4. Implementation of Stack Application.

*********To implement Postfix evaluation**********

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

// The main function that returns value
// of a given postfix expression
int evaluatePostfix(char* exp)
{
        // Create a stack of capacity equal to expression size
        stack<int> st;
        int i;

        // Scan all characters one by one
        for (i = 0; exp[i]; ++i) {

                // If the character is blank space then continue
                if (exp[i] == ' ')
                        continue;

                // If the scanned character is an
                // operand (number here),extract the full number
                // Push it to the stack.
                else if (isdigit(exp[i])) {
                        int num = 0;

                        // Extract full number
                        while (isdigit(exp[i])) {
                                num = num * 10 + (int)(exp[i] - '0');
                                i++;
                        }
                        i--;

                        // Push the element in the stack
                        st.push(num);
                }

                // If the scanned character is an operator, pop two
                // elements from stack apply the operator
                else {
                        int val1 = st.top();
                        st.pop();
                        int val2 = st.top();
                        st.pop();

                        switch (exp[i]) {
                        case '+':
```

```
                                st.push(val2 + val1);
                                break;
                        case '-':
                                st.push(val2 - val1);
                                break;
                        case '*':
                                st.push(val2 * val1);
                                break;
                        case '/':
                                st.push(val2 / val1);
                                break;
                        }
                }
        }
        return st.top();
}

int main()
{
        char exp[] = "100 200 + 2 / 5 * 7 +";

        // Function call
        cout << evaluatePostfix(exp);
        return 0;
}
```

*********To implement Balancing of Parenthesis***********

```
#include <iostream>
#include <stack>

using namespace std;

bool isValid(string s) {
    stack<char> st;
    for (char ch : s) {
        if (ch == '(' || ch == '{' || ch == '[') {
            st.push(ch);
        } else {
            if (st.empty() || (st.top() != '(' && ch == ')') ||
                (st.top() != '{' && ch == '}' ) ||
                (st.top() != '[' && ch == ']')) {
                return false;
            }
            st.pop();
        }
    }
    return st.empty();
```

```
}

int main() {
    string s="{()[]}";

    if (isValid(s)) {
        cout << "true" << endl;
    } else {
        cout << "false" << endl;
    }

    return 0;
}
```

5. Implement all different types of queues.

*********To implement Circular Queue**********

```
#include<iostream>
# define max 4
using namespace std;
class CircularQ
{
        public:
        int cq[max];
        int front, rear;
        CircularQ();
        void enqueue();
        void dequeue();
        void display();
};
CircularQ::CircularQ()
{
        front=rear=-1;
}
void CircularQ::enqueue()
{
        int num;
        //checking overflow
        if(front==(rear+1)%max)//user if(front==0 && rear==max)
        {
                cout<<"Queue is full\n";
                return;
        }
        else
        {
                cout<<"enter number::";
                cin>>num;
```

```
                //queue is empty
                if(front==-1)
                rear = front=0;
                else
                rear=(rear+1)%max;
        }
cq[rear]=num;
cout<<num<<"is inserted..";
}
void CircularQ::dequeue()
{
        int num;
        if(front==-1)
        cout<<"Queue is empty";
        else
        {
                num=cq[front];
                cout<<"deleted item is::"<<num;
                if(front==rear)
                front=rear=-1;
                else
                front=(front+1)%max;
        }
        }
        void CircularQ::display()
        {
                int i;
                if(front==-1)
                cout<<"Queue is empty";
                else
                {
                        cout<<"\n queue elements are:: \n";
                        for(i=front;i<=rear;i++)
                        cout<<cq[i]<<"\t";
                }
                if(front>rear)
                {
                        for(i=front;i<max;i++)
                        cout<<cq[i]<<"t";
                        for(i=0;i<=rear;i++)
                        cout<<cq[i]<<"\t";
                }
        }
                int main()
                {
                        CircularQ c;
                        int choice;
                        while(1)
                        {
                                cout<<"\n ---- Circular Queue Operation---\n";
```

```
                                cout<<"\n 1. Enqueue \n 2. Dequeue \n 3.Display\n
4. Exit";
                                cout<<"\n Enter the Choice::";
                                cin>>choice;
                                switch(choice)
                                {
                                        case 1:
                                                c.enqueue();
                                                break;
                                        case 2:
                                                c.dequeue();
                                                break;
                                        case 3:
                                                c.display();
                                                break;
                                        case 4:
                                            exit(0);
                                        default:
                                                cout<<"Wrong Choice";
                                }
                        }
                        return 0;
                }
```

6. Demonstrate application of queue.

*********To implement Priority Queue**********

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

struct node
{
        int priority;
        int info;
        struct node *link;
};

class Priority_Queue
{
    private:
        node *front;
    public:
        Priority_Queue()
        {
            front = NULL;
        }
```

```
void insert(int item, int priority)
{
    node *tmp, *q;
    tmp = new node;
    tmp->info = item;
    tmp->priority = priority;
    if (front == NULL || priority < front->priority)
    {
        tmp->link = front;
        front = tmp;
    }
    else
    {
        q = front;
        while (q->link != NULL && q->link->priority <= priority)
            q=q->link;
        tmp->link = q->link;
        q->link = tmp;
    }
}

void del()
{
    node *tmp;
    if(front == NULL)
        cout<<"Queue is underflow\n";
    else
    {
        tmp = front;
        cout<<"Deleted item is: "<<tmp->info<<endl;
        front = front->link;
        free(tmp);
    }
}

void display()
{
    node *ptr;
    ptr = front;
    if (front == NULL)
        cout<<"Queue is empty\n";
    else
    {   cout<<"Queue is :\n";
        cout<<"Priority       Item\n";
        while(ptr != NULL)
        {
            cout<<ptr->priority<<"                 "<<ptr->info<<endl;
            ptr = ptr->link;
        }
```

```
            }
        }
};

int main()
{
    int choice, item, priority;
    Priority_Queue c;
    do
    {
        cout<<"1.Insert\n";
        cout<<"2.Delete\n";
        cout<<"3.Display\n";
        cout<<"4.Quit\n";
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            cout<<"Enter the element in queue : ";
            cin>>item;
            cout<<"Enter its priority : ";
            cin>>priority;
            c.insert(item, priority);
            break;
        case 2:
            c.del();
            break;
        case 3:
            c.display();
            break;
        case 4:
            break;
        default :
            cout<<"Wrong choice\n";
        }
    }
    while(choice != 4);
    return 0;
}
```

7. Implementation of all types of linked list.

*********To implement Single Linked List***********

```
#include <iostream>
```

```
using namespace std;

// Node struct to represent elements in the linked list
struct Node {
    int data;
    Node* next;

    Node(int data) : data(data), next(nullptr) {}
};

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Insert a new node at the end of the linked list
    void append(int data) {
        Node* newNode = new Node(data);

        if (head == nullptr) {
            head = newNode;
        } else {
            Node* current = head;
            while (current->next != nullptr) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    // Display the linked list
    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " -> ";
            current = current->next;
        }
        cout << "nullptr" << endl;
    }

    // Destructor to free memory
    ~LinkedList() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
```

```
        }
    }
};

int main() {
    LinkedList myList;

    int n, data;
    cout << "Enter the number of elements: ";
    cin >> n;

    for (int i = 0; i < n; i++) {
        cout << "Enter element " << i + 1 << ": ";
        cin >> data;
        myList.append(data);
    }

    cout << "Linked List: ";
    myList.display();

    return 0;
}
```

**********To implement Double Linked List**********

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

class Node
 {
public:
    int data;
    Node* next;
    Node* prev;

    Node()
    {
        data = 0;
        next = NULL;
        prev = NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->next = NULL;
        this->prev = NULL;
    }
```

```
};

class Linkedlist
{
    Node* head;
public:
    Linkedlist()
        {
        head = NULL;
        }

    void insertNode(int data)
    {
    Node* newNode = new Node(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    temp->next = newNode;
    newNode->prev=temp;
        }

    void printList()
        {
    Node* temp = head;

    if (head == NULL) {
        cout << "List empty" << endl;
        return;
    }

    while (temp != NULL) {
        cout << temp->data << "<-->";
        temp = temp->next;
    }
    cout<<"null";
 }
        void deleteNode(int index)
        {
        Node* temp = head;
        if(head==NULL)
        {
                cout<<"List is empty";
                return;
        }
```

```
        if(temp->next==NULL)
        {
                head=NULL;
                return;
        }
        Node* previous;
        Node* nextNode;
        while(index!=0)
        {
                previous=temp;
                temp=temp->next;
                nextNode=temp->next;
                index--;
        }
        previous->next=temp->next;
        nextNode->prev=previous;
        free(temp);
        return;
        }
};

int main()
{
        int ch,val,indx;
        Linkedlist ll;
        cout<<"\n1.Insert\n2.Delete\n3.DisplayList\n4.Exit";
        while(1)
        {
                cout<<"\nEnter your choice: ";
                cin>>ch;
                switch(ch)
                {
                        case 1: int size,val;
                                        cout<<"Enter size of the LinkedList ";
                                        cin>>size;
                                        for(int i=0;i<size;i++)
                                          {
                                         cout<<"\nInsert element-"<<i+1<<": ";
                                          cin>>val;
                                          ll.insertNode(val);
                                          }
                        break;
                        case 2:
cout<<"\nEnter the index of node to delete: ";
                                  cin>>indx;
                                  ll.deleteNode(indx);
                                  cout<<"\nElements after deletion: ";
                                  ll.printList();
                                 break;
                        case 3:ll.printList();
```

```
                        break;
                case 4:
                        exit(0);
                        break;
                default: cout<<"Invalid Input";
            }
        }
}
```

**********To implement Circular Linked List***********

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

struct Node
{
   int data;
   struct Node *next;
};

struct Node *insertInEmpty(struct Node *last, int new_data)
{
   if (last != NULL)
   return last;
   struct Node *temp = new Node;
   temp -> data = new_data;
   last = temp;
   last->next = last;
   return last;
}

struct Node *insertAtBegin(struct Node *last, int new_data)
{
   //if list is empty then add the node by calling insertInEmpty
   if (last == NULL)
   return insertInEmpty(last, new_data);

   //else create a new node
   struct Node *temp = new Node;

   //set new data to node
   temp -> data = new_data;
   temp -> next = last -> next;
   last -> next = temp;

   return last;
}
```

```
struct Node *insertAtEnd(struct Node *last, int new_data)
{
   if (last == NULL)
   return insertInEmpty(last, new_data);
   struct Node *temp = new Node;
   temp -> data = new_data;
   temp -> next = last -> next;
   last -> next = temp;
   last = temp;
   return last;
}



void traverseList(struct Node *last) {
   struct Node *p;

   if (last == NULL) {
      cout << "Circular linked List is empty." << endl;
      return;
      }
p = last -> next;

do {
      cout << p -> data << "==>";
      p = p -> next;
      } while(p != last->next);
   if(p == last->next)
   cout<<p->data;
   cout<<"\n\n";
   }

void deleteNode(Node** head, int key)
{
   // If linked list is empty retun
   if (*head == NULL)
   return;
   if((*head)->data==key && (*head)->next==*head) {
      free(*head);
      *head=NULL;
      }
Node *last=*head,*d;

if((*head)->data==key) {
   while(last->next!=*head)
   last=last->next;
   last->next=(*head)->next;
   free(*head);
   *head=last->next;
```

```
   }

while(last->next!=*head&&last->next->data!=key)
{
   last=last->next;
}
if(last->next->data==key) {
      d=last->next;
      last->next=d->next;
      cout<<"The node with data "<<key<<" deleted from the list"<<endl;
      free(d);
      cout<<endl;
      cout<<"Circular linked list after deleting "<<key<<" is as follows:"<<endl;
      traverseList(last);
      }
   else
   cout<<"The node with data "<< key << " not found in the list"<<endl;
   }

int main()
{
        int size,val,ch;
   struct Node *last = NULL;
   cout<<"=======MENU======";
   cout<<"\n1.Insert In EmptyList\n2.Insert at Begin\n3.Insert at
End\n4.Display\n5.Delete\n6.Exit";
   while(1)
   {
        cout<<"\nEnter your choice: ";
        cin>>ch;
        switch(ch)
        {
        case 1: cout<<"\nInsert element in EmptyList: ";
                        cin>>val;
                        last=insertInEmpty(last,val);
           break;
        case 2:cout<<"\nInsert element At begin: ";
                                cin>>val;
                                last=insertAtBegin(last,val);
        break;
        case 3:cout<<"\nInsert element At End: ";
                                cin>>val;
                                last=insertAtBegin(last,val);
        break;
        case 4:cout<<"\nThe circular linked list created is as follows:"<<endl;
                        traverseList(last);
        break;
        case 5:cout<<"\nEnter an element to delete: ";
                        cin>>val;
                        deleteNode(&last,val);
```

```
        break;
        case 6: exit(0);
        default:
        cout<<"Invalid Choice";
        }
   }
   return 0;
}
```

8. Demonstrate application of linked list.

*********To implement Polynomial Addition**********

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

// Node structure containing power
// and coefficient of variable
struct Node
{
        int coeff;
        int pow;
        struct Node* next;
};

// Function to create new node
void create_node(int x, int y,
                                struct Node** temp)
{
        struct Node *r, *z;
        z = *temp;
        if (z == NULL)
        {
                r = (struct Node*)malloc(sizeof(struct Node));
                r->coeff = x;
                r->pow = y;
                *temp = r;
                r->next = (struct Node*)malloc(sizeof(struct Node));
                r = r->next;
                r->next = NULL;
        }
        else
        {
                r->coeff = x;
                r->pow = y;
                r->next = (struct Node*)malloc(sizeof(struct Node));
                r = r->next;
                r->next = NULL;
        }
}
```

```
// Function Adding two polynomial
// numbers
void polyadd(struct Node* poly1,
                        struct Node* poly2,
                        struct Node* poly)
{
        while (poly1->next &&
                poly2->next)
        {
                // If power of 1st polynomial is greater
                // than 2nd, then store 1st as it is and
                // move its pointer
                if (poly1->pow > poly2->pow)
                {
                        poly->pow = poly1->pow;
                        poly->coeff = poly1->coeff;
                        poly1 = poly1->next;
                }

                // If power of 2nd polynomial is greater
                // than 1st, then store 2nd as it is and
                // move its pointer
                else if (poly1->pow < poly2->pow)
                {
                        poly->pow = poly2->pow;
                        poly->coeff = poly2->coeff;
                        poly2 = poly2->next;
                }

                // If power of both polynomial numbers
                // is same then add their coefficients
                else
                {
                        poly->pow = poly1->pow;
                        poly->coeff = (poly1->coeff +
                                                poly2->coeff);
                        poly1 = poly1->next;
                        poly2 = poly2->next;
                }

                // Dynamically create new node
                poly->next =
                        (struct Node*)malloc(sizeof(struct Node));
                poly = poly->next;
                poly->next = NULL;
        }
        while (poly1->next || poly2->next)
        {
                if (poly1->next)
```

```
                {
                        poly->pow = poly1->pow;
                        poly->coeff = poly1->coeff;
                        poly1 = poly1->next;
                }
                if (poly2->next)
                {
                        poly->pow = poly2->pow;
                        poly->coeff = poly2->coeff;
                        poly2 = poly2->next;
                }
                poly->next =
                        (struct Node*)malloc(sizeof(struct Node));
                poly = poly->next;
                poly->next = NULL;
        }
}

// Display Linked list
void show(struct Node* node)
{
        while (node->next != NULL)
        {
                printf("%dx^%d", node->coeff,
                                node->pow);
                node = node->next;
                if (node->coeff >= 0)
                {
                        if (node->next != NULL)
                                printf("+");
                }
        }
}

// Driver code
int main()
{
        struct Node *poly1 = NULL,
                                *poly2 = NULL,
                                *poly = NULL;

        // Create first list of 5x^2 +
        // 4x^1 + 2x^0
        create_node(5, 2, &poly1);
        create_node(4, 1, &poly1);
        create_node(2, 0, &poly1);

        // Create second list of -5x^1 -
        // 5x^0
        create_node(-5, 1, &poly2);
```

```
        create_node(-5, 0, &poly2);

        printf("1st Number: ");
        show(poly1);

        printf("2nd Number: ");
        show(poly2);

        poly = (struct Node*)malloc(sizeof(struct Node));

        // Function add two polynomial
        // numbers
        polyadd(poly1, poly2, poly);

        // Display resultant List
        printf("Added polynomial: ");
        show(poly);

        return 0;
}
```

*********To implement Sparse Matrix***********

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

int main()
{
        int a[10][10]={{0,0,9},{5,0,8},{7,0,0}};
        int i,j,count=0;
        int row=3,col=3;

        for(i=0;i<row;i++){
                for(j=0;j<col;j++)
                {
                        if(a[i][j]==0)
                        count++;
                }
        }


        cout<<"The matrix is:"<<endl;
        for(i=0;i<row;i++)
        {
                for(j=0;j<col;j++)
                {
                        cout<<a[i][j]<<" ";
                }
                cout<<endl;
```

```
        }



        cout<<"the number of zeros in the matrix are"<<count<<endl;

        if(count>((row*col)/2))
        cout<<"this is not sparse matrix"<<endl;

        else
        cout<<"This is not sparse matrix"<<endl;


        return 0;
}
```