Parvatibai Genba Moze College of Engineering Wagholi, Pune

SE COMPUTER ENGINEERING

LAB MANUAL

DATA STRUCTURE LABORATORY

Prepared By

Prof. Vidya S Y

Academic Year 2023-24

Parvatibai Genba Moze College of Engineering Wagholi, Pune

DEPARTMENT OF COMPUTER ENGINEERING

Lab Manual

Second Year Engineering
Semester-III
DATA STRUCTURES LAB
Subject Code: 210247

Class: Second Year

Prepared By:
Prof. Vidya S Y

Academic year 2023-24

DATA STRUCTURES LAB
(210247)

Teaching Scheme	Credit	Examination Scheme
PR: 04 Hours/Week	02	TW: 25 Marks PR: 50 Marks

Guidelines for Instructor's Manual

The instructor,,s manual is to be developed as a hands-on resource and reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface etc), University syllabus, conduction & Assessment guidelines, topics under consideration- concept, objectives, outcomes, set of typical applications/practicals/ guidelines, and references.

Guidelines for Student Journal

The laboratory practicals are to be submitted by student in the form of journal. Journal consists of prologue, Certificate, table of contents, and handwritten write-up of each practical (Title, Objectives, Problem Statement, Outcomes, software & Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, test cases, conclusion/analysis. Program codes with sample output of all perform practical‟s are to be submitted as softcopy.

As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal may be avoided. Use of DVD containing students programs maintained by lab In-charge is highly encouraged. For reference one or two journals may be maintained with program prints at Laboratory.

Guidelines for Assessment

Continuous assessment of laboratory work is done based on overall performance and lab practicals performance of student. Each lab practical assessment will assign grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each lab practical assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness.

Guidelines for Practical Examination

Both internal and external examiners should jointly set problem statements. During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement. The supplementary and relevant questions may be asked at the time of evaluation to test the students for advanced learning, understanding of the fundamentals, effective and efficient implementation. So encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. So adhering to these principles will consummate our team efforts to the promising start of the student's academics.

Guidelines for Laboratory Conduction

The instructor is expected to frame the practicals by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The practical framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. The instructor may set multiple sets of practicals and distribute among batches of students. It is appreciated if the practicals are based on real world problems/applications. Encourage students for

appropriate use of Hungarian notation, proper indentation and comments. Use of open source software is to be encouraged.

In addition to these, instructor may assign one real life application in the form of a mini- project based on the concepts learned. Instructor may also set one practical or mini-project that is suitable to respective branch beyond the scope of syllabus.

Set of suggested practical list is provided in groups- A, B, C, D, and E. Each student must perform at least 13 practicals as at least 3 from group A, 3 from group B, 2 from group C, 2 from group D and 3 from group E.

Group A and B assignments should be implemented in Python without using built-in methods for major functionality of assignment. Use List data structure of Python as array**. Group C, D and E assignments should be implemented in C++ language.**

Operating System recommended:- 64-bit Open source Linux or its derivative

Programming tools recommended: - Open Source Python,

Programming tool like Jupyter Notebook, Pycharm, Spyder, G++/GCC.

Practical No.	Laboratory Assignments
	GROUP - A
1	A-01)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)
2	A-02) 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
3	A-03) 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 given string is palindrome or not d) To display index of first substring e) To count the occurrences of each word in string
4	A-04)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
	GROUP - B
5	B-11) a) Write a **Python** program to store roll numbers of student in array who attended training program in random order. Write function for searching whether particular student attended training program or not, using Linear search and Sentinel search. a) Write a **Python** program to store roll numbers of student array who attended training program in sorted order. Write function for searching whether particular student attended training program or not, using Binary search and Fibonacci search
6	B-14) 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.
7	B-16) 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.

GROUP - C	
8	C-19) 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 Two linked lists exists for two divisions. Concatenate two lists.
9	C-20) The ticket booking system of Cinemax theater has to be implemented using C++ program. There are 10 rows and 7 seats in each row. Doubly circular linked list has to be maintained to keep track of free seats at rows. Assume some random booking to start with. Use array to store pointers (Head pointer) to each row. On demand a) The list of available seats is to be displayed b) The seats are to be booked The booking can be cancelled.
10	C-21) Write C++ program for storing appointment schedule for day. Appointments are booked randomly using linked list. Set start and end time and min and max duration for visit slot. Write functions for- A) Display free slots B) Book appointment C) Sort list based on time D) Cancel appointment (check validity, time bounds, availability) Sort list based on time using pointer manipulation
GROUP – D	
11	D-26) 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.
12	D-27) 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. Only '+', '-', '*' and '/ ' operators are expected.
GROUP – E	
13	E-29) 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.
14	E-31) 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.

Practical No: 01 (A)

Practical Title:

To perform the Set Operations without using SET built-in functions

Aim: -

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- Do not use SET built-in functions)

Prerequisite:

- Python Programming

Objectives:

- To understand implementation of Array data structure.
- Understand the implementation of set with various operations like, Union, Intersection and Difference etc.

Input:

N number of students.

Outcome:

On completion of this assignment students will be able to -

- Implement the array data structure.
- Solve real world problem of set operations logically using array data structure.

Software & Hardware requirements:

- Open-Source C Programming tools like G++/GCC or Eclipse.
- 64-bit Open-source Linux or its derivative.

Theory- Concept:

What is array?

Whenever we want to work with large number of data values, we need to use that much number of different variables. As the numbers of variables are increasing, complexity of the program also increases and programmers get confused with the variable names. There may be situations in which we need to work with large number of similar data values. To make this work more easily, C programming language provides a concept called "Array"

An array is a variable which can store multiple values of same data type at a time.

An array can also be defined as follows... "Collection of similar data items stored in continuous memory locations with single name". To understand the concept of arrays, consider the following

example declaration. int a, b, c;

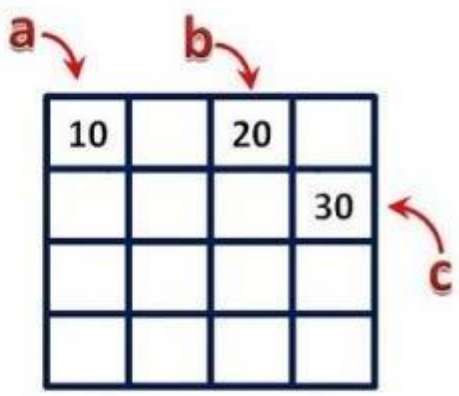
Here, the compiler allocates 2 bytes of memory with name 'a', another 2 bytes of memory with name 'b' and more 2 bytes with name 'c'. These three memory locations are may be in sequence or may not be in sequence. Here these individual variables can store only one value at a time.

In computer memory is organized as shown in figure. Here assume that each box is of 2 bytes of memory.

2 byte for 'a', another 2 bytes for 'b' and 2 more bytes for 'c'.

If we assign following values they will inserted into that memory locations.

a = 10;
b = 20;
c = 30;

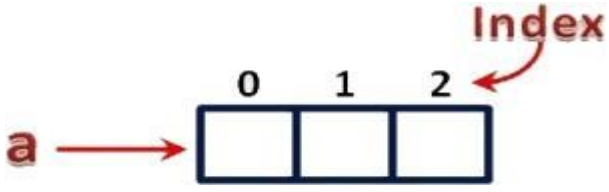


Now consider the following declaration...

int a[3];

Here, the compiler allocates total 6 bytes of continuous memory locations with single name 'a'. But allows storing three different integer values (each in 2 bytes of memory) at a time. And memory is organized as follows...

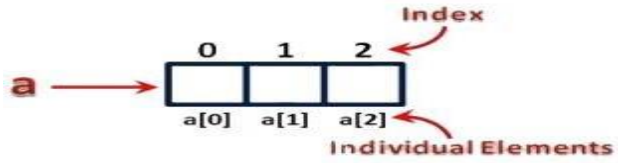
That means all these three memory locations are named as 'a'. But "how can we refer individual elements?" is the big question. Answer for this question is, compiler not only allocates memory, but also assigns a numerical value to each individual element of an array. This numerical value is called as "Index". Index values for the above example are as follows...



The individual elements of an array are identified using the combination of 'name' and 'index' as follows...

arrayName [indexValue]

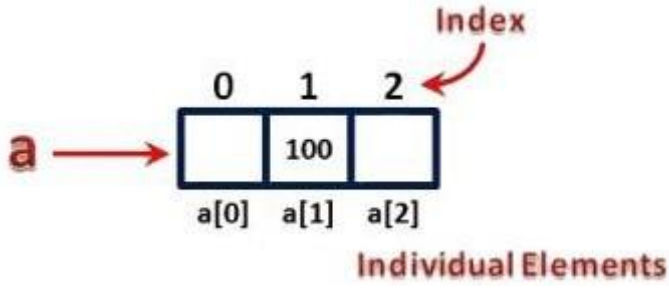
For the above example the individual elements can be referred to as follows...



If I want to assign a value to any of these memory locations (array elements), we can assign as follows...

a[1] = 100;

The result will be as follows..



Sets in Python:

- A set is a structure, representing an unordered collection (group) of zero or more distinct (different) objects.
- Rules:
- Every elements of set must be unique; no two members may be identical.
- All elements of set having same data type.
- Sets are a collection of unordered elements that are unique. Meaning that even if the data is repeated more than one time, it would be entered into the set only once. It resembles the sets that you have learnt in arithmetic. The operations also are the same as is with the arithmetic sets. An example program would help you understand better.

Set Operations:

Sets

Creating a set

Sets are created using the flower braces but instead of adding key-value pairs, you just pass values to it.

```
1  my_set = {1, 2, 3, 4, 5, 5, 5} #create set
2  print(my_set)
```

Output:

{1, 2, 3, 4, 5}

Adding elements

To add elements, you use the add() function and pass the value to it.

```
1 my_set = {1, 2, 3}
2 my_set.add(4) #add element to set
3 print(my_set)
```

Output:

{1, 2, 3, 4}

Operations in sets

The different operations on set such as union, intersection and so on are shown below.

```
1 my_set = {1, 2, 3, 4}
2 my_set_2 = {3, 4, 5, 6}
3 print(my_set.union(my_set_2), '            ', my_set | my_set_2)
4 print(my_set.intersection(my_set_2), '            ', my_set & my_set_2)
5 print(my_set.difference(my_set_2), '            ', my_set - my_set_2)
6 print(my_set.symmetric_difference(my_set_2), '            ', my_set ^ my_set_2)
7 my_set.clear()
8 print(my_set)
```

- The union() function combines the data present in both sets.
- The intersection() function finds the data present in both sets only.
- The difference() function deletes the data present in both and outputs data present only in the set passed.
- The symmetric_difference() does the same as the difference() function but outputs the data which is remaining in both sets.

Output:

{1, 2, 3, 4, 5, 6} ———- {1, 2, 3, 4, 5, 6}
{3, 4} ———- {3, 4}
{1, 2} ———- {1, 2}
{1, 2, 5, 6} ———- {1, 2, 5, 6}

set()

1. **Union**

   - For sets A, B, their union A U B is the set containing all elements that are either in A, or (“Ú”) in B (or, of course, in both).
   - Example if set A= {2,3,5} and set B= {3,5,7} then, A U B = { 2,3,5,7 }

2. **Intersection**

   - For sets A, B, their intersection A^B is the set containing all elements that are simultaneously in A and (“^”) in B.
   - Example if set A= {2,3,5} and set B= {3,5,7} then, A ^ B = { 3,5 }

3. **Difference**

- For sets A, B, the difference of A and B, written A-B, is the set of all elements that are in A but not B.
- Example- if set A= {2,3,5} and set B= {3,5,7} then, A - B = { 2 }

Implementation:

```
a = '''
In second year computer engineering class, group A student's play cricket, group B students play badminton and group C students play football.\n
   Write a python program using functions to compute following: - \n
   a)\tList of students who play both cricket and badminton \n
   b)\tList of students who play either cricket or badminton but not both \n
   c)\tNumber of students who play neither cricket nor badminton\n
   d)\tNumber of students who play cricket and football but not badminton.\n

'''
print(a)

groupA=[] #for cricket
groupB=[] #for badminton
groupC=[] #for football

n1=int(input("Enter the student count who play cricket"))
print("Enter %i students name:"%n1)
groupA=list(map(str,input().split(" ")))

n2=int(input("Enter the student count who play Bandmiton"))
print("Enter %i students name:"%n2)
groupB=list(map(str,input().split(" ")))

n3=int(input("Enter the student count who play Football"))
print("Enter %i students name:"%n3)
groupC=list(map(str,input().split(" ")))

print("GroupA=",groupA)
print("GroupB=",groupB)
print("GroupC=",groupC)



def first (groupA, groupB):
   newlist=[]
   lena=len(groupA)
   lenb=len(groupB)

   for i in range(lena):
      for j in range (lenb):
         if(groupA[i]==groupB[j]):
            newlist.append(groupA[i])
```

```
            break
    print("List of students who play both cricket and badminton")
    print(newlist)



def second(groupA, groupB):
    newlist=[]
    lena = len(groupA)
    lenb = len(groupB)
    flag=0
    for  i  in  range(lena):
        for j in range(lenb):
            if(groupA[i]==groupB[j]):
                flag=1
                break
        if(flag==0):
            newlist.append(groupA[i])
        flag=0

    for i in range(lenb):
        for j in range(lena):
            if(groupB[i]==groupA[j]):
                flag=1
                break;

        if(flag==0):
            newlist.append(groupB[i])
        flag=0
    print("List of students who play either cricket or badminton but not both")
    print(newlist)



def third(groupA, groupB,groupC):

    newlist=[]
    lena=len(groupA)
    lenb=len(groupB)
    lenc=len(groupC)
    flag=0
    for  i  in  range(lenc):
        for j in range(lena):
            if(groupC[i]==groupA[j]):
                flag=1
                break
        for var in range(lenb):
            if(groupC[i]==groupB[var]):
                flag=1
                break
```

```
        if(flag==0):
            newlist.append(groupC[i])
        flag=0

    print("Number of students who play neither cricket nor badminton")
    print(newlist)


def fourth(groupA,groupB,groupC):
    list1=[]
    newlist = []
    lena = len(groupA)
    lenb = len(groupB)
    lenc = len(groupC)
    flag=0
    for  i  in  range(lena):
        for j in range(lenc):
            if(groupA[i]==groupC[j]):
                list1.append(groupA[i])
                break

    newl=len(list1)
    for i in range(newl):
        for j in range(lenb):
            if(list1[i]==groupB[j]):
                flag=1
                break
        if(flag==0):
            newlist.append(list1[i])
        flag=0

    print("Number of students who play cricket and football but not badminton")
    print(newlist)

first(groupA,groupB)
second(groupA,groupB)
third(groupA,groupB,groupC)
fourth(groupA,groupB,groupC)
```

Conclusion:

Hence, we have learned the Array Data structure and how to use it to implement Set Operations.

Program Codes with sample output:

Questions:

1. What is Data structure?
2. What are the different types of Data Structure based on organizing method?
3. Explain Array is of which type of Data Structure?
4. What are different types array? 5. What is an Algorithm?

Practical No: 02(A)

Practical Title: Write a python program to store marks for N students.

Aim: - 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

Prerequisite:

- Python Programming

Objectives:

To understand the use functions for N students record.

Input: N number of students.

Outcome: Resulting average, highest and lowest marks operation.

Theory:

List Data Structure:

- A **list** is a data structure in Python that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ].

- Lists are great to use when you want to work with many related values. They enable you to keep data together that belongs together, condense your code, and perform the same methods and operations on multiple values at once.

- When thinking about Python lists and other data structures that are types of collections, it is useful to consider all the different collections you have on your computer: your assortment of files, your song playlists, your browser bookmarks, your emails, the collection of videos you can access on a streaming service, and more.

- To get started, let's create a list that contains items of the string data type:

```
sea_creatures = ['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
```

When we print out the list, the output looks exactly like the list we created:

```
print(sea_creatures)
```

Indexing Lists:

- Each item in a list corresponds to an index number, which is an integer value, starting with the index number 0.

- For the list sea_creatures, the index breakdown looks like this:

‘shark’	'cuttlefish’	'squid’	'mantis shrimp’	'anemone’
0	1	2	3	4

Dictionary

Dictionaries are used to store **key-value** pairs. To understand better, think of a phone directory where hundreds and thousands of names and their corresponding numbers have been added. Now the constant values here are Name and the Phone Numbers which are called as the keys. And the various names and phone numbers are the values that have been fed to the keys. If you access the values of the keys, you will obtain all the names and phone numbers. So that is what a key-value pair is. And in Python, this structure is stored using Dictionaries. Let us understand this better with an example program.

Creating a Dictionary

- Dictionaries can be created using the flower braces or using the dict() function. You need to add the key-value pairs whenever you work with dictionaries.

```
1
2
3
4
my_dict = {} #empty dictionary
print(my_dict)
my_dict = {1: 'Python', 2: 'Java'} #dictionary with elements
print(my_dict)
```

Output: {}
{1: ‘Python’, 2: ‘Java’}

Tuple: Tuples are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed. The example program will help you understand better.

Creating a Tuple:

You create a tuple using parenthesis or using the tuple() function.

```
1
2
my_tuple = (1, 2, 3) #create tuple
print(my_tuple)
```

Output:
(1, 2, 3)

Set Data Structure:

- Sets are a collection of unordered elements that are unique. Meaning that even if the data is repeated more than one time, it would be entered into the set only once. It resembles the sets that you have learnt in arithmetic. The operations also are the same as is with the arithmetic sets. An example program would help you understand better.

Creating a set

Sets are created using the flower braces but instead of adding key-value pairs, you just pass values to it.

```
1  my_set = {1, 2, 3, 4, 5, 5, 5} #create set
2  print(my_set)
```

Output:

{1, 2, 3, 4, 5}

Functions Used :

Write algorithm/pseudo code for each function.

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

```
print("\n      1 : Accept FDS Marks\n
              2 : Average score of class\n
              3 : Highest score and lowest score of class\n
              4 : Count of students who were absent for the test\n
              5 : Display mark with highest frequency\n
              6 :      Exit\n
              ")

def accept():
        fds=[]
        n=int(input("Enter the Student count of class:"))
        for i in range(n):
                print("Enter the marks scored in FDS for student %i: "%(i+1))
                var=int(input())
                fds.append(var)
        print("\n1. Marks stored in fds:")
```

```
        for i in range(n):
                print("Student %i:"%(i+1),fds[i])
        return fds

def average(fds):

        sum=0
        for i in range(len(fds)):
                if(fds[i]!=-1):
                        sum=sum+fds[i]
        avg=sum/len(fds)

        print("2. Average score of class:",avg)
        print("SUM=", sum)

def highlow(fds):
        print("3. Highest score and lowest score of class:")
        large=fds[0]
        for i in range(len(fds)):
                if(large<fds[i]):
                        large=fds[i]
        print("Highest marks of class:",large)

        for i in range (len(fds)):
                if(fds[i]!=-1):
                        small=fds[i]
                        break
        for j in range(len(fds)):
                if(fds[j]!=-1):
                        if(small>fds[j]):
                                small=fds[j]
        print("Smallest marks of class:",small)

def absent(fds):
        count=0
        print("4. Count of students who were absent for the test: ")
        for i in range(len(fds)):
                if(fds[i]==-1):
                        count=count+1
                        print("Absent Student roll is :%i"%(i+1))

        print("Absent student count is:",count)


def highfre(fds):
        max1=0

        for i in range(len(fds)):
                if(fds[i]!=-1):
                        var=fds.count(fds[i])
```

```
                if(var>max1):
                        max1=var
                        res=fds[i]


        print("5. Mark with highest frequency",res)
        print("count:",max1)

while True:

        ch=int(input("Enter your choice:"))
        if(ch==1):
                fds=accept()
        elif(ch==2):
                average(fds)
        elif(ch==3):
                highlow(fds)
        elif(ch==4):
                absent(fds)
        elif(ch==5):
                highfre(fds)
        if(ch==6):
                break
```

Conclusion:

Hence, we have learned and understand the data structures in python.

A	P	J	Total	Dated Sign
3	4	3	10	

Questions:

1. Basics of python Programming.
2. What are functions?
3. What are the features of python Programming?
4. What are array?

Practical No: 3 (A)

Practical Title: Write a python program to perform String operations.

Aim: 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

Pre-requisite:

Basics of String operations.

Objectives:

- To understand the use standard library functions for string operations.
- To perform the string operations.

Input: One or Two Strings

Output: Resulting string after performing string operation.

Theory :

What is String in Python?

- ✓ A string is a sequence of characters.
- ✓ A character is simply a symbol. For example, the English language has 26 characters.
- ✓ Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0s and 1s.
- ✓ This conversion of character to a number is called encoding, and the reverse process is
- ✓ Strings can be created by enclosing characters inside a single quote or double-quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

How to create a string in Python?

```
# defining strings in Python
# all of the following are equivalent
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
        the world of Python"""
print(my_string)
```

When you run the program, the output will be:

```
Hello
Hello
Hello
Hello, welcome to
        the world of Python
```

How to access characters in a string?

- ✓ We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an IndexError. The index must be an integer. We can't use floats or other types, this will result into TypeError.
- ✓ Python allows negative indexing for its sequences.
- ✓ The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator :(colon).

```
#Accessing string characters in Python
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
```

Parvatibai Genba Moze College of Engineering Wagholi, Pune

```
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

When we run the above program, we get the following output:

```
str = programiz
str[0] =  p
str[-1] = z
str[1:5] = rogr
str[5:-2] =  am
```

String operation (explain each operation in detail with example)

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

```
print('''Write a python program to compute following operations on String:
    1)To display word with the longest length
    2)To determines the frequency of occurrence of particular character in the string
    3)To check whether given string is palindrome or not
    4)To display index of first appearance of the substring
    5)To count the occurrences of each word in a given string
    6)exit''')

def one():

    str1=input("Enter the string:")
    list1=str1.split()     #spilt the string and store into list as single element
    m=0
    word=0
    print(list1)
    for i in range(len(list1)):
        len(list1[i])
        if m<len(list1[i]):
            m=len(list1[i])
            word=i
    print("a) The word with longest length:",list1[word])
```

```
def two():
    str1=input("Enter the string:")
    char=input("Enter  character")
    print(str1)
    counter=0
    for i in range(len(str1)):
        if char==str1[i]:
            counter+=1  #counter=counter+1

    print("Character %s is present %i times in string %s"% (char,counter,str1))


def three():
    str2 = input("Enter string")
    lenstr2=len(str2)
    j=lenstr2-1

    flag=0
    for i in range(int (lenstr2/2+1)):
        if(str2[i]==str2[j]):
            flag=1
        else:
            break
        j=-1

    if(flag==1):
        print("It is palindrome")
    else:
        print("it is not palindrome")

def four():
    str1 = input("Enter the string:") #i am ganesh 11
    sub1=input("Enter substring:")     #ane
    sublen=len(sub1)     //3
    index1=0
    j=0
    for i in range(len(str1)):
        if(sub1[j]==str1[i]):

            j=j+1
            if(j==sublen):
                index1=i-(sublen-1)
                break
        else:
            j=0
```

```
    print("substring index :", index1)

def five():
    str1 = input("Enter input string:")
    list1= str1.split()
    list2=set(list1)                #Delete duplicates \n", \
    list3=list(list2)               #convert set again into List\n", \
    print(list1)
    print(list3)
    list4=[]
    list5=[]
    counter=0
    for i in range(len(list3)):
        counter=0
        for j in range(len(list1)):
            if(list3[i]==list1[j]):
                counter+=1
        list4=list3[i],counter
        list5.append(list4)

    print(list5)

while True:
    ch=int(input("Enter your choice:"))
    if(ch==1):
        one()
    elif(ch==2):
        two()
    elif(ch==3):
        three()
    elif(ch==4):
         four()
    elif(ch==5):
        five()
    elif(ch==6):
        break
```

Conclusion:

Hence, we studied and implemented String operation in Python.

A	P	J	Total	Dated Sign
3	4	3	10	

Questions:

1. Write a program to find the length of string.
2. Write a program to display string from backward.
3. Write a program to count number of words in string.
4. Write a program to concatenate one string contents to another.
5. Write a program to compare two strings they are exact equal or not.
6. Write a program to check a string is palindrome or not.
7. WAP to find a substring within a string. If found display its starting position.
8. Write a program to reverse a string.
9. Write a program to convert a string in lowercase.
10. Write a program to convert a string in uppercase.

Practical No : 04 (A)

Practical Title: Python program that computes the net amount of a bank account.

Aim :

A-4 Write a **Python** program that computes the net amount of a bank account based a transaction log from console input. The transaction log format is shown as following: D 100 W 200 (Withdrawal is not allowed if balance is going negative. Write functions for withdraw and deposit) D means deposit while W means withdrawal.

Suppose the following input is supplied to the program:

D 300, D 300 , W 200, D 100 Then, the output should be: 500

Pre-requisite:

Knowledge of representing function in Python

Objectives:

To understand the concept of functions in python

Input :

Input is supplied to the program:

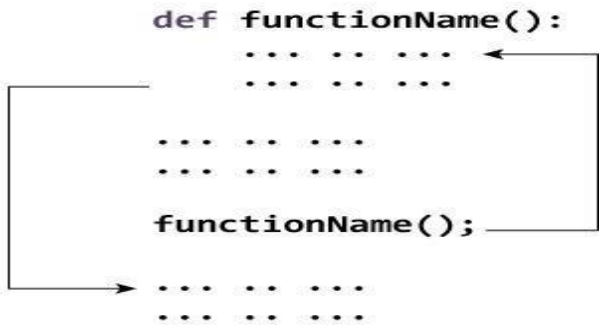
D 300, D 300 , W 200, D 100 Then, the output should be: 500

Outcome:

D 300, D 300 , W 200, D 100 Then, the **output should be: 500**

Theory:

How Function works in Python?

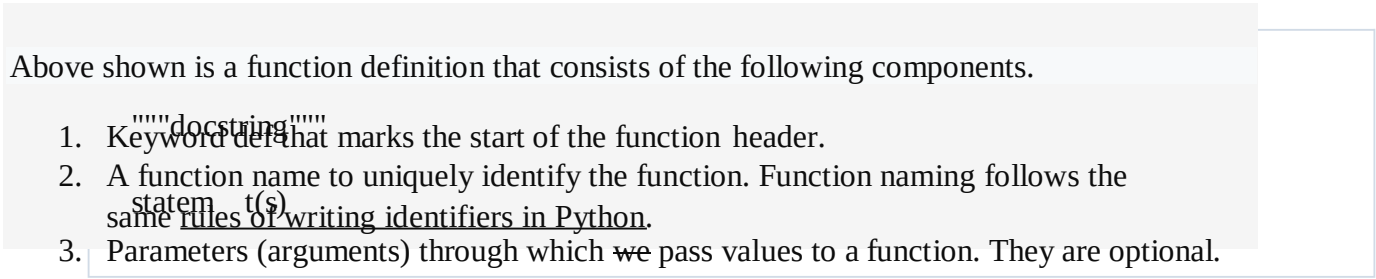


What is a function in Python?

- ✓ In Python, a function is a group of related statements that performs a specific task.
- ✓ Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

- ✓ Furthermore, it avoids repetition and makes the code reusable.

Syntax of Function



Above shown is a function definition that consists of the following components.

1. Keyword def that marks the start of the function header.
2. A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of the function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.

Example of a function

```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
```

How to call a function in python?

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

```
>>> greet('Paul')
Hello, Paul. Good morning!
```

Note: Try running the above code in the Python program with the function definition to see the output.

```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")

greet('Paul')
```

The return statement

The `return` statement is used to exit a function and go back to the place from where it was called.

Syntax of return

```
return [expression_list]
```

This statement can contain an expression that gets evaluated and the value is returned. If there is no expression in the statement or the `return` statement itself is not present inside a function, then the function will return the `None` object.

For example:

```
>>> print(greet("May"))
Hello, May. Good morning!
None
```

Here, `None` is the returned value since `greet()` directly prints the name and no `return` statement is used.

Example of return

```
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""

    if num >= 0:
        return num
```

```
    else:
        return -num


print(absolute_value(2))

print(absolute_value(-4))
```

Output

Types of Functions

2

Basically, we can divide functions into the following two types:

1. Built-in functions - Functions that are built into Python.
2. User-defined functions - Functions defined by the users themselves.

Implementation:

```
print( '''
    Write a python program that computes the net amount of a bank account based a transaction log from console input.
    The transaction log format is shown as following:
    D 100 W 200 (Withdrawal is not allowed if balance is going negative. Write functions for withdraw and deposit) D means deposit while W means withdrawal.
    Suppose the following input is supplied to the program:
    D 300 , D 300 , W 200 , D 100 ,
    Then, the output should be: 500
  ''')

def deposit(num):
    global balance
    balance =balance+num

def withdrawal(num):
    global balance
    if(balance >0):
        balance=balance-num
    else:
        print("Withdraw not possible because balance is less.")
```

```
list1 = []

balance=0
while True:
    data = input("Please enter the transaction details:")
    if ('Exit' == data):
        break
    list1.append(data.split())
print(list1)

for var in list1:
    if(var[0]=='D'):
        deposit(int(var[1]))
    elif (var[0]=='W'):
        withdrawal(int(var[1]))


print("Balance =",balance)
```

Conclusion:

Hence, we studied, how to use functions in python programming**.**

A	**P**	**J**	**Total**	**Dated Sign**
3	4	3	10	

GROUP - B

Practical No: 5(B)

Practical Title: Python Program on Linear search, Sentinel search, Binary search and Fibonacci search

Aim : B-11) a) Write a **Python** program to store roll numbers of student in array who attended training program in random order. Write function for searching whether particular student attended training program or not, using Linear search and Sentinel search.

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

Pre-requisite:

Knowledge of Fibonacci series, and basic searching concepts.

Objectives:

- To search record using linear, binary and sentinel searching.

Input:

- Array Record for storing the record.
- Search key elements.

Output:

If Searched Sufessfully then function returns index of searched key otherwise return "Search key not present in the list"

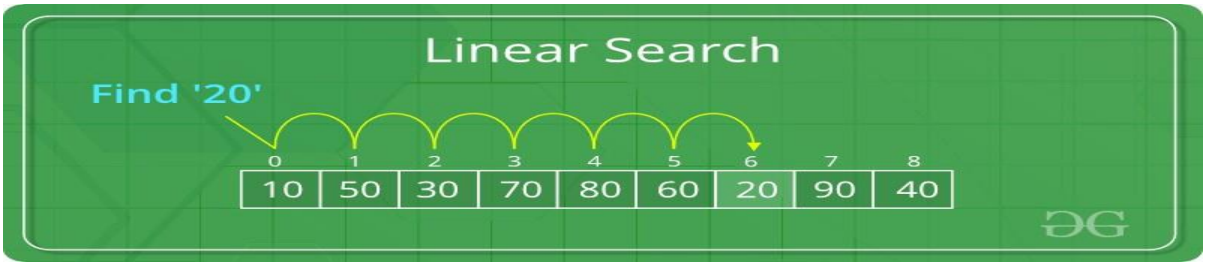
Theory:

1. Linear Search:

In computer science, a **linear search** or **sequential search** is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

A simple approach is to do a **linear search**, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.



The **time complexity** of the above algorithm is O(n).

Pseudocode

```
procedure linear_search (list, value)

   for each item in the list
      if match item == value
         return the item's location
      end if
   end for

end procedure
```

Complexity of algorithm

Complexity	Best Case	Average Case	Worst Case
Time	O(1)	O(n)	O(n)
Space			O(1)

Linear Search in Python:

```
def linearsearch(arr, x):
   for i in range(len(arr)):                    ---  it

      if arr[i] == x:                    --  n      ====   2n+1    =  O(n)
         return i
   return -1
arr = ['t','u','t','o','r','i','a','l']
x = 'a'
m=linearsearch(arr,x)
if(m==-1):
            print("Elementi is not found in list")
else:
            printf("")
```

2. Sentinel Linear Search

Linear Search : The idea behind linear search is to compare the search item with the elements in the list one by one (using a loop) and stop as soon as we get the first copy of the search element in the list. Now considering the worst case in which the search element does not exist in the list of size **N** then the **Simple Linear Search** will take a total of **2N+1** comparisons (**N** comparisons against every element in the search list and **N+1** comparisons to test against the end of the loop condition).

Sentinel Linear Search : Here the idea is to reduce the number of comparisons required to find an element in a list. Here we replace the last element of the list with the search element itself and run a **while loop** to see if there exists any copy of the search element in the list and quit the loop as soon as we find the search element. See the code snippet for clarification.

	15	30	10	7	8	9
	0	1	2	3	4	5

X=12

I=5

Last=a[n-1]=9

```
int last = array[N-1];
array[N-1] = item;    // Here item is the search element.
int i = 0;
while(array[i]!=item)                    n
{
   i++;
}
array[N-1] = last;
if( (i < N-1) || (item == array[N-1]) )-------- 2  ======== n+2,,,,,,,,,2n+1        >>>O(n)
{
   cout << " Item Found @ "<<i;
}
else
{
   cout << " Item Not Found";
}
```

Here we see that the **while loop** makes only one comparison in each iteration and it is sure that it will terminate since the last element of the list is the search element itself. So in the worst case (if the search element does not exists in the list) then there will be at most **N+2** comparisons

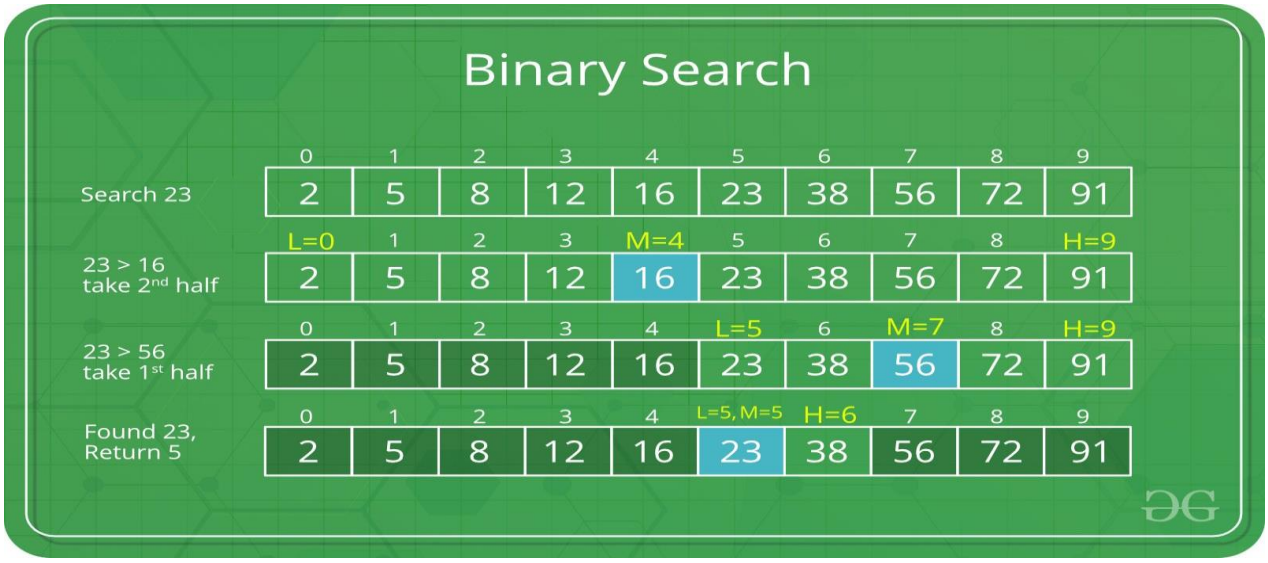
(**N** comparisons in the while loop and **2** comparisons in the if condition). Which is better than (**2N+1**) comparisons as found in **Simple Linear Search**.

Take note that both the algorithms have time complexity of **O(n)**.

3. Binary Search

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].

A simple approach is to do linear search**.** The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.
Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.
Example :



The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Python Implementation

```
# Iterative Binary Search Function method Python Implementation
# It returns index of n in given list1 if present,
# else returns -1
def binary_search(list1, n):
```

```
    low = 0
    high = len(list1) - 1
    mid = 0

    while low <= high:
        # for get integer result
        mid = (high + low) // 2

        # Check if n is present at mid
        if list1[mid] < n:
            low = mid + 1

        # If n is greater, compare to the right of mid
        elif list1[mid] > n:
            high = mid - 1

        # If n is smaller, compared to the left of mid
        else:
            return mid

            # element was not present in the list, return -1
    return -1


# Initial list1
list1 = [12, 24, 32, 39, 45, 50, 54]
n = 45

# Function call
result = binary_search(list1, n)

if result != -1:
    print("Element is present at index", str(result))
else:
    print("Element is not present in list1")
```

Output:

```
Element is present at index 4
```

Calculating Time complexity:

- Let say the iteration in Binary Search terminates after **k** iterations. In the above example, it terminates after 3 iterations, so **here k = 3**
- At each iteration, the array is divided by half. So let's say the length of array at any iteration is **n**
- At **Iteration 1**,
  Length of array = **n**
- At **Iteration 2**,
  Length of array = $n/2$
- At **Iteration 3**,
  Length of array = $(n/2)/2 = n/2^2$
- Therefore, after **Iteration k**,
  Length of array = $n/2^k$
- Also, we know that after
  After k divisions, the **length of array becomes 1**
- Therefore
- Length of array = $n/2^k = 1$
- => $n = 2^k$
- Applying log function on both sides:
- => $\log_2(n) = \log_2(2^k)$
- => $\log_2(n) = k \log_2(2)$
- As $(\log_a(a) = 1)$
  Therefore,
- => $k = \log_2(n)$

Hence, the time complexity of Binary Search is
$\log_2(n)$

Worst Case & Average Case time Complexity: log2(n)
Best case: O(1)

4. Fibonacci Search:

Given a sorted array arr[] of size n and an element x to be searched in it. Return index of x if it is present in array else return -1.

Examples:

Input: arr[] = {2, 3, 4, 10, 40}, x = 10

Output: 3

Element x is present at index 3.

Input: arr[] = {2, 3, 4, 10, 40}, x = 11

Output: -1

Element x is not present.

Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.

Similarities with Binary Search:

1. Works for sorted arrays
2. A Divide and Conquer Algorithm.
3. Has Log n time complexity.

Differences with Binary Search:

1. Fibonacci Search divides given array into unequal parts.
2. Binary Search uses a **division** operator to divide range. Fibonacci Search doesn't use /, but **uses + and -.** The division operator may be costly on some CPUs.
3. Fibonacci Search examines relatively closer elements in subsequent steps. So when the input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful.

Algorithm:

- Let the searched element be x.
- The idea is to first find the smallest Fibonacci number that is greater than or equal to the length of the given array. Let the found Fibonacci number be fib (m'th Fibonacci number). We use (m-2)'th Fibonacci number as the index (If it is a valid index). Let (m-2)'th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for subarray after i, else we recur for subarray before i.

Below is the complete algorithm
Let arr[0..n-1] be the input array and the element to be searched be x.

Algorithm Steps:

1. Find the smallest Fibonacci Number greater than or equal to n. Let this number be fibM [m'th Fibonacci Number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)'th Fibonacci Number] and fibMm2 [(m-2)'th Fibonacci Number].
2. While the array has elements to be inspected:
    1. Compare x with the last element of the range covered by fibMm2
    2. **If** x matches, return index
    3. **Else If** x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
    4. **Else** x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate the elimination of approximately front one-third of the remaining array.

Example:

15	25	27	38	45	85	110
0	1	2	3	4	5	6

Fibonacci Series: 0,1,1,2,3,5,8,
N=13 M=8 **m1=5 & m2=3** offset=0 and Searched Element (x) = 45

i=m2+offset==== 3

Compare x with Range of M2

1. If (x==a[i]) return index;

2. Else if (x>a[i]) move m 1 position down

M=5,m1=3, m2=2
m=m1; m1=m2; m2=m-m1
Offset=m2=3

3. Else if(x<a[i]) move m 2 position down
M=m2; m1=m-1; m2=m-m1
M=2 m1=1, m2=1

Complexity:
Worse Case Time Complexity: O(logn)

i	1	2	3	4	5	6	7	8	9	10	11	12	13
ar[i]	**10**	**22**	**35**	**40**	**45**	**50**	**80**	**82**	**85**	**90**	**100**	-	-

fibMm2	*fibMm1*	*fibM*	*offset*	i=min(offset+fibLn)	*arr[i]*	*Consequence*
5	8	13	0	5	45	Move one down, reset offset
3	5	8	5	8	82	Move one down, reset offset
2	3	5	8	10	90	Move two down
1	1	2	8	9	85	Return i

Conclusion:

Hence, e Studied Linear Search, Binary Search, Fibonacci Search, Sentinel Search

A	**P**	**J**	**Total**	**Dated Sign**
3	4	3	10	

Practical No:06(B)

Practical Title: Sorting of an array using **selection and bubble sort.**

Aim: 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.of club**.**

Pre-requisite:

Knowledge of sorting techniques

Objective:

- To sort array of floating point numbers in ascending order using Selection Sort b) Bubble sort and
- Display top five scores.

Input:

- Size of array Elements of array

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 in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner.

Sorting is also used to represent data in more readable formats.

Following are some of the examples of sorting in real-life scenarios −

- **Telephone Directory** − The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.
- **Dictionary** − The dictionary stores words in an alphabetical order so that searching of any word becomes easy.

Internal and External Sorting:

An **internal sort** is any data sorting process that takes place entirely within the main memory of a computer. This is possible whenever the data to be sorted is small enough to all be held in the main memory.

Some common internal sorting algorithms include:

1. Bubble Sort
2. Insertion Sort
3. Quick Sort
4. Heap Sort
5. Radix Sort
6. Selection sort

External sorting is a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory.

Example: Merge Sort

Comparison Based Sorting:

1. Bubble Sort
2. Selection sort
3. Merge Sort
4. Insertion Sort
5. Quick Sort
6. Shell Sort

Non-comparison Based Sorting Methods-

1. Radix Sort,
2. Counting Sort,
3. Bucket Sort

1. Bubble sort:

- Bubble sort is a simple sorting algorithm.
- This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order
- Bubble sort will start by comparing the first element of array with the second elements. if the first element is greater than the second elements, it will swap the both elements and then move on to compare second and third elements and so on .
- It is known as bubble sort because every complete iteration the largest element in the given array, move towards the last place and or highest index and small small elements bubbles up toward the upward directions just like water bubble rises up to the water surface.

Examples:

If the elements are n then n-1 passes are required to sort the bubble sort.

ass 1						
	15	**7**	**9**	**25**	**-3**	**7**
Step1	**7**	**15**	**9**	**25**	**-3**	**7**
Step2	**7**	**9**	**15**	**25**	**-3**	**7**
Step3	**7**	**9**	**15**	**25**	**-3**	**7**
Step4	**7**	**9**	**15**	**-3**	**25**	**7**
Step5	**7**	**9**	**15**	**-3**	**7**	**25**

Pass 2	**0**	**1**	**2**	**3**	**4**	**5**
	7	**9**	**15**	**-3**	**7**	**25**
Step1	**7**	**9**	**15**	**-3**	**7**	**25**
Step2	**7**	**9**	**15**	**-3**	**7**	**25**
Step3	**7**	**9**	**-3**	**15**	**7**	**25**
Step4	**7**	**9**	**-3**	**7**	**15**	**25**

Pass 3	0	1	2	3	4	5
	7	9	-3	7	15	25
Step1	7	9	-3	7	15	25
Step2	7	-3	9	7	15	25
Step3	7	-3	7	9	15	25

Pass 4	0	1	2	3	4	5
	7	-3	7	9	15	25
Step1	-3	7	7	9	15	25
Step2	-3	7	7	9	15	25

Pass 5	0	1	2	3	4	5
	-3	7	7	9	15	25
Step1	-3	7	7	9	15	25

Analysis of Bubble Sort:

Total number of Comparison:
=(n-1)+(n-2)+(n-3)+……+3+2+1
=n(n+1)/2
=O(n2)
If in improved version of bubble sort if record already sorted: O(n)

Best Case, Average Case and Worst-Case complexity: O(n2)

2. Selection Sort:
- in place comparison sorting.
- Record is divided into two subparts:
    1. Sorted record
    2. Un sorted
- We select the smallest element from the record and place it at the beginning of the record.
- Initially sorted subarray is empty, whole list/ record is unsorted subarray.

If elements are n then pass: n-1 7 , 6

Pass 1	18	5	0	7	-8	20	3
Step1	5	18	0	7	-8	20	3
Step2	0	18	5	7	-8	20	3
Step3	0	18	5	7	-8	20	3
step4	-8	18	5	7	0	20	3
Step5	-8	18	5	7	0	20	3

Step6	**-8**	**18**	**5**	**7**	**0**	**20**	**3**

Pass2	-8	18	5	7	0	20	3
Step1	-8	5	18	7	0	20	3
Step2	-8	5	18	7	0	20	3
Step3	-8	0	18	7	5	20	3
Step4	-8	0	18	7	5	20	3
Step5	-8	0	18	7	5	20	3

Pass3	-8	0	18	7	5	20	3
Step1	-8	0	7	18	5	20	3
Step2	-8	0	5	18	7	20	3
Step3	-8	0	5	18	7	20	3
Step4	-8	0	3	18	7	20	5

Pass4	-8	0	3	18	7	20	5
Step1	-8	0	3	7	18	20	5
Step2	-8	0	3	7	18	20	5
Step3	-8	0	3	5	18	20	7

Pass5	-8	0	3	5	18	20	7
Step1	-8	0	3	5	18	20	7
Step2	-8	0	3	5	7	20	18

Pass6	-8	0	3	5	7	20	18
Step1	-8	0	3	5	7	18	20

Analysis: n
=(n-1)+(n-2)+(n-3)+……+3+2+1
=n(n-1)/2 =(n^2-n)/2 = O(n2)
Best, Average, Worse : O(n^2)

Algorithm

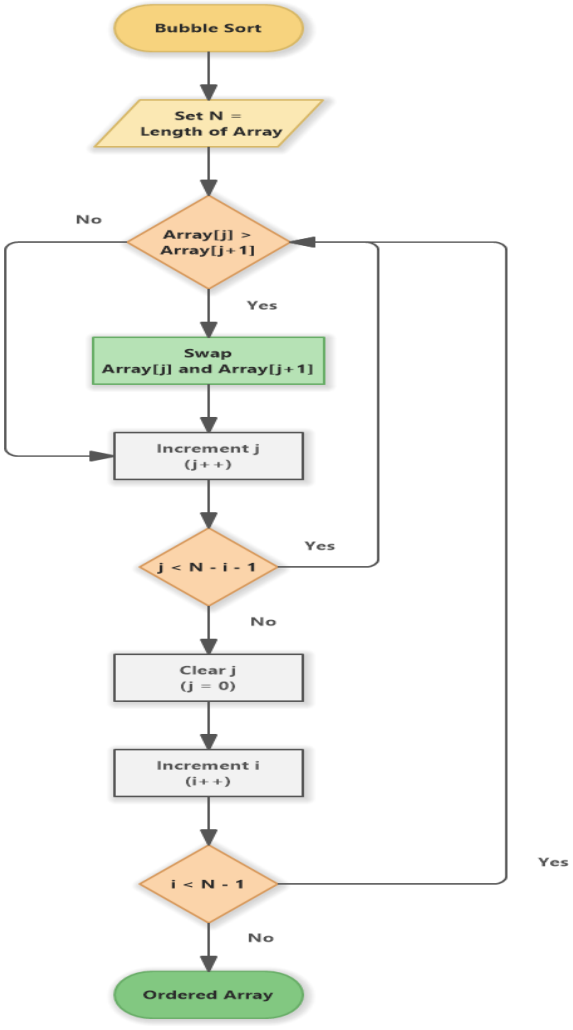
We assume **list** is an array of **n** elements. We further assume that **swap** function swaps the values of the given array elements.

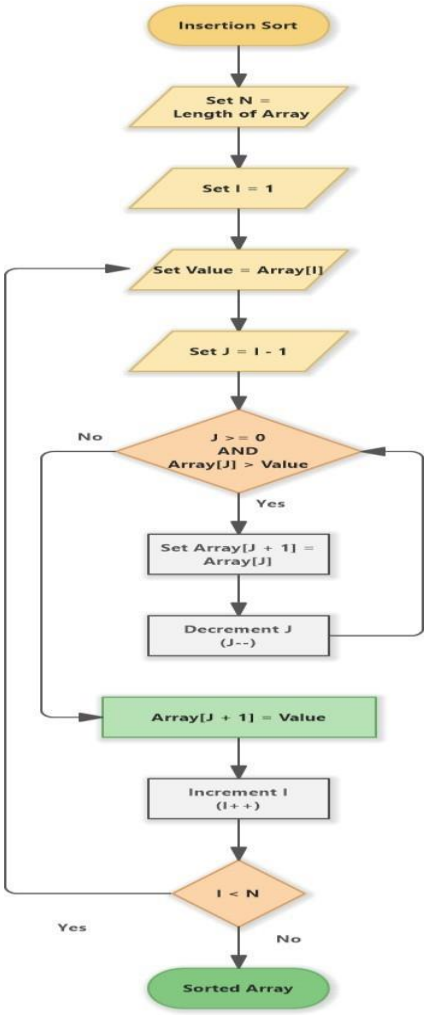
```
begin BubbleSort(list)

   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
```

Selection Sort Algorithm

```
selectionSort(array, size)
  repeat (size - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
end selectionSort
```

Flowchart:





Conclusion:

Hence, we studied sorting of an array using selection and bubble sort.

A	P	J	Total	Dated Sign
3	4	3	10	

Question Bank:

1. Explain the sorting?
2. What are the different types of sorts in datastructures
3. .Define the bubble sort?
4. Define the selection sort?
5. How many passes are required in selection sort?
6. What is the time complexity of selecion and bubble sort?

Practical No:07(B)

Practical Title: Sorting array of floating-point numbers in ascending order using **quick sort**.

Aim: 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.

Pre-requisite:

- Knowledge of sorting techniques

Objective:

- To sort array using quick sort

Input: Size of array

First year percentage of students.

Outcome:

- To sort array using quick sort
- To display top five scores.

Theory :

Quick Sort: (partition Exchange Sort)

- ✓ Quick Sort is a Divide and Conquer algorithm.
- ✓ It picks an element as pivot and partitions the given array around the picked pivot.
- ✓ There are many different versions of quickSort that pick pivot in different ways.
- ✓ Always pick first element as pivot.
- ✓ Always pick last element as pivot
- ✓ Pick a random element as pivot.
- ✓ Pick median as pivot.
- ✓ The key process in quickSort is partition().
- ✓ Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Examples:

	0	1	2	3	4	5	6	7	8
a	25	9	11	21	95	0	17	5	32

P=0,i=0, j=8 h=0,h=8
a[i]<=a[p] i++ if[j]>a[p] j—
if(i<j) swap I and j

if I and j cross
then swap p with j

	0	1	2	3	4	5	6	7	8
	17	9	11	21	5	0	25 j	95 i	32

a(l,j-1) a(j+1,h)

Complexity of Quicksort:

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

$T(n) = T(0) + T(n-1) + (n)$

which is equivalent to

$T(n) = T(n-1) + (n)$

The solution of above recurrence is $O(n_2)$.

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

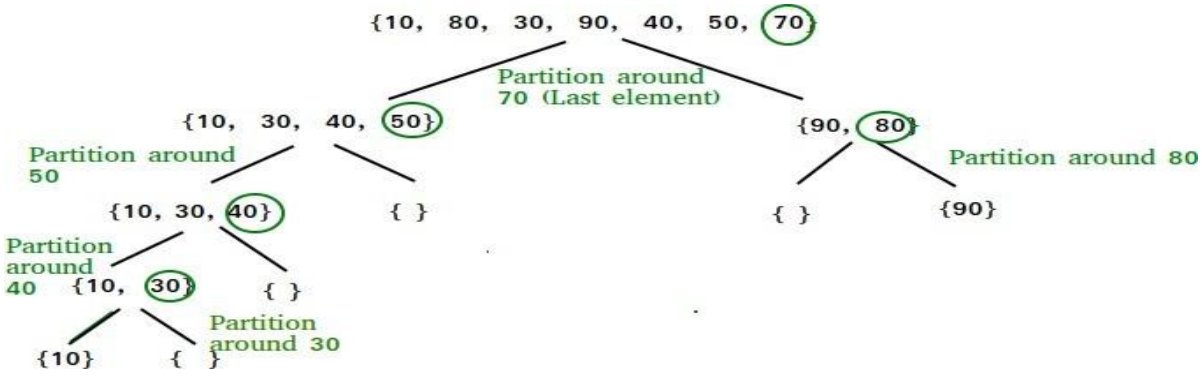
$T(n) = 2T(n/2) + (n)$

The solution of above recurrence is O(nLogn).

Average Case:
To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.
We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

$T(n) = T(n/9) + T(9n/10) + (n)$

Solution of above recurrence is also O(nLogn)



Pseudocode of Quick Sort Algorithm:

Quick Sort

```
/**
* The main function that implements quick sort.
* @Parameters: array, starting index and ending index
*/
quickSort(arr[], low, high)
{
   if (low < high)
   {
      // pivot_index is partitioning index, arr[pivot_index] is now at correct place in sorted array
      pivot_index = partition(arr, low, high);

      quickSort(arr, low, pivot_index - 1); // Before pivot_index
      quickSort(arr, pivot_index + 1, high); // After pivot_index
   }
}
```

Partition Method

```
/**
* The function selects the last element as pivot element, places that pivot element correctly in the arr
ay in such a way
* that all the elements to the left of the pivot are lesser than the pivot and
* all the elements to the right of pivot are greater than it.
* @Parameters: array, starting index and ending index
* @Returns: index of pivot element after placing it correctly in sorted array
*/
partition (arr[], low, high)
{
   // pivot - Element at right most position
   pivot = arr[high];
   i = (low - 1); // Index of smaller element
   for (j = low; j <= high-1; j++)
   {
      // If current element is smaller than the pivot, swap the element with pivot
      if (arr[j] < pivot)
      {
         i++;    // increment index of smaller element
         swap(arr[i], arr[j]);
      }
   }
   swap(arr[i + 1], arr[high]);
   return (i + 1);
}
```

 # Python program for implementation of Quicksort Sort

```
# This function takes last element as pivot, places
# the pivot element at its correct position in sorted
# array, and places all smaller (smaller than pivot)
# to left of pivot and all greater elements to right
# of pivot


def partition(arr, low, high):
    i = (low-1)         # index of smaller element
    pivot = arr[high]   # pivot

    for j in range(low, high):

        # If current element is smaller than or
        # equal to pivot
        if arr[j] <= pivot:

            # increment index of smaller element
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort


def quickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr, low, high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)


# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
```

```
n = len(arr)
quickSort(arr, 0, n-1)
print("Sorted array is:")
for i in range(n):
        print("%d" % arr[i]),

# This code is contributed by Mohit Kumra
#This code in improved by https://github.com/anushkrishnav
```

Conclusion:

Hence We Studied Quick Sort algorithm with examples.

A	P	J	Total	Dated Sign
3	4	3	10	

Question Bank:

1. Explain the sorting?
2. What are the different types of sorts in data structures?
3. Define the quick sort?
5. How many passes are required in quick sort? 6. What is the time complexity of quick sort?

GROUP - C

Practical No : 08 (C)

Assignment Title: Write C++ program to maintain club member ‘s information using singly linked list.

Aim: Department of Computer Engineering has student's 1/8 club named 'Pinnacle Club'. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member‘s information using singly linked list. Store student PRN and Name. Write functions to

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

Input: Individual details

Output: Maintain information of the Club member's

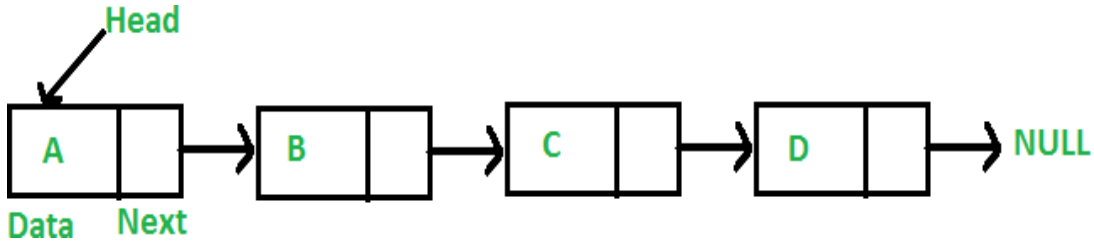
Objectives:

- To maintain club member's information by performing different operations like add, delete, reverse, concatenate on singly linked list.

Theory:

Linked List :

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

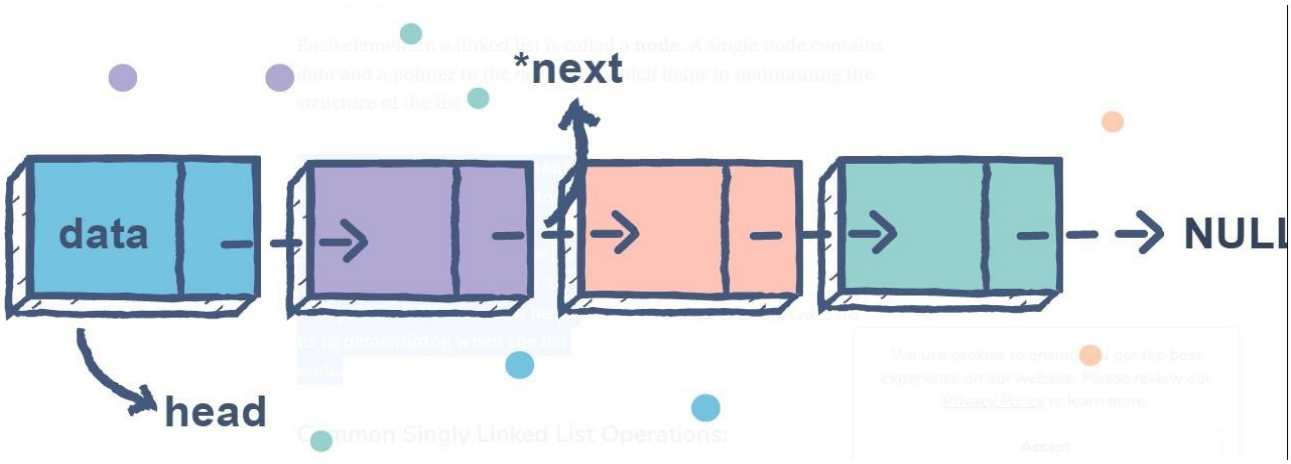
TYPES OF LINKED LIST:

There are 3 different implementations of Linked List available, they are:

1. Linear/Singly Linked List
2. Doubly Linked List
3. Circular Linked List

1. Linear/Singly Linked List:

- It is the basic form of the linked list.
- Each element in a linked list is called a **node**. A single node contains *data* and a pointer to the *next* node which helps in maintaining the structure of the list.
- A **singly linked list** is a type of linked list that is *unidirectional*, that is, it can be traversed in only one direction from head to the last node (tail).



- The first node is called the head; it points to the first node of the list and helps us access every other element in the list. The last node, also sometimes called the tail, points to *NULL* which helps us in determining when the list ends.

Primitive Operation performed on Singly Linked List:

- ❑ Create:
- ❑ Traverse:
- ❑ Search:
- ❑ Insert:
- ❑ Delete:
- ❑ Sort:
- ❑ Concatenation:

Advantages of Linked List

Dynamic Data Structure

- ❑ Linked list is a dynamic data structure so it can grow and shrink at runtime by allocating and deallocating memeory. So there is no need to give initial size of linked list.

Insertion and Deletion

- ❑ Insertion and deletion of nodes are really easier. Unlike array here we don't have to shift elements after insertion or deletion of an element. In linked list we just have to update the address present in next pointer of a node.

No Memory Wastage

- ❑ As size of linked list can increase or decrease at run time so there is no memory wastage. In case of array there is lot of memory wastage, like if we declare an array of size 10 and store

only 6 elements in it then space of 4 elements are wasted. There is no such problem in linked list as memory is allocated only when required.

Implementation

- Data structures such as stack and queues can be easily implemented using linked list.

Linked List Operation Pseudocode:

1. Creation of Linked List:

,we will assume that the linked list has three nodes 1 --->2 --->3 with node structure as below:

```
struct node
{
  int data;
  struct node *next;
};
```

2. Traverse a Linked List

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

3. Insert Elements to a Linked List

When temp
You can add elements to either the beginning, middle or end of the linked list.
while loop.

1. Insert at the beginning

```
struct node *temp = head;
printf("\n\nList elements are - \n");
wh
{
    p
    t
}
```

- Allocate memory for new node
- Store data
- Change next of new node to point to head

- Change head to point to recently created node

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
newNode->next = head;
head = newNode;
```

2. Insert at the End

- Allocate memory for new node
- Store data
- Traverse to last node
- Allocate memory and store data for new node

3. Insert at the Middle

- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
struct node *temp = head;
while(temp->next != NULL){
temp = temp->next;
}
temp->next = newNode;
```

```
for(int i=2; i < position; i++) {
   if(temp->next != NULL) {
        temp = temp->next;
    }
}
newNode->next = temp->next;
temp->next = newNode;
```

4. Delete from a Linked List

You can delete either from the beginning, end or from a particular position.

1. Delete from beginning

- Point head to the second node

```
head = head->next;
```

2. Delete from end

- Traverse to second last element
- Change its next pointer to null

3. Delete from middle

- Traverse to element before the element to be deleted
- Change next pointers to exclude the node from the chain

```
str
wh
te
}
```

```
for(int i=2; i< position; i++) {
   if(temp->next!=NULL) {
        temp = temp->next;
    }
}

temp->next = temp->next->next;
```

Conclusion:

Hence, we studied, how to maintain club member ‘s information using singly linked list and various operation of singly liked list.

A	P	J	Total	Dated Sign
3	4	3	10	

Questions: (No need to write answers)

1. What is a Linked list?
2. Can you represent a Linked list graphically?
3. How many pointers are required to implement a simple Linked list?
4. How many types of Linked lists are there?
5. How to represent a linked list node?
6. Describe the steps to insert data at the starting of a singly linked list.
7. How to insert a node at the end of Linked list?
8. How to delete a node from linked list?
9. How to reverse a singly linked list?
10. What is the difference between singly and doubly linked lists?
11. What are the applications that use Linked lists?
12. What will you prefer to use a singly or a doubly linked lists for traversing through a list of elements?

Practical No: 09 (C)

Practical Title:

The ticket booking system of Cinemax theater has to be implemented using C++ program. There are 10 rows and 7 seats in each row. Doubly circular linked list has to be maintained to keep track of free seats at rows. Assume some random booking to start with. Use array to store pointers (Head pointer) to each row. On demand

a) The list of available seats is to be displayed
b) The seats are to be booked
c) The booking can be cancelled.

Pre-requisite:

- Knowledge of Doubly Circular Linked List
- Representation of Circular Linked list
- Knowledge of ticket booking

Objective:

- To perform Doubly Circular linked list for cinemax ticket booking.
- To display available seats.
- To book and cancel seats

Input: Row no and seat no to book seat

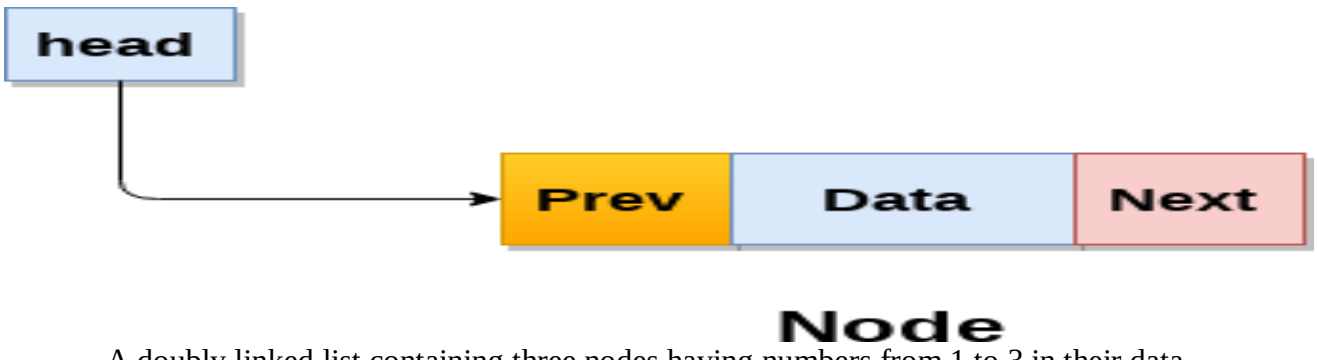
Outcome:

- Diplay available seats to book movie ticket.
- Display status of Booked seat/ cancel seat.

Theory:

Doubly linked list:

- Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence.
- Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.
- In doubly linked list, from every node the list can be traversed in both the directions.
-



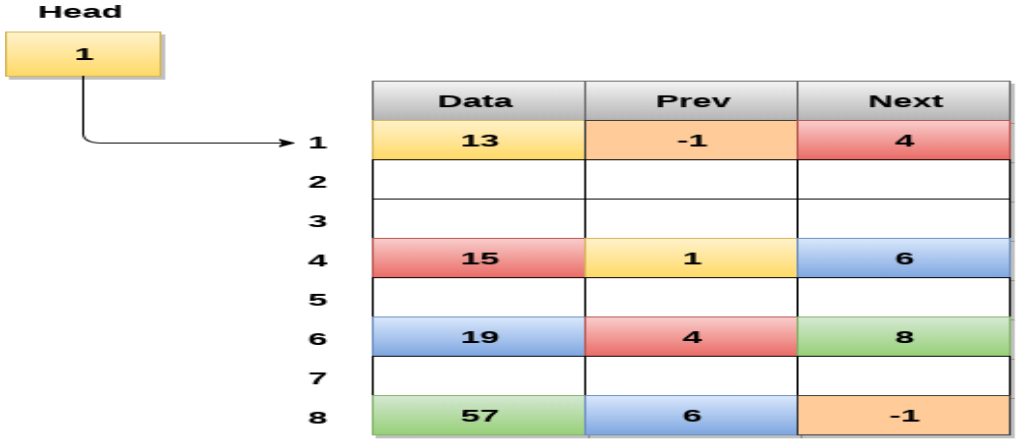
A doubly linked list containing three nodes having numbers from 1 to 3 in their data

part, is shown in the following image:

Doubly Linked List

```
struct node
{
   struct node *prev;
   int data;
   struct node *next;
}
```

Memory Representation of a doubly linked list:



Memory Representation of a Doubly linked list

Advantages of Doubly linked list:

Doubly linked list is one of the important data structures. Here are various advantages of doubly linked list.

1. As like singly linked list it is the easiest data structures to implement.
2. Allows traversal of nodes in both direction which is not possible in singly linked list.
3. Deletion of nodes is easy when compared to singly linked list, as in singly linked list deletion requires a pointer to the node and previous node to be deleted. Which is not in case of doubly linked list we only need the pointer which is to be deleted.
4. Reversing the list is simple and straightforward.
5. Can allocate or de-allocate memory easily when required during its execution.
6. It is one of most efficient data structure to implement when traversing in both direction is required.

Disadvantages of Doubly linked list:

Not many but doubly linked list has few disadvantages also which can be listed below:

1. It uses extra memory when compared to array and singly linked list.
2. Since elements in memory are stored randomly, hence elements are accessed sequentially no direct access is allowed.

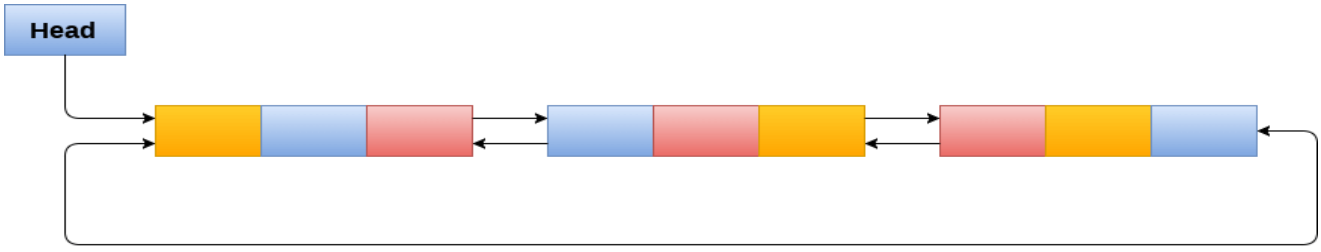
Applications/Uses of doubly linked list in real life:
There are various application of doubly linked list in the real world. Some of them can be listed as:

1. Doubly linked list can be used in navigation systems where both front and back navigation is required.
2. It is used by browsers to implement backward and forward navigation of visited web pages i.e. **back** and **forward** button.
3. It is also used by various application to implement **Undo** and **Redo** functionality.
4. It can also be used to represent deck of cards in games.
5. It is also used to represent various states of a game.

Circular Doubly Linked List

Circular doubly linked list is a more complexed type of data structure in which a node contains pointers to its previous node as well as the next node. Circular doubly linked list doesn't contain NULL in any of the node. The last node of the list contains the address of the first node of the list. The first node of the list also contains address of the last node in its previous pointer.

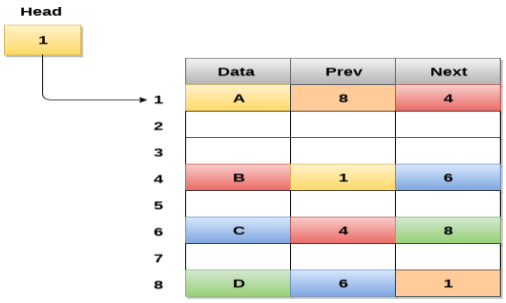
A circular doubly linked list is shown in the following figure:



Circular Doubly Linked List

Memory Management of Circular Doubly linked list

The following figure shows the way in which the memory is allocated for a circular doubly linked list. The variable head contains the address of the first element of the list i.e. 1 hence the starting node of the list contains data A is stored at address 1. Since, each node of the list is supposed to have three parts therefore, the starting node of the list contains address of the last node i.e. 8 and the next node i.e. 4. The last node of the list that is stored at address 8 and containing data as 6, contains address of the first node of the list as shown in the image i.e. 1. In circular doubly linked list, the last node is identified by the address of the first node which is stored in the next part of the last node therefore the node which contains the address of the first node, is actually the last node of the list.



Memory Representation of a Circular Doubly linked list

Operations on circular doubly linked list :

There are various operations which can be performed on circular doubly linked list. The node structure of a circular doubly linked list is similar to doubly linked list. However, the operations on circular doubly linked list is described in the following table.

SN	Operation	Description
1	Insertion at beginning	Adding a node in circular doubly linked list at the beginning.
2	Insertion at end	Adding a node in circular doubly linked list at the end.
3	Deletion at beginning	Removing a node in circular doubly linked list from beginning.
4	Deletion at end	Removing a node in circular doubly linked list at the end.

Traversing and searching in circular doubly linked list is similar to that in the circular singly linked list.

Implementation:

This program is a implementation of Ticket Booking System using Doubly Circular Linked List. Data structures used in the program- Doubly Circular Linked List.

```
-----------------------------------------
|                Platinum               |
| A1   A2   A3   A4   A5   A6   A7  |
| B1   B2   B3   B4   B5   B6   B7  |
| C1   C2   C3   C4   C5   C6   C7  |
-----------------------------------------
|                  gold                 |
| D1   D2   D3  D4   D5   D6   D7  |
| E1   E2   E3  E4   E5   E6   E7  |
| F1   F2    F3 F4    F5   F6   F7   |
| G1  G2   G3  G4   G5   G6   G7  |
| H1  H2   H3  H4   H5   H6   H7  |
-----------------------------------------
|                 Silver                |
| I1   I2    I3   I4    I5   I6    I7  |
| J1   J2    J3   J4    J5   J6    J7  |
-----------------------------------------
```

FUNCTIONS USED : -

1] create() :- To create 70 nodes internally (in memory), each node represent one seat.

Each node has 6 parameters

1) previous pointer of node* type to store address of previous node;
2) next pointer of node* type to store address of next node;
3) seat_no- to store seat number (int type)
4) row_no- to store row no(char type)
5)booking status- to show wether seat is alredy booked or available for booking(char type)
6)pin- different pin for each seat(int type)
So this function will create 70 nodes in memory locations & it will assign each node a seat_no,booking_status and pin.

2] display() :- To display the seating arrangement ie to display the linked list.
The display will show row number followe by seat no followe by booking status (‘a’ or ‘b’ -> available/booked) of each seat/node.
“In display additional lines of codes are added to make display more powerful for LINUX TERMINAL”

3] book_seat() :- it will take input from user as seat no. If booking status of that seat is ‘a’ then avaibility status of that seat will change from (a) to (b) . else it will show an error message. If Seat is booked succesfully the it will display pin for each seat.

4] cancle_seat() :- the purpose of assigning pin to each seat is only one who has booked the seat can cancel it. So at the time of cancelation pins are required .
Now at cancelation time it wil ask for seat number and pin if seat number and pin combination matches and the avaibility status of that seat is booked then that seat will be cancled. Else it will give an error message.

Conclusion:

Hence, we studied how to we can book or cancel movie ticket using doubly Circular linked lists.

A	P	J	Total	Dated Sign
3	4	3	10	

Questions: **(No need to write answers)**

1. What is Doubly Circular Linked List?
2. How to represent doubly circular linked list?
3. How to book or cancel seat?
3. What is doubly linked list?
4. How to insert and delete elements from doubly circulars linked list?

Practical No:10 (C)

Practical Title:

Write C++ program for storing appointment schedule for day. Appointments are booked randomly using **linked list**. Set start and end time and min and max duration for visit slot.Write functions for

a) Display free slots
b) Book appointment
c) Cancel appointment (check validity, time bounds, availability etc)
d) Sort list based on time
e) Sort list based on time using pointer

Pre-requisite:

- Basics of Singly Linked List
- Different Operations that can be perforemd on Singly linked list

Objective:

- To book or cancel appointment using linked list.

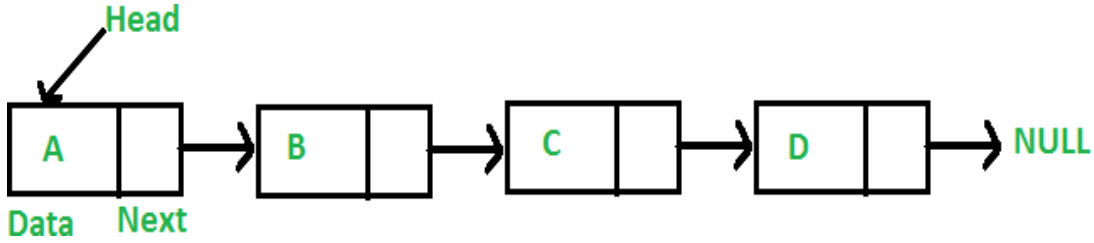
Input: Start time, end time,
min and max time
Status as booked or
free

Outcome:

- Display Appointment schedule
- Shoe appointment booking status
- Result of Sort appointment as per start time on linked list

Theory:

Linked List:
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

TYPES OF LINKED LIST:

There are 3 different implementations of Linked List available, they are:

1. Linear/Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Implementation:

Functions Used in the implementation:

1. void create_app()
2. void display_SLL()
3. void book_app()
4. void cancel_app()
5. void sort_app()

```
#include<iostream>
#include<string.h>
using namespace std;
int nodes;
struct appoint
{
 int status;
 char start[10];
 char end[10];
 char max[10];
 char min[10];
 struct appoint *Next;
}*head;

void create_app()
{
    int i ;
    appoint *temp, *p;

     head = NU  LL;
    for(i=0; i<nodes; i++)
    cout<<"\n\"  How many Appointments";
    {
    in>>nodes;
     cout<<" NEW APPOINTMENT ";
       temp = new(struct appoint); //Step 1: Allocate Memory
    temp->status=0;

       cout<<"\n\t Enter Start Time: "; //Step 2: Store data and address
```

```
      cin>>temp->start;
      cout<<"\n\t Enter End Time: "; //Step 2: Store data and address
      cin>>temp->end;
      cout<<"\n\t Enter min Time: "; //Step 2: Store data and address
      cin>>temp->min;
      cout<<"\n\t Enter max Time: "; //Step 2: Store data and address
      cin>>temp->max;

      temp->Next = NULL;

      if(head == NULL) //Step 3: Attach node in linked List
      {
        head = temp;
        p = head;
      }
      else
      {
        p->Next = temp;
        p = p->Next;
      }

    }
      }

void display_SLL()
{
 appoint *p;

   p = head;    cout<<"Status\tStart Time\tEnd Time\tMin Time\tMax Time\n";
                while(p != NULL)
   {
   if(p->status==0)
   {
   cout<<"Free";
   }
   else
   {
    cout<<"Booked";
 }
      cout<<"\t\t"<<p->start<<"\t\t"<<p->end<<"\t\t"<<p->min<<"\t\t"<<p->max<<"\t\n";
      p = p->Next;
   }

}
void book_app()
{
char time[10];
struct appoint *p;
cout<<"\n\n\tEnter The Time Slot";
cin>>time;
```

```
p=head;
 while(p!=NULL)
 {
  if(strcmp(time,p->start) == 0)
  {
    if(p->status == 0)
    {
     p->status=1;
     cout<<"Your Appointment Is Booked\n\n";
    }
    else
    cout<<"Appointment slot is Busy\n\n";
    break;
  }
  else
   p=p->Next;
 }

 if(p==NULL)
 cout<<"\n\n Appointment slot is Not available\n\n";
 display_SLL();

}

void cancel_app()
{
char time[10];
struct appoint *p;
cout<<"\n\n\tEnter Cancellation Time";
cin>>time;
p=head;
 while(p!=NULL)
 {
  if(strcmp(time,p->start) == 0)
  {
   if(p->status == 1)
    {
     p->status=0;
     cout<<"Your Appointment Is Cancelled\n\n";
    }
    else
    cout<<"Appointment slot is Busy\n\n";
    break;
   }
   else
   p=p->Next;
 }

 if(p==NULL)
 cout<<"\n\n Appointment slot is Not available\n\n";
 display_SLL();
```

```
}

void sort_app()
{
char str[10];
struct appoint *p;
int i;
for(i=0;i<nodes-1;i++)
 {
  p = head;
  while(p->Next!=NULL)
  {
   if(strcmp(p->start,p->Next->start)>0)
   {
    int tmp=p->status;
    p->status=p->Next->status;
    p->Next->status=tmp;

    strcpy(str,p->start);
    strcpy(p->start,p->Next->start);
    strcpy(p->Next->start,str);

    strcpy(str,p->end);
    strcpy(p->end,p->Next->end);
    strcpy(p->Next->end,str);

    strcpy(str,p->min);
    strcpy(p->min,p->Next->min);
    strcpy(p->Next->min,str);

    strcpy(str,p->max);
    strcpy(p->max,p->Next->max);
    strcpy(p->Next->max,str);
   }
   p=p->Next;
  }
 }
  cout<<"\n\nSORTED\n";
  display_SLL();
}


int main()
{
create_app();

display_SLL();

book_app();

cancel_app();
```

```
sort_app();

return 0;
}
```

Conclusion:

Hence, we studied how to implement linked list to book or cancel appointment.

A	P	J	Total	Dated Sign
3	4	3	10	

Question Bank: (No need to write answers)

1. What is a Linked list?
2. Can you represent a Linked list graphically?
3. How many pointers are required to implement a simple Linked list?
4. How many types of Linked lists are there?
5. How to represent a linked list node?
6. Describe the steps to insert data at the starting of a singly linked list.
7. How to insert a node at the end of Linked list?
8. How to delete a node from linked list?
9. How to reverse a singly linked list?
10. What is the difference between singly and doubly linked lists?
11. What are the applications that use Linked lists?
12. What will you prefer to use a singly or a doubly linked lists for traversing through a list of elements?

GROUP - D

Practical No:11(D)

Practical Title: Write C++ program to check well formedness of parenthesis using stack.

Aim: 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.

Pre-requisite:

- Basics of stack.
- Different operations that can be performed on stack

Objective:

- To check whether the given expression is well parenthesized or not.

Input:

Expression using {},(),[].

Outcome:

- Result of checking well formedness of parenthesis.

Theory:

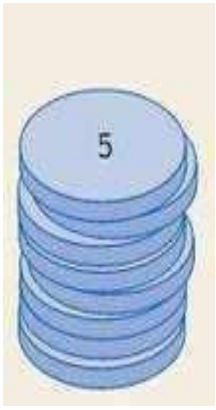
Definition:

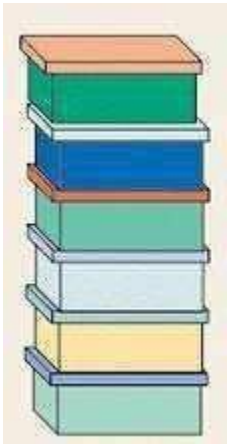
"Stack is data structure in ehich addition and removal of an element is allowed at the same end is called as top of stack."

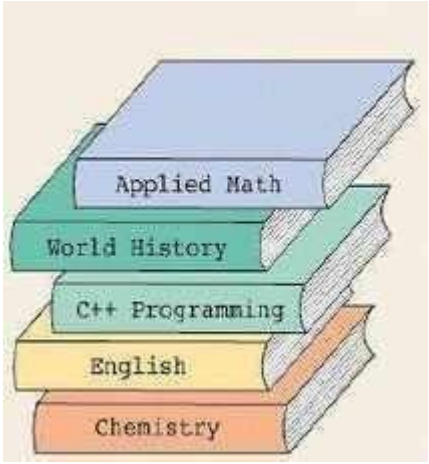
- Stack is also called as Last In First Out(LIFO) list.

- It means element which get added at last will be removed first.

Examples of Stack:







Stack is an abstract data type which is defined by the following structure and operation.

Stack operation :

1. createstack( )

2. Push( )

3. Pop( )

4. Peek( )

5. IsEmpty( )

6. IsFull( )

7. Size( )

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

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

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

isfull()

Algorithm of isfull() function −

```
begin procedure isfull

   if top equals to MAXSIZE
      return true
   else
      return false
   endif

end procedure
```

isempty()

Algorithm of isempty() function −

```
begin procedure isempty

   if top less than 1
      return true
   else
      return false
   endif

end procedure
```

Algorithm for PUSH Operation

A simple algorithm for Push operation can be derived as follows −

```
begin procedure push: stack, data
```

```
  if stack is full
    return null
  endif

  top ← top + 1
  stack[top] ← data

end procedure
```

Algorithm for Pop Operation

A simple algorithm for Pop operation can be derived as follows −

```
begin procedure pop: stack

  if stack is empty
    return null
  endif

  data ← stack[top]
  top ← top - 1
  return data

end procedure
```

Time

Complexities of operations on stack:

- push(), pop(), isEmpty() and peek() all take O(1) time. We do not run any loop in any of these operations.

Applications of stack:

- Balancing of symbols
- Infix to Postfix /Prefix conversion
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
- Backtracking is one of the algorithm designing technique .Some example of back tracking are Knight-Tour problem,N-Queen problem,find your way through maze and game like chess or checkers in all this problems we dive into someway if that way is not efficient we come back to the previous state and go into some another path. To get back from current state we need to store the previous state for that purpose we need stack.
- In Graph Algorithms like Topological Sorting and Strongly Connected Components
- In Memory management any modern computer uses stack as the primary-management for a running purpose.Each program that is running in a computer system has its own memory allocations
- String reversal is also a another application of stack.Here one by one each character get inserted into the stack.So the first character of string is on the bottom of the stack

and the last element of string is on the top of stack. After Performing the pop operations on stack we get string in reverse order .

Implementation:

There are two ways to implement a stack:

- Using array
- Using linked list

Algorithms :

Functions Used:

1. void push(char ch);
2. char pop();
3. bool isEmpty();
4. bool isFull();
5. void display();
6. void checkParenthesis();

Implementation:

```
#include<iostream>
                using namespace std;
                const int MAX=20;
                class Stack
                {
                        char str[MAX];
                        int top;
                        public:
                                Stack()
                                {
                                        top=-1;
                                }

                                void push(char ch);
                                char pop();
                        //      char getTop();
                                bool isEmpty();
                                bool isFull();
                                void display();
                                void checkParenthesis();
                };
                bool Stack::isEmpty()
                {
                        if(top==-1)
                                return 1;
                        else return 0;
                }

                bool Stack::isFull()
                {
```

```
        if(top==MAX-1)
                return 1;
        else
                return 0;
}


void Stack :: display()
{
        if(isEmpty()==1)
                cout<<"\nStack is empty";
        else
        {
                for(int i=0;i<=top;i++)
                {
                        cout<<" "<<str[i];
                }
        }
}
void Stack::push(char ch)
{
        if(!isFull())
        {
                top++;
                str[top]=ch;
        }
}

char Stack::pop()
{
        if(!isEmpty())
        {
                char ch=str[top];
                top--;
                return ch;
        }
        else
        {
                return '\0';
        }
}


void Stack::checkParenthesis()
{
        cout<<"\nEnter # as a deliminator after expression(At the end)\n";
        cout<<"\nEnter Expression: ";
        cin.getline(str,MAX,'#');
        char ch;
        bool flag=0;
```

```
        for(int i=0;str[i]!='\0';i++)
        {
                if(str[i]=='(' || str[i]=='[' || str[i]=='{')
                        push(str[i]);
                if(str[i]==')'||str[i]==']'||str[i]=='}')
                {
                        ch=pop();
                        if((str[i]==')'&& ch!='(') ||(str[i]==']'&&
ch!='[')||(str[i]=='}'&& ch!='{'))
                        {
                                cout<<"\nNot parenthesized At "<<i<<" =
"<<str[i];
                                flag=1;
                                break;
                        }
                }
        }

        if(isEmpty()==1 && flag==0)
                cout<<"\nExpresseion is Well Parenthesized.";
        else
                cout<<"\nExpression is not Well Parenthesized.";
}


int main()
{

        int choice;
        do
        {
                Stack s;
                s.checkParenthesis();
                cout<<"\nDO you want to continue?{1/0)";
                cin>>choice;
        }while(choice!=0);

        return 0;
}
```

Conclusion:

Hence, we studied implementation of stack operation.

A	P	J	Total	Dated Sign
3	4	3	10	

Practical No:12(D)

Practical Title: Write a C++ program for expression conversion as **infix to postfix** and its evaluation using stack

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

i. Operands and operator, both must be single character.

ii. Input Postfix expression must be in a desired format.

iii. Only '+', '-', '*' and '/ ' operators are expected

Pre-requisite:

- Basics of stack.
- Different operations that can be performed on stack

Objective:

- To convert the expression from infix to postfix
- Evaluate the expression

Input:

Infix expression

Outcome:

- Equivalent postfix expression
- Result of evaluation of an expression.

Theory :

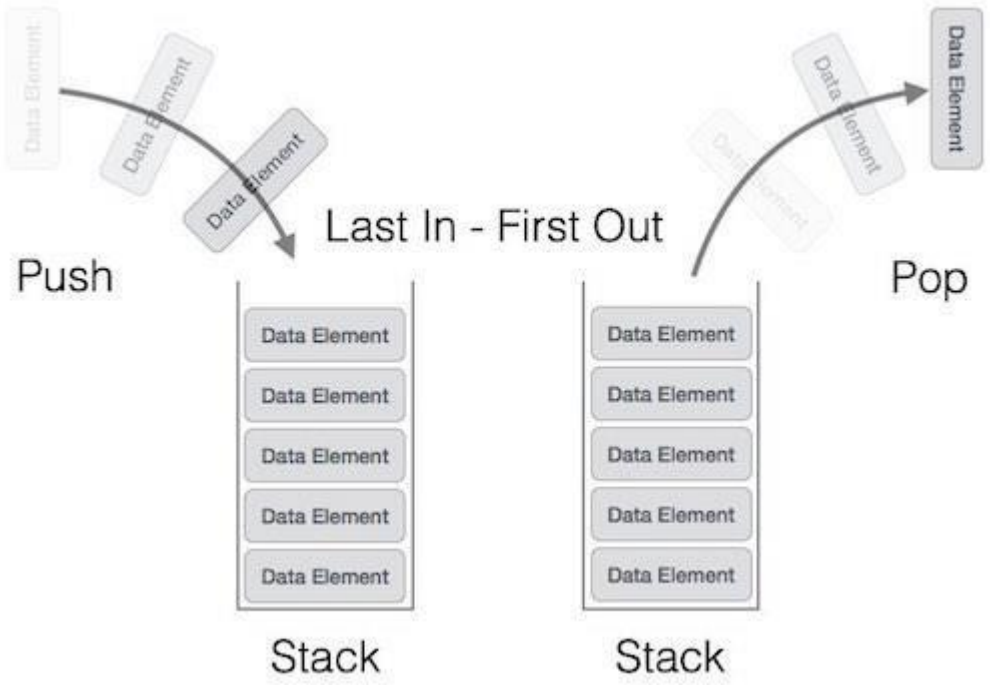
A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called **PUSH** operation and removal operation is called **POP** operation.

Stack Representation

The following diagram depicts a stack and its operations −



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

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

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

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named **top**. The **top** pointer provides top value of the stack without actually removing it.

APPLICATION OF STACKs

- Convert infix expression to postfix and prefix expressions
- Evaluate the postfix expression
- Reverse a string
- Check well-formed (nested) parenthesis
- Reverse a string
- Process subprogram function calls
- Parse (analyze the structure) of computer programs
- Simulate recursion

In computations like decimal to binary conversion
In Backtracking algorithms (often used in optimizations and in games

Polish Notation:

The way to write arithmetic expression is known as a **notation**. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are −

- Infix Notation
- Prefix (Polish) Notation
- Postfix (Reverse-Polish) Notation

These notations are named as how they use operator in expression. We shall learn the same here in this chapter.

Infix Notation

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

Prefix Notation

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

Postfix Notation

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

The following table briefly tries to show the difference in all three notations −

Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) ∗ c	∗ + a b c	a b + c ∗
3	a ∗ (b + c)	∗ a + b c	a b c + ∗
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) ∗ (c + d)	∗ + a b + c d	a b + c d + ∗
6	((a + b) ∗ c) - d	- ∗ + a b c d	a b + c ∗ d -

Postfix Evaluation Algorithm

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

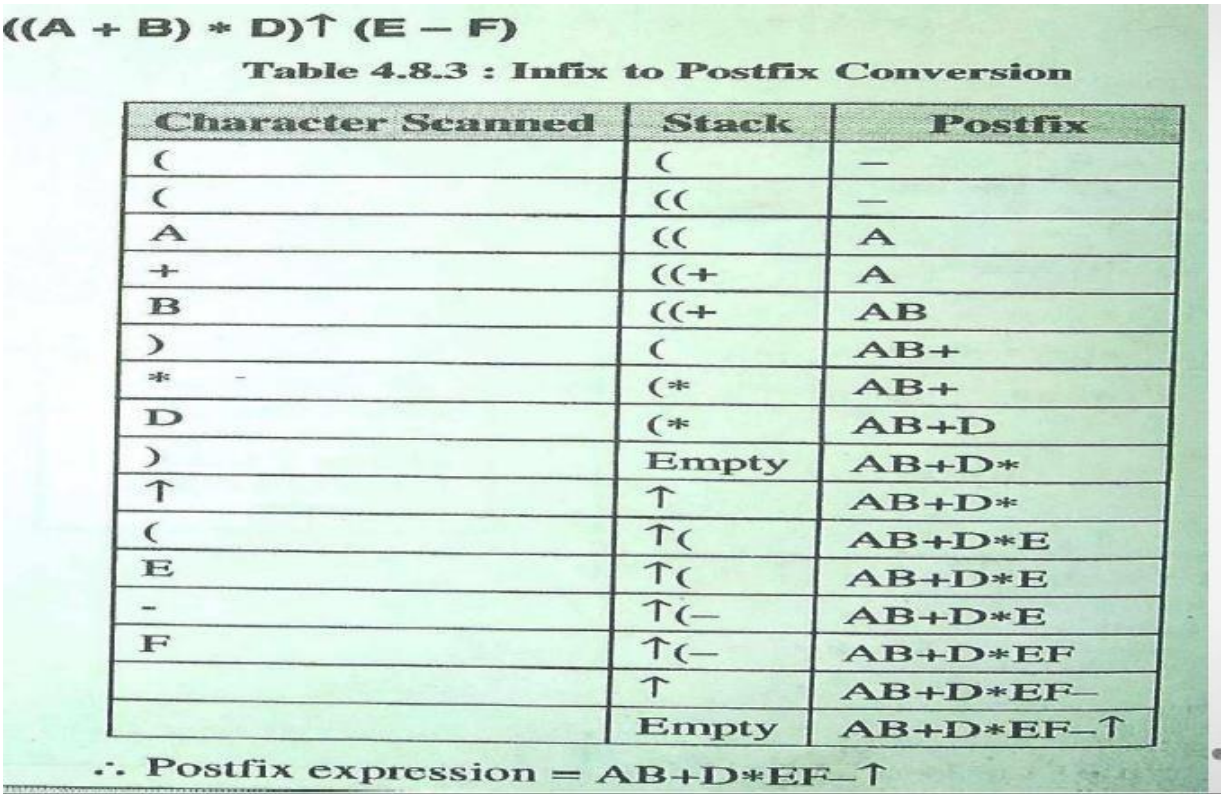
Step 1 − scan the expression from left to right
Step 2 − if it is an operand push it to stack
Step 3 − if it is an operator pull operand from stack and perform operation
Step 4 − store the output of step 3, back to stack
Step 5 − scan the expression until all operands are consumed
Step 6 − pop the stack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

Algorithm to Convert Infix to Postfix Expression Using Stack

Following is the **algorithm** to convert infix expression into Reverse Polish notation.

1. Initialize the Stack.
2. Scan the operator from left to right in the infix expression.
3. If the leftmost character is an operand, set it as the current output to the Postfix string.
4. And if the scanned character is the operator and the Stack is empty or contains the '(', ')' symbol, push the operator into the Stack.
5. If the scanned operator has higher precedence than the existing **precedence** operator in the Stack or if the Stack is empty, put it on the Stack.
6. If the scanned operator has lower precedence than the existing operator in the Stack, pop all the Stack operators. After that, push the scanned operator into the Stack.
7. If the scanned character is a left bracket '(', push it into the Stack.
8. If we encountered right bracket ')', pop the Stack and print all output string character until '(' is encountered and discard both the bracket.
9. Repeat all steps from 2 to 8 until the infix expression is scanned.
10. Print the Stack output.
11. Pop and output all characters, including the operator, from the Stack until it is not empty.

Example:



((A + B) * D)↑ (E – F)

Table 4.8.3 : Infix to Postfix Conversion

Character Scanned	Stack	Postfix
(	(	–
(	((	–
A	((	A
+	((+	A
B	((+	AB
)	(	AB+
*	(*	AB+
D	(*	AB+D
)	Empty	AB+D*
↑	↑	AB+D*
(	↑(	AB+D*E
E	↑(	AB+D*E
-	↑(–	AB+D*E
F	↑(–	AB+D*EF
	↑	AB+D*EF–
	Empty	AB+D*EF–↑

∴ Postfix expression = AB+D*EF–↑

Implementation:

```
1. #include<iostream>
2. #include<stack>
3. using namespace std;
4. // defines the Boolean function for operator, operand, equalOrhigher precedence and the stri
   ng conversion function.
5. bool IsOperator(char);
6. bool IsOperand(char);
7. bool eqlOrhigher(char, char);
8. string convert(string);
9.
10. int main()
11. {
12. string infix_expression, postfix_expression;
13. int ch;
14. do
15. {
16. cout << " Enter an infix expression: ";
17. cin >> infix_expression;
18.  postfix_expression = convert(infix_expression);
```

```
19.  cout << "\n Your Infix expression is: " << infix_expression;
20. cout << "\n Postfix expression is: " << postfix_expression;
21. cout << "\n \t Do you want to enter infix expression (1/ 0)?";
22. cin >> ch;
23. //cin.ignore();
24. } while(ch == 1);
25. return 0;
26. }
27.
28. // define the IsOperator() function to validate whether any symbol is operator.
29. /* If the symbol is operator, it returns true, otherwise false. */
30. bool IsOperator(char c)
31. {
32. if(c == '+' || c == '-' || c == '*' || c == '/' || c == '^' )
33. return true;
34. return false;
35. }
36.
37. // IsOperand() function is used to validate whether the character is operand.
38. bool IsOperand(char c)
39. {
40. if( c >= 'A' && c <= 'Z') /* Define the character in between A to Z. If not, it returns False.*
    /
41. return true;
42. if (c >= 'a' && c <= 'z') // Define the character in between a to z. If not, it returns False. */
43. return true;
44. if(c >= '0' && c <= '9') // Define the character in between 0 to 9. If not, it returns False. */
45. return true;
46. return false;
47. }
48. // here, precedence() function is used to define the precedence to the operator.
49. int precedence(char op)
50. {
51. if(op == '+' || op == '-')                /* it defines the lowest precedence */
52. return 1;
53. if (op == '*' || op == '/')
54. return 2;
55. if(op == '^')                             /* exponent operator has the highest precedence *
```

```
56. return 3;
57. return 0;
58. }
59. /* The eqlOrhigher() function is used to check the higher or equal precedence of the two op
    erators in infix expression. */
60. bool eqlOrhigher (char op1, char op2)
61. {
62. int p1 = precedence(op1);
63. int p2 = precedence(op2);
64. if (p1 == p2)
65. {
66. if (op1 == '^' )
67. return false;
68. return true;
69. }
70. return (p1>p2 ? true : false);
71. }
72.
73. /* string convert() function is used to convert the infix expression to the postfix expression
    of the Stack */
74. string convert(string infix)
75. {
76. stack <char> S;
77. string postfix ="";
78. char ch;
79.79.
80. S.push( '(' );
81. infix += ')';
82.
83. for(int i = 0; i<infix.length(); i++)
84. {
85. ch = infix[i];
86.
87. if(ch == ' ')
88. continue;
89. else if(ch == '(')
90. S.push(ch);
91. else if(IsOperand(ch))
```

```
92. postfix += ch;
93. else if(IsOperator(ch))
94. {
95. while(!S.empty() && eqlOrhigher(S.top(), ch))
96. {
97. postfix += S.top();
98. S.pop();
99. }
100.        S.push(ch);
101.        }
102.        else if(ch == ')')
103.        {
104.        while(!S.empty() && S.top() != '(')
105.        {
106.        postfix += S.top();
107.        S.pop();
108.        }
109.        S.pop();
110.        }
111.        }
112.        return postfix;
113.        }
```

Conclusion:

By this way, we can perform expression conversion as infix to postfix and its evaluation using stack

A	P	J	Total	Dated Sign
3	4	3	10	

Question Bank:

1. What is Stack?
2. Which are the different operations that can be performed on stack?
3. Explain PUSH, POP operations on stack
4. What are the applications of stack?
5. What is infix, postfix and prefix expression?
6. Conversion – infix to postfix, infix to prefix , etc.
7. Evaluation of infix, postfix and prefix expression

GROUP - E

Practical No:13 (E)

Practical Title: Perform different operations on Queue.

Aim: 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.

Pre-requisite:

- Basics of Queue
- Different operations that can be performed on queue

Objective:

- To perform addition and deletion operations on queue.

Input:

- Size of queue Elements in ququе

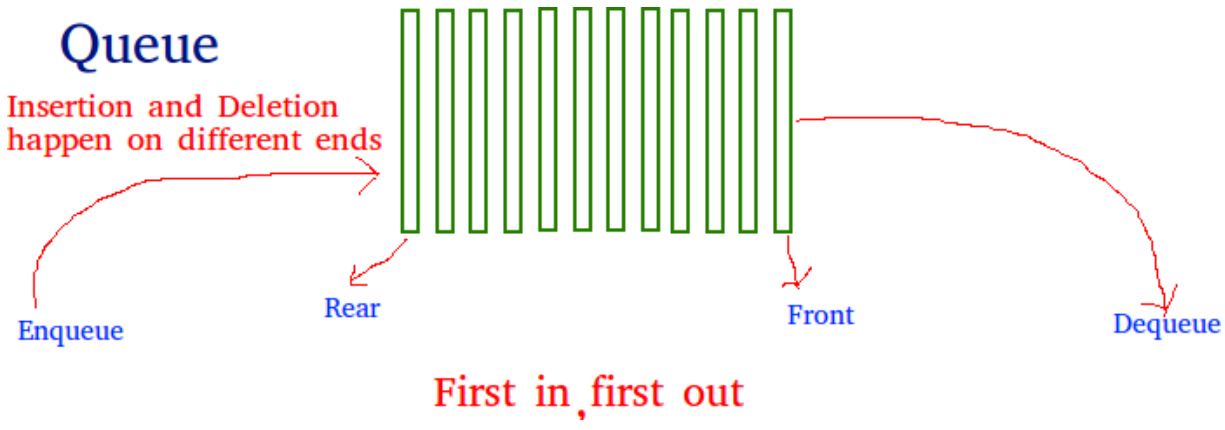
Outcome:

- Result of addition of job operation on queue.
- Result of deletion of job operation on queue.

Theory :

Queue:

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



Basic Operations

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

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

Few more functions are required to make the above-mentioned queue operation efficient. These are −

- **peek()** − Gets the element at the front of the queue without removing it.
- **isfull()** − Checks if the queue is full.
- **isempty()** − Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by **front** pointer and while enqueing (or storing) data in the queue we take help of **rear** pointer.

isfull()

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

Algorithm

```
begin procedure isfull

  if rear equals to MAXSIZE
    return true
  else
    return false
  endif

end procedure
```

isempty()

Algorithm of isempty() function −

Algorithm

```
begin procedure isempty

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

```
  endif

end procedure
```

Algorithm for enqueue operation

```
procedure enqueue(data)

  if queue is full
    return overflow
  endif

  rear ← rear + 1
  queue[rear] ← data
  return true

end procedure
```

Algorithm for dequeue operation

```
procedure dequeue

  if queue is empty
    return underflow
  end if

  data = queue[front]
  front ← front + 1
  return true

end procedure
```

Applications of Queue:

Queue is used when things don't have to be processed immediatly, but have to be processed in **F**irst **I**n**F**irst **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.

2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc..

Implementation:

```
#include <iostream>
#define MAX 10
using namespace std;
struct queue
```

```
{       int data[MAX];
          int front,rear;
};
class Queue
{    struct queue q;
  public:
     Queue(){q.front=q.rear=-1;}
     int isempty();
     int isfull();
     void enqueue(int);
     int delqueue();
     void display();
};
int Queue::isempty()
{
          return(q.front==q.rear)?1:0;
}
int Queue::isfull()
{    return(q.rear==MAX-1)?1:0;}
void Queue::enqueue(int x)
{q.data[++q.rear]=x;}
int Queue::delqueue()
{return q.data[++q.front];}
void Queue::display()
{ int i;
   cout<<"\n";
   for(i=q.front+1;i<=q.rear;i++)
             cout<<q.data[i]<<" ";
}
int main()
{      Queue obj;
          int ch,x;
          do{    cout<<"\n 1. insert job\n 2.delete job\n 3.display\n 4.Exit\n Enter your choice:";
               cin>>ch;
          switch(ch)
          { case 1: if (!obj.isfull())
                       { cout<<"\n Enter data:";
                              cin>>x;
                              obj.enqueue(x);
                       }
                    else
                          cout<< "Queue is overflow";
                     break;
            case 2: if(!obj.isempty())
                                 cout<<"\n Deleted Element="<<obj.delqueue();
                       else
                              { cout<<"\n Queue is underflow";  }
                       cout<<"\nremaining jobs :";
                       obj.display();
                     break;
           case 3: if (!obj.isempty())
                 { cout<<"\n Queue contains:";
```

```
                obj.display();

            }
            else      cout<<"\n Queue is empty";

           break;


        case 4: cout<<"\n Exit";
     }
    }while(ch!=4);
return 0;
}
```

Conclusion:

By this way, we can perform different operations on queue

A	P	J	Total	Dated Sign
3	4	3	10	

Question Bank:

1. What is Queue?
2. What are the different operations that can be performed on queue?
3. Explain all the operations on queue
4. Which are different types of queues , Explain.

Practical No:14 (E)

Practical Title: Perform operations on Double ended queue.

Aim: 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.

Pre-requisite:

- Knowledge of Queue
- Types of queue
- Knowledge of double ended queue and different operations that can be performed on it

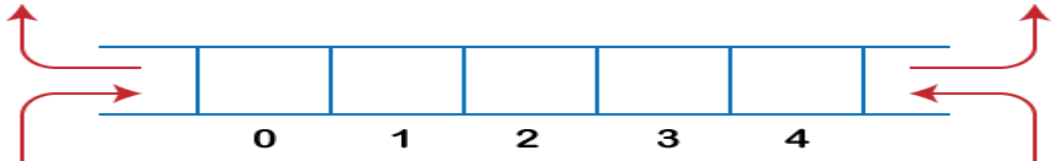
Objective:
- To simulate deque with functions to add and delete elements from either end of the deque.

Input: Size of array Elements in the queue

Outcome:
- Result of deque with functions to add and delete elements from either end of the deque.

Theory :

The dequeue stands for **Double Ended Queue**. In the queue, the insertion takes place from one end while the deletion takes place from another end. The end at which the insertion occurs is known as the **rear end** whereas the end at which the deletion occurs is known as **front end**.



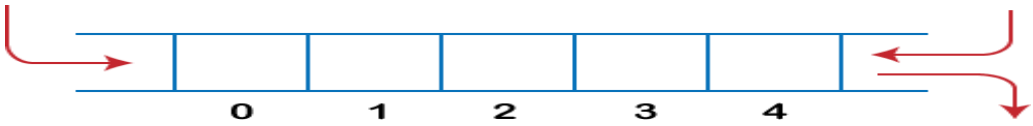
Deque is a linear data structure in which the insertion and deletion operations are performed from both ends. We can say that deque is a generalized version of the queue.

There are two types of Queues, **Input-restricted queue**, and **output-restricted queue**.

1. **Input-restricted queue:** The input-restricted queue means that some restrictions are applied to the insertion. In input-restricted queue, the insertion is applied to one end while

the deletion is applied from both the ends.

2. **Output-restricted queue:** The output-restricted queue means that some restrictions are applied to the deletion operation. In an output-restricted queue, the deletion can be applied only from one end, whereas the insertion is possible from both ends.



Operations on Deque

The following are the operations applied on deque:

- **Insert at front**
- **Delete from end**
- **insert at rear**
- **delete from rear**

Other than insertion and deletion, we can also perform **peek** operation in deque.
Through **peek** operation, we can get the **front** and the **rear** element of the dequeue.

We can perform two more operations on dequeue:

- **isFull():** This function returns a true value if the stack is full; otherwise, it returns a false value.
- **isEmpty():** This function returns a true value if the stack is empty; otherwise it returns a false value.

Algorithm for Insertion at rear end

```
Step -1:
    [Check for
    overflow]
    if(rear==M
    AX)
    Print("Queu
    e is
    Overflow");
    return;
Step-2: [Insert element] else     rear=rear+1;
```

```
        q[rear]=no;
        [Set rear and front pointer]

    if rear=0
        rear=1;
    if front=0
        front=1;
    Step-3: return
```

Implementation of Insertion at rear end

```
    void add_item_rear()
    {
        int num;
        printf("\n Enter Item to insert : ");
        scanf("%d",&num);
        if(rear==MAX)
        {
        printf("\n Queue is Overflow");
        return;
      }
        else
        {
     rear++;
     q[rear]=num;
     if(rear==0)
     rear=1;
     if(front==0)
     front=1;
    }
     }
```

Algorithm for Insertion at font end

```
    Step-1 : [Check for the front position]
     if(front<=1)
     Print ("Cannot add item at front end");
     return;
    Step-2 : [Insert at front]
     else
    front=front-1;
    q[front]=no;
    Step-3 : Return
```

Implementation of Insertion at font end

```
    void add_item_front()
    {
     int num;
     printf("\n Enter item to insert:");
      scanf("%d",&num);
```

```
 if(front<=1)

{
 printf("\n Cannot add item at front end");
return;
 }
 else
 {
 front--;
 q[front]=num;
}
}
```

Algorithm for Deletion from front end

```
Step-1 [ Check for front pointer]
 if front=0
 print(" Queue is Underflow”);
  return;
   Step-2 [Perform deletion]
  else
    no=q[front];
 print(“Deleted element is”,no);
 [Set front and rear pointer]
 if front=rear
 front=0;
 rear=0;
 else
 front=front+1;
 Step-3 : Return
```

Implementation of Deletion from front end

```
 void delete_item_front()
 {
 int num;
 if(front==0)
 {
 printf("\n Queue is Underflow\n");
 return;
 }
 else
 {
 num=q[front];
 printf("\n Deleted item is %d\n",num);
 if(front==rear)
 {
 front=0;
 rear=0;
 }
 else
 {
 front++;
```

```
    }

    }
}
```

Algorithm for Deletion from rear end

Step-1 : [Check for the rear pointer]
if rear=0
print(“Cannot delete value at rear end”);
return;
Step-2: [perform deletion]
else
no=q[rear];
[Check for the front and rear pointer]
if front= rear
front=0;
rear=0;
else
rear=rear-1;
print(“Deleted element is”,no);
Step-3 : Return

Implementation of Deletion from rear end

```
   void delete_item_rear()
 {
   int num;
   if(rear==0)
   {
   printf("\n Cannot delete item at rear end\n");
   return;
   }
   else
   {
   num=q[rear];
   if(front==rear)
   {
   front=0;
   rear=0;
   }
   else
   {
   rear--;
   printf("\n Deleted item is %d\n",num);
   }
   }
  }
```

Conclusion:

By this way, we can perform operations on double ended queue

A	P	J	Total	Dated Sign
3	4	3	10	

Question Bank:

1. What is queue?
2. Types of queue
3. What is double ended queue?
4. How to insert the new node in Doubly Link List?
5. How to delete the node from front of Doubly Link List ?
6. How to delete the node from end of Doubly Link List ?
7. How to delete the node in between of Doubly Link List

THANKS..

PROF. Vidya Yenegur
ASSISTANT PROFESSOR
PARVATIBAI GENBA MOZE COLLEGE OF ENGINEERING WAGHOLI PUNE 412207