PUNE INSTITUTE OF COMPUTER TECHNOLOGY

DHANKAWADI, PUNE – 43.

LAB MANAUAL

ACADEMIC YEAR: 2023-2024

COMPUTER ENGINEERING DEPARTMENT

CLASS: S.E. **SEMESTER: I**

SUBJECT: DATA STRUCTURES LABORATORY

Sr. No.	PROBLEM STATEMENT	Revised on
	Group A	
1	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)	18/8/2023
2	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 (Do not use string built-in functions)	25/8/2023
3	Write a python program to compute following computation on matrix: a) Addition of two matrices b) Subtraction of two matrices c) Multiplication of two matrices d) Transpose of a matrix **OR** Write the Python Program that determines the location of saddle point of matrix if exist. A m*n Matrix is said to have saddle point if some entry a[i][j] is the smallest value in row I and largest value in j.	01/9/2023
	Group B	
4	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	08/9/2023
5	Write a python program to store second year percentage of students in array. Write function for sorting array of floating-point numbers in ascending order using a) Bubble sort b) Selection sort c) Insertion sort d) Shell Sort and display top five scores.	22/9/2023
6	Write a python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores.	29/9/2023

	Group C	
7	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. **OR** Write C++ program for storing binary number using doubly linked lists. Write functions- a) To compute 1's and 2's complement b) Add two binary numbers	13/10/2023
8	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) d) Sort list based on time e) Sort list based on time using pointer manipulation **OR** Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C++ program to store two sets using linked list. compute and display- a) Set of students who like both vanilla and butterscotch b) Set of students who like either vanilla or butterscotch or not both c) Number of students who like neither vanilla nor butterscotch	20/10/2023
	Group D	
9	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.	27/10/2023
10	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.	03/11/2023
	Group E	
11	Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job, display job and delete job from queue.	09/11/2023
12	Write program to implement a priority queue in C++ using an order list/array to store the items in the queue. Create a class that includes the data items (which should be template) and the priority (which should be int). The order list/array should contain these objects, with operator <= overloaded so that the items with highest priority appear at the beginning of the list/array (which will make it relatively easy to retrieve the highest item.)	24/11/2023
13	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.	01/12/2023

[illegible]

Ms. Priyanka Makkar

Subject Coordinator

[illegible]

Dr. Geetanjali Kale

HOCD

:F-LTL-UG / 03 / R1

Assignment no	1
Aim	**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)**
Objective	To understand the concept of functions in programming languages To understand , implement SET data structure and its operations To use list or array in python to implement derived SET data structures
Outcome	To understand ,design and implement SET data structure using list or array in python To write/implement user defined functions/modules for different operations of SET in python To write menu driven, modular program in Python
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with Python plugin or Pycharm IDE

Theory related to assignment:

In this assignment we will implements derived data structure SET data structure using list or array as primitive data structure (NOTE: we are not supposed to use inbuilt SET in python)

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements.

Lists

- Python lists are very flexible and can hold arbitrary data.
- Lists are a part of Python’s syntax, so they do not need to be declared first.
- Resize quickly
- Store heterogeneous data
- Mathematical functions can be applied directly
- List consume more memory

Arrays

- Arrays need to first be imported, or declared, from other libraries (i.e. numpy).
- Store homogenous data
- Wide range of mathematical functions can be applied directly
- Arrays are compact in size

How to use list in python:
lst=[ ]// empty list
lst.append(1) //append or add 1 element to list
print(lst) //print the list

How to use array in python:
import array as arr
a=arr.array('i',[1,2,3]) // a is array of integer

We can use list as array but we can restricts to type of elements

ADT :

Set is Objects or value definition: A finite collection with zero or more elements
Functions or operator definition:
For all S є Set and item є element
createSet() : Set //creates the empty Set
addEle(Set,item) :void //adds unique items to Set
Union(Set,Set):Set // Returns union of 2 sets
isemptySet(Set):Boolean //if Set is empty returns TRUE else returns FALSE
Intersection(Set): Set //Returns Intersection of 2 sets
End Set

ADT representation using class for Set
Class SET {
int *a; // value definition in ADT
int n; //indicate size of set
public: //Operator definition in ADT
Set create(); //creates the empty Set with n=0 and allocate memory for array a
void addelement(i,val); // appends val in Set at ith location provided val is not present in set and increment n;
void addelement(val) // append val at end and increment n
void read(); //read whole set
int read(i); // returns value at ith location from Set
bool exists(item); // if item exists in Set returns true else false
Set union(Set ,Set) // return union of 2 sets

Set intersection(Set ,Set) // return intersection of 2 sets
}

Step for implementation:

Step 1:

Program should defined six list variables which are empty initially. Declare the List for 1) students who play both cricket 2) students who play badminton 3) students who play football 3) students who play both cricket and badminton 4) students who play either cricket or badminton but not both 5) Number of students who play neither cricket nor badminton 6) Number of students who play cricket and football but not badminton

Step 2:

Accept the three sets for cricket, badminton and football by calling createSet() function which will take care of unique elements

Step 3: Define functions of union, intersection, difference/subset which accept the two list as arguments and return the answer list

Step 4: call corresponding functions with list as arguments according to menu.

Pseudo Code:

Union of two set:

procedure Set union(Set s1, Set s2)

> **Purpose : Finds the union of sets s1 and s2 represented as objects of Set with length of set as ‘n’**
>
> **Pre-condition: nothing**
>
> **Post condition: Union of two sets s3 is returned as Set**

```
1.  begin
2.  for i=0 to i=s1.n-1
3.  begin
4.  s3.addelement(s1.read(i))
5.  end for
6.  for i=0 to i=s2.n-1
7.  begin
8.  if(s3.exists(s2.read(i))==false)
9.  s3.addelement(s2.read(i))
10. end if
11. end for
12. return s3
```

end union

Intersection of two sets:

Set intersection(Set s1, Set s2)

Purpose : Finds the union of sets s1 and s2 represented as objects of Set with length of set as ‘n’

Pre-condition: nothing

Post condition: intersection of two sets s3 is returned as Set

```
1. begin
2. s1.read();
3. s2.read();
4. for i=0 to i=s1.n -1
5. begin
6. if(s2.exists(s1.read(i))==true) s3.addelement(s1.read(i))
7. end if
8. end for
9. return s3
```

end intersection

Subset of two sets:

Procedure Boolean subset (Set s1, Set s2)

Purpose : Finds the whether set2 is subset of of sets s1

Pre condition: nothing

Post condition: if set s2 is subset of set s1 then it is returned as true

```
1. begin
2. Subsetflag=True
3. for i=0 to s2.length-1 do
4. If s1.exists(s2[i])==0
5. Subsetflag=false
6. end for
7. return Subsetflag;
```

end subset;

Time Complexity: O(n)

Space Complexity: O(n)

Conclusion:

The functions for different set operations are implemented successfully using list as primitive data structure.

Review Questions:

1. What is set data structure and its applications?
2. Explain the different operations of set data structure?
3. How to use list to implement SET data structure?
4. How to use array to implement SET data structure?
5. What is list data type in python?

6. Can we add duplicate elements in SET?
7. Can we add duplicate elements in list or array?
8. SET is homogeneous or heterogeneous data type?
9. Explain the complexities of different operations of SET?

Assignment no.	2
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 (Do not use string built-in functions)
Objective	To understand the concept of Strings in Python To apply string manipulation operations
Outcome	After executing this assignment, Students will be able to perform all the tasks without using built in functions. Students will be familiar with String module Students will be in position to use string manipulation operations
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit)

Theory related to assignment:

In this assignment We shall Learn Data type String.

- Python treats strings as contiguous series of characters delimited by single, double or even triple quotes.
- Python has a built-in string class named "str" that has many useful features.
- We can simultaneously declare and define a string by creating a variable of string type. This can be done in several ways which are as follows:

name = "India"

graduate = 'N'

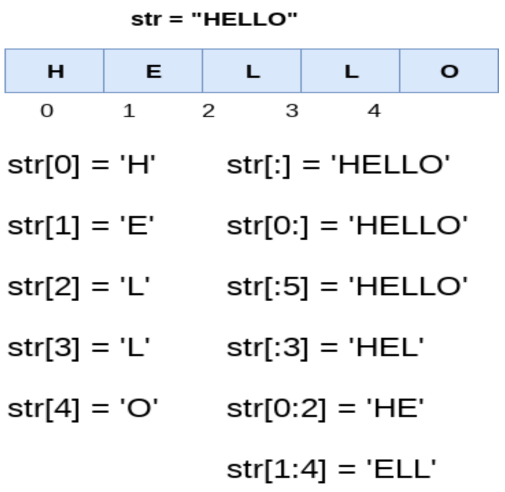
country = name

nationality = str("Indian")

In python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

str = "Hi PICT !"

print(type(str)), then it will **print** string (str).



```
str[0] = 'H'      str[:] = 'HELLO'

str[1] = 'E'      str[0:] = 'HELLO'

str[2] = 'L'      str[:5] = 'HELLO'

str[3] = 'L'      str[:3] = 'HEL'

str[4] = 'O'      str[0:2] = 'HE'

                  str[1:4] = 'ELL'
```

Indexing: Individual characters in a string are accessed using the subscript ([ ]) operator.

- The expression in brackets is called an index. The index specifies a member of an ordered set and in this case it specifies the character we want to access from the given set of characters in the string.
- The index of the first character is 0 and that of the last character is n-1 where n is the number of characters in the string.
- If you try to exceed the bounds (above n-1), then an error is raised.

Traversing a String: A string can be traversed by accessing character(s) from one index to another. For example, the following program uses indexing to traverse a string from first character to the last.

Example: Program to demonstrate string traversal using indexing

```
message= “Hello!”

index=0

for i in message:

        print(“message[“, index, “]=”, i)

        index+=1
```

Output:

```
message[ 0 ] = H

message[ 1 ] = e

message[ 2 ] = l
```

message[3] = l

message[4] = o

message[5] = !

a) **To display word with the longest length :**

Algorithm/Pseudocode:

```
Step 1: Read a Sentence
Step 2: Break up the sentence into an array of individual words (list)
Step 3:  Initialize a counter variable
Step 4:  Loop through each word in the array(list)
Step 5:  Get the length of each word
Step 6: If the length is greater than the counter, set the counter
Step 7: Return the counter
```

Time Complexity: O(n), where n is the length of string.
Space: O(n), where n is the length of string.

===

b) **To determines the frequency of occurrence of particular character in the string:**

Algorithm/ Pseudocode:

```
1. Start
2. Declare a string
3. Ask the user to initialize it.
4. Use a frequency array to store the frequency of each character.
5. Convert the string to a character array (list)
6. Use two for loops to calculate the frequency of each element.
7. Use the first for loop to iterate through each character of the array(list).
8. Initialize each element of the frequency array as 1(list).
9. Use another for loop to iterate through the remaining characters.
10. Check for the total occurrence of the element.
11. If the element occurs again, increment the value in the frequency array(list).
12. Set the character array to 0 to avoid counting visited characters.
13. Print the characters and their corresponding frequency.
14. Stop.
```

Time Complexity: O(n).
Space: O(1)

===

c) **To check whether given string is palindrome or not :**

We are starting the algorithm by taking the string to be checked as input from the user. After that, the length of the string is calculated and stored in a variable, say ‘length’. To check whether a string is palindrome or not, the given string must be reversed. To store the reversed string, we are initializing a variable ‘rev’ as an empty string. That being done, we are starting a loop with initial value i = length – 1. In this loop, we are reversing the string, one character at a time by performing: rev = rev + character at position i. Here, the ‘+’ operator performs the concatenation of the characters of the string in reverse order. After that, the value of ‘i’ is decremented by 1. This loop runs until i >= 0. Once this loop ends, we have the reversed string in the variable rev.

We will now check whether both the strings are equal or not, ignoring the cases of the characters. If both the strings are equal, the given string is palindrome, else, the given string is not palindrome.

Algorithm/ Pseudocode:

```
Step 1. Start
Step 2. Read the string from the user
Step 3. Calculate the length of the string
Step 4. Initialize rev = “ ”[empty string]
Step 5. Initialize i = length - 1
Step 6. Repeat until i>=0:
            6.1: rev = rev + Character at position ‘i’ of the string
            6.2: i = i – 1
Step 7. If string = rev:
            7.1: Print “Given string is palindrome”
Step 8. Else:
            8.1: Print “Given string is not palindrome”
Step 9. Stop
```

Time complexity : O(n)
Space : O(1)
==

d) To display index of first appearance of the substring

Algorithm/Pseudocode:

```
Step1: Read String 1
Step2: Read substring 2
Step3: Apply loop till length of string 1
Step4: Apply Loop till length of string 2
Step5: Check if ith index of str1 is equal to jth index of str 2
Step6: If true, print I and come out of the loop using Break
Step7: Print Substring not present
```

Time Complexity: O(n).
Space: O(1)
==

e) To count the occurrences of each word in a given string

Algorithm/ Pseudocode:

```
Step1: Read String 1
Step2: Convert the string in to list
Step3: Create one more list CountList initialized with 1, of length 4
Step4:  apply for loop till length of list using counter variable i
Step5: apply for loop till length of list using counter variable j
Step6: check if lst [i] is equal to list[j]
Step7: if true, increment ith element of countlist
Step8: Complete internal loop of counter variable j
Step9: Complete outer loop of counter variable i
Step10: Display the countlist
```

Time Complexity: O(n).
Space: O(1)

==

Conclusion:

Students Successfully Executed the string manipulation operations without using string built in methods. Students learnt to find length of string, to calculate frequency of words in the string manually.

Review Questions:

1 What is the output of the following code ?

```
example = "snow world"

example[3] = 's'

print example
```

A. snow
B. snow world
C. Error
D. snos world

2. What is the output of “hello”+1+2+3 ?

A. hello123
B. hello
C. Error

D. hello6

3. Suppose i is 5 and j is 4, i + j is same as

A. i.__add(j)

B. i.__add__(j)

C. i.__Add(j)

D. i.__ADD(j)

4. The Index of the first character of the string is:

A. 0
B. 1
C. N-1
D. N

5. What is string in python explain String indexing and String traversing with examples.

6.Explain Concatenating, Appending and Multiplying Strings with examples.

7. Explain str () with examples.

8. String is mutable or immutable? Explain with an example.

9. Explain String formatting operator with example.

10. Explain any seven in-built string methods with example

Assignment no.	3
Aim	Write a **python** program to compute following computation on matrix: a) Addition of two matrices b) Subtraction of two matrices c) Multiplication of two matrices d) Transpose of a matrix
Objective	1. To understand the standard and abstract data representation methods. 2. To identify the appropriate data structure and algorithm design method for a specified application.
Outcome	1. To demonstrate a detailed understanding of behaviour of data structures like array, linked list, stack, and queue by developing programs. 2. To analyse and use effective and efficient data structures in solving various Computer Engineering domain problems. 3. To design the algorithms to solve the programming problems.
OS/Programming tools used	• 64-bit Open-source Linux or its derivative • Jupyter Notebook

Theory

Matrix is a set of elements in tabular form. The matrix can be represented using two-dimensional array data structure as shown below. An item of the matrix can be accessed using its row and column index. The basic operations which can be performed on matrix includes addition, subtraction, multiplication and transpose.

$$\text{Matrix A} = \begin{matrix} 23 & 78 & 98 \\ 04 & 21 & 33 \\ 67 & 44 & 33 \end{matrix}$$

Some of the basic constraints are to be checked before performing the operations on matrices. For example, for performing addition of two matrices, we need to have the matrices with same dimensions while multiplying two matrices the number of rows and columns of first matrix need to be same as number of columns and rows of second matrix respectively.

In python, the matrix is to be defined as a nested list. Following example shows the same.

```
M1 = [[8, 14, -6], [12,7,4], [-11,3,21]]
#To print the matrix
print(M1)
```

ADT Matrix

{A set of lists, where each list contains integers.}

Operations

1. void getMatrix(r, c): Reads a Matrix that can hold r*c elements information.
2. Matrix Transpose(A): return the matrix produced by interchanging the row and column value of every triple.
3. Matrix Add(A,B): if dimensions of a and b are the same return the matrix produced by corresponding items, namely those with identical row and column values else return error.
4. Matrix Multiply(A,B): if number of columns in a equals number of rows in B return the matrix D produced by multiplying A by B according to the formula:

   D[i][j]= Σ(A[i][k]*B[k][j])

   where D[i][j] is the $(i,j)^{th}$ element else return error.

Pseudo code:

Algorithm getMatrix()

```
{// to read input matrices
//M is the input matrix, r is the number of rows, c is the number of colums
1. initialse M:=[]
2. read r,c from user
3. for i:=0 to i:= r do
        3.1 row=[]
        3.2 for j:=0 to j:=c do
                3.2.1 read element at (i,j) position of matrix
                3.2.2 append element to row
4. M.append(row)
}
```

Algorithms

1. Matrix addition

// A and B are the matrices of dimension rows1*columns1 and rows2*columns2 respectively

// C is zero matrix of order rows1*columns1

1. Start
2. If (rows1 is not equal to rows2 or columns1 is not equal to columns2) then return (“Addition
   Not Possible”)
3. Initialize row to zero and column to zero
4. Set C [row, column] as addition of A [row, column] and B [row, column]
5. Increment column by 1
6. Repeat step 4 and 5 till column<columns1
7. Increment row by 1
8. Repeat step 4 to 7 till row<rows1
9. Return C
10. Stop

Pseudo code

Algorithm ADD(M1,M2)

```
{// takes 2 matrices as input
// returns matrix M3 which is the addition of M1 and M2
1.if(M1.r==M2.r and M1.c==M2.c)   //->1 check if dimensions of input matrix are same
{
        1.Initialize M3:=[]                         //->1
        2. for i:=1 to M1.r do                      //-> m+1
                2.1 row=[]                          //->m
                2.2 for j:=0 to M1.c                //->m*(n+1)
                        2.2.1 sum=M1[i][j]+M2[i][j] //->m*n
                        2.2.2 row.append(sum)       //->m*n
                        2.2.3 sum:=0                //->m*n
                2.3 M3.append(row)                  //->m
        3. return M3                                //->1
}
```

2. Matrix subtraction

// C matrix is a zero matrix of order rows1*columns1

1. Start
2. If (rows1 is not equal to rows2 or columns1 is not equal to columns2) then return (“Subtraction Not Possible”)
3. Initialize row to zero and column to zero
4. Set C [row, column] as subtraction of B [row, column] from B [row, column]
5. Increment column by 1
6. Repeat step 4 and 5 till column<columns1
7. Increment row by 1
8. Repeat step 4 to 7 till row<rows1
9. Return C
10. Stop

Pseudo code

Algorithm SUB(M1,M2)

```
{// takes 2 matrices as input
// returns matrix M3 which is the subtraction of M1 and M2
1.if(M1.r==M2.r and M1.c==M2.c)     //->1 check if dimensions of input matrix are same
{
    1.Initialize M3:=[]                         //->1
    2. for i:=1 to M1.r do                      //->m+1
        2.1 row=[]                              //->m
        2.2 for j:=0 to M1.c                    //->m(n+1)
            2.2.1 diff=M1[i][j]-M2[i][j]        //->mn
            2.2.2 row.append(diff)              //->mn
            2.2.3 diff:=0                       //->mn
        2.3 M3.append(row)                      //->m
    3. return M3                                //->1
}
2. else display "difference cannot be found"
}
```

3. Matrix multiplication

// C matrix is zero matrix of order rows1*columns2

1. Start
2. If (rows1 is not equal to columns2 or columns1 is not equal to rows2) then return ("Multiplication Not Possible")
4. Initialize row to zero and column to zero
5. Initialize i to zero
6. Set C [row, column] as C [row, column] added with multiplication of A [row, i] and B [i, column])
7. Increment i by 1
8. Repeat step 6 and 7 till i <columns1
9. Increment column by 1
10. repeat step 6 to 9 till column<columns2
11. Increment row by 1
12. repeat step 6 to 11 till row<rows1
13. return C
14. stop

Pseudo code

Algorithm MULT(M1,M2)

```
{// takes 2 matrices as input and returns matrix M3 which is the product of M1 and M2
1. if M1.c==M2.r do                                  //->1
{
    1.Initialize M3:=[]                              //->1
    2. for i:=0 to M1.r do                           //->m+1
        2.1 row=:[]                                  //->m
        2.2 for j:=0 to j:=M2.c                      //->m(n+1)
            2.2.1 sum=0                              //->mn
            2.2.2 for k:=0 to k:=M2.r do             //->mn(p+1)
                2.2.2.1 sum=sum+M1[i][k]*M2[k][j]            //->mnp
            2.2.3 row.append(sum)                            //->mn
        2.3. M3.append(row)                                  //->m
    4. return M3                                             //->1
}
```

2. else display "product cannot be found" *//->1*

4. Matrix transpose

// C is zero matrix of order column*rows

1. Start

2. Initialize row to zero and column to zero

4. Set C [column, row] as A [row, column]

5. Increment column by 1

6. Repeat step 4 and 5 till column<columns

7. Increment row by 1

8. repeat step 4 to 7 till row<rows

9. Return C

10. Stop

Pseudo code:

Algorithm Transpose ()

```
{// takes matrix as input and returns its transpose
1. input matrix                          //->1
2. for i:=0 to M1.r do                   //->m+1
        2.1 for j:=0 to M1.c do          //->m(n+!)
                2.1.1 M3[j][i]=M1[i][j]  //->mn
3. return M3                             //->1
```

Conclusion

Two-dimensional array is the data structure identified and used for representing the matrices. Matrix operations are performed using list data structure in Python.

Review questions

1. Which data structure is to be chosen for storing matrix? Justify the same.
2. Analyse the time complexity for matrix addition and multiplication.
3. What is the space complexity for matrix subtraction operation?

Assignment no. 4	4
Aim	**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.**
Objective	To understand the concept of searching method To find the performance of searching method To apply searching method to check whether an element is present in the given array/list.
Outcome	To design and implement different searching methods To calculate time and space complexity of different searching method To apply functions of different searching method on given data for checking the presence of a particular element.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open-source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with Python plugin or Pycharm IDE

Theory related to assignment:

In this assignment we will implement the different searching techniques.

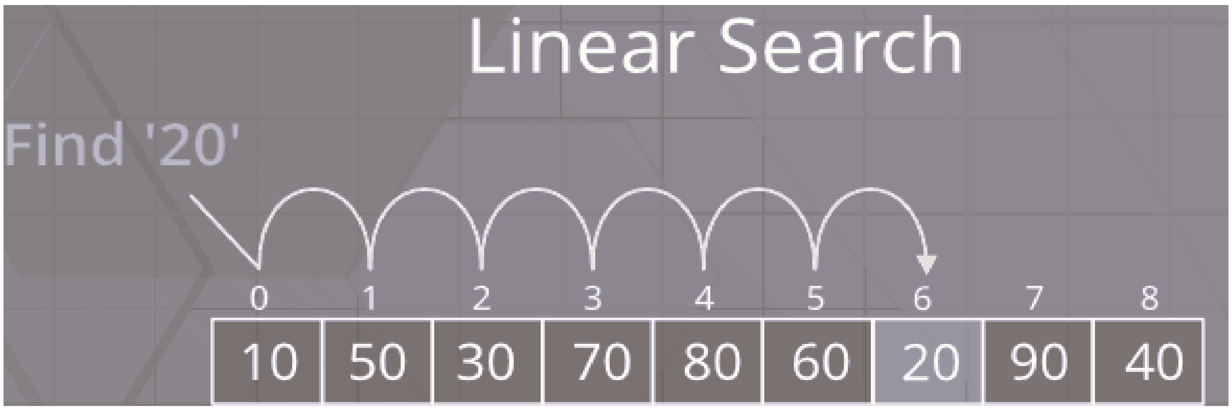
For this purpose, we need to accept the student's data in array or list. Thereafter, apply the following searching operations to search an element within an array or list.

The program should display the position of an element if it is found in input list or display not found message.

Linear Search:

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 linear search runs in at worst linear time and makes at most *n* comparisons, where *n* is the length of the list. If each element is equally likely to be searched, then linear search has an average case of (n+1)/2 comparisons, but the average case can be affected if the search probabilities for each element vary.

Example of linear search:



Steps to perform linear search

Linear Search (Array A, Value x)

```
Step 1: Set i to 0
Step 2: if i = n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

Pseudocode for Linear search:

int search(arr[], int count, int key)

```
1.  found := false
2.  position := –1, index := 0
3.  while (index < count and !found)
4.          If (key == arr[index]) then
5.                  found := true
6.                  position := index
7.                  break;
8.          End If
9.           index:=index + 1
10. end while
11. return position
```

end search

Time Complexity:

Best Case – O(1)
Worst Case – O(n)
Average Case – O(n)

Space Complexity: O(1)

Sentinel Search or Sentinel Linear Search:

Sentinel Linear Search as the name suggests is a type of Linear Search where the number of comparisons is reduced as compared to a traditional linear search. When a linear search is performed on an array of size N then in the worst case a total of N comparisons are made when the element to be searched is compared to all the elements of the array and (N + 1) comparisons are made for the index of the element to be compared so that the index is not out of bounds of the array which can be reduced in a Sentinel Linear Search.

Pseudocode Sentinel Search

procedure Sentinel_Search(int arr[], int n, int x)

```
1.  int last := arr[n - 1];     // Last element of the array
2.  arr[n - 1] := x;       // Element to be searched is placed at last index
3.  int i = 0;
4.  while (arr[i] != x)
5.          i++;
6.  arr[n - 1] := last;       // Put the last element back
7.  if ((i < n - 1) || (x == arr[n - 1]))
8.          cout << x << " is present at index " << i;
9.  else
10.         cout << "Not found“
```

end Sentinel_search

Example of sentinel search

Consider the following array of elements

5	3	6	-12	20

We have to find the element 6. The first step would be to assign the last element to the int last and place the element to be found at last index position.

i.e. int last=20 and arr[4]=6. Our array would now look like this.

5	3	6	-12	6

Now traverse though the entire array from the beginning to search for the element 6 just like linear search.

5	3	6	-12	6

5	3	6	-12	6

You will find the element at arr[2]

5	3	6	-12	6

As your element was found before the last index position, it implies that it was initially present in the array at that particular position (Refer to step 7 of pseudocode).

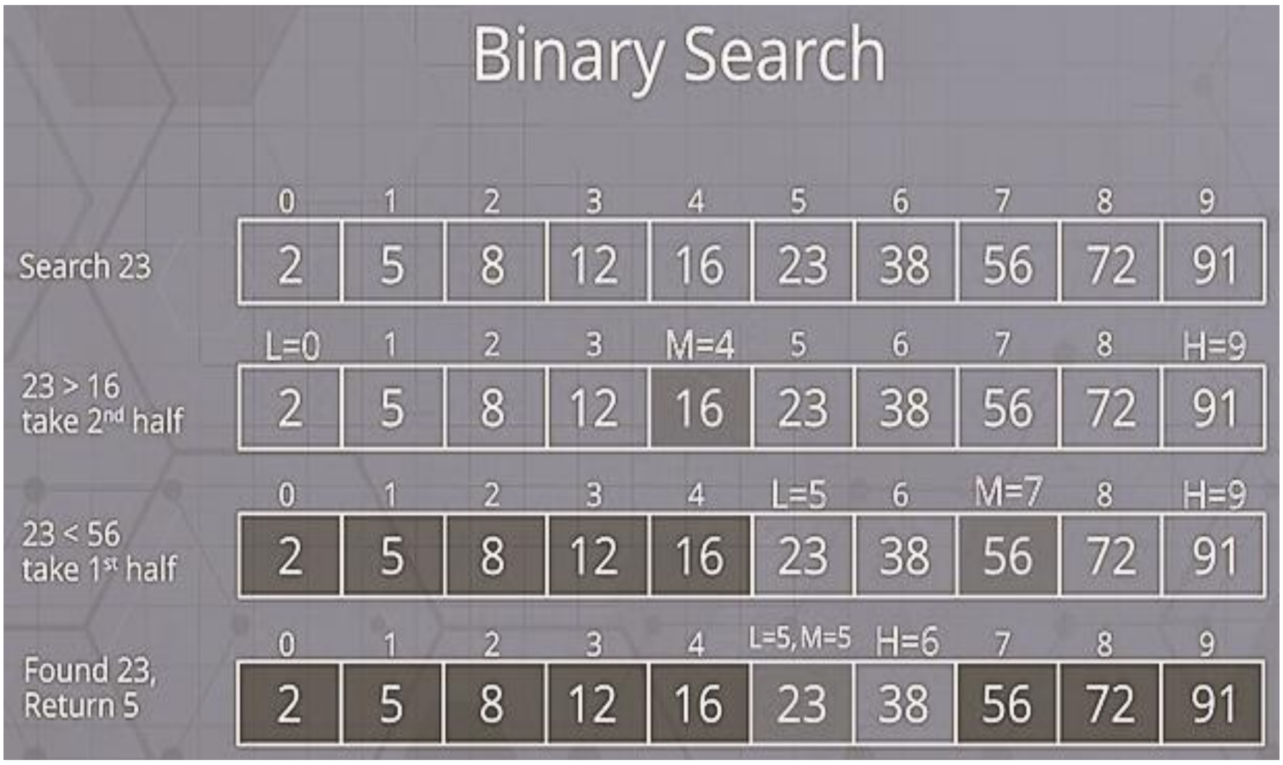
Time complexity

	Linear Search	**Sentinel Search**
Best Case	**O(1) – two comparisons**	**O(1) – three comparisons**
Worst Case	**O(n) – 2N comparisons**	**O(n) – N+2 comparisons**
Average Case	**O(n) - 2N comparisons**	**O(n) – N+2 comparisons**
Space complexity	**O(1)**	**O(1)**

Binary Search

- Search a sorted array by repeatedly dividing the search interval in half.
- It begins 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.
- If the value of the search key is greater than the item in the middle of the interval, narrow it to the upper half.
- Repeatedly check until the value is found or the interval is empty.
- We ignore half of the elements just after one comparison.
- Upper bound of Loop is decreasing by n/2 after each comparison.
- Time complexity is O(log n).

Consider the following example:



Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Binary Search Algorithm can be implemented in two ways:

1. Iterative Method
2. Recursive Method

The procedure for implementation of binary search using iterative method is as follows:

Pseudocode for Binary Search

```
procedure binarySearch(int arr[], int s, int e, int x)

        {

        // Precondition :- arr should be sorted array
        // Postcondition :- will return position of x if present otherwise -1
    1.  while (s<= e)
    2.  {
    3.  m :=floor((s+e) / 2);            //find mid element

    4.  if (arr[m] == x)                 // Check if x is present at mid
    5.          return m;
    6.  if (arr[m] < x)                  // If x greater, ignore left half
    7.          s := m + 1;
    8.  else
    9.          e := m - 1;              // If x is smaller, ignore right half
    10. }
    11.  return -1;               // if we reach here, then element was not present
        }
end binarySearch
```

The following is the procedure for implementation of binary search using recursive method:

```
Procedure binarySearch(int arr[], int s, int e, int x)

{

Precondition :- arr should be sorted array

Postcondition :- will return position of x if present otherwise -1

    1.  if(s<=e)
    2.  {
    3.  int m :=floor((s+e) / 2);
    4.  if (arr[m] == x)                 // Check if x is present at mid
    5.  return m;
    6.  if (arr[m] < x)                  // If x greater, ignore left half
    7.  s := m + 1;
    8.  return(binarySearch(arr,s,e,x));
    9.  else
    10. e := m - 1;               // If x is smaller, ignore right half
    11. return(binarySearch(arr,s,e,x));
    12. }
    13. else
    14. return -1;                  // if we reach here, then element was not present
    15. }
```

end binarySearch

Time complexity

- Best case – O(1)
- Average, worst case - O(log n)

Space complexity

- Best case – O(1)
- Average, worst case - O(log n)

Fibonacci Search

- Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
- It is very much similar to binary search
- Works on sorted arrays
- A Divide and Conquer Algorithm
- Time complexity is O(log n)
- Binary search divides given array at mid but Fibonacci search divides in unequal parts
- Binary Search uses division operator to divide range. Fibonacci Search doesn’t use /, but uses + and -. The division operator may be costly on some CPUs.
- Fibonacci Search examines relatively closer elements in subsequent steps. So when input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful.
- It is applied on nonlinear unimodal function

Pseudocode for Fibonacci Search

Procedure Fibonacci_Search(arr[ ], x, N):

```
1.    m = 0
2.    while Fibo(m) < N             // repeat till mth Fibonacci no. is less than N
3.        m = m + 1
4.    offset = -1
5.    while (Fibo(m) > 1)
6.        mid = min( offset + Fibo(m - 2) , N - 1)
7.        if (x > arr[mid])
8.            m = m - 1
9.            offset = mid
10.       elif (x < arr[mid])
11.           m = m - 2
12.       else
13.           return mid
14.   end while
15.   if(!Fibo(m - 1) and arr[offset + 1] == x)
16.       return offset + 1
17.   return -1
   End Fibonacci_Search
```

Example of Fibonacci search

Search 70 in the following array.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	20	30	40	50	60	70	80	90	100

Size=10 Fm=13 Fm-1=8

70<90. So, take first half.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	20	30	40	50	60	70	80	90	100

70>60. So, take second half.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	20	30	40	50	60	70	80	90	100

Size=8 Fm=8 Fm-1=5

Arr[0+5]=Arr[5]

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	20	30	40	50	60	70	80	90	100

70<80. Take first half.

Size=2 Fm=2 Fm-1=1

Arr[6+1]=Arr[7]

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	20	30	40	50	60	70	80	90	100

70 found.

Size=1 Fm=1 Fm-1=0

Arr[6+0]=Arr[6]

Time complexity

Best case – O(1)
Average, worst case - O(log n)

Space complexity

Best case – O(1)
Average, worst case - O(log n)

Conclusion:

The functions for different searching methods are implemented successfully on student data with efficient time and space complexity.

Review Questions:

1. What are the various applications of linear search?
2. When to use linear search?
3. Can linear search be done on Linked Lists?
4. What is the best-case time complexity for Linear Search?

5. Which search algorithm is best if data keeps changing frequently?
6. Can linear search be made parallel for execution on multiple CPU cores?
7. What is the time complexity of Linear Search for string data?
8. What is the time complexity of Linear Search for Integer data?
9. Why is the worst-case time complexity same as average case for Linear Search?
10. Compare Binary Search vs Linear Search
11. Explain how does the Sentinel Search work?
12. Is Sentinel Linear Search better than normal Linear Search?
13. Explain why complexity of Binary Search is O(log n)?
14. What is the time complexity of Fibonacci Search?

Assignment no. 5	5
Aim	**Write a python program to store second year percentage of students in array. Write function for sorting array of floating-point numbers in ascending order using a) Insertion sort b) Shell Sort and display top five scores**
Objective	To understand the concept of a sorting method To find the performance of sorting method To apply sorting method to sort the given array/list in ascending order and display top five elements.
Outcome	To design and implement the sorting techniques. To calculate time and space complexity of the given sorting methods To apply the functions of sorting methods on given data for sorting the elements in ascending order and display top five elements.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open-source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with Python plugin or Pycharm IDE

Theory related to assignment:

In this assignment we will implement the given sorting techniques.

For this purpose, we need to accept the student's data of percentages in array or list. The percentages should be accepted as floating-point numbers. Thereafter, apply the following sorting operations to sort the percentages in ascending order.

The program should display the sorted elements in ascending order as well as display the top five scores amongst them.

Sorting is the process of arranging the given data in some pre-defined order or sequence. The order can be either ascending or descending.

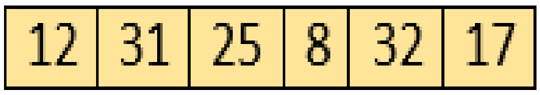
Insertion Sort:

- Insertion sort works similar to the sorting of playing cards in hands.
- It is assumed that the first card is already sorted in the card game, and then we select an unsorted card.

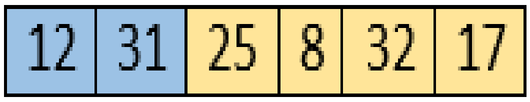
- If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.
- The same approach is applied in insertion sort. The idea behind the insertion sort is that first take one element, iterate it through the sorted array.
- Although it is simple to use, it is not appropriate for large data sets as the time complexity of insertion sort in the average case and worst case is O(n2), where n is the number of items.
- Insertion sort is less efficient than the other sorting algorithms like heap sort, quick sort, merge sort, etc

Example of Insertion sort

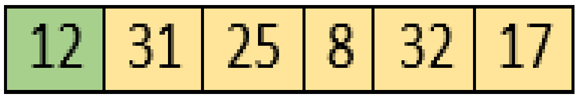
Let the elements of array are –



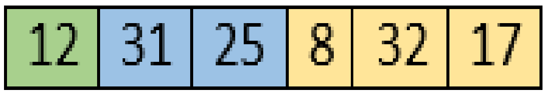
Initially, the first two elements are compared in insertion sort.

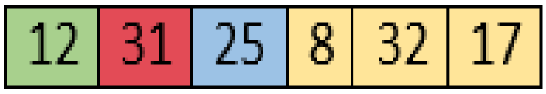


Here, 31 is greater than 12. That means both elements are already in ascending order. So, for now, 12 is stored in a sorted sub-array.



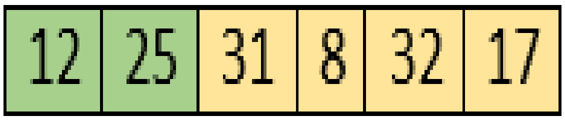
Now, move to the next two elements and compare them.



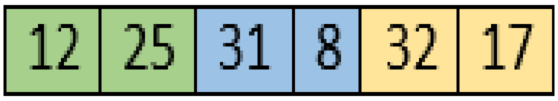


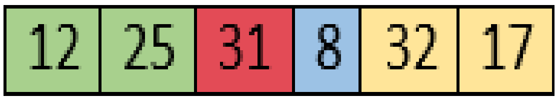
Here, 25 is smaller than 31. So, 31 is not at correct position. Now, swap 31 with 25. Along with swapping, insertion sort will also check it with all elements in the sorted array.

For now, the sorted array has only one element, i.e. 12. So, 25 is greater than 12. Hence, the sorted array remains sorted after swapping.

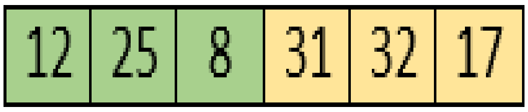


Now, two elements in the sorted array are 12 and 25. Move forward to the next elements that are 31 and 8.

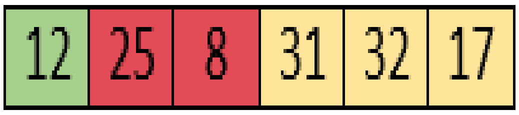




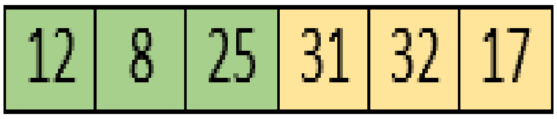
Both 31 and 8 are not sorted. So, swap them.



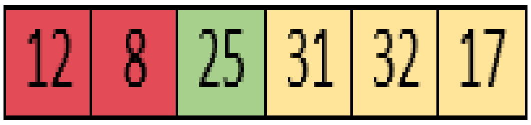
After swapping, elements 25 and 8 are unsorted.



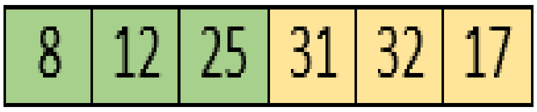
So, swap them.



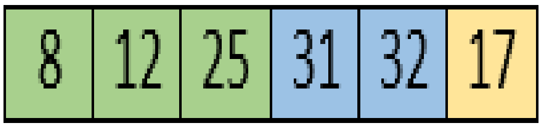
Now, elements 12 and 8 are unsorted.



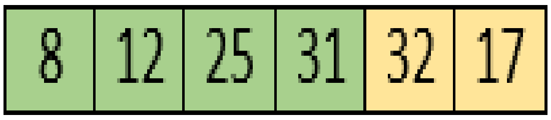
So, swap them too.



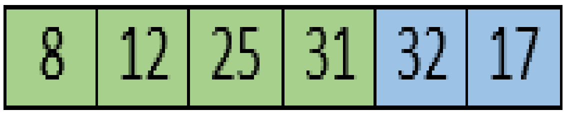
Now, the sorted array has three items that are 8, 12 and 25. Move to the next items that are 31 and 32.



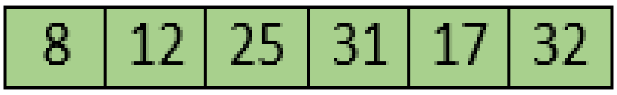
Hence, they are already sorted. Now, the sorted array includes 8, 12, 25 and 31.

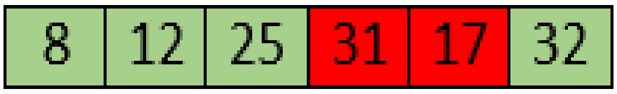


Move to the next elements that are 32 and 17.

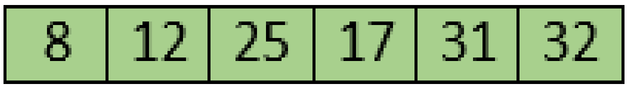


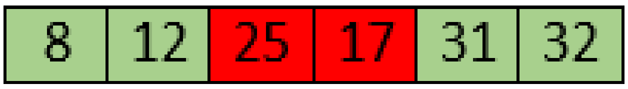
17 is smaller than 32. So, swap them.



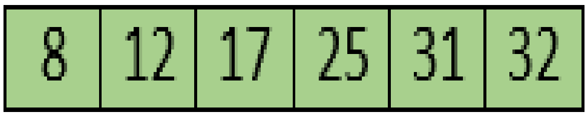


Swapping makes 31 and 17 unsorted. So, swap them too.





Now, swapping makes 25 and 17 unsorted. So, perform swapping again.



Now, the array is completely sorted.

Pseudocode for Insertion Sort:

```
Algorithm insertionSort(A [] ) {
    for i = 1 to length(A) inclusive do:
    /* select value to be inserted */
    valueToInsert = A[i]
    holePosition = i
    /*locate hole position for the element to be inserted */
    while holePosition > 0 and A[holePosition-1] > valueToInsert do:
        A[holePosition] = A[holePosition-1]
        holePosition = holePosition -1
    end while
    /* insert the number at hole position */
    A[holePosition] = valueToInsert
    end for
end procedure
```

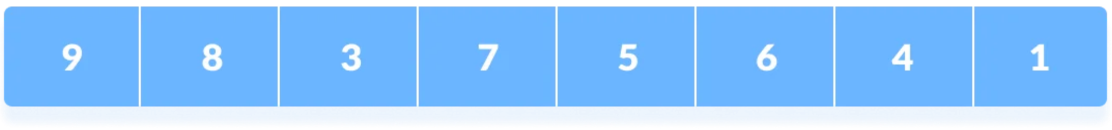
Time Complexity:
Worst Case – $O(n^2)$
Average Case – $O(n^2)$

Shell Sort:

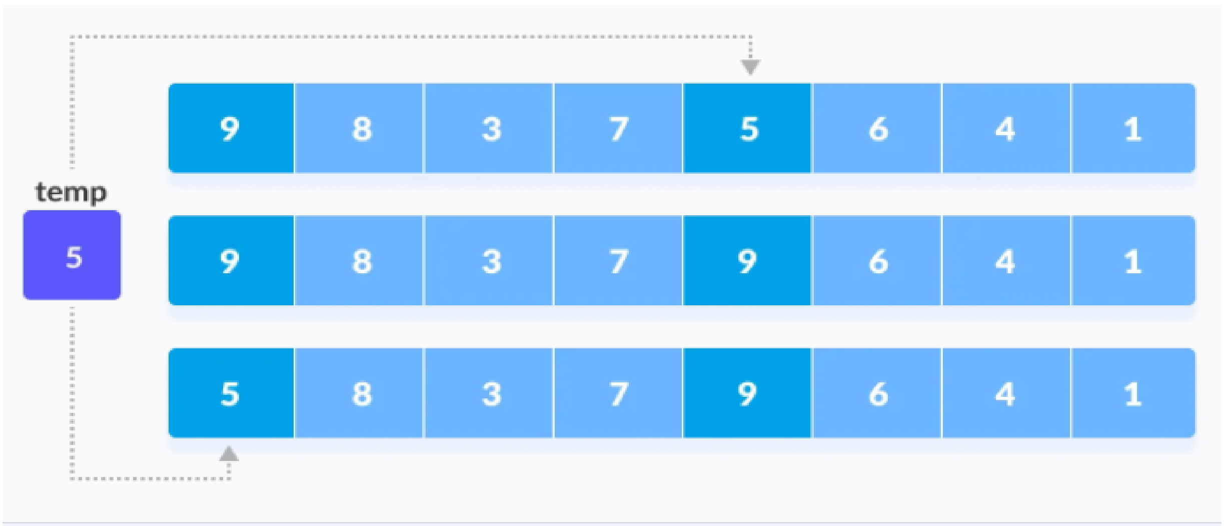
- Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm.
- This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.
- This algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval.
- This algorithm is quite efficient for medium-sized data sets as its average and worst-case complexity of this algorithm depends on the gap sequence the best known is O(n), where n is the number of items. And the worst case space complexity is O(n).

Example of Shell Sort

Suppose, we need to sort the following array.

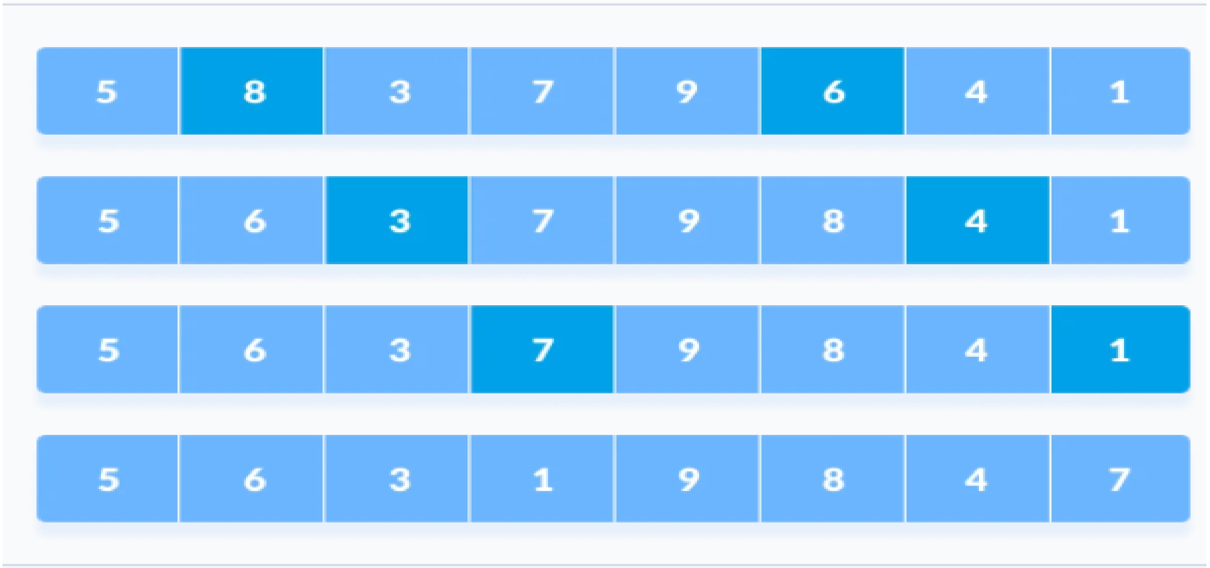


- We are using the shell's original sequence (N/2, N/4, ...1) as intervals in our algorithm.
- In the first loop, if the array size is N = 8 then, the elements lying at the interval of N/2 = 4 are compared and swapped if they are not in order.
- The 0th element is compared with the 4th element.
- If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



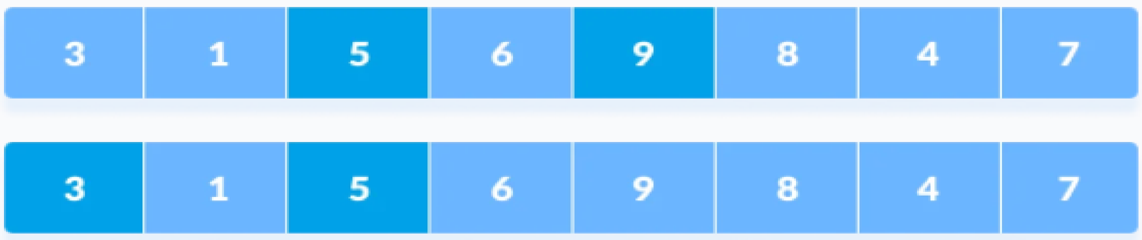
Rearrange the elements at n/2 interval

- This process goes on for all the remaining elements.



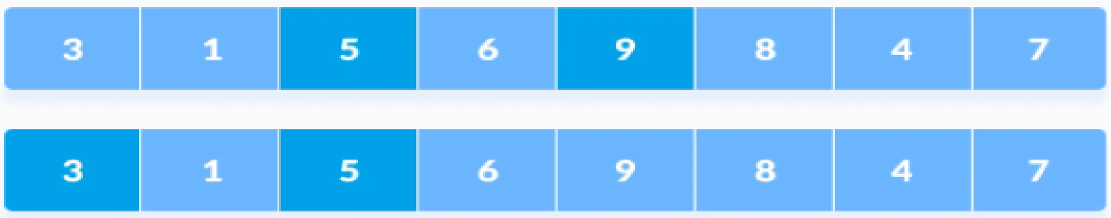
Rearrange all the elements at n/2 interval

- In the second loop, an interval of N/4 = 8/4 = 2 is taken and again the elements lying at these intervals are sorted.

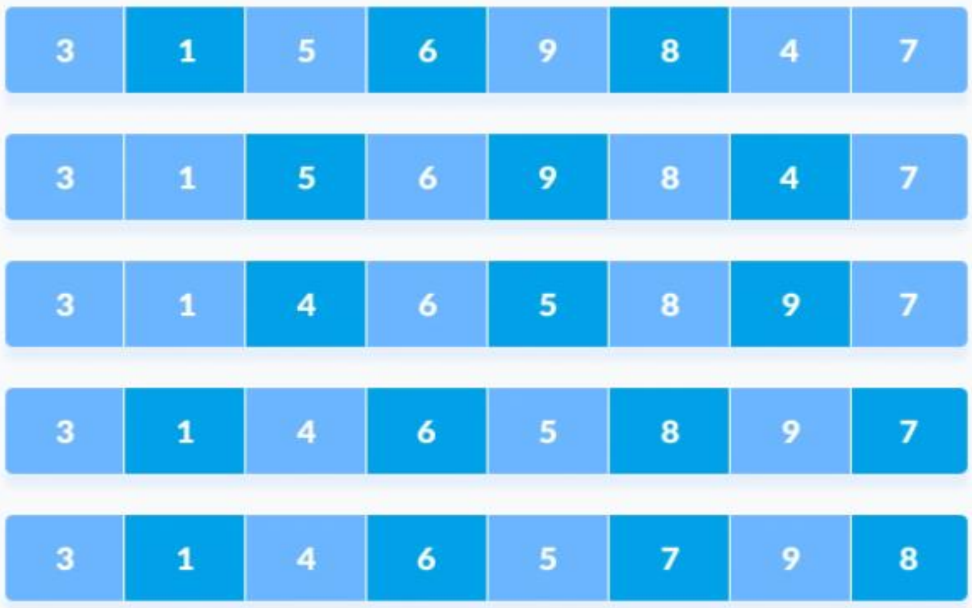


All the elements in the array lying at the current interval are compared.

- The elements at 4th and 2nd position are compared. The elements at 2nd and 0th position are also compared. All the elements in the array lying at the current interval are compared.
- The same process goes on for remaining elements.



All the elements in the array lying at the current interval are compared.



Rearrange all the elements at n/4 interval

- Finally, when the interval is N/8 = 8/8 =1 then the array elements lying at the interval of 1 are sorted. The array is now completely sorted.

Pseudocode Shell Sort

```
Algorithm shellSort(A[]) {
  /* calculate interval*/
  while interval < A.length /3 {
    interval = interval * 3 + 1
}

  while interval > 0 {

    for outer = interval to A.length; {

    /* select value to be inserted */
    valueToInsert = A[outer]
    inner = outer;

      /*shift element towards right*/
      while inner > interval -1 && A[inner - interval] >= valueToInsert {
            A[inner] = A[inner - interval]
            inner = inner - interval
}

    /* insert the number at hole position */
    A[inner] = valueToInsert

} //end for

  /* calculate interval*/
  interval = (interval -1) /3;
} //  end while

end procedure
```

Time Complexity:
Worst case, Average case: O(n)

Conclusion:

The functions for different sorting methods are implemented successfully on student percentage data with efficient time and space complexity and displayed top five scores amongst them.

Review Questions:

1. Explain the concept of sorting?
1. Define and explain insertion sort?
2. Define and explain shell sort?
3. How many passes are required in insertion and shell sort?
4. What is the time complexity of insertion sort?
5. What is the time complexity of shell sort?
6. Trace the step-by-step procedure of insertion sort on a sample array of elements.
7. Trace the step-by-step procedure of shell sort on a sample array of elements.

Assignment no	**6**
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.**
Objective	1. To understand the standard and abstract data representation Methods. 2. To identify the appropriate data structure and algorithm design method for a specified application. 3. To get the sorted array of percentage of first year students using quick sort. 4. To get the top five score of students after quick sort.
Outcome	1. To understand ,design and implement quick sort using list or array in python. 2. To analyse the quick sort time complexity 3. To design the algorithms to solve the programming problems.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with Python plugin

Theory related to assignment:

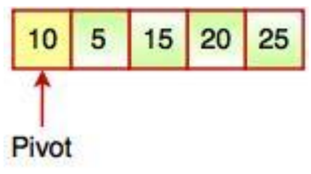
In this assignment we will implements quick sort. List or array as primitive data structure can be used to perform quick sort.

Quick Sort

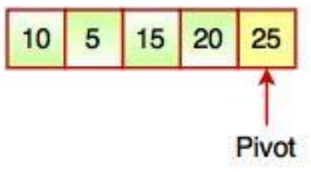
- Quick sort is also known as **Partition-exchange sort** based on the rule of **Divide and Conquer.**
- It is a highly efficient sorting algorithm.
- Quick sort is the quickest comparison-based sorting algorithm.
- It is very fast and requires less additional space, only O(n log n) space is required.
- Quick sort picks an element as pivot and partitions the array around the picked pivot.

There are different versions of quick sort which choose the pivot in different ways:

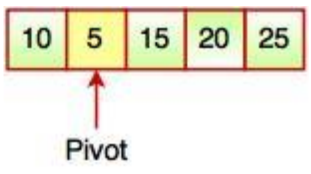
1. First element as pivot



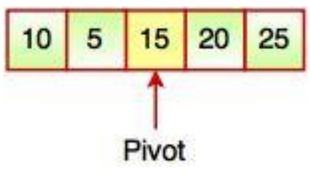
2. Last element as pivot



3. Random element as pivot



4. Median as pivot



Algorithm for Quick Sort

Step 1: Choose the highest index value as pivot.

Step 2: Take two variables to point left and right of the list excluding pivot.

Step 3: Left points to the low index.

Step 4: Right points to the high index.

Step 5: While value at left < (Less than) pivot move right.

Step 6: While value at right > (Greater than) pivot move left.

Step 7: If both Step 5 and Step 6 does not match, swap left and right.

Step 8: If left = (Less than or Equal to) right, the point where they met is new pivot.

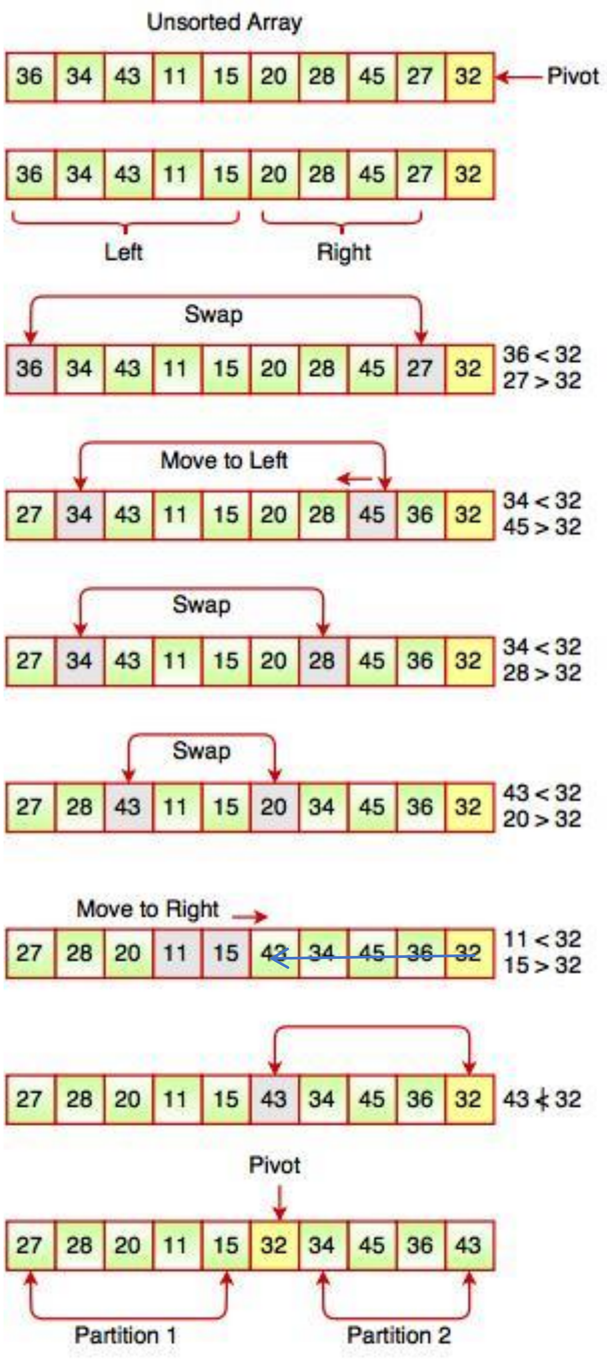


Fig. Finding Pivot Value in an Array

Pseudo Code for recursive QuickSort function:

/* low –> Starting index, high –> Ending index */

```
quickSort(arr[], low, high) {
    if (low < high) {
        /* pi is partitioning index, arr[pi] is now at right place */
        pi = partition(arr, low, high);
        quickSort(arr, low, pi – 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Pseudo code for partition ()

/ 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 */*

```
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
pivot = arr[high];
 i = (low – 1)  // Index of smaller element and indicates the
// right position of pivot found so far
for (j = low; j <= high- 1; j++){
 // If current element is smaller than the pivot
if (arr[j] < pivot){
i++;    // increment index of smaller element
 swap arr[i] and arr[j]
     }
 }
    swap arr[i + 1] and arr[high])
return (i + 1)
```

Time Complexity: O(n log n)

Space Complexity: O(logn)

Conclusion:

Quick sort implemented successfully for getting top 5 students.

Review Questions:

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

Assignment no	**7**
Aim	**7 A)** The ticket booking system of Cinemax theatre 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. **OR** **7 B)** Write C++ program for storing binary number using doubly linked lists. Write functions- a) To compute 1‘s and 2‘s complement b) Add two binary numbers
Objective	1. To understand concept of OOP. 2. To understand DLL, CDLL data structure and its class structures DLL in C++. 3. To understand primitive operations of DLL, CDLL.
Outcome	1. To implement DLL, CDLL and its primitive operations in C++ 2. To use DLL and DCLL data structure in applications.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with Python plugin

7 A) The ticket booking system of Cinemax theatre 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.

Doubly Linked List

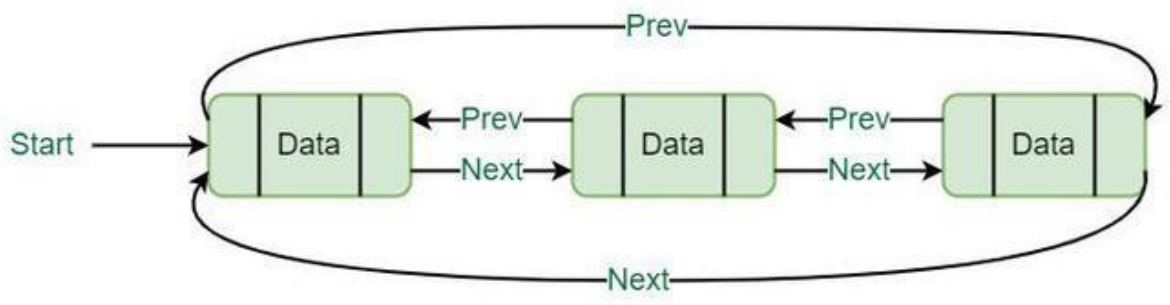
Doubly Linked List is a variation of Linked list in which navigation is possible in both ways either forward or backward easily as compared to Single Linked List. Following are important terms to understand the concepts of doubly Linked List:

Link − Each Link of a linked list can store a data called an element.

Next − Each Link of a linked list contain a link to next link called Next.

Prev − Each Link of a linked list contain a link to previous link called Prev.

Circular LinkedList − A LinkedList contains the connection link to the first Link called First and to the last link called Last.



Algorithm:

I. Book seat

1. Start

2. Create header array to store starting address of 10 rows which represented by circular doubly linked list.

3. Take input form the user to book the seat.

4. Transverse to particular row (CDLL) using corresponding header node and use seat number to reach to particular node in the CDLL.

5. Check the data in selected node whether the seat is already booked

5. Then the return seat already booked.

6. Else change the data to “X” (booked).

7. Display the all 10 rows to see the status

8. Stop.

II. Cancelling Booking

1. Start.

2. Take the input of the seat numbers to be cancelled.

3. Transverse to particular row and the node we are looking for

4. Check the data if it is “__” (not booked) return the seat was not yet booked.

5. Else change the data to “__”.

6. Display the updated rows.

7. Stop.

III. Printing the book status.

1. Start
2. Point a temporary pointer to the start of the row using corresponding header node.
3. Iterate through the list until the next pointer of this node is not equal to the header node.
4. Print the data for the given nodes.
5. Stop.

Pseudocode:

1. Booking the seat

```
Algorithm bookseat (row, seatno){
Temp:= header[row]
Count :=0
While(count<seatno)
    Temp:=temp->next
    Count:=count+1
If (temp->data == “X”}
Throw exception “Already Booked”,
}
Else {
Temp->data=’X’ //mark as booked
}
}
```

2. Cancel the seat booking

```
Algorithm bookseat (row, seatno){
Temp:= header[row]
Count :=0
While(count<seatno)
    Temp:=temp->next
    Count:=count+1
```

```
If (temp->data == " __"){
throw exception "seat not yet booked"
}
else {
temp->data='__
}
```

Conclusion: We got to know and understand doubly circular linked list and implemented it for booking by accessing and editing particular node.

7 B) Write C++ program for storing binary number using doubly linked lists. Write functions- a) To compute 1‘s and 2‘s complement b) Add two binary numbers

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

Given a Binary Number as a string, print its 1’s and 2’s complements.

1’s complement of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.In the 1’s complement format , the positive numbers remain unchanged . The negative numbers are obtained by taking the 1’s complement of positive counterparts.

for example +9 will be represented as 00001001 in eight-bit notation and -9 will be represented as 11110110, which is the 1’s complement of 00001001.

Examples:

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

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

2’s complement of a binary number is 1, added to the 1’s complement of the binary number. In the 2’s complement representation of binary numbers, the MSB represents the sign with a ‘0’ used for plus sign and a ‘1’ used for a minus sign. the remaining bits are used for representing magnitude. Positive magnitudes are represented in the same way as in the case of sign-bit or 1’s complement representation. Negative magnitudes are represented by the 2’s complement of their positive counterparts.

Examples:

2's complement of "0111" is “1001"

2's complement of "1100" is “0100"

Another trick to finding two’s complement:

Step 1: Start from the Least Significant Bit and traverse left until you find a 1. Until you find 1, the bits stay the same

Step 2: Once you have found 1, let the 1 as it is, and now

Step 3: Flip all the bits left into the 1.

Algorithm:

```
A. Prepare Binary (n)
   1. Start
   2. Assign new node (no/02) to start and last.
   3. Assign n/2 to n.
   4. While n is greater than 0.
         a. Make node pointer new node = Node (No / 02)
         b. Declare start → prev = new node
         c. New node → next = start
         d. Start = start → prev.
         e. Assign n/2 to n.
   5. Repeat step 4
   6. End

B. Print Function
   1. Start
   2. Assign Start to pointer loop.
   3. While temp is not equal to NULL.
         a. Print the data at temp.
         b. Assign temp → next to temp.
   4. End

C. One’s Complement
   1. Start
   2. Declare a node printer ‘temp’ = start
   3. While temp is not equal to NULL.
         a. If data at temp is ‘0’ change it to ‘1’.
         b. Else change it to ‘1’.
         c. Assign temp → next to temp.
   4. Repeat step 3.
   5. End.

D. Two’s Complement
   1. Start
   2. Call Ones Complement function
```

```
3. Initialize integer variable 'carry' equal to '1'.
4. Initialize node pointer 'temp' equal to lost.
5. While temp not equal to NULL
        a. If temp → data = 1 and carry = 1, change data to temp equal to zero and
           carry equal to '1'.
        b. Elseif temp → data = 0 and carry = 1 , change data temp equal to 1 and
           carry to 0.
        c. If carry = 0, break
6. Assign temp → prev to temp.
7. Repeat step 5.
8. End.
```

E. "+" operator overloading [Binary operator + (Binary b2)]

```
1. Start
2. Declare binary 'sum' object.
3. Initialize node pointer a to last and b to b2lat and initialize carry equal to "0".
4. While a is not equal to NULL and b is not equal to NULL.
        a. Initialize additional variable as:
                (a → data) ^ (b → data) ^ (carry)
        b. Add this addition node to sum of carry own node at start (addition).
        c. Make changes in the carry in similar way.
        d. Move a to its previous node, do same for b.
5. If a is having more bites than b
        a. While (a not equal to NULL)
        b. Add (a → data) ^ Carry node in the sum.
6. If b is having max bit than 'a'
   While (b not equal to NULL)
   Call sum node at start to add another node.
7. If carry is obtained, we will add carry using node at start carry.
8. Stop
```

F. Node at Start Function

```
1. Start.
2. Initialize node pointer 'new node' to new node (data).
3. If start is NULL, assign start and last equal to new node.
4. Else, connect the start of previous node with newly created node i.e. link the
   previous node and current new node.
```

<u>CONCLUSION:</u>

1. We learnt about user defined data types, doubly linked list.
2. We also learnt about operator overloading.

Assignment no	8
Aim	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) d) Sort list based on time e) Sort list based on time using pointer manipulation OR Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C++ program to store two sets using linked list. compute and display- a) Set of students who like both vanilla and butterscotch b) Set of students who like either vanilla or butterscotch or not both c) Number of students who like neither vanilla nor butterscotch
Objective	To understand the concept of linked list data structure To understand, implement linked list data structure and its operations
Outcome	After executing this assignment, Students will be able to identify and use linked list data structure for given application. Design and implement linked list data structures and its operations.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit), Eclipse

Theory related to assignment:

In this assignment we will implements linked list data structure and its operations

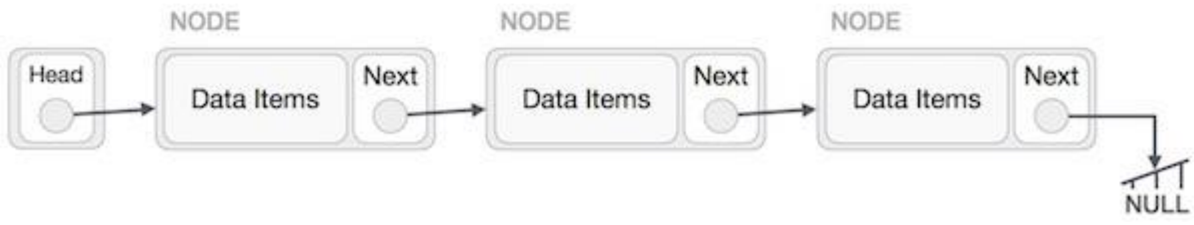
A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

Link − Each link of a linked list can store a data called an element.
Next − Each link of a linked list contains a link to the next link called Next.
LinkedList − A Linked List contains the connection link to the first link called First.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm[1]

As per the above illustration, following are the important points to be considered.

1. Linked List contains a link element called first.
2. Each link carries a data field(s) and a link field called next.
3. Each link is linked with its next link using its next link.
4. Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.

1. **Simple Linked List** − Item navigation is forward only.
2. **Doubly Linked List** − Items can be navigated forward and backward.
3. **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

Insertion − Adds an element at the beginning of the list.
Deletion − Deletes an element at the beginning of the list.
Display − Displays the complete list.
Search − Searches an element using the given key.
Delete − Deletes an element using the given key.

1.Creation

Step 1 - Define a Node structure with two members data and next

Step 2 - Define a Node pointer 'head' and set it to NULL.

2.Insertion

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

2.1 Inserting at Beginning of the list

2.2 Inserting at End of the list

2.3 Inserting at Specific location in the list

2.1 Inserting at Beginning of the list

We can use the following steps to insert a new node at beginning of the single linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set newNode→next = NULL and head = newNode.

Step 4 - If it is Not Empty then, set newNode→next = head and head = newNode.

2.2 Inserting at End of the list

We can use the following steps to insert a new node at end of the single linked list...

Step 1 - Create a newNode with given value and newNode → next as NULL.

Step 2 - Check whether list is Empty (head == NULL).

Step 3 - If it is Empty then, set head = newNode.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).

Step 6 - Set temp → next = newNode.

2.3 Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the single linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set newNode → next = NULL and head = newNode.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

Step 6 - Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.

Step 7 - Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode'

3. Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows...

3.1 Deleting from Beginning of the list

3.2 Deleting from End of the list

3.3 Deleting a Specific Node

3.1 Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4 - Check whether list is having only one node (temp → next == NULL)

Step 5 - If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)

Step 6 - If it is FALSE then set head = temp → next, and delete temp.

3.2 Deleting from End of the list

We can use the following steps to delete a node from end of the single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and

terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and

initialize 'temp1' with head.

Step 4 - Check whether list has only one Node (temp1 → next == NULL)

Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate the function. (Setting Empty list condition)

Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node.

Repeat the same until it reaches to the last node in the list.

(until temp1 →next == NULL)

Step 7 - Finally, Set temp2 → next = NULL and delete temp1.

3.3 Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and

terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and

initialize 'temp1' with head.

Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the

last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its

next node.

Step 5 - If it is reached to the last node then display 'Given node not found in the list!

Deletion not possible!!!'. And terminate the function.

Step 6 - If it is reached to the exact node which we want to delete, then check whether list

is having only one node or not

Step 7 - If list has only one node and that is the node to be deleted, then

set head = NULL and delete temp1 (free(temp1)).

Step 8 - If list contains multiple nodes, then check whether temp1 is the first node in the

list (temp1 == head).

Step 9 - If temp1 is the first node then move the head to the next node (head = head →

next) and delete temp1.

Step 10 - If temp1 is not first node then check whether it is last node in the list (temp1 → next == NULL).

Step 11 - If temp1 is last node then set temp2 → next = NULL and

delete temp1 (free(temp1)).

Step 12 - If temp1 is not first node and not last node then set temp2 → next = temp1 →

next and delete temp1 (free(temp1)).

4. Displaying a Single Linked List

We can use the following steps to display the elements of a single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last

node

Step 5 - Finally display temp → data with arrow pointing to NULL (temp → data --->

NULL)

ADT:

Node class of each Appointment

```
class SLL_Node
{
   int start;
   int end;
   int min;
   int max;
   int flag;
  SLL_Node *next;
public: SLL_Node();//constructor
}
```

Class for SLL class:

```
Class SLL{
SLL_Node *head,*last;
public: SLL(){head=NULL, last=NULL;}//constructor
create_Shedule()   ;
display_Shedule() ;
book_App()           ;
cancel_App()         ;
```

```
                sort_App();
                }
```

Pseudo code:

1. **Function Definition to create Appointment Schedule**

```
Algorithm create_Shedule()    {
   Print("Enter the Appointment Slots:");
   Read(size);
   For i:=0 to size-1    {
     temp = new  SLL_Node;        // Step 1: Dynamic Memory Allocation
     // Step 2: Assign Data & Address
     Read(temp->start);
     Read(temp->end);
     Read(temp->min);
     Read(temp->max);
     temp->flag = 0;
     temp->next = NULL;
      if (head == NULL)      {
        head = temp;
        last = head;
     }
     else {
        last->next = temp;
        last = last->next;
     }
   }
}
```

2.Function Definition to Display Appointment Schedule

```
Algorithm display_Shedule()  {
  cnt:=0
   Print(Appointment Schdule)
   Print(“ Srno.\tStart\tEnd\tMin_Dur\tMax_Dur\tStatus");
   temp = head;
   while(temp != NULL)
   {
      Print(cnt);
      Print(temp->start);
      Print(temp->end);
      Print (temp->min);
      Print (temp->max)
      if(temp->flag)
         Print (Booked);
      else
        print(Free);
      temp = temp->next;
      cnt++;
   }
}
```

3. Function Definition to Book Appointment

```
Algorithm book_App()     {
 Print(Please enter Appointment time:);
 Read(start);
 temp = head;
 while(temp != NULL)    {
      if(start == temp->start)
      {
         if(temp->flag == 0)
         {
            Print(“Appointment Slot is Booked”);
            temp->flag = 1;
         }
         else
```

```
            print(“ Appointment Slot is not Available!!!";
        }
        temp = temp->next;
    }
}
```

4. Function Definition to Cancel Appointment

```
Algorithm cancel_App()    {
Print(“Please enter Appointment time to Cancel: ");
Read(start);
temp = head;
while(temp != NULL)  {
        if(start == temp->start)   {
            if(temp->flag == 1)     {
                print(“Your Appointment Slot is Canceled!!!");
                temp->flag = 0;
            }
            else
                print(“Your Appointment was not Booked!!!");
        }
        temp = temp->next;
    }
}
```

5.Function Definition to Sort Appointments

```
Algorithm  sort_App() {
For i:=0 to size-1   {
         temp = head;
         while(temp->next != NULL)        {
             if(temp->start > temp->next->start)
             {
                 val = temp->start;
                     temp->start = temp->next->start;
```

```
            temp->next->start = val;


          val = temp->end;
            temp->end = temp->next->end;
            temp->next->end = val;


          val = temp->min;
            temp->min = temp->next->min;
            temp->next->min = val;


          val = temp->max;
            temp->max = temp->next->max;
            temp->next->max = val;


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

Time Complexity: O(n)

Space Complexity: O(n)

Conclusion:

The functions for different appointment schedule are implemented successfully using linked list data structure.

Review Questions:

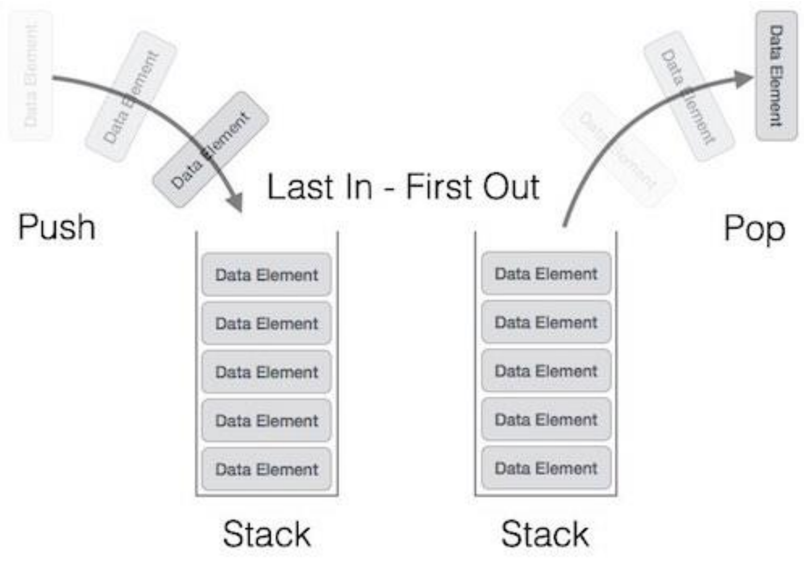
1. What is Singly Linked List ?
2. What is Circular Linked List ?
3. What is Doubly Linked List ?
4. Write a function to delete a linked List ?
5. Why Quick Sort preferred for Arrays and Merge Sort for Linked List?
6. What is a Linked list? 2. Can you represent a Linked list graphically?
7. How many pointers are required to implement a simple Linked list?
8. How many types of Linked lists are there?
9. How to represent a linked list node?
10. Describe the steps to insert data at the starting of a singly linked list.
11. How to insert a node at the end of Linked list?
12. How to delete a node from linked list?
13. How to reverse a singly linked list?

14. What is the difference between singly and doubly linked lists?
15. What are the applications that use Linked lists?
16. What will you prefer to use a singly or a doubly linked lists for traversing through a list of elements?

Assignment no	9
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
Objective	To understand the concept of Stack data structure To implement Stack data structure and its operations for given application.
Outcome	After executing this assignment, Students will be able to identify and use Stack data structure for given application. Students design and implement stack data structure and its operations
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse

Theory related to assignment:
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.

The following diagram depicts a stack and its operations −



A stack can be implemented by means of Array, Structure, Pointer, and Linked List.

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.

Application of Stack:
Expression Evolution
Expression Conversion Infix to Postfix, Infix to Prefix, Postfix to Infix Prefix to Infix Parsing.

Simulation of recursion Function call Algorithm:
1) Declare a character stack S.
2) Now traverse the expression string exp.
a) If the current character is a starting bracket (‘(‘or ‘{‘or ‘[‘) then push it to stack.
b) If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack
and if the popped character is the matching starting bracket, then fine else parenthesis is not balanced.
3) After complete traversal, if there is some starting bracket left in stack then “not balanced”

Test Cases
1. (A+(B*C) Expression is invalid
2. (A+(B*C)) Expression is valid

Example:
Input:
Enter any expression having number of opening & closing brackets.
Output:
Entered expression is well parenthesized or not.
Algorithm:
Step 1: Declare a character stack S.
Step 2: Now traverse the expression string exp.

```
If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘)
then
    push it to stack.
If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’)
then
   pop from stack and
        if the popped character is the matching starting bracket
        then
             fine
        else
             parenthesis are not balanced
```

Step 3: After complete trave
rsal, if there is some starting bracket left in stack
then “not balanced”
Step 4: Exit

ADT:

```
class Stack
{
   char s[MAX];
   int top;
   public:
      Stack()
      {
         top=-1;
      }
      void push(char ch);
      char pop();
      bool isEmpty();
      bool isFull();
      bool checkParenthesis(char expr[]);
};
```

Pseudo Code:

1. isEmpty()

```
Algorithm isEmpty() {

   if(top==-1)

      return 1;

   else

      return 0;

}
```

2 isFull()

```
Algorithm isFull() {

   if(top==MAX-1)

      return 1;

   else

      return 0;

}
```

3.push

```
Algorithm push(char ch) {
    if(!isFull())     {
        top++;
        s[top]=ch;
    }
}
```

4. pop

```
 Algorithm pop() {
    if(!isEmpty())     {
        char ch=s[top];
        top--;
        return ch;
    }
    else
        return '\0';
}
```

5.checkParanthesis

```
Algorithm checkParenthesis(char expr[])
{
   // Traversing the Expression
    For i:=0 to length(expr)     {
        if (expr[i]=='('||expr[i]=='['||expr[i]=='{')
        {
            // Push the element in the stack
            push(expr[i]);
            continue;

        }

    // IF current current character is not opening  bracket, then it must be closing. So stack
//cannot be empty at this point.

        if (isEmpty())

           return false;

         switch (expr[i])        {

        case ')':

            // Store the top element in a

            x = pop();

            if (x=='{' || x=='[')

                return false;

            break;

    case '}':

            // Store the top element in b

            x = pop();

            if (x=='(' || x=='[')

                return false;

            break;

        case ']':

            // Store the top element in c

            x = pop();

              if (x =='(' || x == '{')

                return false;

            break;

        }

    }
    // Check Empty Stack

    return (isEmpty());

}
```

Conclusion:
Different stack operations are implemented successfully.

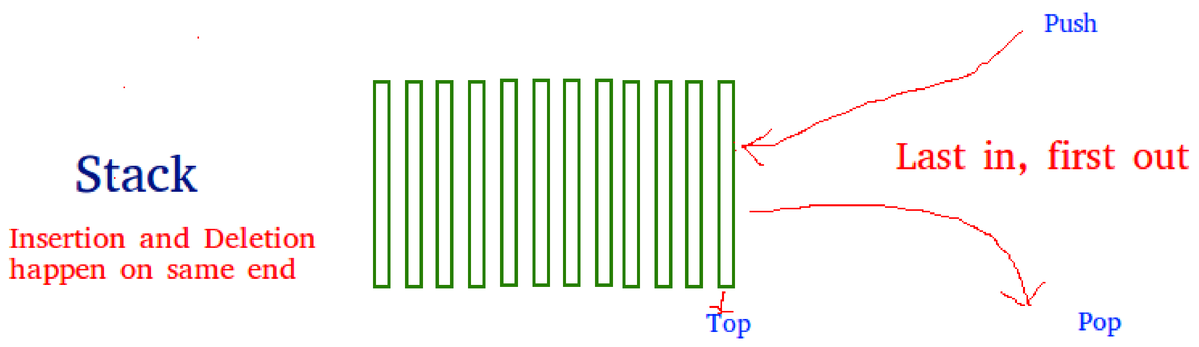
Review Questions:

1. Which data structure is required to check whether an expression contains balanced parenthesis?
2. What data structure would you mostly likely see in a non-recursive implementation of a recursive algorithm?
3. Process of removing an element from stack is called….
4. In a stack, if a user tries to remove an element from empty stack it is called---
5. Convert the following infix expressions into its equivalent postfix expressions.
   (A + B -D)/(E – F)+G
6. Consider the following operation performed on a stack of size
   Push(1);Pop();Push(2);Push(3)Pop();Push(4);Pop();Pop();Push(5);
7. If the elements “A”, “B”, “C” and “D” are placed in a stack and are deleted one at a time, what is the order of removal?
8. Convert the following Infix expression to Postfix form using a stack.
   x + y * z + (p * q + r) * s, Follow usual precedence rule and assume that the expression is legal.
9. Define a stack?
10. Stack data structure uses which principle?
11. Stack belongs to which type of data structure?
12. What do you mean by stack underflow?
13. What do you mean by stack overflow?
14. List out the basic operations of a stack? 7. How to implement stack?

Assignment no	**10**
Aim	**Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.**
Objective	To understand the stack data structure and its use in real-time applications To understand , expression conversion as infix to postfix and its evaluation using stack
Outcome	To implement primitive operations on stack. To make use of stack for converting infix expression to postfix.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with C++ plugin

Theory:-

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



There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

Consider a simple expression: A + B .This notation is called Infix Notation.

A + B in Postfix notation is A B +

As you might have noticed, in this form the operator is placed after the operands (Hence the name 'post'). Postfix is also known as Reverse Polish Notation. Similarly for Infix, the operator is placed inside the operands. Likewise the equivalent expression in prefix would be + A B, where the operator precedes the operands.

Examples of Infix, Prefix, and Postfix		
Infix Expression	**Prefix Expression**	**Postfix Expression**
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

An Expression with Parentheses		
Infix Expression	**Prefix Expression**	**Postfix Expression**
(A + B) * C	* + A B C	A B + C *

ADT:

```
class Stack
{
        finite ordered list with 0 or more elements.
        int top;
public:
        Stack();//create an empty stack and initializes top;
        void push( ele);//to insert element into stack;
        void pop();//to remove element from stack;
        int top();//returns value of top
        Isempty();//check wheather stack is empty or not
        Isfull();
}
```

Algorithm:-

```
1. Scan infix expression Q from left to right and repeat step 2 to 5 for each element of
   Q until the STACK is empty.
2. If an operand is encountered add it to P(answer).
3. If a left parenthesis is encountered push it onto the STACK.
4. If an operator is encountered, then
     • Repeatedly pop from STACK and add to P each operator which has same
       precedence as or higher precedence than the operator encountered.
     • Push the encountered operator onto the STACK.
5. If a right parenthesis is encountered, then
     • Repeatedly pop from the STACK and add to P each operator until a left
       parenthesis is encountered.
     • Remove the left parenthesis; do not add it to P.
6. Exit
```

```
 for (each character ch in the infix expression){
   switch(ch){
     case operand:    // append operand to end of PE
       postfixExp = postfixExp + ch
       break
     case '(':        // save '(' on stack
       aStack.push(ch)
       break
     case ')':        // pop stack until matching '('
       while (top of stack is not '('){
         postfixExp = postfixExp + (top of aStack)
         aStack.pop()
       }  // end while
       aStack.pop()   // remove the '('
       break
     case operator:   // process stack operators of
                      // greater precedence
       while (!aStack.isEmpty() and
              top of stack is not '(' and
              precedence(ch) <= precedence(top of aStack)){
         postfixExp = postfixExp + (top of aStack)
         aStack.pop()
       } // end while
       aStack.push(ch)   // save new operator
       break
   } // end switch
 } // end for
// append to postfixExp the operators remaining on the stack
while(!aStack.isEmpty()){
  postfixExp = postfixExp + (top of aStack)
  aStack.pop()
} // end while
```

Evaluating Postfix expression:

Start with an empty stack.
We scan P from left to right.
While (we have not reached the end of P)
If an operand is found push it onto the stack End-If
If an operator is found

    Pop the stack and call the value A
    Pop the stack and call the value B
    Evaluate B op A using the operator just found.
    Push the resulting value onto the stack

• End-If
End-While Pop the stack (this is the final value)

Complexity:

Time complexity = O(n).

Conclusion:
Implemented C++ program for expression conversion as infix to postfix and its evaluation using stack

Review Questions:

1. What is Stack and where it can be used?
2. What are Infix, prefix, Postfix notations?
3. Why Are Stacks Useful?
4. Explain what are Infix, Prefix and Postfix Expressions?
5. Why do we need Prefix and Postfix notations?

Assignment no	11 (Group E)
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, display job and delete job from queue.**
Objective	To understand the concept of queue data structure To understand job queue of operating system & implement it using queue To use array in C++ to implement job queue scheduling (add job, display job & delete job)
Outcome	To understand ,design and implement queue data structure using array in C++ To write/implement user defined functions/modules for different operations of queue (create, insert, delete, display) To write menu driven, modular program in C++.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse IDE with C++ (CDT Plugin) & linux gcc compiler

Theory related to assignment:

Queue is a collection whose elements are added at one end (the ***rear*** or ***tail*** of the queue) and removed from the other end (the ***front*** or ***head*** of the queue) i.e. queue is opened at both end. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).

A queue is a ***FIFO*** (first in, first out) data structure i.e., the data item stored first will be accessed first.

Example: Any waiting line is a queue:

- The check-out line at a grocery store
- The cars at a stop light
- An assembly line

Queue Operations:

enqueue : add an element to the tail of a queue

dequeue : remove an element from the head of a queue

first : examine the element at the head of the queue ("peek")

Other useful operations **(e.g. is the queue empty)**

<u>It is not legal to access the elements in the middle of the queue!</u>

ADT :

```
ADT representation using class for Set
Class Queue {
int item[MAX];          // array to hold queue elements of size MAX
int FRONT;
int REAR;
public: //Operator definition in ADT
void qInsert( q,val); // insert val in queue, pointed by q
void show(); //displays complete queue
int qDelete(Q); // delete an element from Q & returns it.
}
```

Pseudo Code:

Insert an item into queue

```
procedure qInsert(q, val)
        Purpose: To insert val into queue pointed by q
        Pre-condition: queue should not be full
        Post condition: element by name val will be inserted into queue
        begin
    1.  if q->Rear = MAX-1   // check if queue is full
            a.  print "Overflow" and go to step 5
    end if
    2.  if q->FRONT = -1    // check if queue is empty
            a.  set q->FRONT = 0
    end if
    3.  set q->REAR = q->REAR +1    //increment rear by 1
    4.  set q->item[q->REAR]=val      // insert val
    5.  end
```

Delete an item from queue

```
procedure qDelete(q)
        Purpose : To delete an item from queue pointed by q
        Pre-condition: queue should not be empty
        Post condition: element deleted from queue will be returned
        Begin
    1.  if q->FRONT = -1   // check if queue is EMPTY
            a.  print "Underflow" and go to step 5
            b.  return 0 and go to step 5
        end if
    2.  set del_val=q->item[q->FRONT]   // item to be deleted saved in del_val
    3.  if q->FRONT = q->REAR    // check if only 1 element in queue
            a.  set q->FRONT = q->REAR = -1   // set queue to initial condition as empty
        else
            set  q->FRONT = q->FRONT+1    // increment FRONT by 1
        end if
    4.  retun del_val
    5.  end
```

In case of array implementation

Time Complexity: O(1)

Space Complexity: O(1)

Conclusion:

The queue data structure is used to simulate the job queue and performed primitive operations using C++

Review Questions:

1. What is set queue data structure and its applications?
2. What is priority queue?
3. How to use linked list to implement queue data structure?
4. How to use queue to implement stack data structure?
5. How to implement priority queue
6. How queue is used in computing problems?
7. What is circular queue & how it is different than normal linear queue?
8. What will be time complexity in case of linked list implementation of queue?

Assignment no	12
Aim	**Write program to implement a priority queue in C++ using an order list/array to store the items in the queue. Create a class that includes the data items (which should be template) and the priority (which should be int). The order list/array should contain these objects, with operator <= overloaded so that the items with highest priority appear at the beginning of the list/array (which will make it relatively easy to retrieve the highest item.)**
Objective	• To understand concept of Priority queue. • To Understand how this can be used to implement specific application • To understand the concept of operator overloading.
Outcome	• To implement operations on priority queue. • To write functions to use queue for job scheduling and prioritizing jobs. •
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with C++ plugin

Theory related to assignment:

Priority Queue

- It is an abstract data type that is similar to a queue, and every element has some priority value associated with it.
- The priority of the elements in a priority queue determines the order in which elements are served (i.e., the order in which they are removed).
- If in any case the elements have same priority, they are served as per their ordering in the queue.

Therefore, all the elements are either arranged in an ascending or descending order.

Properties of Priority Queue

A priority Queue is an extension of the queue with the following properties.

- Every item has a priority associated with it.
- An element with high priority is dequeued before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.

Operations of a Priority Queue:

A typical priority queue supports the following operations:

1) Insertion in a Priority Queue

When a new element is inserted in a priority queue, it moves to the empty slot. However, if the element is not in the correct place, then it will be compared with other elements already in queue and are swapped . The swapping process continues until all the elements are placed in the correct position.

2) Deletion in a Priority Queue

It will remove the element which has maximum priority first. Thus, you remove 1st element from the queue as already elements are arranged according to priority so can be removed using FIFO principle

How to Implement Priority Queue?

Priority queue can be implemented using the following data structures:

Arrays

Linked list

Heap data structure

Implement Priority Queue Using Array:

A simple implementation is to use an array of the following structure/class

```
 Class item {
       int id;
       int priority;
}
```

<u>Operations on Priority Queue</u>

enqueue(): This function is used to insert new data into the queue.

dequeue(): This function removes the element with the highest priority from the queue.

peek()/top(): This function is used to get the highest priority element in the queue without removing it from the queue.

Pseudocode

<u>Enqueue Operation</u>

Algorithm Enqueue(q[N], f, r, item(process no , priority))

```
{//q  is an array of  size N ,item is element to be inserted.
1. if(r==N-1)
          1.1 Print “overflow"
2. if(f==-1)
             2.1 f=0 , r=0
             2.2 Enter process no and priority  ,item
             2.3 q[r]=item
  else
        2.1        r=r+1
        2.2 Enter process no and priority, item // object of structure
        2.3 a[r]=item
             2.2 j=r-1
             2.4 while( (q[j]  <= item ) && (j >= F))
             2.4.1q[j+1] = q[j]
             2.4.2 j=j-1
             2.5 q[j+1] = item;
        }
```

Dequeue Operation

```
Algorithm Dequeue(q[N],item,f,r)
  {
        1. if (( f == – 1 ) || (f == r+1))
       1.1 Print  “QUEUE EMPTY”
else
       1.1  f = f +1
 }
```

In the above code we have have to use concept of operator overloading as mention in problem statement

```
int operator <= (item obj1,item ob2) {
    if(ob1 . p < =ob2 . p)
   return 1;
```

```
        else
    return 0;
}
```

Time Complexity

enqueue(): O(n)

dequeue():O(1)

peek()/top(): O(1)

operator <= overloaded

ADT :

```
Class Priority queue
{
Item A[] //a finite ordered list with zero or more elements(Process id and priority associated with them )
Public:
Queue();//create queue with some initial capacity.
Bool isempty();//to check queue is empty or not.
int peek();//return element at front.
void enqueue();//insert item at rear and arrange
void dequeue;//delete front element of the queue.
}
```

Conclusion:

The various operation of priority queue are implemented successfully using array data structure.

Review Questions:

1. What is priority queue?
2. Explain different operation of priority queue?
3. How priority queue is different from simple queue?
4. How can we use list to implement priority queue?
5. What is the complexity of various operations of queue?
6. What is an ADT? How can we define ADT of priority queue?
7. What are the various application of priority queue?
8. What is the advantage and disadvantage of implementing priority queue using an Array?

Assignment no	13
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.**
Objective	To understand the concept of a double-ended queue To study the basic operations of Deque To study the applications of Deque
Outcome	To understand ,design and implement basic operations of Deque in C++ To write menu driven, modular program in C++
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with Python plugin or Pycharm IDE

Theory related to assignment:

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

The double-Ended queue is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail). It is also often called a head-tail linked list, though properly this refers to a specific data structure implementation. In a sense, this hybrid linear structure provides all the capabilities of stacks and queues in a single data structure.

<u>Implementation Details:</u>

Create a class Deque with elements and operations. The Deque can be implemented using

1. **Single-linked list or Doubly linked list**
2. **Array**
3. **Singly or doubly circular linked list**

Operations :

1. **Insert at first**
    - **a.** Check the position of the front
    - **b.** If front< 1 reinitialize the front to the last index
    - **c.** Else decrease the front by 1 for the array
    - **d.** Add the new key element into the array/LinkedList
2. **Insert at rear**
    - a. Check if the array is full.
    - b. If the deque is full, reinitialize rear =0
    - c. Else, increase the rear by 1
    - d. Add the new key element into the array/linked list
3. **Delete from front**
    - a. Check if the deque is empty.
    - b. If the deque is empty, deletion not possible
    - c. If the deque has only one element i.e front = =rear, set front=-1 and rear=-1
    - d. Else if the front is at the end front==n-1, set front=0
    - e. Else front= front +1

3, **Delete from the rear**

- **a.** Check if the deque is empty.
- b. If the deque is empty, deletion not possible
- c. If the deque has only one element i.e. front == rear, set front=-1 and rear=-1
- d. Else if the rear is at the front set rear=n-1
- e. Else rear = rear -1

4. **Deque empty**

    If front==-1 the deque is empty.

5. **Deque full**

    If front==0 and rear == n-1 OR front = rear +1 then the deque is full.

Time complexity:

Each operation time complexity is O(1).

Conclusion:

Double-ended queue operations are studied and implemented. The time complexity is analysed and it is O(1).

Review Questions:

1. What is a double-ended queue and its applications?
2. Explain the different operations of Dequeue
3. How to use class to implement Dequeue operations?
4. How to use the array to implement Dequeue operations?

5. How to use SLL/DLL to implement Dequeue operations?
6. How to use Circular SLL/DLL to implement Dequeue operations?
7. Explain the complexities of different operations on Dequeue.