LAB FILE

DATA STRUCTURES ALGORITHMS & APPLICATIONS CT-159

SYED AHMED ALI

DT - 301

CSIT (DATA SCIENCE)

LAB 01 - ARRAYS AND DYNAMIC MEMORY ALLOCATION

Question1.cpp

```cpp
//<----Lab 01 - Arrays and Dynamic Memory Allocation---->

//Q1. Write a C++ code to copy data of a 2D array in a 1D array using Column Major Order.

#include<bits/stdc++.h>
using namespace std;
int main(){
int m,n;
cout<<"Enter Rows of 2d Array: ";
cin >> m;
cout<<"Enter Columns of 2d Array: ";
cin >> n;
int arr1d[m*n],arr2d[m][n];
for(int i=0;i<m;i++){
for(int j=0;j<n;j++){
cout<<"Enter ARRAY2D ["<<i+1<<"]["<<j+1<<"] :"<<endl;
cin>>arr2d[i][j];
}
}
cout<<"Copying to 1darray in column major order"<<endl;
int count=0;
for(int i=0;i<n;i++){
for(int j=0;j<m;j++){
arr1d[count++]=arr2d[j][i];
}
}

cout<<"printing 1d array"<<endl;
for(int i=0;i<m*n;i++){
cout<<arr1d[i]<<endl;
}
}
```

Question2.cpp

```cpp
//<----Lab 01 - Arrays and Dynamic Memory Allocation---->

// Q2. Write a program to calculate the CGPA of students of all subjects of a single semester.
// Assume all the courses have the same credit hour (let’s assume 3 credit hours).

#include<bits/stdc++.h>
using namespace std;

//This assumes CGPAs are already given
int main(){
float students[5][5]={{3.66,3.33,4.0,3.0,2.66},{3.33,3.0,3.66,3.0,0},{4.0,3.66,2.66,0,0},
{2.66,2.33,4.0,0,0},{3.33,3.66,4.0,3.0,3.33}};
float count=0,total=0;
for(int i=0;i<5;i++){
for(int j=0;j<5;j++){
total+=students[i][j];
(students[i][j] != 0 )?count=count+1:count=count;
}
        cout<<"CGPA (ignoring -- subjects) of student #"<<i+1<<" "<<total/count<<endl;
count=0;total=0;
}
for(int i=0;i<5;i++){
for(int j=0;j<5;j++){
total+=students[i][j];
}
cout<<"CGPA of student #"<<i+1<<" "<<total/5<<endl;
count=0;total=0;
}
}

/* This one takes input of all GPAs then calculates for those
#include<iostream>
using namespace std;

int main(){
float cgpa;
int NoOfSub;
int CreditHours=3;
float gradePoints=0;
float grades;
cout<<"enter total number of subjects"<<endl;
cin>>NoOfSub;

for(int i=1;i<=NoOfSub;i++){
cout<<"enter the grades of courses"<<i<<" "<<endl;
cin>>grades;
gradePoints+=grades*3;
CreditHours=3*NoOfSub;
}

cgpa=gradePoints/CreditHours;
cout<<"cgpa of sem is"<<cgpa<<endl;
return 0;
}
*/

```

Question3.cpp

```cpp
//<----Lab 01 - Arrays and Dynamic Memory Allocation---->

//Q3. Suppose you are planning a picnic. You want to store the names of the students of your
// class who have submitted their contribution money. Write a program that uses jagged
// array to store the students' names and then perform a search for your name.

#include<iostream>
#include<string>
using namespace std;

int main(){
char** names;
string name;
int n,loc;
cout<<"Enter number of students: ";
cin>>n;
names = new char*[n];
for(int i=0;i<n;i++){
cout<<"Enter name of student #"<<i+1<<" : ";
fflush(stdin);
getline(cin, name);
names[i]=new char[name.length()+1];
for(int j=0;j<name.length();j++){
names[i][j]=name[j];
}
names[i][name.length()]='\0';
}
cout<<endl<<endl;
for(int i=0;i<n;i++){
int j=0;
while(names[i][j]!='\0'){
cout<<names[i][j];
j++;
}
cout<<endl;
}
bool found=false;
cout<<endl<<endl;
cout<<"Enter name to search :";
getline(cin, name);
for(int i=0;i<n;i++){
int j=0;
while(names[i][j]==name[j] && names[i][j]!='\0'){
j++;
}
if(names[i][j]=='\0'){
found=true;
loc=i;
}
}
if(found==true){
cout<<"The name was found at index "<<loc<<endl;
}
    else{
        cout<<"Name not found"<<endl;
    }
}
```

Question4.cpp

```cpp
//<----Lab 01 - Arrays and Dynamic Memory Allocation---->

//Q4. Using the abstract data Type of a Matrix given below, write a program that
// 1. Input a 4*3 matrix from user in 2D array
// 2. Map this array in 1D array using Row major order
// 3. Input second matrix of 3*4 in 2D array
// 4. Map this array in 1D array using Row major order.
// 5. Now perform matrix multiplication on these 1D arrays
// 6. Save the result back in a 2D array.

// Implement this question for any number of rows and columns using class “matrix”.

#include<iostream>
using namespace std;
class matrix{
int **p;
int r;
int c;
int *rowmajor;
int *multiply1D;
public:

matrix(int row, int col);
void disp2D(){
 cout<<"2D matrix element\t";
 for(int i=0;i<r;i++){
 for(int j=0;j<c;j++){
 cout<<p[i][j]<<" ";
  }
 cout<<endl;
   }
}
void dispRowMajor(){
 cout<<"row major order matrix"<<endl;
 for(int i=0;i<r*c;i++){
 cout<<rowmajor[i]<<" ";
 }
 cout<<endl;
}
void Multiply_rowMajor(matrix & x){
 if(c!=x.r){
 cout<<"multiplication not possible";
 return;
 }
 for(int i=0;i<r;i++){
 for(int j=0;j<c;j++){
 multiply1D[i*x.c+j]=0;
 for(int k=0;k<c;k++){
 multiply1D[i*x.c+j]+=rowmajor[i*c+k]*x.rowmajor[k*x.c+j];
   }
  }
 }
}
```

```
void rowMajor_2D(){
 int C = sizeof(multiply1D)/r;
         cout<<"result of 2D in row major"<<endl;
 for(int i=0;i<r;i++){
 for(int j=0;j<C;j++){
 cout<<multiply1D[i*c+j]<<" ";
  }
 cout<<endl;
 }
}
~matrix(){
 for(int i=0;i<r;i++){
 delete[] p[i];
 }
 delete[] p;
 delete[] rowmajor;
 delete[] multiply1D;
}
};

int main()
{
matrix a(4,3);
matrix b(3,4);
a.disp2D();
a.dispRowMajor();
b.disp2D();
b.dispRowMajor();
a.Multiply_rowMajor(b);
a.rowMajor_2D();
}

matrix::matrix(int row,int col){

 r=row;
 c=col;
 p = new int*[r];
 rowmajor = new int[r * c];
 multiply1D = new int[r * c];

 for (int i = 0; i < r; i++) {
 p[i] = new int[c];
 for (int j = 0; j < c; j++) {
 p[i][j] = (i + j);
 rowmajor[i * c + j] = p[i][j];
}
   }
}




//First different approach
//#include<iostream>
//using namespace std;
//
//class matrix{
//int **p;
```

```
//int r;
//int c;
//int *rowmajor;
//int *multiply1D;
// public:
//matrix(int row, int col);
//// Constructor
//void disp2D(){
//for(int i=0;i<r;i++){
//for(int j=0;j<c;j++){
// cout<<p[i][j]<<"\t";
//}
// cout<<endl;
//}
//}
//// displays the elements of **p
//void dispRowMajor(){
//int arr[r*c],x=0;
//for(int i=0;i<r;i++){
//for(int j=0;j<c;j++){
// arr[x]=p[i][j];
// x++;
//}
//}
//for(int i=0;i<r*c;i++){
// cout<<arr[i]<<endl;
//}
//}
//// converts 2D into 1D using row major
////and displays the elements Row Major Order Matrix
//void Multiply_rowMajor(matrix & x){
//int k=0;
//int A[r * c],B[x.r * x.c];
//for(int i=0;i<r;i++){
//for(int j=0;j<c;j++){
// A[k]=p[i][j];
// k++;
//}
//}
// k=0;
//for(int i=0;i<x.r;i++){
//for(int j=0;j<x.c;j++){
// B[k]=x.p[i][j];
// k++;
//}
//}
// int res[r * x.c];
//
// int i, j;
//
// for (i = 0; i < r; i++) {
// for (j = 0; j < x.c; j++) {
// res[j + i * x.c] = 0;
//
// for (k = 0; k < c; k++) {
// res[j + i * x.c] += A[k + i * c] * B[j + k * x.c];
```

```
// }
// }
// }
// cout<<endl<<endl;
// for (i = 0; i < r * x.c; i++)
// cout<<res[i]<<endl;
//}
//void rowMajor_2D(){
// return;
//}
//// Maps the elements stored in row major order to
//// the 2D array and print the results
// ~matrix(){
//delete p;
//}
//// Destructor
//};
//
//int main()
//{
//matrix a(4,3);
//matrix b(3,4);
// a.disp2D();
// a.dispRowMajor();
// b.disp2D();
// b.dispRowMajor();
// a.Multiply_rowMajor(b);
// a.rowMajor_2D();
//}
//
//matrix::matrix(int row,int col)
//{
// r=row;
// c=col;
//p = new int*[r];
//for(int i=0;i<r;i++)
//{
//p[i]=new int[c];
//for(int j=0;j<c;j++)
// p[i][j]=(i+j);
//}
//// CODE FOR STORING DATA FROM
//// **P TO *rowmajor ROW MAJOR
//}
```

Question5.cpp

```cpp
//<----Lab 01 - Arrays and Dynamic Memory Allocation---->

// Q5. Write a program that creates a 2D array of 5x5 values of type boolean. Suppose indices
// represent cities and that the value at row i, column j of a 2D array is true just in case i and
// j are direct neighbors and false otherwise. Use initializer list to instantiate and initialize
// your array to represent the following configuration: (* means “neighbors”)

// Write a method to check whether two cities have a common neighbor. For example, in the
// example above, 0 and 2, 3 and 4 are neighbors with 1 (so they have a common neighbor),
// whereas 0 and 1 have no common neighbors.

#include<iostream>
using namespace std;

class array2d{
const bool city[5][5]={{false,true,false,false,false},{true,false,true,false,false},{false,true,false,false,true},{false,true,true,false,false},{false,true,true,true,false}};
int row;
int col;
bool** arr;
public:
array2d(int r=5,int c=5):row(r),col(c){
arr = new bool*[5];
for(int i=0;i<5;i++){
arr[i]=new bool[5];
}
for(int i=0;i<5;i++){
for(int j=0;j<5;j++){
arr[i][j]=city[i][j];
}
}

}
void display(){
cout<<endl;
for(int i=0;i<row;i++){
cout<<"\t";
for(int j=0;j<col;j++){
cout<<arr[i][j]<<"\t";
}
cout<<endl;
}
cout<<endl;
}
void searchneighbours(int city1,int city2){
for(int i=0;i<5;i++){
if(arr[city1][i]==arr[city2][i] && arr[city1][i]== true && arr[city2][i]==true){
                cout<<"City "<<city1<<" and city "<<city2<<" share neighbour at index "<<i<<". "<<endl;
}
}
```

```
}
};

int main(){
int c1,c2;
array2d cities;
cities.display();
cout<<"Enter cities to check"<<endl;
cin>>c1>>c2;
cities.searchneighbours(c1,c2);
}

//First Attempt

//#include<iostream>
//using namespace std;
//class Array{
//int row;
//int column;
//bool arr[5][5];
// public:
//Array(int r=5,int c=5):row(r),column(c){
//for(int i=0;i<r;i++){
//for(int j=0;j<c;j++){
// arr[i][j]=false;
//}
//}
// arr[0][1]=true;
// arr[1][0]=true;
// arr[1][2]=true;
// arr[2][1]=true;
// arr[2][4]=true;
// arr[3][1]=true;
// arr[3][2]=true;
// arr[4][1]=true;
// arr[4][2]=true;
// arr[4][3]=true;
//}
//void display(){
//for(int i=0;i<5;i++){
//for(int j=0;j<5;j++){
//cout<<arr[i][j]<<" ";
//}
// cout<<endl;
//}
//}
//void search(int city1,int city2){
//for(int i=0;i<5;i++){
//if(arr[city1][i]==arr[city2][i] && arr[city2][i]==true && arr[city1][i]==true ){
//cout<<"Common Neighbour of City1 and City2 are index "<<i<<endl;
//}
//}
//}
//};
//
//int main(){
//Array cities;
//  cities.search(0,2);
// cities.search(1,4);
// cities.display();
//}
```

LAB 02- SORTING TECHNIQUES

Question1.cpp

```cpp
//<----Lab 02- Sorting Techniques---->

// Q1. If the array is already sorted, we don’t want to continue with the comparisons. This can
// be achieved with modified bubble sort. Update the code in example 02 to have a
// modified bubble sort function.

#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n){
 bool Swap = false;
 int i, j;
 for (i = 0; i < n - 1; i++){
 // Last i elements are already in place
 for (j = 0; j < n - i - 1; j++){
 if (arr[j] > arr[j + 1]){
 Swap = true;
 swap(arr[j], arr[j + 1]);
   }
   }
 if(!Swap){
 break;
   }
   }
}
// Function to print an array
void printArray(int arr[], int size){
 int i;
 for (i = 0; i < size; i++)
 cout << arr[i] << " ";
 cout << endl;

}

// Driver code
int main(){
 int arr[] = { 5, 1, 4, 2, 8};
 int N = sizeof(arr) / sizeof(arr[0]);
 cout << "Unsorted array: \n";
 printArray(arr, N);
bubbleSort(arr, N);
 cout << "Sorted array: \n";
 printArray(arr, N);
 return 0;
}
```

Question2.cpp

```cpp
//<----Lab 02- Sorting Techniques---->

// Q2. Create a Person class which has following attributes:

// First Name
// Last Name
// Birth Year
// Birth Month
// Birth Date
// Develop C++ solution such that day month and year are taken as input for N persons and
// perform Sorting based on year, month and day using Selection Sort.

#include<iostream>
using namespace std;

class Person{
 string FirstName;
 string LastName;
 int BirthYear;
 int BirthMonth;
 int BirthDate;

 public:

 Person(string f="Random",string l="Person",int by=0,int bm=0,int bd=0):FirstName(f),
LastName(l),BirthYear(by),BirthMonth(bm),BirthDate(bd){}
 Person(int by,int bm, int bd):BirthYear(by),BirthMonth(bm),BirthDate(bd){}
 void display(){
 cout<<FirstName<<endl<<LastName<<endl<<BirthYear<<"/"<<BirthMonth<<"/"<<BirthDate<
<endl<<endl;
   }
 int getBY(){
 return BirthYear;
   }
 int getBM(){
 return BirthMonth;
   }
 int getBD(){
 return BirthDate;
   }
 void setBY(int by){
 BirthYear=by;
   }
 void setBM(int bm){
 BirthMonth=bm;
   }
 void setBD(int bd){
 BirthDate=bd;
   }

};

int main(){
 int n,temp;
```

```
 cout<<"Enter number of persons:"<<endl;
 cin>>n;
 Person people[n];
 for(int i=0;i<n;i++){
 cout<<"Enter birthyear"<<endl;
 cin>>temp;
people[i].setBY(temp);
 cout<<"Enter birthmonth"<<endl;
 cin>>temp;
people[i].setBM(temp);
 cout<<"Enter birthdate"<<endl;
 cin>>temp;
people[i].setBD(temp);
   }
 cout<<endl;
 for(int i=0;i<n;i++){
 people[i].display();
   }

 int i,j,min_index;
 for(i=0;i<n-1;i++){
 min_index = i;
 for(j=i+1;j<n;j++){
 if((people[j].getBY()<people[min_index].getBY()) ||
 ((people[j].getBM()<people[min_index].getBM())&&(people[j].getBY()==
people[min_index].getBY())) ||
            ((people[j].getBD()<people[min_index].getBD())&&(people[j].getBM()==
people[min_index].getBM())&&(people[j].getBY()==people[min_index].getBY())))
   {
 min_index=j;
 }
}
 if(min_index!=i){
 Person temp;
 temp=people[min_index];
 people[min_index]=people[i];
 people[i]=temp;
   }
   }

 cout<<endl;
 for(int i=0;i<n;i++){
 people[i].display();
   }
}
```

Question3.cpp

```cpp
//<----Lab 02- Sorting Techniques---->

// Q3. Given an array arr[ ] of length N consisting cost of N toys and an integer K the amount
// with you. The task is to find maximum number of toys you can buy with K amount.
// Test Case:
// Input: N = 7, K = 50, arr[] = {1, 12, 5, 111, 200, 1000, 10}
// Output: 4
// Explanation: The costs of the toys. You can buy are 1, 12, 5 and 10.

#include<iostream>
using namespace std;

int main(){
 int n,k;
 cout<<"Enter Number of Toys:";
 cin>>n;
 int toycosts[n];
 for(int i=0;i<n;i++){
 cout<<"Enter cost of Toy #"<<i<<" :"<<endl;
 cin>>toycosts[i];
   }

cout<<"Enter amount you have:"<<endl;
 cin>>k;
 int remaining = k;

for(int i=0;i<n-1;i++){
 for(int j=0;j<n-i-1;j++){
 if(toycosts[j]>toycosts[j+1]){
 int temp;
 temp = toycosts[j];
 toycosts[j]=toycosts[j+1];
 toycosts[j+1]=temp;
   }
   }
   }
 int count=0;
 cout<<"The Cost of items you can buy are ";
 for(int i=0;i<n;i++){
 if(toycosts[i]<=remaining){
 count++;
 remaining-=toycosts[i];
 cout<<toycosts[i]<<" ";
   }
 if(toycosts[i]>remaining){
 break;
   }
   }
 cout<<endl<<"Total Toys : "<<count<<endl;

}
```

Question4.cpp

```cpp
//<----Lab 02- Sorting Techniques---->

// Q4. Create a single class Sort, which will provide the user the option to choose between all 4
// sorting techniques. The class should have following capabilities:
// * Take an array and a string (indicating the user choice for sorting technique) as
// input and perform the desired sorting.
// * Should allow the user to perform analysis on a randomly generated array. The
// analysis provides number of comparisons and number of swaps performed for
// each technique.
// * After printing all the results, the class should highlight the best and worst
// techniques.

#include<iostream>
using namespace std;
class Sort{
int* arr;
int n;
string choice;
int cmp[3],swap[3];
public:
Sort(int x): n(x){
arr = new int[x];
}
void bubbleSort(){
for(int i=0;i<n;i++){
cmp[0]++;
for(int j=0;j<n-i-1;j++){
cmp[0]++;
if(arr[j]>arr[j+1]){
int temp;
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
swap[0]++;
}
cmp[0]++;
}
}
}
void InsertionSort(){
int key;
for(int i=1;i<n;i++){
cmp[1]++;
key=arr[i];
int j=i-1;
while( j >=0 && arr[j]>key){
arr[j+1]=arr[j];
j--;
cmp[1]++;
swap[1]++;
}
arr[j+1]=key;
```

```
swap[1]++;
}
}
void SelectionSort(){
int min_index=0;
for(int i=0;i<n;i++){
cmp[2]++;
min_index=i;
for(int j=i+1;j<n;j++){
cmp[2]++;
if(arr[j]<arr[min_index]){
min_index=j;
}
cmp[2]++;
}
if(min_index!=i){
int temp;
temp=arr[i];
arr[i]=arr[min_index];
arr[min_index]=temp;
swap[2]++;
}
}
}
void choose(){
cout<<"Enter Data of Array:"<<endl;
for(int i=0;i<n;i++){
cin>>arr[i];
}
cout<<"\nBefore Sort:\n";
Display();
    cout<<"enter choice b:bubble i:insertion s:selection"<<endl;
cin>>choice;
if(choice=="b"){
bubbleSort();
}
else if(choice=="i"){
InsertionSort();
}
else{
SelectionSort();
}
cout<<"\nAfter Sort:\n";
Display();

}
void comparison(){
for(int i=0;i<3;i++){
cmp[i]=0;
swap[i]=0;
}
int temp[n];
for(int i=0;i<n;i++){
arr[i]=rand();
temp[i]=arr[i];
}
```

```
bubbleSort();
for(int i=0;i<n;i++){
arr[i]=temp[i];
}
InsertionSort();
for(int i=0;i<n;i++){
arr[i]=temp[i];
}
SelectionSort();

int BestS=99999,WorstS=-1,BestC=99999,WorstC=-1;
 int tBS,tWS,tBC,tWC;

for(int i=0;i<3;i++){
 if(swap[i]<BestS){
 BestS=swap[i];tBS=i;
}
if(swap[i]>WorstS){
 WorstS=swap[i];tWS=i;
}
if(cmp[i]<BestC){
BestC=cmp[i];tBC=i;
}
if(cmp[i]>WorstC){
 WorstC=cmp[i];tWC=i;
}
}

cout<<"Test Array:\n";
for(int i=0;i<n;i++){
cout<<temp[i]<<" ";
}
cout<<endl<<endl;

             cout<<"Bubble Sort for "<<n<<" Size array - Comparisons: "<<cmp[0]<<" Swaps: "<<
swap[0]<<endl;
             cout<<"Insertion Sort for "<<n<<" Size array - Comparisons: "<<cmp[1]<<" Swaps:
"<<swap[1]<<endl;
             cout<<"Selection Sort for "<<n<<" Size array - Comparisons: "<<cmp[2]<<" Swaps:
"<<swap[2]<<endl<<endl<<endl;
cout<<"Technique 0 = BubbleSort, 1 = InsertionSort, 2 = SelectionSort"<<endl<<
endl;
cout<<"Best Technique (Comparisons) : "<<tBC<<" - at "<<BestC<<" Comparisons."<<
endl<<endl;
            cout<<"Worst Technique (Comparisons) : "<<tWC<<" - at "<<WorstC<<" Comparisons."<
<endl<<endl;
cout<<"Best Technique (Swaps) : "<<tBS<<" - at "<<BestS<<" Swaps."<<endl<<endl;
            cout<<"Worst Technique (Swaps) : "<<tWS<<" - at "<<WorstS<<" Swaps."<<endl<<endl;

}

void Display(){
for(int i=0;i<n;i++){
cout<<arr[i]<<"";
}
cout<<endl<<endl;
}
```

```
};

int main(){
int n;
cout<<"Enter Size of Array to sort:\n";
cin>>n;
Sort x(n);
x.choose();
    cout<<"\n\nEnter Size of Array to Compare sorts:\n";
cin>>n;
Sort y(n);
y.comparison();
}
```

LAB 03 - SINGLY LINKED LIST

Question1.cpp

```cpp
// <----Lab 03 - Singly Linked List---->

// Q1. Implement a singly linked list class with the following functions:
// a) Insert a node at head
// b) Insert a node at tail/end/back
// c) Insert a node at any position
// d) Delete a node by value
// e) Delete head
// f) Delete tail
// g) Delete a node at any position.

#include<iostream>
using namespace std;

class node {
public:
int data;
node* next;
node(int value) {
data=value;
next=NULL;
}
node(int value,node* nxt) {
data=value;
next=nxt;
}

};



class SLL { //SLL = Singly Linked List
node* head=NULL;
public:
void insertAttail(int value) {
node* n=new node(value);
             if(head==NULL) { //Check if Linked List Empty.
head=n;
return;
}
node* temp=head;
while(temp->next!=NULL) {
temp=temp->next;
}
temp->next=n;
}
void insertAtPos(int posvalue,int value) {
int count=0;
node* temp=head;
while(temp->next!=NULL&&count<posvalue-1) {
temp=temp->next;
count++;
}
            node* n=new node(value,temp->next);
temp->next=n;
```

```
}
void display() {
node* temp=head;
cout<<"[HEAD] ";
while(temp!=NULL) {
        cout<<temp->data<<"| "<<temp->next<<" -> ";
temp=temp->next;
}
cout<<"NULL [TAIL]"<<endl;
}
void insertAthead(int value) {
node* n=new node(value);
n->next=head;
head=n;
}
void deletion(int value) {
if(head==NULL) {
return;
}
node* temp=head;
if(head==NULL){
cout<<"Empty Linked List, returning\n";
return;
}
if(head->data==value){
head=head->next;
return;
}
while(temp->next->data!=value ) {
if(temp->next->next==NULL){
            cout<<"Value not found... Returning\n";
return;
}
temp=temp->next;
}
node* todelete=temp->next;
temp->next=temp->next->next;

delete todelete;
}
void deleteAthead() {
if(head==NULL) {
return;
}
node* todelete=head;
head=head->next;
delete todelete;
}
void deleteAtPos(int posvalue) {
if(head==NULL) {
return;
}
int count=0;
node* temp=head;
while(temp->next!=NULL && count<posvalue-1) {
temp=temp->next;
```

```
        count++;
        }
        node* todelete=temp->next;
        temp->next=temp->next->next;
        delete todelete;
        }
        void deleteAttail() {
        if(head==NULL) { //If linked list empty.
        return;
        }
        node* temp=head;
            if(head->next==NULL) { //If linked list has 1 item only.
        head=NULL;
        delete temp;
        return;
        }
        while(temp->next->next!=NULL) {
        temp=temp->next;
        }
        delete temp->next;
        temp->next=NULL;
        }


};


int main() {
SLL list;
float input=0;
int value;
while(input!=0.5) {
cout<<"-----------------------------------------------------\n";
cout<<"CURRENT LINKED LIST:\n";
list.display();
cout<<"-----------------------------------------------------\n";
cout<<"What would you like to do with the linked list?\n";
cout<<"1. Insert\t2. Delete\nEnter 0.5 to Exit\n[Anything else will default to Delete]\n";
cin>>input;
if(input==1) {
cout<<"Enter Value to insert: ";
cin>>value;
cout<<"Where to Insert in Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default to Insertion at Head]\n";
cin>>input;
if(input == 2){
list.insertAttail(value);
}
else if(input == 3){
int pos;
cout<<"Enter the Position to insert into: ";
cin>>pos;
list.insertAtPos(pos,value);
}
else{
list.insertAthead(value);
```

```
}

}
else if(input==0.5){
break;
}
else{
cout<<"Where to Delete from Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. At specified Position\t 4. Delete a specific Value\n[Any other value will default to Deletion from Head]\n";
cin>>input;
if(input == 2){
list.deleteAttail();
}
else if(input == 3){
int pos;
cout<<"Enter the Position to Delete from: ";
cin>>pos;
list.deleteAtPos(pos);
}
else if(input == 4){
int pos;
cout<<"Enter the Value to Delete: ";
cin>>value;
list.deletion(value);
}
else{
list.deleteAthead();
}
}


    }
}
```

Question2.cpp

```cpp
// <----Lab 03 - Singly Linked List---->

// Q2.Solve the following problem using a Singly Linked List. Given a singly linked list of
// characters, write a function to make word out of given letters in the list. Test Case:
// Input:C->S->A->R->B->B->E->L->NULL,
// Output:S->C->R->A->B->B->L->E->NULL

#include<iostream>
using namespace std;

string dictionary[]={"hello","rehan","maha","ali","class","university","section","data","
structure","algorithm","sir",
"madam","word","sentence","biryani","cold","drink","village","book","
library","bookshelf","campus",
"charger","phone","cable","computer","laptop","television",
"money","alot","large","amount","small","vocabulary","
pneumonoultramicroscopicsilicovolcaniosis","goodbye","CGPA","GPA"};



string unsorted[38];

void sortdictionary(){ //Sorting due to anagram detection logic

cout<<"DICTIONARY: Word => SortedWord"<<endl;
cout<<"_________________________________________"<<endl;
for(int i=0;i<38;i++){
unsorted[i]=dictionary[i];
}

for(int i=0;i<38;i++){ // for array
//for bubble sort
int count=0;
while(dictionary[i][count]!='\0'){
count++;
}

for(int j=0;j<count-1;j++){
for(int k=0;k<count-1-j;k++){
if(dictionary[i][k]>dictionary[i][k+1]){
char temp = dictionary[i][k];
dictionary[i][k] = dictionary[i][k+1];
dictionary[i][k+1] = temp;
}
}
}
}

for(int i=0;i<38;i++){
          cout<<unsorted[i]<<" => "<<dictionary[i]<<endl; //printing to check array.
}
cout<<endl<<endl<<endl;



}
```

```cpp

class node {
public:
char data;
node* next;
node(char value) {
data=value;
next=NULL;
}
        node(char value,node* nxt) {
data=value;
next=nxt;
}
};


class SLL { //SLL = Singly Linked List
node* head=NULL;
public:
void insertAttail(char value) {
node* n=new node(value);
             if(head==NULL) { //Check if Linked List Empty.
head=n;
return;
}
node* temp=head;
while(temp->next!=NULL) {
temp=temp->next;
}
temp->next=n;
}
        void insertAtPos(int posvalue,char value) {
int count=0;
node* temp=head;
while(temp->next!=NULL&&count<posvalue-1) {
temp=temp->next;
count++;
}
            node* n=new node(value,temp->next);
temp->next=n;
}
void display() {
node* temp=head;
cout<<"[HEAD] ";
while(temp!=NULL) {
                cout<<temp->data<<"| "<<temp->next<<" -> ";
temp=temp->next;
}
cout<<"NULL [TAIL]"<<endl;
}
void insertAthead(char value) {
node* n=new node(value);
n->next=head;
head=n;
}
void deletion(char value) {
```

```
if(head==NULL) {
return;
}
node* temp=head;
while(temp->next->data!=value ) {
temp=temp->next;
}
node* todelete=temp->next;
temp->next=temp->next->next;

delete todelete;
}
void deleteAthead() {
if(head==NULL) {
return;
}
node* todelete=head;
head=head->next;
delete todelete;
}
void deleteAtPos(int posvalue) {
if(head==NULL) {
return;
}
int count=0;
node* temp=head;
while(temp->next!=NULL && count<posvalue-1) {
temp=temp->next;
}
node* todelete=temp->next;
temp->next=temp->next->next;

delete todelete;

}
void deleteAttail() {
if(head==NULL) { //If linked list empty.
return;
}
node* temp=head;
    if(head->next==NULL) { //If linked list has 1 item only.
head=NULL;
delete temp;
}
while(temp->next->next!=NULL) {
temp=temp->next;
}
delete temp->next;
temp->next=NULL;
}

string sortlist(){ //Sorting by data not value
node* temp=head;
while(temp->next!=NULL){
node* temp2=head;
while(temp2->next!=NULL){
```

```
if(temp2->data>temp2->next->data){
char tempchar = temp2->data;
temp2->data = temp2->next->data;
temp2->next->data = tempchar;
}
temp2=temp2->next;
}
temp=temp->next;
}

string check="";
temp=head;
while(temp->next!=NULL){
check+=temp->data;
temp=temp->next;
}
check+=temp->data;
return check;
}

void reverse() {
node* prev=NULL;
node* after=NULL;
while(head!=NULL) {
after=head->next;
head->next=prev;
prev=head;
head=after;
}
head=prev;
}

void FormWord() {

string sortedWord = sortlist();
 for (int i = 0; i < 38; i++) {
 if (sortedWord == dictionary[i]) {
 cout << "Found matching word in dictionary: " << unsorted[i] << endl;
 formWord(unsorted[i]);
 return;
   }
   }
 cout << "No matching word found in dictionary." << endl;
   }

 void formWord(string targetWord) {
 node* newHead = NULL;

 for (char ch : targetWord) { //For each character in targetword "Computer"
 node* temp = head;
 node* prev = NULL;
 while (temp != NULL && temp->data != ch) { //finding a character that goes into the position
 prev = temp;
 temp = temp->next;
   }
```

```cpp

 if (temp == NULL) { //if char not found then invalid
 cout << "Error: Cannot form the word." << endl;
 return;
   }

 if (prev == NULL) { //checks if character is in 1st position or not
 head = temp->next;
 } else {
 prev->next = temp->next; //link skips temp
   }
 temp->next = newHead; //placed at head
 newHead = temp;
   }
 head = newHead; //Generates word but in reverse due to head logic
 reverse(); //fixes the reversed word generated by formword.
   }
};


int main(){
sortdictionary();
cout<<"-----------------------------------------"<<endl;
cout<<"DICTIONARY above this point\n";
cout<<"-----------------------------------------\n"<<endl;
SLL word;
char input=' ';
while(input!='@'){
        cout<<"Enter character to put in linked list [Enter '@' to end input phase] \n(CASE SENSITIVE! use lower case): ";
cin>>input;
if(input!='@'){
word.insertAttail(input);
}
}
word.display();
cout<<endl<<"Checking if word can be formed.\n\n"<<endl;
cout<<endl;
word.FormWord();
cout<<endl;
word.display();


}

```

Question3.cpp

```cpp
// <----Lab 03 - Singly Linked List---->

// Q3. Use the class of SLL created by you during the lab task 1. Do the following:
// a) Reverse the linked list
// b) Sort the contents of linked list
// c) Find the duplicates in the linked list

#include<iostream>
using namespace std;

class node {
public:
int data;
node* next;
node(int value) {
data=value;
next=NULL;
}
node(int value,node* nxt) {
data=value;
next=nxt;
}

};


class SLL { //SLL = Singly Linked List made in task 1
node* head=NULL;
public:
void insertAttail(int value) {
node* n=new node(value);
             if(head==NULL) { //Check if Linked List Empty.
head=n;
return;
}
node* temp=head;
while(temp->next!=NULL) {
temp=temp->next;
}
temp->next=n;
}
void insertAtPos(int posvalue,int value) {
int count=0;
node* temp=head;
while(temp->next!=NULL&&count<posvalue-1) {
temp=temp->next;
count++;
}
            node* n=new node(value,temp->next);
temp->next=n;
}
void display() {
node* temp=head;
cout<<"[HEAD] ";
```

```
while(temp!=NULL) {
cout<<temp->data<<"| "<<temp->next<<" -> ";
temp=temp->next;
}
cout<<"NULL [TAIL]"<<endl;
}
void insertAthead(int value) {
node* n=new node(value);
n->next=head;
head=n;
}
void deletion(int value) {
if(head==NULL) {
return;
}
node* temp=head;
while(temp->next->data!=value ) {
temp=temp->next;
}
node* todelete=temp->next;
temp->next=temp->next->next;

delete todelete;
}
void deleteAthead() {
if(head==NULL) {
return;
}
node* todelete=head;
head=head->next;
delete todelete;
}
void deleteAtPos(int posvalue) {
if(head==NULL) {
return;
}
int count=0;
node* temp=head;
while(temp->next!=NULL && count<posvalue-1) {
temp=temp->next;
}
node* todelete=temp->next;
temp->next=temp->next->next;

delete todelete;

}
void deleteAttail() {
if(head==NULL) { //If linked list empty.
return;
}
node* temp=head;
    if(head->next==NULL) { //If linked list has 1 item only.
head=NULL;
delete temp;
}
```

```
while(temp->next->next!=NULL) {
temp=temp->next;
}
delete temp->next;
temp->next=NULL;
}

void sortlist() { //sorting by swapping values
node* temp=head;
while(temp->next!=NULL) {
node* temp2=head;
while(temp2->next!=NULL) {
if(temp2->data>temp2->next->data) {
char tempchar = temp2->data;
temp2->data = temp2->next->data;
temp2->next->data = tempchar;
}
temp2=temp2->next;
}
temp=temp->next;
}
}

void reverse() {
node* prev=NULL;
node* after=NULL;
while(head!=NULL) {
after=head->next;
head->next=prev;
prev=head;
head=after;
}
head=prev;
}

void duplicates() {
         int* items = new int[100]; // Assuming a maximum of 100 unique items
int* itemcount = new int[100];
int size = 0;

node* temp = head;

while(temp != NULL) {
bool duplicate = false;
for(int i = 0; i < size; i++) {
if(temp->data == items[i]) {
duplicate = true;
itemcount[i]++;
break;
}
}
if(!duplicate) {
items[size] = temp->data;
itemcount[size] = 1;
size++;
}
```

```
temp = temp->next;
}
cout<<"Unique Items: "<<size<<endl;
for(int i = 0; i < size; i++) {
                cout<<"Data: "<<items[i]<<", Count :"<<itemcount[i]<<endl;
}

delete[] items;
delete[] itemcount;

}

};

int main() {
SLL list;
float input=0;
while(input!=0.5) {
cout<<"Enter integer to put in linked list [Enter 0.5 to end input phase]: ";
cin>>input;
if(input!=0.5) {
list.insertAthead((int)input);
}
}
cout<<"\n\n";
list.display();
char option;
     cout<<"\n\n\nDo you want to... \n'R'\tReverse the Linked List.\n'S'\tSort the Linked List.\n'D'\tFind Duplicates in Linked List?\n";
cin>>option;
if(option=='R'||option=='r'){
cout<<"Reversing the Linked List."<<endl;
list.sortlist();
list.display();
}
else if(option=='S'||option=='s'){
cout<<"Sorting the Linked List."<<endl;
list.sortlist();
list.display();
}
else if(option=='D'||option=='d'){
cout<<"Finding Duplicates: "<<endl;
list.duplicates();
}
else{
cout<<"Invalid Input. defaulting to Sort."<<endl;
list.sortlist();
list.display();
}


}
```

LAB 04 - DOUBLY AND CIRCULAR LINKED LIST

Question1.cpp

```cpp
// <----Lab 04 - Doubly and Circular Linked List---->

// Q1. Create a doubly link list and perform the mentioned tasks.
// a. Insert a new node at the end of the list.
// b. Insert a new node at the beginning of list.
// c. Insert a new node at given position.
// d. Delete any node.
// e. Print the complete doubly link list.


#include<iostream>
using namespace std;

class node {
public:
int data;
          node* nextnode; //to point to node after it
         node* prevnode; //to point to node before it
node() {
data=0;
nextnode=NULL;
prevnode=NULL;
}
node(int value) {
data=value;
nextnode=NULL;
prevnode=NULL;
}
        node(int value, node* nn, node* pn) {
data=value;
nextnode=nn;
prevnode=pn;
}
};

class DLL {
int nodecount=0;
node* head=NULL;
public:
void insertAttail(int value) {
if(head==NULL) { // if list was empty
node* n=new node(value);
head=n;
nodecount++;
return;
}
node* temp=head;
            while(temp->nextnode!=NULL) {
temp=temp->nextnode;
}
node* n=new node(value,NULL,temp);
temp->nextnode=n;
nodecount++;
```

```
}
void insertAthead(int value) {
node* n=new node(value,head,NULL);
if(head!=NULL){
head->prevnode=n;
}
head=n;
nodecount++;
}
void insertAtPos(int pos,int value) {
if(pos<0){
cout<<"Position less than 0, Inserting at head.\n";
insertAthead(value);
return;
}
if(pos>nodecount-1){
cout<<"Position more than nodes in list, Inserting at tail.\n";
insertAttail(value);
return;
}
int count=0;
node* temp=head;
while(temp->nextnode!=NULL && count<pos-1) {
temp=temp->nextnode;
count++;
}
node* n=new node(value,temp->nextnode,temp);
temp->nextnode=n;
n->nextnode->prevnode=n;
nodecount++;
}
void display() {
node* temp=head;
cout<<"HEAD | ";
while(temp!=NULL) {
                    cout<<" <--"<<temp->prevnode<<" | "<<temp->data<<" | "<<temp->nextnode<<" --> ";
temp=temp->nextnode;
}
cout<<"| TAIL"<<endl;
}
//Assuming ANY node means any of the 4 types (head,tail,position,value)
void deleteAtHead() {
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* todelete=head;
head=head->nextnode;
head->prevnode=NULL;
delete todelete;
nodecount--;
}
void deletion(int value) {
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
```

```
return;
}
node* temp=head;
if(head->data==value) {
deleteAtHead();
return;
}
while(temp->data!=value) {
if(temp->nextnode==NULL) {
cout<<"Value not found, Returning\n";
return;
}
temp=temp->nextnode;
}
if(temp->nextnode==NULL){
deleteAtTail();
return;
}
node* todelete=temp;
temp->prevnode->nextnode=temp->nextnode;
temp->nextnode->prevnode=temp->prevnode;
delete todelete;
nodecount--;
}
void deleteAtPos(int pos) {
if(pos<0){
        cout<<"Position less than zero, INVALID. Returning..."<<endl;
return;
}
if(pos==0){
deleteAtHead();
return;
}
else if(pos==nodecount-1){
deleteAtTail();
return;
}
if(pos>nodecount-1){
cout<<"Invalid Position, Returning"<<endl;
return;
}
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
int count=0;
node* temp=head;
while(temp->nextnode!=NULL && count<pos-1) {
temp=temp->nextnode;
count++;
}
node* todelete=temp->nextnode;
temp->nextnode=temp->nextnode->nextnode;
temp->nextnode->prevnode=temp;
delete todelete;
nodecount--;
```

```
}
void deleteAtTail() {
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* temp=head;
while(temp->nextnode!=NULL) {
temp=temp->nextnode;
}
node* todelete=temp;
temp=temp->prevnode;
temp->nextnode=NULL;
delete todelete;
nodecount--;
}
};

int main() {
DLL list;
float input=0;
int value;
while(input!=0.5) {
cout<<"-----------------------------------------------------\n";
cout<<"CURRENT LINKED LIST:\n";
list.display();
cout<<"-----------------------------------------------------\n";
cout<<"What would you like to do with the linked list?\n";
cout<<"1. Insert\t2. Delete\nEnter 0.5 to Exit\n[Anything else will default to
Delete]\n";
cin>>input;
if(input==1) {
cout<<"Enter Value to insert: ";
cin>>value;
cout<<"Where to Insert in Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will
default to Insertion at Head]\n";
cin>>input;
if(input == 2){
list.insertAttail(value);
}
else if(input == 3){
int pos;
cout<<"Enter the Position to insert into: ";
cin>>pos;
list.insertAtPos(pos,value);
}
else{
list.insertAthead(value);
}

}
else if(input==0.5){
break;
}
else{
cout<<"Where to Delete from Linked List?\n";
```

```
            cout<<"1. At head\t2. At tail\t3. At specified Position\t 4. Delete a specific Value\n[Any other value will default to Deletion from Head]\n";
cin>>input;
if(input == 2){
list.deleteAtTail();
}
else if(input == 3){
int pos;
cout<<"Enter the Position to Delete from: ";
cin>>pos;
list.deleteAtPos(pos);
}
else if(input == 4){
int pos;
cout<<"Enter the Value to Delete: ";
cin>>value;
list.deletion(value);
}
else{
list.deleteAtHead();
}
}



    }
}
```

Question2.cpp

```cpp
// <----Lab 04 - Doubly and Circular Linked List---->

// Q2. Create two doubly link lists, say L and M . List L should contain all even elements from 2 to
// 10 and list M should contain all odd elements from 1 to 9. Create a new list N by
// concatenating list L and M.

#include<iostream>
using namespace std;

class node {
public:
int data;
node* nextnode; //to point to node after it
node* prevnode; //to point to node before it
node() {
data=0;
nextnode=NULL;
prevnode=NULL;
}
node(int value) {
data=value;
nextnode=NULL;
prevnode=NULL;
}
node(int value, node* nn, node* pn) {
data=value;
nextnode=nn;
prevnode=pn;
}
};

class DLL {
int nodecount=0;
node* head=NULL;
public:
void insertAttail(int value) {
if(head==NULL) { // if array was empty
node* n=new node(value);
head=n;
nodecount++;
return;
}
node* temp=head;
while(temp->nextnode!=NULL) {
temp=temp->nextnode;
}
node* n=new node(value,NULL,temp);
temp->nextnode=n;
nodecount++;
}
void insertAthead(int value) {
node* n=new node(value,head,NULL);
```

```
if(head!=NULL){
head->prevnode=n;
}
head=n;
nodecount++;
}
void insertAtPos(int pos,int value) {
if(pos<0){
cout<<"Position less than 0, Inserting at head.\n";
insertAthead(value);
return;
}
if(pos>nodecount-1){
cout<<"Position more than nodes in list, Inserting at tail.\n";
insertAttail(value);
return;
}
int count=0;
node* temp=head;
while(temp->nextnode!=NULL && count<pos-1) {
temp=temp->nextnode;
count++;
}
node* n=new node(value,temp->nextnode,temp);
temp->nextnode=n;
n->nextnode->prevnode=n;
nodecount++;
}
void display() {
node* temp=head;
cout<<"HEAD | ";
while(temp!=NULL) {
                    cout<<" <--"<<temp->prevnode<<" | "<<temp->data<<" | "<<temp->nextnode<<" --> ";
temp=temp->nextnode;
}
cout<<"| TAIL"<<endl;
}
//Assuming ANY node means any of the 4 types (head,tail,position,value)
void deleteAtHead() {
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* todelete=head;
head=head->nextnode;
head->prevnode=NULL;
delete todelete;
nodecount--;
}
void deletion(int value) {
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* temp=head;
```

```
if(head->data==value) {
deleteAtHead();
return;
}
while(temp->data!=value) {
if(temp->nextnode==NULL) {
cout<<"Value not found, Returning\n";
return;
}
temp=temp->nextnode;
}
if(temp->nextnode==NULL){
deleteAtTail();
return;
}
node* todelete=temp;
temp->prevnode->nextnode=temp->nextnode;
temp->nextnode->prevnode=temp->prevnode;
delete todelete;
nodecount--;
}
void deleteAtPos(int pos) {
if(pos<0){
        cout<<"Position less than zero, INVALID. Returning..."<<endl;
return;
}
if(pos==0){
deleteAtHead();
return;
}
else if(pos==nodecount-1){
deleteAtTail();
return;
}
if(pos>nodecount-1){
cout<<"Invalid Position, Returning"<<endl;
return;
}
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
int count=0;
node* temp=head;
while(temp->nextnode!=NULL && count<pos-1) {
temp=temp->nextnode;
count++;
}
node* todelete=temp->nextnode;
temp->nextnode=temp->nextnode->nextnode;
temp->nextnode->prevnode=temp;
delete todelete;
nodecount--;
}
void deleteAtTail() {
if(head==NULL) {
```

```
cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* temp=head;
while(temp->nextnode!=NULL) {
temp=temp->nextnode;
}
node* todelete=temp;
temp=temp->prevnode;
temp->nextnode=NULL;
delete todelete;
nodecount--;
}
void concatlist(DLL &obj){
node* temp=obj.head;
while(temp!=NULL){
insertAttail(temp->data);
temp=temp->nextnode;
}
}
};

int main(){
DLL l,m;
for(int i=2;i<11;i+=2){ //Initializing L and M with evens and odds respectively
l.insertAttail(i);
m.insertAttail(i-1);
}
cout<<"-----------LIST L (Evens)-------------------"<<endl;
l.display();//Showing L for surity of values
cout<<"-----------LIST M (odds)-------------------------"<<endl;
m.display();//Showing M for surity of values
cout<<"------------------------------------"<<endl;
DLL n;//Creates empty linked list
n.concatlist(l); //concats L into N
n.concatlist(m); //concats M into N
    cout<<"-----------LIST N after concatenating L & M-------------------------"<<endl;
n.display();
}
```

Question3.cpp

```cpp
// <----Lab 04 - Doubly and Circular Linked List---->

// Q3. Using the above created list N, sort the contents of list N is descending order.

#include<iostream>
using namespace std;

class node {
public:
int data;
node* nextnode; //to point to node after it
node* prevnode; //to point to node before it
node() {
data=0;
nextnode=NULL;
prevnode=NULL;
}
node(int value) {
data=value;
nextnode=NULL;
prevnode=NULL;
}
node(int value, node* nn, node* pn) {
data=value;
nextnode=nn;
prevnode=pn;
}
};

class DLL {
int nodecount=0;
node* head=NULL;
public:
void insertAttail(int value) {
if(head==NULL) { // if array was empty
node* n=new node(value);
head=n;
nodecount++;
return;
}
node* temp=head;
while(temp->nextnode!=NULL) {
temp=temp->nextnode;
}
node* n=new node(value,NULL,temp);
temp->nextnode=n;
nodecount++;
}
void insertAthead(int value) {
node* n=new node(value,head,NULL);
if(head!=NULL){
head->prevnode=n;
}
```

```
head=n;
nodecount++;
}
void insertAtPos(int pos,int value) {
if(pos>nodecount-1){
cout<<"Position more than nodes in list, Inserting at tail.\n";
insertAttail(value);
return;
}
int count=0;
node* temp=head;
while(temp->nextnode!=NULL && count<pos-1) {
temp=temp->nextnode;
count++;
}
node* n=new node(value,temp->nextnode,temp);
temp->nextnode=n;
n->nextnode->prevnode=n;
nodecount++;
}
void display() {
node* temp=head;
cout<<"HEAD | ";
while(temp!=NULL) {
                    cout<<" <--"<<temp->prevnode<<" | "<<temp->data<<" | "<<temp->nextnode<<" --> ";
temp=temp->nextnode;
}
cout<<"| TAIL"<<endl;
}
//Assuming ANY node means any of the 4 types (head,tail,position,value)
void deleteAtHead() {
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* todelete=head;
head=head->nextnode;
head->prevnode=NULL;
delete todelete;
nodecount--;
}
void deletion(int value) {
if(head==NULL) {
cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* temp=head;
if(head->data==value) {
deleteAtHead();
return;
}
while(temp->data!=value) {
if(temp->nextnode==NULL) {
cout<<"Value not found, Returning\n";
return;
```

```
}
temp=temp->nextnode;
}
if(temp->nextnode==NULL){
deleteAtTail();
return;
}
node* todelete=temp;
temp->prevnode->nextnode=temp->nextnode;
temp->nextnode->prevnode=temp->prevnode;
delete todelete;
nodecount--;
}
void deleteAtPos(int pos) {
if(pos==0){
deleteAtHead();
return;
}
else if(pos==nodecount-1){
deleteAtTail();
return;
}
if(pos>nodecount-1){
cout<<"Invalid Position, Returning"<<endl;
return;
}
if(head==NULL) {
        cout<<"Empty Linked List, Returning"<<endl;
return;
}
int count=0;
node* temp=head;
while(temp->nextnode!=NULL && count<pos-1) {
temp=temp->nextnode;
count++;
}
node* todelete=temp->nextnode;
temp->nextnode=temp->nextnode->nextnode;
temp->nextnode->prevnode=temp;
delete todelete;
nodecount--;
}
void deleteAtTail() {
if(head==NULL) {
        cout<<"Empty Linked List, Returning"<<endl;
return;
}
node* temp=head;
while(temp->nextnode!=NULL) {
temp=temp->nextnode;
}
node* todelete=temp;
temp=temp->prevnode;
temp->nextnode=NULL;
delete todelete;
nodecount--;
```

```
}
void concatlist(DLL &obj){
node* temp=obj.head;
while(temp!=NULL){
insertAttail(temp->data);
temp=temp->nextnode;
}
}
void sortlist(){//sorting by data , bubble sort
node* temp=head;
while(temp->nextnode!=NULL) {
node* temp2=head;
while(temp2->nextnode!=NULL) {
if(temp2->data<temp2->nextnode->data) {
char tempchar = temp2->data;
temp2->data = temp2->nextnode->data;
temp2->nextnode->data = tempchar;
}
temp2=temp2->nextnode;
}
temp=temp->nextnode;
}
}
};

int main(){
DLL l,m;
for(int i=2;i<11;i+=2){ //Initializing L and M
l.insertAttail(i);
m.insertAttail(i-1);
}
cout<<"-----------LIST L (Evens)------------------"<<endl;
l.display();
cout<<"-----------LIST M (odds)-------------------------"<<endl;
m.display();
cout<<"------------------------------------"<<endl;
DLL n;
n.concatlist(l);
n.concatlist(m);
    cout<<"-----------LIST N after concatenating L & M-------------------------"<<endl;
n.display();

//Question 3 part starts here

cout<<"-----------LIST N after Desc Sort-------------------------"<<endl;
n.sortlist();//calls sorting function
n.display();



}
```

Question4.cpp

```cpp
// <----Lab 04 - Doubly and Circular Linked List---->

// Q4. Create a circular link list and perform the mentioned tasks.
// a. Insert a new node at the end of the list.
// b. Insert a new node at the beginning of list.
// c. Insert a new node at given position.
// d. Delete any node.
// e. Print the complete doubly link list.

#include<iostream>
using namespace std;

class node {
public:
int data;
node* next;
node(int value) {
data=value;
next=NULL;
}
node(int value,node* nxt) {
data=value;
next=nxt;
}
};

class CLL{
node* head=NULL;
int nodecount=0;
public:
void appendNode(int value){ //insert at end of list / tail
if(head==NULL) { //Check if Linked List Empty.
node* n=new node(value,n);
head=n;
head->next=head;
nodecount++;
return;
}
else{
node* temp=head;
while(temp->next!=head) {
temp=temp->next;
}
node* n=new node(value,head);
temp->next=n;
nodecount++;
}
}
void prependNode(int value){ //insert at start of list / head
if(head==NULL) { //Check if Linked List Empty.
node* n=new node(value,n);
head=n;
head->next=head;
```

```
    nodecount++;
    return;
    }
    node* n=new node(value,head);
    node* temp=head;
    while(temp->next!=head) {
    temp=temp->next;
    }
    temp->next=n;
    head=n;
    nodecount++;
    }
    void insertNodeAfter(int posvalue,int value){ // insert at position (i+1)
    if(head==NULL) { //Check if Linked List Empty.
    cout<<"Empty List, adding at Head.\n";
    node* n=new node(value,n);
    head=n;
    nodecount++;
    return;
    }
    if(posvalue>nodecount-1){
    cout<<"Position more than nodes in list, Inserting at tail.\n";
    appendNode(value);
    return;
    }
    int count=0;
    node* temp=head;
    while(temp->next!=head&&count<posvalue) {
    temp=temp->next;
    count++;
    }
    node* n=new node(value,temp->next);
    temp->next=n;
    }
    void deleteathead(){
    node* temp=head;
    while(temp->next=head){
    temp=temp->next;
    }
    temp->next=head->next;
    delete head;
    head = temp->next;
    }
    void deleteattail(){
    node* temp=head;
    while(temp->next->next!=head){
    temp=temp->next;
    }
    node* todelete=temp->next;
    temp->next=head;
    delete todelete;
    }
    void deleteNodeByKey(int value){ // delete by value
    node* temp=head;
    node* prev=temp;
    while(temp->next!=head&&temp->data!=value) {
```

```
prev=temp;
temp=temp->next;
}
if(temp->data==value){
prev->next=temp->next; //skip temp (i.e delete)
delete temp;
}
if(temp->next==head){
cout<<"Value not in Linked List.\n";
return;
}
}
void updateNodeByKey(int value){ // update by value
node* temp=head;
while(temp->next!=head&&temp->data!=value) {
node* prev=temp;
temp=temp->next;
}
if(temp->data==value){
cout<<"Enter new value: ";
cin>>temp->data;
}
if(temp->next==head){
cout<<"Value not in Linked List.\n";
return;
}
}
void print(){
node* temp=head;
cout<<"[HEAD] ";
if(head!=NULL){
cout<<temp->data<<"| "<<temp->next<<" -> ";
temp=temp->next;
while(temp!=head) {
cout<<temp->data<<"| "<<temp->next<<" -> ";
temp=temp->next;
}
cout<<"head [TAIL]"<<endl;
}
else{
cout<<"NULL [TAIL]"<<endl;
}
}
};

int main(){
CLL list;
float input=0;
int value;
while(input!=0.5) {
cout<<"-----------------------------------------------------\n";
cout<<"CURRENT LINKED LIST:\n";
list.print();
cout<<"-----------------------------------------------------\n";
cout<<"What would you like to do with the linked list?\n";
        cout<<"1. Insert\t2. Delete\t3.Update\nEnter 0.5 to Exit\n[Anything else will default
```

```
to Delete]\n";
cin>>input;
if(input==1) {
cout<<"Enter Value to insert: ";
cin>>value;
cout<<"Where to Insert in Linked List?\n";
cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default to Insertion at Head]\n";
cin>>input;
if(input == 2){
list.appendNode(value);
}
else if(input == 3){
int pos;
cout<<"Enter the Position to insert After: ";
cin>>pos;
list.insertNodeAfter(pos,value);
}
else{
list.prependNode(value);
}

}
else if(input==0.5){
break;
}
else if(input==3){
cout<<"Enter the Value to Update: ";
cin>>value;
list.updateNodeByKey(value);
}
else{
cout<<"Where to Delete from Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. Delete a specific Value\n[Any other value will default to Deletion from Head]\n";
cin>>input;
if(input == 2){
list.deleteattail();
}
else if(input == 3){
int pos;
cout<<"Enter the Value to Delete: ";
cin>>value;
list.deleteNodeByKey(value);
}
else{
list.deleteathead();
}
}



    }
}
```

Question5.cpp

```cpp
// <----Lab 04 - Doubly and Circular Linked List---->

// Q5. Break the above-created circular linked list into two halves.

#include<iostream>
using namespace std;

class node {
public:
int data;
node* next;
node(int value) {
data=value;
next=NULL;
}
node(int value,node* nxt) {
data=value;
next=nxt;
}
};

class CLL{
node* head=NULL;
int nodecount=0;
public:
void setlist(node* h,int n){
head=h;
nodecount=n;
}
void makehalves(CLL &obj){
int middleposition = (int)nodecount/2; //Amount for 1st half
int secondhalf = nodecount-middleposition; //Amount for 2nd half
int count=0;
node* temp=head;
node* prev=temp;
          while(temp->next!=head&&count<middleposition) { // prev is last element of first half, temp is head of second half
prev=temp;
temp=temp->next;
count++;
}
prev->next=head; //detach prev from remaining half, circular link back to start
node* secondhead=temp;
while(temp->next!=head){
temp=temp->next;
}
temp->next=secondhead;
obj.setlist(secondhead,secondhalf);
}

void appendNode(int value){ //insert at end of list / tail
if(head==NULL) { //Check if Linked List Empty.
node* n=new node(value,n);
```

```
    head=n;
    head->next=head;
    nodecount++;
    return;
    }
    else{
    node* temp=head;
    while(temp->next!=head) {
    temp=temp->next;
    }
    node* n=new node(value,head);
    temp->next=n;
    nodecount++;
    }
    }
    void prependNode(int value){ //insert at start of list / head
    if(head==NULL) { //Check if Linked List Empty.
    node* n=new node(value,n);
    head=n;
    head->next=head;
    nodecount++;
    return;
    }
    node* n=new node(value,head);
    node* temp=head;
    while(temp->next!=head) {
    temp=temp->next;
    }
    temp->next=n;
    head=n;
    nodecount++;
    }
    void insertNodeAfter(int posvalue,int value){ // insert at position (i+1)
    if(head==NULL) { //Check if Linked List Empty.
    cout<<"Empty List, adding at Head.\n";
    node* n=new node(value,n);
    head=n;
    nodecount++;
    return;
    }
    if(posvalue>nodecount-1){
    cout<<"Position more than nodes in list, Inserting at tail.\n";
    appendNode(value);
    return;
    }
    int count=0;
    node* temp=head;
    while(temp->next!=head&&count<posvalue) {
    temp=temp->next;
    count++;
    }
    node* n=new node(value,temp->next);
    temp->next=n;
    }
    void deleteathead(){
    node* temp=head;
```

```
    while(temp->next=head){
    temp=temp->next;
    }
    temp->next=head->next;
    }
    void deleteattail(){
    node* temp=head;
    while(temp->next->next=head){
    temp=temp->next;
    }
    temp->next=head;
    }
    void deleteNodeByKey(int value){ // delete by value
    node* temp=head;
    node* prev=temp;
    while(temp->next!=head&&temp->data!=value) {
    prev=temp;
    temp=temp->next;
    }
    if(temp->data==value){
             prev->next=temp->next; //skip temp (i.e delete)
    delete temp;
    }
    if(temp->next==head){
    cout<<"Value not in Linked List.\n";
    return;
    }
    }
    void updateNodeByKey(int value){ // update by value
    node* temp=head;
    while(temp->next!=head&&temp->data!=value) {
    node* prev=temp;
    temp=temp->next;
    }
    if(temp->data==value){
    cout<<"Enter new value: ";
    cin>>temp->data;
    }
    if(temp->next==head){
    cout<<"Value not in Linked List.\n";
    return;
    }
    }
    void print(){
    node* temp=head;
    cout<<"[HEAD] ";
    if(head!=NULL){
    cout<<temp->data<<"| "<<temp->next<<" -> ";
    temp=temp->next;
    while(temp!=head) {
    cout<<temp->data<<"| "<<temp->next<<" -> ";
    temp=temp->next;
    }
    cout<<"head [TAIL]"<<endl;
    }
    else{
```

```
cout<<"NULL [TAIL]"<<endl;
}
}
};

int main(){
CLL list,list2;
float input=0;
int value;
while(input!=0.5) {
cout<<"-----------------------------------------------------\n";
cout<<"CURRENT LINKED LIST:\n";
list.print();
cout<<"-----------------------------------------------------\n";
cout<<"What would you like to do with the linked list?\n";
        cout<<"1. Insert\t2. Delete\t3. Update\t4. Halve the Linked List [Halving will end the program]\nEnter 0.5 to Exit\n[Anything else will default to Delete]\n";
cin>>input;
if(input==1) {
cout<<"Enter Value to insert: ";
cin>>value;
cout<<"Where to Insert in Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default to Insertion at Head]\n";
cin>>input;
if(input == 2){
list.appendNode(value);
}
else if(input == 3){
int pos;
cout<<"Enter the Position to insert After: ";
cin>>pos;
list.insertNodeAfter(pos,value);
}
else{
list.prependNode(value);
}

}
else if(input==0.5){
break;
}
else if(input==3){
cout<<"Enter the Value to Update: ";
cin>>value;
list.updateNodeByKey(value);
}
else if(input==4){
cout<<"Halving list"<<endl;
list.makehalves(list2);
cout<<"First Half list--------------------------------"<<endl;
list.print();
cout<<"Second Half list-------------------------------"<<endl;
list2.print();
return 0;
}
else{
```

```
cout<<"Where to Delete from Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. Delete a specific Value\n[Any other value will default to Deletion from Head]\n";
cin>>input;
if(input == 2){
list.deleteattail();
}
else if(input == 3){
int pos;
cout<<"Enter the Value to Delete: ";
cin>>value;
list.deleteNodeByKey(value);
}
else{
list.deleteathead();
}
}


    }
}
```

LAB 05 - STACKS

Question1.cpp

```cpp
// <----Lab 05 - Stacks---->

// Q1. Please write a program which performs the following tasks:
// a. Make a left to right scan of the postfix expression
// b. If the element is an operand push it on Stack
// c. If the element is operator, evaluate it using as operands the correct number from stack
// and pushing the result onto the stack
#include<iostream>
#include<string>

using namespace std;

class stack {
float arr[20];
int top=-1;
public:
stack() {
for(int i=0; i<20; i++) {
arr[i]=0;
}
}
bool full() {
return top==19;
}
bool empty() {
return top==-1;
}

void push(float value) {
if(!full()) {
top++;
arr[top]=value;
} else {
cout<<"Array is Full.\n";
return;
}
}
float pop() {
if(!empty()) {
int temp = arr[top];
top--;
return temp;
} else {
cout<<"Array is Empty.\n";
return -1;
}
}

};

float solveeq(stack &obj,string exp) {
string curr="";
for(int i=0; i<exp.length(); i++) {
```

```
curr = exp[i];
if(curr>="0"&&curr<="9") {
obj.push(stoi(curr));
} else if(exp[i]=='+') {
float temp=obj.pop();
obj.push(obj.pop()+temp);
} else if(exp[i]=='-') {
float temp=obj.pop();
obj.push(obj.pop()-temp);
} else if(exp[i]=='/') {
float temp=obj.pop();
obj.push(obj.pop()/temp);
} else if(exp[i]=='*') {
float temp=obj.pop();
obj.push(obj.pop()*temp);
} else {
cout<<"Invalid Character!"<<endl;
}
}
return obj.pop();
}

int main() {
string exp;
stack e;
     cout<<"Enter Postfix Expression without Exponents (Numbers will be read as Single Digits and as INTEGERS): \n";
cin>>exp;
cout<<"Answer of your Expression: "<<solveeq(e,exp)<<endl;
}
```

Question2.cpp

```cpp
// <----Lab 05 - Stacks---->

// Q2. A palindrome is a word, phrase, number, or another sequence of characters that reads the same
// backward and forwards. Can you determine if a given string, s, is a palindrome? Write a
// Program using stack for checking whether a string is palindrome or not.

#include<iostream>
#include<string>

using namespace std;

class stack {
char arr[20];
int top=-1;
public:
stack() {
for(int i=0; i<20; i++) {
arr[i]=0;
}
}
bool full() {
return top==19;
}
bool empty() {
return top==-1;
}

void push(char value) {
if(!full()) {
top++;
arr[top]=value;
} else {
cout<<"Array is Full.\n";
return;
}
}
char pop() {
if(!empty()) {
char temp = arr[top];
top--;
return temp;
} else {
cout<<"Array is Empty.\n";
return -1;
}
}
void display() {
cout<<"\n--";
for(int i=0; i<=top; i++) {
cout<<arr[i];
}
cout<<"--\n";
```

```
}
};

bool checkp(string s) {
stack word;
for(int i=0; i<s.length(); i++) {
word.push(s[i]);
}
string reversed="";
for(int i=0; i<s.length(); i++) {
reversed+=word.pop();
}
return s==reversed;
}

int main() {
string str;
    cout<<"Enter String to check for palindrome:\n";
getline(cin,str);
if(checkp(str)) {
cout<<"String is Palindrome\n";
} else {
cout<<"String is NOT Palindrome\n";
}
}
```

Question3.cpp

```cpp
// <----Lab 05 - Stacks---->

// Q3. Write a program using stacks which takes an expression as input and determines whether the
// delimiters are matched or not.

#include<iostream>
#include<string>

using namespace std;

class stack {
char arr[20];
int top=-1;
public:
stack() {
for(int i=0; i<20; i++) {
arr[i]=0;
}
}
bool full() {
return top==19;
}
bool empty() {
return top==-1;
}

void push(char value) {
if(!full()) {
top++;
arr[top]=value;
} else {
cout<<"Array is Full.\n";
return;
}
}
void pop() {
if(!empty()) {
top--;
}
else{
cout<<"Stack is Empty!"<<endl;
}
}
char Top(){
if(!empty()){
return arr[top];
}
else{
cout<<"Array Empty, Returning 'X'...\n";
return 'X';
}

        }
```

```
void display() {
cout<<"\n--";
for(int i=0; i<=top; i++) {
cout<<arr[i];
}
cout<<"--\n";
}
};

bool MatchDelim(string ex){
stack delims;
for(int i=0;i<ex.length();i++){
if(ex[i]=='{'||ex[i]=='('||ex[i]=='['){
delims.push(ex[i]);
}
else if(ex[i]==')'||ex[i]=='}'||ex[i]==']'){
if(ex[i]==')' && delims.Top()=='('){
delims.pop();
}
else if(ex[i]==']' && delims.Top()=='['){
delims.pop();
}
else if(ex[i]=='}' && delims.Top()=='{'){
delims.pop();
}
else{
                cout<<"Open and Close Brackets Mismatched at position "<<i<<endl;
return false;
}
}
}
if(delims.empty()){
cout<<"Expression is valid\n";
return true;
}
else{
cout<<"Unclosed Delimiters Remain\n";
delims.display();
return false;
}
}

int main(){
string exp;
cout<<"Enter Expressions to match the Delimiters:\n";
getline(cin,exp);
MatchDelim(exp);
}
```

LAB 06 - QUEUES

Question01.cpp

```cpp
// <----Lab 06 - Queues---->

// Q1. Write the definition of all the functions listed in Example 01.

#include <iostream>
using namespace std;

class Queue
{
private:
 int front, rear;
 unsigned capacity;
 int *array;

public:
 Queue(int capacity)
  {
 array = new int[capacity];
 front = -1;
 rear = -1;
 this->capacity = capacity;
  }

 bool isEmpty()
  {
 return (rear == -1 && front == -1);
  }

 bool isFull()
  {
 return ((rear + 1) % capacity == front);
  }

 void enqueue(int element)
  {
 if (isFull())
  {
 cerr << "Queue is full cannot enqueue." << endl;
 return;
  }
 else if (isEmpty())
  {
 front = rear = 0;
  }
 else
  {
 rear = (rear + 1) % capacity;
  }
 array[rear] = element;
 cout << array[rear] << " is enqueued in the queue." << endl;
  }

 void dequeue()
```

```
  {
 if (isEmpty())
  {
 cerr << "Queue is Empty cannot dequeue." << endl;
 return;
  }
 else if (front == rear)
  {
 cout << array[front] << " is dequeued from the queue." << endl;
 rear = front = -1;
  }
 else
  {
 cout << array[front] << " is dequeued from the queue." << endl;  front =
(front + 1) % capacity;
  }
  }

 int peek()
  {
 if (isEmpty())
  {
 cerr << "Queue is empty." << endl;
  }
 return array[front];
  }

 int queue_size()
  {
 if (isEmpty())
 return 0;
 else if (rear >= front)
  {
 return rear - front + 1;
  }
 else
  {
 return (capacity - front + rear) % capacity;
  }
  }
};

int main()
{
Queue q(10);
 q.enqueue(1);
 q.enqueue(2);
 q.enqueue(3);
 q.enqueue(4);
 q.enqueue(5);
 q.enqueue(6);
 q.enqueue(7);
 q.enqueue(8);
 cout << "Front element " << q.peek() << endl;
 cout << "Size of queue " << q.queue_size() << endl;
 return 0;
}
```

Question02.cpp

```cpp
// <----Lab 06 - Queues---->

// Q2. Please implement the Generic Queue definition using singly linked list (you may use the
// Singly Linked List that you already developed in Lab # 3), you may also add any
// functions needed in the Singly Linked List definition given in Lab # 3. Your
// implementation should work for the main function given below.
#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>
using namespace std;

template <class T>
class Node
{
public:
 T data;
 Node *next;
 Node(T data)
   {
 this->data = data;
 next = nullptr;
   }
};

template <class T>
class Queue
{
private:
 Node<T> *rear;
 Node<T> *front;
 int size;
 int capacity;

public:
 Queue(int capacity) : rear(nullptr), front(nullptr)
   {
 this->capacity = capacity;
 size = 0;
   }

 bool isEmpty()
   {
 return (rear == nullptr && front == nullptr);
   }

 bool isFull()
   {
 return (size == capacity);
   }

 void Put(T value)

```

```
  {
Node<T> *newnode = new Node<T>(value);
if (isFull())
  {
cerr << "Cannot enqueue queue is full" << endl;
delete newnode;
return;
  }
else if (isEmpty())
  {
rear = front = newnode;
  }
else
  {
rear->next = newnode;
rear = newnode;
++size;
  }
  }

void dequeue()
  {
if (isEmpty())
  {
cerr << "Queue is empty cannot dequeu." << endl;
return;
  }
else if (rear == front)
  {
rear = front = nullptr;
  }
else
  {
Node<T> *temp = front;
front = front->next;
delete temp;
  }
--size;
  }

T Get()
  {
T value = front->data;
dequeue();
return value;
  }
};
#endif
```

Queue.h

```
// <----Lab 06 - Queues---->
#include <iostream>
#include "Queue.h"
using namespace std;

int main()
{
Queue<int> *q = new Queue<int>(10);
 if (q->isEmpty())
   {
 cout << "Queue is empty." << endl;
   }
 q->Put(1);
 q->Put(2);
 q->Put(3);
 while (!q->isEmpty())
   {
 cout << q->Get() << endl;
   }
 return 0;
}
```

Question03.cpp

```cpp
// <----Lab 06 - Queues---->

// Q3. Give the C++ code to implement the Generic Queue using array. Please change the
// private data members as you are now using an array.


#include <iostream>
using namespace std;

template <class T>
class Queue
{
private:
 int front, rear;
 unsigned capacity;
 T *array;

public:
 Queue(int capacity)
  {
 array = new T[capacity];
 front = -1;
 rear = -1;
 this->capacity = capacity;
  }

 bool isEmpty()
  {
 return (rear == -1 && front == -1);
  }

 bool isFull()
  {
 return ((rear + 1) % capacity == front);
  }

 void enqueue(T element)
  {
 if (isFull())
  {
 cerr << "Queue is full cannot enqueue." << endl;
 return;
  }
 else if (isEmpty())
  {
 front = rear = 0;
  }
 else
  {
 rear = (rear + 1) % capacity;
  }
 array[rear] = element;
 cout << array[rear] << " is enqueued in the queue." << endl;
```

```
  }

 void dequeue()
  {
 if (isEmpty())
  {
 cerr << "Queue is Empty cannot dequeue." << endl;
 return;
  }
 else if (front == rear)
  {
 cout << array[front] << " is dequeued from the queue." << endl;
 rear = front = -1;
  }
 else
  {
 cout << array[front] << " is dequeued from the queue." << endl;
 front = (front + 1) % capacity;
  }
  }

 T peek()
  {
 if (isEmpty())
  {
 cerr << "Queue is empty." << endl;
  }
 return array[front];
  }

 int queue_size()
  {
 if (isEmpty())
 return 0;
 else if (rear >= front)
  {
 return rear - front + 1;
  }
 else
  {
 return (capacity - front + rear) % capacity;
  }
  }
};

int main()
{
Queue<char> q(10);
 q.enqueue('a');
 q.enqueue('b');
 q.enqueue('c');
 q.enqueue('d');
 q.enqueue('e');
 q.enqueue('f');
 q.enqueue('g');
 q.enqueue('h');
```

```
 q.enqueue('h');
 q.enqueue('h');
 q.enqueue('h');
 q.enqueue('h');
 q.enqueue('h');
 q.dequeue();
 q.dequeue();
 cout << "Front element " << q.peek() << endl;
 cout << "Size of queue " << q.queue_size() << endl;
 return 0;
}
```