked List :** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



- **Doubly Linked List :** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.

- **Circular Linked List :** In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.

FLOWCHART:


Start
Create a linked list having which is having positive and negative nos.
Create a new node and insert it at the end of linked list
Display the linked list
Delete the negative no. using free()
Create a two linked list one is containing positive nos. and second is containing negative nos.
Merge both the linked list using merge sort
Display the linked list

Test Cases:

Sr.No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Create a linked list	Insert node to the linked list	Linked list containing positive and negative nos.	Linked list containing positive and negative nos.	Pass
2	ID2	Input linked list	Delete the node	Delete the negative no. from the linked list	Delete the negative no. from the linked list	Pass
3	ID3	Input 2 linked list	One containing positive nos. second containing negative nos.	One list of positive nos. and another list of negative nos.	One list of positive nos. and another list of negative nos.	Pass
4	ID4	Input linked list	Both positive and negative linked list	Sorting the linked list	Sorting the linked list	Pass

Conclusion: Thus, we have studied linked list and implemented insertion, deletion and sorting operations on linked list successfully.

FAQ:

1. Explain the difference in singly and doubly linked list
2. Linked list is not an indexed structure. Comment.
3. Discuss the drawbacks of Linked list.
4. Explain how to delete last node from singly linked list?
5. Explain how to insert an item to the beginning of singly linked list?

ASSIGNMENT NO: 14

TITLE: Study of shortest path algorithm (Dijkstra's algorithm or single source algorithm).

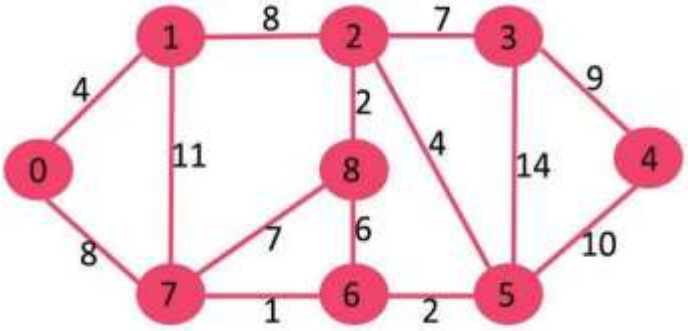
OBJECTIVE: To study shortest path algorithm.

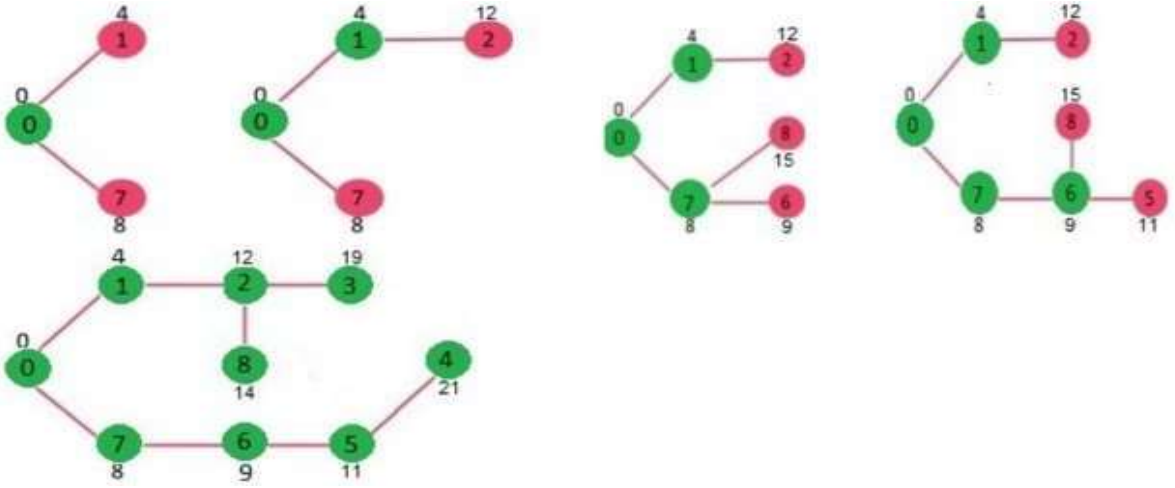
PROBLEM STATEMENT: Write a program to find shortest path for given source & destination of a given graph using C.

OUTCOME: Student will be able to implement shortest path algorithm.

THEORY:

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms. For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. Example of Dijikstra's Algorithm:





ALGORITHM:

1. Create the cost matrix[ ][ ] from adjacency matrix adj[ ][ ].
2. If there is no edge between vertices i & j then c[i][j] is infinity
3. Array visited[ ] is initialized to zero For (i=0;i<n;i++) visited[i]=0;
4. If the vertex 0 is the source vertex then visited[0] is marked as 1
5. Create the distance array by storing the cost of vertices form vertex no 0 to n-1 from source vertex 0

6. For(i=1;i<n;i++)
7. Distance[i]=cost[i][j]

   Initial distance of source vertex is taken as 0. Distance [0]=0

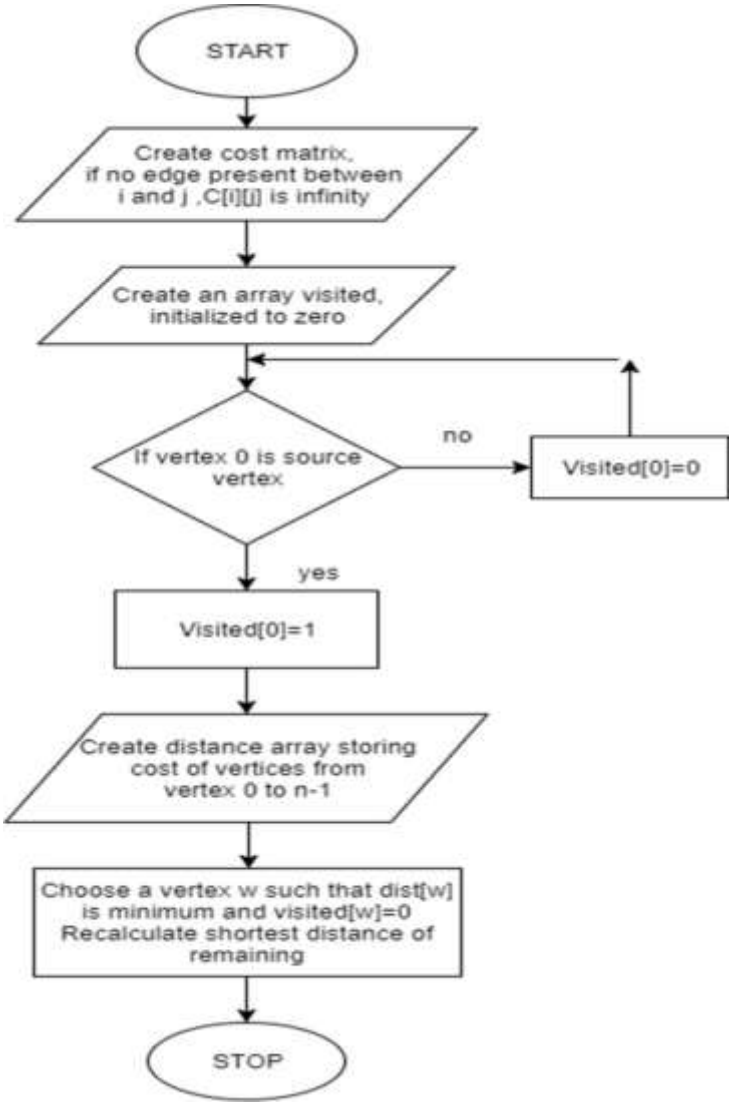
8. For(i=1; i<n; i++)
   - Choose a vertex w such that distance[w] is minimum and visited[w] is o
   - Mark visited[w] as 1
   - Recalculate the shortest distance of remaining vertices from the source . Only the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance

     i.e. for each vertex V if (visited[v]==0)

     distance[V]= min(distance[v], distance[w]+cost[w][v])

Time Complexity: $O(n^2)$

FLOW CHART:



TEST CASES:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	ID1	Select the node to visit with minimum cost and not already visited	Adjacency Matrix	Get the shortest path	Got the shortest path	Pass
2	ID1	Select the node to visit with minimum cost, already visited	Adjacency Matrix	Get the shortest path	Get cycle in the path	Fail

CONCLUSION: The shortest path algorithm can be implemented effectively using Dijkstra's algorithm.

ASSIGNMENT NO: 15

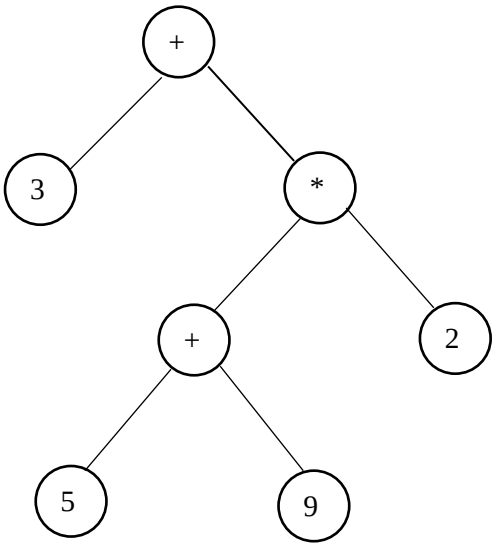
TITLE: Generate Expression Tree

OBJECTIVE: Create expression tree using given polynomial and evaluate expression using it

PROBLEM STATEMENT:

OUTCOME: Student will be able to:
a) Know how to construct expression tree.
b) Know how to evaluate an expression using the tree

THEORY:
Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand so for example expression tree for 3 + ((5+9)*2) would be:



To generate an expression tree, given an expression, we first convert the expression to its postfix form. From the postfix expression, we read one symbol at a time from left to right. If the current symbol is an operand, then we push a tree of one node consisting of the operator onto a stack. If the symbol is an operator, then we pop two trees from the stack and create a tree with the root containing the operator and the result of the pop operations as the right and the left subtrees in that order. The resulting tree is again pushed onto the stack. When we have read all the symbols, the equivalent expression tree is the only element in the stack and a pop operation would give us the tree. The following example illustrates the algorithm. Let the postfix expression be a b + f - c d * e + /.

- When we first read 'a' and 'b' in that order and create two trees containing a single node 'a' and 'b' respectively.
- These trees are then pushed on to the stack in that order.
- When we encounter the '+' operator, we pop two trees from the stack, creating a tree with '+' as the root and the two trees as the right and the left childs respectively.
- On reading the operand 'f', we create a tree with 'f' as a single node and push it on to the stack.

- On reading a '-' we create the tree with '-' as the root and the two previously created trees as its children and push the new tree onto the stack.
- Similarly, on reading 'c' and 'd', the stack has three trees; '-' and two new trees corresponding to nodes 'c' and 'd'.
- On reading a '*', the stack has two trees; '-' and a new tree consisting of '*' as the root and 'c' and 'd' as its children.
- Continuing further, we create the tree with '+' as the root. On reading the last '/', the final tree is created.

CONCLUSION: Expression can be easily evaluated using expression tree.