ARMY INSTITUTE OF TECHNOLOGY

Dighi Hills, Alandi Road, Pune - 411015

DEPARTMENT OF COMPUTER ENGINEERING

DE MME GGGI MMP GGGI

Sr. No.	ASSIGNMENT	Page No.
	GROUP 'A'	
1.	In In Second year Computer Engineering class of M students, set A of students play cricket and set B of students play badminton. Write C/C++ program to find and display- i. Set of students who play either cricket or badminton or both ii. Set of students who play both cricket and badminton iii. Set of students who play only cricket iv. Set of students who play only badminton v. Number of students who play neither cricket nor badminton (Note- While realizing the set duplicate entries are to avoided)	1
2.	Write C/C++ program to store marks scored for first test of subject 'Data Structures and Algorithms' for N students. Compute I. The average score of class ii. Highest score and lowest score of class iii. Marks scored by most of the students iv. list of students who were absent for the test	4
3.	Department library has N books. Write C/C++ program to store the cost of books in array in ascending order. Books are to be arranged in descending order of their cost. Write function for a) Reverse the contents of array without using temporary array. b) Copy costs of books those with cost less than 500 in new array c) Delete the duplicate entries using temporary array d) Delete duplicate entries without using temporary array e) Count number of books with cost more than 500.	6
4	It is decided that weekly greetings are to be furnished to wish the students having their birthdays in that week. The consolidated sorted list with desired categorical information is to be provided to the authority. Write C++ program for array of structures to store students PRNs with date and month of birth. Let Array_A and Array_B be the two arrays for two SE Computer divisions. Arrays are sorted on date and month. Merge these two arrays into third array Array_SE_Comp_DOB resulting in sorted information about Date of Birth of SE Computer students.	8
5.	Write C/C++ program for storing matrix. Write functions for a) Check whether given matrix is upper triangular or not b) Compute summation of diagonal elements c) Compute transpose of matrix d) Add, subtract and multiply two matrices e) An m x n *matrix* is said to have a saddle point if some entry *a[i][j]* is the smallest value in row *i* and the largest value in *j*. Write C/ C++ function that determines the location of a saddle point if one exists.	10
6.	Write C++ program for sparse matrix realization and operations on it- Transpose, Fast Transpose and addition of two matrices	12

7.	Write a C++ program to realize polynomial equation and perform operations. Write function a) To input and output polynomials represented as bmxem+ bm-1xem-1 +….. +b0xe0. Your functions should overload the << and >> operators. 1. b) Evaluates a polynomial at given value of x 2. c) Add two polynomials 3. d) Multiplies two polynomials	14
	GROUP 'B'	
8.	Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member's information using singly linked list. Store student PRN and Name. Write functions to a) Add and delete the members as well as president or even secretary. b) Compute total number of members of club c) Display members d) Display list in reverse order using recursion e) Two linked lists exist for two divisions. Concatenate two lists.	17
9.	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. .	21
10.	Design a linked allocation system to represent and manipulate univariate polynomials with integer coefficients Write and test the following functions: 1. istream&**operator** >>(istream& is, Polynomial& x): Read in an input polynomial and convert it to its circular list representation using a head node. 2. ostream&**operator**<< (ostream&os, Polynomial& x): Convert x from its linked list representation to its external representation and output it. 3. Polynomial:: Polynomial(**const** Polynomial& a) [Copy Constructor]: Initialize the polynomial *this to the polynomial a. 4. **const** Polynomial& Polynomial :: **operator=**(**const** Polynomial& a) [Assignment Operator]: Assign polynomial a to***this**. 5. Polynomial:: Polynomial ( ) [Destructor]: Return all nodes of the polynomial ***this** to the available-space list. 6. *Polynomial* ***operator+*** *(**const** Polynomial& a, **const** Polynomial& b) [Addition]:* Create and return the polynomial a + b. a and b are to be left unaltered. ***7. Polynomial operator* (constPolynomial& a, constPolynomial& b) [Multiplication]:*** Create and return the polynomial a * b. a and b are to be left unaltered. 8. **float*Polynomial ::Evaluate(float x):*** Evaluate the polynomial *this at *x* and return the result.	24

	GROUP C	
11.	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
12.	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions i. Operands and operator, both must be single character. ii. Input Postfix expression must be in a desired format. iii. Only '+', '-', '*' and '/ ' operators are expected	30
	GROUP D	
13	Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.	32
14	Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using circular queue using array.	34
	GROUP E	
15	a) Write C++ 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 C++ 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.	36
16	Write C++ program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using a) Selection Sort b) Bubble sort and display top five scores. c)Insertion sort b) Shell Sort and display top five scores.	39
17	A) Write C++ 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. B) Write C++ program to store Xth percentage of students in array. Write function for sorting array of floating point numbers in ascending order using radix sort and display top five scores.	41

Assignment No -1

Title:

In Second year Computer Engineering class of M students, set A of students play cricket and set B of students play badminton. Write C/C++ program to find and display-

i. Set of students who play either cricket or badminton or both

ii. Set of students who play both cricket and badminton

iii. Set of students who play only cricket

iv. Set of students who play only badminton

v. Number of students who play neither cricket nor badminton

(Note- While realizing the set duplicate entries are to avoided)

Objective:

This assignment will help the students to realize the implementations of different set operations like union, intersection, difference; symmetric difference can be performed using arrays, functions and pointers.

Theory:

1. Arrays:

An array is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They can be used to store the collection of primitive data types such as int, float, double, char, etc of any particular type. An array in C/C++ can also store derived data types such as structures, pointers etc. Array elements are accessed by using an integer index. Array index starts with 0 and goes till the size of the array minus 1.

2. Functions:

A function is a set of statements that take inputs, do some specific computation, and produce output. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.In simple terms, a function is a block of code that only runs when it is called.

It helps in reducing code redundancy , makes code modular and provides abstraction.

There are two types of functions :

1. User defined functions :

   User Defined functions are user/customer-defined blocks of code specially customized to reduce the complexity of big programs. They are also commonly known as “tailor-made functions” which are built only to satisfy the condition in which the user is facing issues meanwhile reducing the complexity of the whole program.

2. Library functions :

   Library functions are also called “builtin Functions“. These functions are a part of a compiler package that is already defined and consists of a special function with special and different meanings. Builtin Function gives us an edge as we can directly use them without defining them whereas in the user-defined function we have to declare and define a function before using them.

   For Example: sqrt(), setw(), strcat(), etc.

Algorithm:

Steps for Union Operation

Union of Set A & Set B is stored in Set C

1. Copy Set A as it is in Set C.
2. K=limit1, limit3=limit1
3. For J=0 to limit2-1
   For I=0 to limit1-1
   If B[J]=A[I]
   Break
   If I=limit1
   C[K]=B[J]
   K=K+1
   Limit3=limit3+1
4. Return (limit3)

Steps for Intersection Operation

Intersection of Set A & Set B is stored in Set C

1. K=0, limit3=0
2. For J=0 to limit2-1
    For I=0 to limit1-1
        If B[J]=A[I]
        C[K]=B[J]
        K=K+1
        limit3=limit3+1
        Break
3. Return (limit3)

Steps for Difference Operation

Difference of Set A & Set B is stored in Set C

1. K=0, limit3=0
2. For I=0 to limit1-1
    For J=0 to limit2-1
        If A[I]=B[J]
            Break
        If J=limit2
            C[K]=A[I]
        K=K+1
        Limit3=limit3+1
3. Return (limit3)

Steps for Complement Operation

Complement of Set A & Set B is stored in Set C

1. K=0, limit3=0
2. For I=0 to limit1-1
    d=0

```
For J=0 to limit2-1
        If A[I]=B[J]
                d=1
        If d !=1
                C[I]=A[I]
```

3. Return (limit3)

Steps for Symmetric Difference operation

Symmetric Difference of Set A & Set B is stored in Set E

1. Find the difference of Set A & Set B in set C
2. Find the difference of Set B & Set A in set D
3. Find Union of Set C & Set D in Set E

Input:

Input is the data / number in the form of set.

Output:

Output is the data/number in the form of set.

Conclusion:

We successfully implemented different set operations using arrays and functions.

Time complexity:

1. Union of two sets : O(n**2)
2. Intersection of two sets. : O(n**2)
3. Difference of two sets : O(n**2)
4. Complement of two sets : O(n**2)
5. Symmetric difference of two sets : O(n**2)

Assignment No - 2

Title:

Write C/C++ program to store marks scored for first test of subject 'Data Structures and Algorithms' for N students. Compute

I. The average score of class

ii. Highest score and lowest score of class

iii. Marks scored by most of the students

iv. List of students who were absent for the test

Objective:

To understand linear data structure array and operations on array

Theory:

An array is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They can be used to store the collection of primitive data types such as int, float, double, char, etc of any particular type. An array in C/C++ can also store derived data types such as structures, pointers etc. Array elements are accessed by using an integer index. Array index starts with 0 and goes till the size of the array minus 1.

Advantages:-

1. Code Optimization: we can retrieve or sort the data efficiently.
2. Random access: We can get any data located at an index position.
3. Traversal through the array becomes easy using a single loop.

Disadvantages:-

1. Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.
2. Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.

Array declaration

1. By specifying size of the array : int arr[size];
2. By initializing the elements : int arr[] = {10,20,30,4};
3. By specifying size and initializing the elements : int arr[3] = {5,6,7};

Algorithm:

Steps for finding the average score of class

1. Sum=0;
2. For i=0 to n
   Sum+=array[i]
3. Avg=sum/total no of students

Steps for Highest score and lowest score of class

1. Highest =0; lowest=0;
2. For i=0 to n
   If array[i]>highest
   Highest=array[i]
   If array[i]<lowest and array[i] is not absent
   lowest =array[i]
3. Print Highest, lowest;

Steps for finding Marks scored by most of the students

1. marks=0 frq=0
2. for i=0 to n
   Cnt=0
   For j=i+1 to n
   If array[i]==array[j]
   Cnt++
   If cnt>frq
   marks=array[i],frq=Cnt

Input:

Input is string: array of marks.

Output:

Highest, lowest, and average of marks

Conclusion:

We successfully Stored marks of n students and applied various operations on it.

Time complexity:

1. Average score : O(n)
2. Highest and lowest score : O(n)
3. Marks scored by Maximum number of students : O(n**2)

Assignment No -3

Title:

Department library has N books. Write C/C++ program to store the cost of books in array in ascending order. Books are to be arranged in descending order of their cost. Write function for

a) Reverse the contents of array without using temporary array.

b) Copy costs of books those with cost less than 500 in new array

c) Delete the duplicate entries using temporary array

d) Delete duplicate entries without using temporary array

e) Count number of books with cost more than 500.

Objective:

To perform various array operations:

1. Reversing the array
2. Delete the duplicate
3. Copying the array elements

Theory:

A function is a set of statements that take inputs, do some specific computation, and produce output. It helps in reducing code redundancy , makes code modular and provides abstraction. Every function has a return type. If a function doesn't return any value, then void is used as a return type.

- There are two types of functions :

1. User defined functions :

   User Defined functions are user/customer-defined blocks of code specially customized to reduce the complexity of big programs.

2. Library functions :

   Library functions are also called "builtin Functions". These functions are a part of a compiler package that is already defined and consists of a special function with special and different meanings.

- Parameter Passing to Functions :

  The parameters passed to function are called actual parameters, while the parameters received by the function are called formal parameters.

  There are two most popular ways to pass parameters:

  1. **Pass by Value**: In this parameter passing method, values of actual parameters are copied to the function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in the actual parameters of the caller.
  2. **Pass by Reference**: Both actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in the actual parameters of the caller.

Algorithm:

Steps for reversing the array:

1. j=limit-1
2. For i=0 to i<j

```
    Swap (book[i], book[j])
    i++;
    j--;
```

Steps for copying the cost less than 500:

Cost of books less then 500 goes into a new array bookless[];

1.j=0;

2. For i=0 to limit-1

```
    If(book[i]<500)
        less[j++]=book[i]
    i++;
```

Steps for Delete the Duplicate entity from the array:

1. For i=0 to limit-1

2.If(book[i]=book[i+1])

```
    For j=i+1 to limit-2
        book[j]=book[j+1]
    limit--;
```

Else

i++;

Steps for counting the no of books cost greater than 500:

Count of books goes into a variable cnt

1.cnt =0

2. For i=0 to limit -1
3. If Book[i]>500

Cnt++;

Input:

Input is the cost of the books of the library in the form of a array.

Output:

Output the cost of the books after performing the various operations

Conclusion:

We successfully Stored cost of n books in the array and applied various operations on it.

Time complexity:

1. Reversing the array : O(n)
2. Deleting duplicate entries : O(n**2)
3. Counting number of books having price > 500 : O(n)

Assignment No -4

Title:

It is decided that weekly greetings are to be furnished to wish the students having their birthdays in that week. The consolidated sorted list with desired categorical information is to be provided to the authority. Write C++ program for array of structures to store students PRNs with date and month of birth. Let Array_A and Array_B be the two arrays for two SE Computer divisions. Arrays are sorted on date and month. Merge these two arrays into third array Array_SE_Comp_DOB resulting in sorted information about Date of Birth of SE Computer students.

Objective:

To merge the two structure array.

Theory:

1. Structures

A structure is a user-defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

The 'struct' keyword is used to create a structure. The general syntax to create a structure is as shown below:

Structures in C++ can contain two types of members:

Data Member: These members are normal C++ variables. We can create a structure with variables of differen data types in C++.
Member Functions: These members are normal C++ functions. Along with variables, we can also include functions inside a structure declaration.

A structure variable can either be declared with structure declaration or as a separate declaration like basic types. Structure members cannot be initialized with declaration. Structure members can be initialized with declaration in C++.

Structure members are accessed using dot (.) operator.

An array of structures finds its applications in grouping the records together and provides for fast accessing

Algorithm:

Steps for merging two arrays

Merged data of class A and class B is stored in class C

1. i=0 ; j=0;k=0;

a. while(i<limit1 && j<limit2)

If A[i].mm<b[j].mm

Copy A[i] into C[k]

k++;

i++;

else If B[j].mm<A[i].mm

Copy B[j] into C[k];

k++;

j++;

else If A[i].dd<=B[j].dd

Copy A[i] into C[k]

k++;

i++;

else

Copy B[j] into C[k];

k++;

j++;

b. while(i<limit1)

Copy A[i] into C[k]

k++;

i++;

c. while(j<limit2)

Copy B[j] into C[k];

k++;

j++;

Input:

Input is the date of birth and PRN no of the students for 2 divisions.

Output:

Merged array of date of birth and PRN no. of students.

Conclusion:

We successfully Stored cost of n books in the array and applied various operations on it.

Time complexity:

O(n)

Assignment No -5

Title:

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

a) Check whether given matrix is upper triangular or not

b) Compute summation of diagonal elements

c) Compute transpose of matrix

d) Add, subtract and multiply two matrices

e) Check whether the given matrix is a magic square or not.

Objective:

- This assignment will help the students to realize how the different operation on matrices like addition, multiplication, transposes.
- To understand the concept of 2D array.
- To understand the concept of double pointer or pointer of arrays or array of pointers.
- Memory allocation and deallocation.
- Analyze the above algorithms.

Theory :

Two – dimensional array is the simplest form of a multidimensional array.

We can see a two – dimensional array as an array of one-dimensional array for easier understanding.

The basic form of declaring a two-dimensional array of size x, y:

Syntax: data_type array_name[x][y];

Here, data_type is the type of data to be stored.

We can declare a two-dimensional integer array say 'x' of size 10,20 as: int x[10][20];

Elements in two-dimensional arrays are commonly referred to by x[i][j] where i is the row number and 'j' is the column number.

A two – dimensional array can be seen as a table with 'x' rows and 'y' columns where the row number ranges from 0 to (x-1) and the column number ranges from 0 to (y-1).

Initializing Two – Dimensional Arrays: There are various ways in which a Two-Dimensional array can be initialized.

First Method:

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

The above array has 3 rows and 4 columns. The elements in the braces from left to right are stored in the table also from left to right. The elements will be filled in the array in order, the first 4 elements from the left in the first row, the next 4 elements in the second row, and so on.

Second Method:

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

Third Method:

```
int x[3][4];
for(int i = 0; i < 3; i++){
   for(int j = 0; j < 4; j++){
      cin >> x[i][j];
   }
}
```

Fourth Method(Dynamic Allocation):

```
int** x = new int*[3];
for(int i = 0; i < 3; i++){
   x[i] = new int[4];
   for(int j = 0; j < 4; j++){
      cin >> x[i][j];
   }
}
```

Accessing Elements of Two-Dimensional Arrays:

Elements in Two-Dimensional arrays are accessed using the row indexes and column indexes. Example:int x[2][1];

The above example represents the element present in the third row and second column.

Algorithm:

Upper triangular matrix

A(m:n) is input

1. Flag = true

2.For i = 0 to m increment by 1

For j = 0 to n increment by 1
If i >=j
If A[i][j] != 0
Flag = false

3. If flag == false

Not upper triangular matrix

Else

Upper triangular matrix

Matrix Addition

A(m:n), B(m;n) are two input, C(p:q) output matrix.

1.For i = 0 to m increment by 1
For j = 0 to n increment by 1
C(i,j) = A(i,j) + B(i,,j)
2. Display C.

Matrix subtraction

A(m:n), B(m;n) are two input, C(p:q) output matrix.

1.For i = 0 to m increment by 1

For j = 0 to n increment by 1

C(i,j) = A(i,j) - B(i,,j)

2. Display C.

Matrix Multiplication

A(m:n), B(n:m) are two input, C(p:q) output matrix.

1.For i = 0 to m increment by 1

For j = 0 to m increment by 1

C(i,j) = 0

For k = 0 to n increment by 1

C(i,j) = C(i,j) + A(i,k)*B(k,j)

2. Display C.

Transpose of Matrix

A(m:n) is input, C(p:q) output matrix.

1.For i = 0 to m increment by 1

For j = 0 to m increment by 1

C(j,i) = A(i,j)

2. Display C.

Sum of diagonal elements

A(m:n) is input, sum is the output.

1. For i=0 to m increment by 1

Sum += A[i][j]

2. display sum

Magic Square

A(m:n) is input

1. flag = false
2. if sum of primary diagonal == sum of secondary diagonal
   if sum of every row == sum of primary diagonal
   if sum of every column == sum of primary diagonal
   flag = true
   else
   break
   else
   break
   else
   break
3. if flag = true
   it is magic square matrix
   else
   not a magic square matrix

Input: 2D array.

Output: Resultant 2D array.

Conclusion:

We successfully Stored data in form of matrix and applied various operations on it.

Time complexity:

Upper triangular matrix : $O(n^2)$

Matrix Addition : $O(n^2)$

Matrix Subtraction : $O(n^2)$

Matrix Multiplication : $O(n^3)$

Transpose of Matrix : $O(n^2)$

Sum of diagonal elements : O(n)

Magic Square : $O(n^2)$

Assignment No – 6

Title:

Write a C++ program to realize polynomial equation and perform operations. Write function

a) To input and output polynomials represented as $b_m x^m + b_{m-1} x^{m-1} + \ldots + b_0 x^0$ .Your

functions should overload the << and >> operators.

1. b) Evaluates a polynomial at given value of x
2. c) Add two polynomials
3. d) Multiplies two polynomials

Objective:

To understand concept of structure and array.

To understand how to pass structure elements to functions.

THEORY:

There are 2 types of polynomial representation :

1. creating an array of size equal to max exponent in polynomial + 1. We assign coefficient to the array at index whose value is equal to the exponent of that coefficient.
2. Creating a 2d array of size equal to number of terms in polynomial and storing the elements in this array as a pair of coefficient, exponent.

Algorithm:

Steps for Addition Operation

Addition of polynomial P1 & P2 is stored in P3.

1. Find the similar power terms in P1 and P2.
2. Assign power of that term from P1 to the power of P3 term.
3. Add the coefficients of P1 and P2 for that term.
4. Then copy the remaining terms with their coefficients and power from P1 to P3.
5. Then copy the remaining terms with their coefficients and power from P2 to P3.
6. Also keep track of total no. of terms in resultant polynomial P3.

Steps for Subtraction Operation

Subtraction of polynomial P1 & P2 is stored in P3.

1. Find the similar power terms in P1 and P2.

2. Assign power of that term from P1 to the power of P3 term.
3. Substract the coefficients of P1 and P2 for that term.
4. Then copy the remaining terms with their coefficients and power from P1 to P3.
5. Then copy the remaining terms with their coefficients and power from P2 to P3.

6. Also keep track of total no. of terms in resultant polynomial P3.

Steps for Multiplication Operation

Multiplication of polynomial P1 & P2 is stored in P3.

Multiply each term from P1 with each term from P2.

While doing this:

1. Add powers of the term from p1 and P2 and store it in power of P3.
2. Multiply coefficient of the terms from P1 and P2 and store it in the coefficient of P3.
3. Now, find the similar power terms in resultant polynomial P3 and do addition of their coefficients.
4. Also keep track of total no. of terms in resultant polynomial P3.

Steps for Evaluation of a given polynomial

Evaluation of a given polynomial is done after providing some value for the variable.

Ex. In Polynomial 3x2+4x+5, if we place x=2, it evaluates to 25.

1. Find out the power of the term for given value.
2. Multiply this with the coefficient of that term.
3. Store this result in some Res variable.
4. Repeat this until the no. of terms in the given polynomial finishes.
5. Finally print Res.

Input: Two arrays of structure elements which represents polynomial.

Output: Resultant array of structure.

Conclusion:

We successfully implemented polynomial equations and applied various operations on it.

Time complexity:

Polynomial Addition : O(n)

Polynomial subtraction : O(n)

Polynomial Multiplication : O(n^2)

Polynomial evaluation : O(n)

Assignment No - 7

Title:

Write C++ program for sparse matrix realization and operations on it- Transpose, Fast Transpose and addition of two matrices

Objective:

This assignment is designed for sparse matrix. Input is simple matrix and we convert it into sparse matrix to perform different operations such as addition, simple transpose, fast transpose.Time Complexity for simple transpose is O(nt). Time Complexity for fast transpose is O (n+t).

Theory

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix ?

Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.

Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements..

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with triples- (Row, Column, value).

2D array is used to represent a sparse matrix in which there are three rows named as

Row: Index of row, where non-zero element is located

Column: Index of column, where non-zero element is located

Value: Value of the non zero element located at index – (row,column)

Algorithm:

Algorithm Addition (A, B)

Read 2D array A1, B1 and convert it into sparse matrix.

1. (m1 ,n1 ,q) ← (A(0).r) , A(0).c, A(0).val)
2. (m2,n2 ,r) ← (B(0).r) , B(0).c, B(0).val) , K=0
3. if t <= 0 then return
4. for i = 1 to q increment by 1

    for j = 1 to r increment by 1

        if A(i,1) = B(j,1) && A(i,2) = B(j,2)

            C[k,1) = A(i,1)

            C[k,2) = A(i,2)

            C[k,1) = A(i,3) + B(j,3)

            K++

        else

            C[k,1) = A(i,1)

            C[k,2) = A(i,2)

            C[k,1) = A(i,3)

            K++

        C[k,1) = B(i,1)

        C[k,2) = B(i,2)

        C[k,1) = B(j,3)

        K++

Algorithm TRANSPOSE (A, B)

// A is a matrix represented in sparse form. B is set to be its transpose.

```
1.  (m ,n ,t)   ←  (A[0].r) , A[0].c, a[0].val )
2.  (B[0].r) , B[0].c, B[0].val )  ←  (n , m ,t)
3. if t <= 0 then return
4.        q = 1
5.   for col ← 1  to n do
6.    for p  ← 1  to n do
7.       if  A[p].c = col
8.           then (B[0].r) , B[0].c, B[0].val )
                        ← (A[0].c) , A[0].r, a[0].val )
9.        q ← q + 1
10.   end
11. end
12. End TRANSPOSE
```

Algorithm FAST –TRANSPOSE (A, B)

// S(1:n) , T(1 : n);

```
1.  (m ,n ,t)   ←  (A(0).r) , A(0).c, A(0).val )
2.  (B(0).r) , B(0).c, B(0).val )  ←  (n , m ,t)
3.   if t <= 0 then return
4. for i ← 1 to n do S(i) ← 0 end
5. for i ← 1 to n do
6.                  S(A(i).c)  ← S(A(i).c) + 1
7.  end
8. T(1)  ← 1;
9. for i ← 2 to n do
10.      T(i)    ← T(i-1) + S(i- 1)
11.  end
12.  for i ← 1 to t do
13.       j  ← A[i].c
```

14. (B(T(j).r , B(T(j).c, B(T(j).val) ←
(A(i).c) , A(i).r, A(i).val)
15. T(j) ← T(j) + 1
16. end
17. end FAST –TRANSPOSE.

Input:

Input to this assignment is simple matrix.

Output:

Manipulated data in the form of sparse matrix.

Conclusion:

We successfully carried out sparse matrix notations and applied various operations on it.

Time complexity:

Addition : O(n)

Simple transpose : $O(n^2)$

Fast transpose : O(n)

Assignment No – 8

Title:

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

a) Add and delete the members as well as president or even secretary.

b) Compute total number of members of club

c) Display members

d) Display list in reverse order using recursion

e) Two linked lists exists for two divisions. Concatenate two lists.

Objective:

This assignment will help to understand the dynamic memory allocation concepts as well as the linked representation of the data elements. How to overcome limitations of arrays.

Theory:

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers. They include a series of connected nodes. Here, each node stores the data and the address of the next node.

Advantages of Linked Lists over arrays: Dynamic Array, Ease of Insertion/Deletion.

Drawbacks of singly Linked Lists:

1.Random access is not allowed. We have to access elements sequentially starting from the first node(head node). So we cannot do a binary search with linked lists efficiently with its default implementation.

2.Extra memory space for a pointer is required with each element of the list.

3.Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

4.It takes a lot of time in traversing and changing the pointers.

5.Reverse traversing is not possible in singly linked lists.

6.It will be confusing when we work with pointers.

7.Direct access to an element is not possible in a linked list as in an array by index.

8.Searching an element is costly and requires O(n) time complexity.

9.Sorting of linked list is very complex and costly.

Types of Linked Lists:

1. Simple Linked List
2. Doubly Linked List
3. Circular Linked List
4. Doubly Circular Linked List
5. Header Linked List.

Algorithm:

Insertion in ordered list

// x is the data element of the node to be inserted in linked list

// list is the starting node of the linked list

// next refers to the next node of the linked list

// info refers to info field of the node

```
Getnode (new)
Info (new) =x
next (new) =null
if list = null
 list = new
else
        p=list
        if x < info (p)
        then next (new) = list
```

```
                    list = new
            else

                while (next (p) <> null) and (x >= info (next (p))) do
                    p= next (p)

                next (new)= next(p)
                next (p) = new
```

Deleting from a linked list

```
// current initially refers to first node of the linked list
// trail refers to one node previous to current
            current = list
            trail = null
            while ( current <> null) and (info (current) <> x) do
            begin
                trail = current
                current = next (current)
            end/* end of the loop to search the node */
            if current not null
            then if trail = null
                then
                    list = next (list) /* delete the first node */
                else
                    next (trail) = next (current)
                freenode (current)
            else print ("node with the given value doesn't exist in the list" )
```

Search a node in a given linked list

```
// current initially refers to the first node of the linked list
// next refers to the next node
            count=0
```

```
while (current <> null) and info (current) <> x) do
        current = next (current)
        count = count +1
        end // end of the loop

if (current = null)
        call print(" node with the given value doesn't exist in the list")
else
        display the position and the content of the node
```

Display the content of linked list

```
// list is the starting node of the linked list
        if (list = null)
                call print(" linked list is empty ")
        else
        while (current <> null ) do
                call print( info(current))
                current = next (current)
                end //end of loop
                end //end else
```

Print the content of the linked list in reverse order

```
// list is the starting node of the linked list
// current is assigned to the first node of linked list
Reverseprint(list)
begin
        current= list
        if (current <> null)
        Reverseprint (next(current) )
        call print ( info (current ))
end if // end of if
```

end of Reverseprint

Input:

Input is the data/element that is to be added to the linked list as a new node.

Output:

Output is the data/elements present in the linked list.

Conclusion:

We successfully stored information using singly linked list and performed some operations on it.

Time complexity:

Add and delete the members : O(n)

Display members : O(n)

reverse order using recursion : 0(n)

Assignment No – 9

Title:

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

a) The list of available seats is to be displayed

b) The seats are to be booked

c) The booking can be cancelled.

Objective:

To understand concept of Doubly Linked List & different operations performed on it.

To understand difference between singly & doubly linked list.

Theory:

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

Advantages of DLL over the singly linked list:

1. A DLL can be traversed in both forward and backward directions.
2. The delete operation in DLL is more efficient if a pointer to the node to be deleted is given.
3. We can quickly insert a new node before a given node.
4. In a singly linked list, to delete a node, a pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using the previous pointer.

Disadvantages of DLL over the singly linked list:

1. Every node of DLL Requires extra space for a previous pointer. It is possible to implement DLL with a single pointer though (See this and this).
2. All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with the next pointers. For example in the following functions for insertions at different positions, we need 1 or 2 extra steps to set the previous pointer.

Algorithm:

Insertion in a doubly linked list

```
Getnode  (new)
Info (new ) = x
if list = null
        then list = new
        next (list) = null
        back (list)= null

else
        travel = list
        while (next (travel) <> null )
                travel = next (travel)
        back (new ) = travel
        next (new) = next (travel)
        next (travel) = new
```

Deleting node from a doubly linked list

```
Delete (list, x)
// list is the starting node of the doubly linked list
// x is the data element that is to be deleted
current = list
        If (current = null)
                call print (" list empty ")
                Return
        while ( current <> null) and (info (current) <> x) do
                current = next (current)

        if (current = null)
                call print ("node with the given value doesn't exist in the list")
        else
                next (back (p) ) =next (p)
                back (next (p) ) =back (p)
                return
```

Search a given node in the doubly linked list

```
// current initially refers to the first node of the linked list
                        // next refers to the next node
count=0
while (current <> null) and info (current) <> x) do
        current = next (current)
        count = count +1

if (current = null)
        call print(" node with the given value doesn't exist in the list")
else
        display the position and the content of the node
```

Display the content of doubly linked list

```
// list is the starting node of the linked list
```

```
if (list = null)
        call print(" linked list is empty ")
else
        while (current <> null ) do
                call print( info(current))
                current = next (current)
```

Input:

Some random booking data

Output:

List of available and booked seats.

Conclusion:

We successfully stored information using doubly linked list and performed some operations on it.

Time complexity:

Insertion / deletion : O(n)

Display members : O(n)

searching : O(n)

Assignment No – 10

Title:

Design a linked allocation system to represent and manipulate univariate polynomials with integer coefficients (use circular linked lists with head nodes). Each term of the polynomial will be represented as a node Thus. a node in this system will have three data members

To erase polynomials efficiently, we need to use an available-space list and associated functions. The external (i.e.. for input or output) representation of a univariate polynomial will be assumed to be a sequence of integers of the form: $n, c_1, e_1, c_2, e_2, c_3, e_3 \ldots, c_n e_n$ where e_i represents an exponent and c_i a coefficient; n gives the number of terms in the polynomial. The exponents are in decreasing order — i.e., $e_1 > e_2 > \ldots > e_n$.

Write and test the following functions:

1. istream&**operator** >>(istream& is, Polynomial& x): Read in an input polynomial and convert it to its circular list representation using a head node.
2. ostream&**operator**<< (ostream&os, Polynomial& x): Convert x from its linked list representation to its external representation and output it.
3. Polynomial:: Polynomial(**const** Polynomial& a) [Copy Constructor]: Initialize the polynomial *this to the polynomial a.
4. **const** Polynomial& Polynomial :: **operator=**(**const** Polynomial& a) [Assignment Operator]: Assign polynomial a to***this**.
5. Polynomial:: Polynomial ( ) [Destructor]: Return all nodes of the polynomial ***this** to the available-space list.
6. Polynomial **operator+** (**const** Polynomial& a, **const** Polynomial& b) [Addition]: Create and return the polynomial a + b. a and b are to be left unaltered.
7. ***Polynomial operator* (constPolynomial& a, constPolynomial& b) [Multiplication]:*** *Create and return the polynomial a * b. a and b are to be left unaltered.*
8. **float*Polynomial ::Evaluate(float x):*** Evaluate the polynomial *this at *x* and return the result.

Objective:

This assignment will help to understand the dynamic memory allocation concepts as well as it gives an idea to represent the circular queue using linked representation of the data elements. It also gives an idea how to overcome limitations of arrays.

Theory:

The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL a the end. Circular linked lists are similar to single Linked Lists with the exception of connecting the last node to the first node.

There are generally two types of circular linked lists:

Circular singly linked list: In a circular Singly linked list, the last node of the list contains a pointer to th first node of the list. We traverse the circular singly linked list until we reach the same node where we started. The circular singly linked list has no beginning or end. No null value is present in the next part of any of the nodes.

Circular Doubly linked list: Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous and next pointer and the last node points to the first node by the next pointer and also the first node points to the las node by the previous pointer.

Advantages of Circular Linked Lists:

1. Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.
2. Useful for implementation of a queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use a circular linked list. We can maintain a pointer to the last inserte node and the front can always be obtained as next of last.
3. Circular lists are useful in applications to repeatedly go around the list..
4. Circular Doubly Linked Lists are used for the implementation of advanced data structures like the Fibonacci Heap

Disadvantages of circular linked list:

1. Compared to singly linked lists, circular lists are more complex.
2. Reversing a circular list is more complicated than singly or doubly reversing a circular list.
3. It is possible for the code to go into an infinite loop if it is not handled carefully.
4. It is harder to find the end of the list and control the loop.

Algorithm:

Pseudo Code for Insert:

```
Void insert(node **R,int x)
  {
   node *p;
   p = (node *)malloc(sizeof(node));
   p->data = x;
   if(empty(*R))
         {
          p->next = p;
          *R = p;
         }
   else
    {
          p->next = (*R)->next;
          (*R)->next = p;
          (*R) = p;
    }

  }
```

Pseudo Code for Delete:

```
 int delete (node **R)
 {
   int x;
   node *p;
   p=(*R)->next;
   x=p->data;
   if(p->next == p)
          {
             *R = NULL;
             free(p);
             return(x);
          }
  (*R)->next = p->next;
```

```
free(p);
return(x);}
```

Input:

Polynomials

Output:

Resultant polynomial after addition and multiplication

Conclusion:

We successfully stored information using circular linked list and performed some operations on it.

Time complexity:

Insertion / deletion : O(n)

Assignment No – 11

Title:

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

Objective:

To know concept of STACK. This assignment will help the students to realize how the different operation on stack like push, pop, display can be implemented.

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).

In order to make manipulations in a stack, there are certain operations provided to us.

1. push() to insert an element into the stack
2. pop() to remove an element from the stack
3. top() Returns the top element of the stack.
4. isEmpty() returns true is stack is empty else false.
5. size() returns the size of stack.

Types of Stacks:

Register Stack: This type of stack is also a memory element present in the memory unit and can handle a small amount of data only. The height of the register stack is always limited as the size of the register stack is very small compared to the memory.

Memory Stack: This type of stack can handle a large amount of memory data. The height of the memory stack is flexible as it occupies a large amount of memory data.

stacks can be implemented by using arrays or linked list.

Algorithm:

```
Push(char item ,int S [])
// TOP as pointer which keep track of the top element in the stack.
// This algorithm gives push item in stack A
// Pre – S is stack
// Post – item is pushed in S stack
// Return –nothing
begin
        if top>=n then
                call stack_full()
                return
        end //if
        top=top+1;
        stack[top]=item;
end push;
```

```
Pop(char  S [])
// TOP as pointer which keep track of the top element in the stack.
//  This algorithm gives  push item in stack A
// Pre – S is stack
// Post – item is poped  in S stack
```

```
// Return –item which is top element of a stack.
begin
        if top<= 0 then
                call stack_empty()
                return
        end //if
        item=stack[top];
        top=top-1;
        return item;
end pop;
```

Stack_empty()

```
// TOP as pointer which keep track of the top element in the stack.
If (top == zero)
        Call print("stack is empty");
End //if
End stack_empty;
```

Stack_full()

```
// TOP as pointer which keep track of the top element in the stack.
If (top == n)
        Call print("stack is full");
End //if
End Stack_full;
```

```
void Stack_display (char S[])

Begin
        counter =top
        While counter >0
```

```
            Call print(s[counter])
            counter—
        end //while
    end Stack_display
```

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

Input:

Input is arithmetic expression

Output:

The expression has balanced parentheses or not.

Conclusion:

We successfully checked if the given order of parenthesis is balanced or not using stacks..

Time complexity:

push() :	**O(1)**
pop() :	**O(1)**
Stack_empty() / full() :	**O(1)**
size() :	**O(1)**

Assignment No – 12

Title:

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

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

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

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

Objective:

To understand use of stack as an abstract data type (A.D.T.)

Theory:

Infix expression: The expression of the form an operator b (a + b). When an operator is in-between every pair of operands.

Postfix expression: The expression of the form a b operator (ab+). When an operator is followed by every pair of operands.

The compiler scans the expression either from left to right or from right to left.

Consider the expression: a + b * c + d

The compiler first scans the expression to evaluate the expression b * c, then again scans the expression to add a to it. The result is then added to d after another scan. The repeated scanning makes it very inefficient and Infix expressions are easily readable and solvable by humans whereas the computer cannot differentiate the operators and parenthesis easily so, it is better to convert the expression to postfix(or prefix) form before evaluation.

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

Algorithm:

Let S be a stack to contain operators.

1. Read the infix expression in infix
2. initialize S.
3. n = strlen(infix)
4. n = n-1, k = n decrement k by 1 till k th element will become '\0'
5. if (infix(n)) is an operand

   p(k) = infix(n)

   k--, n—

goto step 8

6. if(infix(n)) is operator then

   if S is empty then push operator on stack

   goto step 8

   if S is non empty check the priority and take action accordingly.
7. If (n>=0) goto step 6
8. pop all the elements of S and place it in P
9. print P
10. stop.

Input:

Input is string which is an expression in infix form.

Output:

Output is string which is an expression in prefix form.

Conclusion:

We successfully converted the given infix expression to postfix using stacks..

Time complexity:

O(n)

Assignment No – 13

Title:

Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.

Objective:

This assignment will help the student to realize the implementation difference between stack and queue. Also this will give an idea for the implementation of queue.

Theory:

A queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.

We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.

Characteristics of Queue:

1. Queue can handle multiple data.
2. We can access both ends.
3. They are fast and flexible.

Queue Representation:

1. Array Representation:

Like stacks, Queues can also be represented in an array: In this representation, the Queue is implemented using the array. Variables used in this case are

Queue: the name of the array storing queue elements.

Front: the index where the first element is stored in the array representing the queue.

Rear: the index where the last element is stored in an array representing the queue.

2. Linked List Representation:

A queue can also be represented in Linked-lists, Pointers, and Structures.

Types of Queue:

There are different types of queue:

Input Restricted Queue

This is a simple queue. In this type of queue, the input can be taken from only one end but deletion can be done from any of the ends.

Output Restricted Queue

This is also a simple queue. In this type of queue, the input can be taken from both ends but deletion can be done from only one end.

Circular Queue

This is a special type of queue where the last position is connected back to the first position. Here also the operations are performed in FIFO order. To know more refer this.

Double-Ended Queue (Dequeue)

In a double-ended queue the insertion and deletion operations, both can be performed from both ends. To know more refer this.

Priority Queue

A priority queue is a special queue where the elements are accessed based on the priority assigned to them

Algorithm:

```
Insertq (item, Q, n, front, rear)
    if rear >= n then
            call QUEUE_FULL
            return
     rear <- rear+1
    Q(rear) = item

Deleteq (item, Q, n, front, rear)
     if front <= 0 then
            call QUEUE_EMPTY
            return
    item = Q(front)
     front = (front+1)
```

```
Displayq(Q, n, front, rear)
        count = front
        while count <= rear
                call print (Q(count))
                count++
```

Input:

Input is the data/element that is to be added to the queue.

Output:

Output is the data/element removed from the queue.

Conclusion:

We successfully implemented the program using queue.

Time complexity:

Insertq() :	**O(1)**
Deleteq() :	**O(1)**

Assignment No – 14

Title:

Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using circular queue using array

Objective:

This assignment will help to realize the implementation concept of circular queue and its operations like insertion, deletion etc.

Theory:

A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'.

Operations on Circular Queue:

1. **Front**: Get the front item from queue.
2. **Rear**: Get the last item from queue.
3. **enQueue(value) :** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.
    a. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
    b. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.
4. **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.
    a. Check whether queue is Empty means check (front==-1).
    b. If it is empty then display Queue is empty. If queue is not empty then step 3
    c. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

Algorithm:

```
INSERTQ (item, Q, n, front, rear)
        if front=(rear +1) mod n then
                call QUEUE_FULL
                return

         rear <- (rear+1)mod n
        Q(rear) = item

DELETEQ (item, Q, n, front, rear)
        if front=rear then
                 call QUEUE_EMPTY
                 return
        front = (front+1)mod n
        item = Q(front)
```

Pseudocode for Circular Queue with count

```
INSERTQ(item, Q, n, front, rear, count)
        if count >=n
                call print("Q is full ")
                return

        rear = (rear+1)mod n
        Q(rear) = x
        count ++

DELETEQ(item, Q, n, front, rear, count)
        if count <=0
                call print("Q is empty")
                return
                item = Q(front)
        front= (front+1) mod n
        count—
```

```
DISPLAYQ(Q, n, front, rear, count)
        i=1, j= front
        if count <=0
                call print(" Q is empty ")
                return
        while i<= count
                print( Q(j) )
                j= (j+1) mod n
                i =i+1
```

Input:

Input is the data/element that is to be added to the queue.

Output:

Output is the data/element removed from the queue.

Conclusion:

We successfully implemented the program using circular queue.

Time complexity:

Insertq() :	**O(1)**
Deleteq() :	**O(1)**

Assignment No – 15

Title:

a) Write C++ 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 C++ 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 different searching methods. Also to understand concept of recursion.

Theory:

Searching Algorithms are designed to check for an element or retrieve an element from any data structur where it is stored. Based on the type of search operation, these algorithms are generally classified into tw categories:

Sequential Search: In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search, sentinel search

Interval Search: These algorithms are specifically designed for searching in sorted data-structures. Thes type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: Binary Search, Fibonacci search.

1. **Linear Search**

   It is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. It is the easiest searching algorithm

2. **Sentinel search**

   in this search, the last element of the array is replaced with the element to be searched and then the linear search is performed on the array without checking whether the current index is inside the index range of the array or not because the element to be searched will definitely be found inside the array even if it was not present in the original array since the last element got replaced with it. So, the index to be checked will never be out of bounds of the array. The number of comparisons in the worst case here will be (N + 2).

3. **Binary search**

   Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

   The basic steps to perform Binary Search are:

a. Begin with the mid element of the whole array as a search key.

b. If the value of the search key is equal to the item then return an index of the search key.

c. Or 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.

d. Otherwise, narrow it to the upper half.

e. Repeatedly check from the second point until the value is found or the interval is empty.

4. Fibonacci search

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

Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1

Algorithm:

1. Sequential Search

X is the number to be searched, A[N] is the Array of elements N, I is the index of array.

1. [Initialize Search]
   A. I=1
   B. A[N+1]=X

2. [Search Vector]
   A. Repeat while A[I]!=X
      I=I+1

3. [Successful Search?]
   A. If I=N+1
   B. Then Print 'Unsuccessful Search'
      a. Return(0)
   C. Else Print 'Successful Search'
      a. Return(I)

2. Binary Search

Given A[N] is the Array of elements N, X is the number to be searched, lower contains lower bound index and upper contains upper bound index of considered list. Variable mid contains middle index of list in consideration.

1. [Initialize]
    A. Lower=1
    B. Upper=N
2. [Perform Search]
    Repeat thru step 4 while lower<=Upper
3. [Obtain index of Midpoint of Interval]
    Mid= (lower+upper)/2
4. [Compare]
    A. If X< A[mid]
    B. Then set upper to mid-1
    C. Else If X>A[mid]
        i. Then set lower to mid+1
        ii. Else Print 'Successful Search'
        iii. Return(Mid)
5. [Unsuccessful Search]
    A. Print 'Unsuccessful Search'
    B. Return(0)

3. Sentinel search

```
/* Returns the smallest i such that array[i] == key,     or -1 if key is not in array[].
  array[] must be an modifiable array of n ints with room for a (n + 1)th element. */
        int seq_sentinel_search (int array[], int n, int key)
        {
        int *p;
          array[n] = key;
          for (p = array; *p != key; p++)

          /* Nothing to do. */;
          return p - array < n ? p - array : -1;
          }
```

Fibonacci search

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

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

Input

Data in the form an array.and number to be searched

Output :

Position if number found

Conclusion:

We successfully implemented various searching algorithms

Time complexity:

Linear search : O(n)
Sentinel search: O(n)
Binary search: O(log(n))
Fibonacci search: O(log(n))

Assignment No – 16

Title:

Write C++ program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using

a) Selection Sort

b) Bubble sort and display top five scores.

c) Insertion sort

b) Shell Sort and display top five scores

Objective:

This assignment is designed to implement different sorting methods. Analyze the given algorithms.

Theory:

A Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

1. **Selection sort**

   The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.

   The algorithm maintains two subarrays in a given array.

   a. The subarray which already sorted.

   b. The remaining subarray was unsorted.

   In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

2. **Bubble sort**
   Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

3. **Insertion sort**
   Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

4. **Shell sort**
   Shell sort is mainly a variation of Insertion Sort. In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved. The idea of ShellSort is to allow the exchange of far items. In Shell sort, we make the array h-sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element are sorted.

Algorithm:

```
SELECTION-SORT(A)
  for j ← 1 to n-1
        smallest ← j
         for i ← j + 1 to n
               if A[ i ] < A[ smallest ]
                     smallest ← i
          Exchange A[ j ] ↔ A[ smallest ]


BUBBLESORT( A )
   n = length(A)
   repeat
     swapped = false
     for i = 1 to n-1 inclusive do
       /* if this pair is out of order */
       if A[i-1] > A[i] then
         /* swap them and remember something changed */
         swap( A[i-1], A[i] )
         swapped = true
  until not swapped


INSERTION-SORT (A)
for j <- 2 to length[A]
      do key <- A[j]
       Insert A[j] into the sorted sequence A[1 . . j - 1].
       i <- j - 1
       while i > 0 and A[i] > key
          do A[i + 1] <- A[i]
            i <-  i - 1
       A[i + 1] <- key
```

SHELL_SORT(NUM,N,KEY)

Assign, span = int(n/2)

while span > 0 do:

a) for i from span to n - 1, Repeat step b,c,e

b) assign num[i] to key and i to j

c) while j = span and num[j - span] > key, Repeat step d

d) swap num[j] and num[j - span]

e) Assign, span = int(span / 2.2)

Use Insertion Sort to sort remaining array of data

Input:

Input in the form of unsorted list(array).

Output:

Output in the form of sorted list.

Conclusion:

We successfully implemented various sorting algorithms

Time complexity:

1. **Selection sort : $O(n^2)$**

2. **Bubble sort**
   Worst and Average Case Time Complexity: O(N2). The worst case occurs when an array is reverse sorted.
   Best Case Time Complexity: O(N). The best case occurs when an array is already sorte

3. **Insertion sort: $O(n^2)$**

4. **Shell sort**
   The **worst-case complexity** for shell sort is $O(n^2)$
   Best Case Complexity
   when the given array list is already sorted the total count of comparisons of each interval is equal to the size of the given array.So best case complexity is $\Omega(n \log(n))$

   THE **Average Case Complexity**: $O(n*\log n) \sim O(n^{1.25})$

Assignment No – 17

Title:

A) Write C++ 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.

B) Write C++ program to store Xth percentage of students in array. Write function for sorting array of floating point numbers in ascending order using radix sort and display top five scores.

Objective:

This assignment is designed to implement different sorting methods. Analyze the given algorithms.

Theory

1. **Quick sort**
   QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

   a. Always pick the first element as a pivot.
   b. Always pick the last element as a pivot (implemented below)
   c. Pick a random element as a pivot.
   d. Pick median as the pivot.

   The key process in quickSort is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.
2. **Radix sort**
   The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.
   Some key points about radix sort are given here

   a. It makes assumptions about the data like the data must be between a range of elements.
   b. Input array must have the elements with the same radix and width.
   c. Radix sort works on sorting based on an individual digit or letter position.
   d. We must start sorting from the rightmost position and use a stable algorithm at each position.
   e. Radix sort is not an in-place algorithm as it uses a temporary count array.

Algorithm:

1. Quick Sort

```
algorithm quicksort(A, low, high)
  if lo < hi then
      p := partition(A, low, high)
      quicksort(A, low, p)
          quicksort(A, p + 1, high)

algorithm partition(A, low, high)
   pivot := A[lo]
   i := low – 1
   j := high + 1
   loop forever
      do
         i := i + 1
      while A[i] < pivot
      do
         j := j – 1
      while A[j] > pivot
      if i >= j then
         return j

      swap A[i] with A[j]
```

2. Radix Sort

```
for j ← 0 to m-1 do                 // initialize m buckets
          initialize queue b[j]
for i ← 0 to n-1 do                          // place in buckets
          b[S[i].getKey()].enqueue( S[i] );
i ← 0
for j ← 0 to m-1 do                 // place elements in
          while not b[j].isEmpty() do       // buckets back in S
                    S[i] ← b[j].dequeue()
                    i ← i + 1
```

Input:

Input in the form of unsorted list (array).

Output:

Output in the form of sorted list.

Conclusion:

We successfully implemented quick sort and radix sort algorithms

Time complexity:

1. **Quicksort :**

   Best Case Complexity - In Quicksort, the best-case occurs when the pivot element is the middle element or near to the middle element. The best-case time complexity of quicksor is **O(n*log(n)).**

   Average Case Complexity - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity o quicksort is **O(n*log(n)).**

   Worst Case Complexity - In quick sort, worst case occurs when the pivot element is either greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is **$O(n^2)$.**

2. **Radix sort**

   Worst-Case Time Complexity : $O(n^2)$.

   In radix sort, the worst case is when all elements have the same number of digits except one, which has a significantly large number of digits. If the number of digits in the largest element equals n, the runtime is **$O(n^2)$.**

   Best Case Time Complexity : O(a*n).

   When all elements have the same number of digits, the best-case scenario occurs. **O(a(n+b)**) is the best-case time complexity. If b equals O(n), the time complexity is **O(a*n).**

   Average Case Time Complexity

   You considered the distribution of the number of digits in the average case. There are 'p' passes, and each digit can have up to 'd' different values. Because radix sort is independent of the input sequence, we can keep n constant. Radix sort has an average case time complexity of O(p*(n+d)).