Assignment - 1

Q1- Differentiate betn array & linked list

Array	linked list
(i) fixed size i.e. has a specified size only	Dynamic size i.e. can be increased or decreased dynamically.
(ii) Contiguous memory allocation.	Non contiguous memory
(iii) index based iteration possible.	sequential access not possible by indexing.
(iv) memory is allocated to single block of array.	each node of LL is given a location in memory.
(v) Access time for elements in O(1) i.e. constant time complexity	Access time for elements is O(n) i.e. linear time complexity
(vi) Insertion/Deletion taken O(n).	Insertion/Deletion taken O(1) complexity.

Q2- Differentiate betn array & stack.

Array	Stack
(i) collection of related values identified by array index.	linear data structure.

Array	Stack
(ii) The elements in array belong to indexes	Stack are based on LIFO principle
(iii) Insertion operation is adding new element.	Insertion operation is push.
(iv) Array has fixed size	Stack has static size.

Q3 - An array arr[5][5] is stored in the memory with each element occupying 4 bytes of space. Assuming the base address of arr to be 1000. Compute the address of arr[2][4] when array is stored (i) Row - wise, (ii) column-wise

$arr[i][j] = B + W(C(I - l_i) + (J - l_c))$

(i) $arr[2][4] = 1000 + 4(5(2-0) + (4-0))$

(row wise) $= 1056.$

(ii) $arr[2][4] = B + W((I - l_i) + 2(C - l_j))$

(column wise)

$= 1000 + 4((2-0) + 5(4-0))$

$= 1000 + 88 = 1088.$

Q4 - WAP to find sum of element below the main diagonal & sum of elements above main diagonal.

```
#include <iostream>
using namespace std;
int main(){
int s1=0, s2=0;
~~for(int i=0; i~~
int m, n;
cout<<"Enter row & column of matrix:";
cin>>m>>n; arr[m][n];
cout<<"Enter matrix elements:";
for(int i=0; i<m; i++){
    for(int j=0; j<n; j++){
        cin>>arr[i][j];
    }
}
for(int i=0; i<m; i++){
    for(int j=0; j<n; j++){
        if(i>j){
            s1+=arr[i][j];
        }
        else if(i<j){
            s2+=arr[i][j];
        }
    }
}
}
```

```
cout<<"Sum of elements above main diagonal
  is: "<<S1<<endl;
cout<<"Sum of elements below main
  diagonal is: "<<S2<<endl;

return 0;

}
```

Q5- WAP to add two polynomials.

```
#include <iostream>
using namespace std;
int main(){
  int d;
  cout<<"Enter highest degree of polynomial:";
  cin>>d;
 int a[d+1], b[d+1];
 cout<<"Enter the coefficient of x^i of poly:";
for(int i=0; i<=d; i++){
  cout<<"x^"<<i<<" : ";
cin>>a[i];
```

```
cout << "Enter coefficient of 2nd poly :";
for(int i=0; i<=d; i++){
    cout << "x^" << n-i ;<< ": ";
    cin >> b[i];

} cout << "poly 1 + poly 2 = ";
for(int i=0; i<=d; i++){
    cout << a[i]+b[i]<< "x^" << n-i <<
        " + ";

}
cout << a[d] + b[d]<< endl;

return 0;

}
```

Output :-

Enter highest degree of polynomial : 4
Enter the coefficient for 1st poly :
x^4 : 20
x^3 : 10
x^2 : 5
x^1 : 0
x^0 : 3

Enter the coefficient of 2nd poly:

x^4 : 0

x^3 : 3

x^2 : 1

x^1 : 5

x^0 : 0

poly1 + poly2 = 20 x^4 + 13 x^3 + 6 x^2 + 5 x^1 + 3

Q6- WAP to merge two sorted arrays & eliminate the duplicates.

A6 -

```
#include <iostream>
using namespace std;
class ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) :
        val(x), next(next) { }
};
```

```
class Solution {
public:
ListNode* mergeTwoSorted(ListNode* l1,
    ListNode* l2) {
    if(l1 == NULL)
        return l2;
elsif (l2 == NULL)
        return l1;

ListNode head;
ListNode *it = &head, *it1 = l1, *it2 = l2;
int small =0;
while (it1 && it2) {
    if (it->val == it1->val) {
    it1 = it1->next;
    continue;
    }
    else if (it->val == it2->val) {
        it2 = it2->next;
        continue;
    }
    else if (it2->val < it1->val) {
    small = it1->val;
    it2 = it2->next;
    }
```

```
else {
    small = it1->val;
    it1 = it1->next;
}
ListNode* temp = new ListNode(small);
it->next = temp;
it = temp;
}
while (it1) {
    ListNode* temp = new ListNode(it1->val);
    it1->next = temp;
    it = temp; it1 = it1->next;
}
while (it2) {
    ListNode* temp = new ListNode(it2->val);
    it2->next = temp;
    it = temp; it2 = it2->next;
}
return head.next;
}
```

~~void printlist(~~

```
ListNode* createlist(const vector<int>&v){
ListNode* head = nullptr; *it = head;
for(int* i : v){
    ListNode* newNode = new ListNode(i);
    if(!head){
      head = newNode;
      it = newNode;
    }
  else{
    it->nxt = newNode;
    it = it->nxt;
  }
return head;
}
void printlist(ListNode* head){
    while(head){
        cout << head->val << " ";
        if(head->nxt) cout << "->";
        head = head->nxt;
}
cout<<endl;}};
```

```
int main() {
Solution S;
int m, n; vector<int> list1, list2;
cout<<" Enter size of list1 = "; cin>>m;
cout<<" Enter size of list2 = "; cin>>n;
cout<<" Enter elements of list1 = ";
for(int i = 0; i<m; i++){
    [illegible]
    cin>>list[i];
}
cout<<" Enter elements of list2 = ";
for(int i = 0; i<n; i++){
    cin>>list2[i];
}
ListNode* l1 = S.createlist(list1);
ListNode* l2 = S.createlist(list2);
cout<<" The merged sorted list = ";
ListNode* head = S.mergeTwoSorted(l1, l2);
printlist(head);
return 0;
}
```

Output

Enter size of list1 : 5

Enter size of list2 : 10

Enter elements of list1 : 3 6 9 10 14

Enter elements of list2 : 1 3 5 6 8 11 12 18 20 21

The merged sorted list is :

1 → 3 → 5 → 6 → 8 → 9 → 10 → 11 → 12 → 14 → 18 → 20 → 21

Q7- Reverse the elements in a circular linked list.

Ans -

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

class ListNode {
    int val;
    ListNode * next;
    ListNode (int x) : val(x), next(nullptr) {}
};
```

```
class solution {

    public:
    void insertNode(ListNode* &head,
    int value) {
     ListNode* newNode = new ListNode(val
     if (!head) {
       head = newNode;
       head->next = head;
     }
     else {
       ListNode* temp = head;
      while (temp->next != head) {
         temp = temp->next;
      }
      temp->next = newNode;
     newNode->next = head;
    }
}
```

```
ListNode* reverseCircularList(ListNode* head){
    if(!head || head->next==head){
        return head;
    }
    ListNode* prev=nullptr, *curr=head,
            *next=nullptr, *lastnode=head;
    do{
        next=curr->next;
        curr->next=prev;
        prev=curr;
        curr=next;
    }
    while(curr!=head);
        lastNode->next=prev;
        return prev;
}
void printList(ListNode* head){
    if(!head) return;
    ListNode* temp=head;
    do{ cout<<temp->val<<"->";
        temp=temp->next;
    } while(temp!=head); cout<<endl;}
```

```
int main() {
    int n, x;
    Solution S;
    cout << "enter no. of elements: "; cin >> n;

    ListNode* head = nullptr;

    for (int i = 0; i < n; i++) {
        cin >> x;
        S.insertNode(head, x);
    }

    cout << "original circular linked list: ";
    printList(head);

    head = reverseCircularList(head);

    cout << "Reversed circular linked list: ";
    printList(head);

    return 0;
}
```

Output :-

Enter no. of elements : 5

10 20 30 40 50

Original circular linked list :

10 → 20 → 30 → 40 → 50

Reverse circular linked list :

50 → 40 → 30 → 20 → 10

Q- Delete a

A- #include <iostream>

using namespace std;

class ListNode {

int val;

ListNode* next;

ListNode(int x) : val(x), next(nullptr) {}

};

class Solution {

public:

```
void insertNode(int value, ListNode* head);
{
  ListNode* newNode = new ListNode(value);
  if (!head){
     head = newNode;
  }
  else{
     ListNode* it = head;
     while(it){
        it = it->next;
     }
     it->next = newNode;
  }
}

void deleteNode(ListNode* head, int value){
  if(!head) return;
  if(head->val == value){
     ListNode* temp = head;
     head = head->next;
     delete temp; return;
```

```
ListNode * curr = head, * prev = nullptr;
while (curr && curr->val != value) {
    prev = curr;
    curr = curr->next;
}
if (!curr) return;
prev->next = curr->next;
delete curr;
}
void printList (ListNode * head) {
    ListNode * it = head;
    while (it) {
        cout << it->val << "->";
        it = it->next;
    }
    cout << endl;
}
};
```

```
int main(){
    Solution s;
    ListNode* head = nullptr;
    int n;n;
    cout<<"Enter the size of linked list : ";
    cin>>n;
    cout<<"Enter elements of the linked list:";
    for(int i=0;i<n;i++){
        cin>>x;
        s.insertNode(x, head);
    }
    cout<<"Enter the value to be deleted : ";
    cin>>x;
    deleteNode(head, x);
    cout<<"The updated linked list is: ";
    printList(head);
    return 0;
}
```

Output

Enter the size of linked list : 5
Enter elements of the linked list : 6 10 13
15 20

Enter the value to be deleted : 15

The updated linked list is :
6 → 10 → 13 → 20