DSA LAB PRACTICAL 2

1) Stack Using LinkedList

```
#include<iostream>
using namespace std;
class stackLinkedList
{
    private:
        struct node
        {
            int data;
            node *ptr;
        }*top;
    public:
        stackLinkedList()
        {
            top = NULL;
        }
        void push(int x)
        {
            node *temp;
            temp = new node;
            temp->data = x;
            temp->ptr = top;
            top = temp;
        }
        void pop()
        {
```

```
            if(top == NULL)
                cout<<"Stack is empty";
            else
            {
                node *temp;
                temp = top;
                top = temp->ptr;
                cout<<"\nThe deleted element in stack is:
"<<temp->data;
                delete temp;
            }
        }
        void display()
        {
            if(top == NULL)
                cout<<"Stack is empty";
            else
            {
                node *temp;
                temp = top;
                cout<<"\nThe elements in stack are: ";
                while(temp != NULL)
                {
                    cout<<temp->data<<"\t";
                    temp = temp->ptr;
                }
            }
        }
};
```

```
int main()
{
    stackLinkedList s;
    int ele,ch;
    do
    {
        cout<<"\nStack operations are:
\n1.Push\n2.Pop\n3.Display\n4.Exit";
        cout<<"\nEnter your choice:";
        cin>>ch;
        switch(ch)
        {
            case 1:
                cout<<"\nEnter an element to insert: ";
                cin>>ele;
                s.push(ele);
                break;
            case 2:
                s.pop();
                break;
            case 3:
                s.display();
                break;
            case 4:
                exit(0);
            default:
                cout<<"Invalid Input";
        }
    } while (true);
```

```
}
```

Output:

Stack operations are: **TABLE**

1.Push

2.Pop

3.Display

4.Exit

Enter your choice:1

Enter an element to insert: 10

TABLE

Enter your choice:1

Enter an element to insert: 20

TABLE

Enter your choice:3

The elements in stack are: 20 10

TABLE

Enter your choice:2

The deleted element in stack is: 20

TABLE

Enter your choice:3

The elements in stack are: 10

TABLE

Enter your choice:4

2) Queue using LinkedList (Rear-Insertion, Front-Deletion)

```
#include <iostream>
```

```
using namespace std;
class QueueList
{
    private:
        struct node
        {
            int data;
            node *ptr;
        } *rear, *front;
    public:
        QueueList()
        {
            rear = front = NULL;
        }
        void insertion(int n)
        {
            node *temp;
            temp = new node;
            temp->data = n;
            temp->ptr = NULL;
            if (front == NULL)
                front = rear = temp;
            else
            {
                rear->ptr = temp;
                rear = rear->ptr;
            }
        }
        void deletion()
```

```
        {
            if (front == NULL)
                cout << "\nQueue is empty";
            else
            {
                node *temp;
                temp = front;
                front = front->ptr;
                cout << "\nDeleted element is: " <<
temp->data;
                delete temp;
            }
        }
        void display()
        {
            if (front == NULL)
                cout << "\nQueue is empty";
            else
            {
                node *temp;
                temp = front;
                cout << "\nElements are: ";
                while(temp != NULL)
                {
                    cout << temp->data<<" ";
                    temp = temp->ptr;
                }
            }
        }
```

```
};
int main()
{
    QueueList l;
    int ele, ch;
    do
    {
        cout << "\nQueue operations are:
\n1)Insertion\n2)Deletion\n3)Display\n4)Exit";
        cout << "\nEnter your choice: ";
        cin >> ch;
        switch (ch)
        {
            case 1:
                cout << "\nEnter element to insert: ";
                cin >> ele;
                l.insertion(ele);
                break;
            case 2:
                l.deletion();
                break;
            case 3:
                l.display();
                break;
            case 4:
                exit(0);
            default:
                cout << "\nInvalid Input";
        }
```

```
    } while (true);
}
```

Output:

Queue operations are: **TABLE**

1)Insertion

2)Deletion

3)Display

4)Exit

Enter your choice: 1

Enter element to insert: 10

TABLE

Enter your choice: 1

Enter element to insert: 20

TABLE

Enter your choice: 1

Enter element to insert: 30

TABLE

Enter your choice: 3

Elements are: 10 20 30

TABLE

Enter your choice: 2

Deleted element is: 10

TABLE

Enter your choice: 3

Elements are: 20 30

TABLE

Enter your choice: 4

3) Binary Search Tree

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

class BST
{
private:
    // Node class for the Binary Search Tree
    class Node
    {
    public:
        int val;
        Node *left;
        Node *right;

        Node(int v)
        {
            val = v;
            left = nullptr;
            right = nullptr;
        }
    };

    Node *root; // Pointer to the root node of the BST
```

```
    // Helper function for inserting a new node with value
'val'
    Node *insert(Node *node, int val)
    {
        if (node == nullptr)
        {
            return new Node(val);
        }

        if (val < node->val)
        {
            node->left = insert(node->left, val);
        }
        else if (val > node->val)
        {
            node->right = insert(node->right, val);
        }
        return node;
    }

    // Helper function for searching a node with value 'val'
    bool search(Node *node, int val)
    {
        if (node == nullptr)
        {
            return false;
        }

        if (val == node->val)
```

```
        {
            return true;
        }
        else if (val < node->val)
        {
            return search(node->left, val);
        }
        else
        {
            return search(node->right, val);
        }
    }

    // Helper function to find the minimum node in a BST
    Node *findMin(Node *node)
    {
        while (node->left != nullptr)
        {
            node = node->left;
        }
        return node;
    }

    // Helper function to delete a node with value 'val'
from the BST
    Node *deleteNode(Node *node, int val)
    {
        if (node == nullptr)
        {
```

```
    return nullptr;
}

if (val < node->val)
{
    node->left = deleteNode(node->left, val);
}
else if (val > node->val)
{
    node->right = deleteNode(node->right, val);
}
else
{
    // Node with the key to be deleted is found

    // Case 1: Node has only one child or no child
    if (node->left == nullptr)
    {
        Node *temp = node->right;
        delete node;
        return temp;
    }
    else if (node->right == nullptr)
    {
        Node *temp = node->left;
        delete node;
        return temp;
    }
```

```
            // Case 2: Node has two children
            Node *temp = findMin(node->right);
            node->val = temp->val;
            node->right = deleteNode(node->right,
temp->val);
        }

        return node;
    }

    // Helper function for in-order traversal of the BST
    void inOrderTraversal(Node *node)
    {
        if (node != nullptr)
        {
            inOrderTraversal(node->left);
            cout << node->val << " ";
            inOrderTraversal(node->right);
        }
    }
public:
    BST()
    {
        root = nullptr;
    }

    // Function to insert a new node with value 'val' into
the BST
    void insert(int val)
```

```
    {
        root = insert(root, val);
    }

    // Function to search for a node with value 'val' in the
BST
    bool search(int val)
    {
        return search(root, val);
    }

    // Function to delete a node with value 'val' from the
BST
    void remove(int val)
    {
        root = deleteNode(root, val);
    }

    // Function to print the in-order traversal of the BST
    void printInOrder()
    {
        inOrderTraversal(root);
        cout << endl;
    }
};

int main()
{
    BST bst;
```

```
    int choice;
    int key;

    while (true)
    {
        // Display the menu for Binary Search Tree
operations
        cout << "Binary Search Tree Operations:" << endl;
        cout << "1. Insert a node" << endl;
        cout << "2. Search for a key" << endl;
        cout << "3. Delete a node" << endl;
        cout << "4. Print in-order traversal" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter the value to insert: ";
            cin >> key;
            bst.insert(key);
            cout << "Value " << key << " has been inserted
into the BST." << endl;
            break;

        case 2:
            cout << "Enter the key to search: ";
```

```
            cin >> key;
            if (bst.search(key))
                cout << "Key " << key << " is found in the
BST." << endl;
            else
                cout << "Key " << key << " is not found in
the BST." << endl;
            break;

        case 3:
            cout << "Enter the key to delete: ";
            cin >> key;
            bst.remove(key);
            cout << "Key " << key << " has been deleted from
the BST." << endl;
            break;
        case 4:
            cout << "In-order traversal of the BST: ";
            bst.printInOrder();
            break;

        case 5:
            cout << "Exiting program." << endl;
            return 0;

        default:
            cout << "Invalid choice. Please try again." <<
endl;
        }
```

```
    return 0;
}
```

Output:

Binary Search Tree Operations: **TABLE**

1. Insert a node
2. Search for a key
3. Delete a node
4. Print in-order traversal
5. Exit

Enter your choice: 1

Enter the value to insert: 4

Value 4 has been inserted into the BST.

TABLE

Enter your choice: 1

Enter the value to insert: 2

Value 2 has been inserted into the BST.

TABLE

Enter your choice: 1

Enter the value to insert: 3

Value 3 has been inserted into the BST.

TABLE

Enter your choice: 4

In-order traversal of the BST: 2 3 4

TABLE

Enter your choice: 2

Enter the key to search: 3

Key 3 is found in the BST.

TABLE

Enter your choice: 3

Enter the key to delete: 3

Key 3 has been deleted from the BST.

TABLE

Enter your choice: 4

In-order traversal of the BST: 2 4

TABLE

Enter your choice: 5

Exiting program.

4) Quick Sort

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

void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Function to partition the array and return the pivot
index
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = low - 1;
```

```
    for (int j = low; j < high; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i + 1], arr[high]);
    return i + 1;
}
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pivotIndex = partition(arr, low, high);

        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}

int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
```

```
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output:

Enter the number of elements: 6

Enter 6 elements: 10 5 9 0 -9 -2

Sorted array: -9 -2 0 5 9 10

5) Insertion Sort

```
#include <iostream>
using namespace std;
void insertionSort(int arr[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int key = arr[i];
        int j = i - 1;
```

```
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    insertionSort(arr, n);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output:

Enter the number of elements: 8

Enter 8 elements: 15 4 0 2 -8 9 -10 5

Sorted array: -10 -8 0 2 4 5 9 15

6) Selection Sort

```
#include <iostream>
using namespace std;
void selectionSort(int arr[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        int minIndex = i;

        // Find the minimum element in the unsorted part of
the array
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[minIndex])
            {
                minIndex = j;
            }
        }
        // Swap the minimum element with the first element
of the unsorted part
        if (minIndex != i)
        {
            int temp = arr[i];
            arr[i] = arr[minIndex];
```

```
            arr[minIndex] = temp;
        }
    }
}
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    selectionSort(arr, n);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output:

Enter the number of elements: 8

Enter 8 elements: 5 9 -2 8 0 7 -8 6

Sorted array: -8 -2 0 5 6 7 8 9

7) Merge Sort

```
#include<iostream>
using namespace std;
// Function to merge two sorted arrays
void mergeArrays(int a[], int n, int b[], int m, int c[])
{
    int i = 0, j = 0, k = 0;

    // Merge the two sorted arrays a and b into c
    while ((i < n) && (j < m))
    {
        if (a[i] <= b[j])
        {
            c[k] = a[i];
            i++;
            k++;
        }
        else if (b[j] < a[i])
        {
            c[k] = b[j];
            j++;
            k++;
        }
    }

    // If there are remaining elements in array a, add them
to c
    while (i < n) {
        c[k] = a[i];
```

```
        k++;
        i++;
    }

    // If there are remaining elements in array b, add them
to c
    while (j < m) {
        c[k] = b[j];
        k++;
        j++;
    }
}
int main()
{
    int a[20], b[20], c[40], n, m, i, j, p;

    cout << "Enter the number of elements in the first
array: ";
    cin >> n;
    cout << "Enter " << n << " numbers in sorted order: ";
    for (i = 0; i < n; i++) {
        cin >> a[i];
    }
    cout << "Enter the number of elements in the second
array: ";
    cin >> m;
    cout << "Enter " << m << " numbers in sorted order: ";
    for (j = 0; j < m; j++) {
        cin >> b[j];
```

```
    }
    p = m + n; // Total number of elements in the merged
array
    // Call the mergeArrays function to merge arrays a and b
into c
    mergeArrays(a, n, b, m, c);
    cout << "Merged Array: ";
    for (i = 0; i < p; i++) {
        cout << c[i] << "\t";
    }
    return 0;
}
```

Output:

Enter the number of elements in the first array: 5
Enter 5 numbers in sorted order: -19 -2 0 8 14
Enter the number of elements in the second array: 5
Enter 5 numbers in sorted order: -25 -10 0 15 69
Merged Array: -25 -19 -10 -2 0 0 8 14 15 69

8) Bubble Sort

```
#include <iostream>
using namespace std;
void bubbleSort(int arr[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        bool swapped = false;
```

```
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap the elements if they are in the
wrong order
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped in the inner
loop, the array is already sorted
        if (!swapped) {
            break;
        }
    }
}
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
```

```
        cin >> arr[i];
    }
    bubbleSort(arr, n);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output:

Enter the number of elements: 5

Enter 5 elements: 85 99 -8 -55 0

Sorted array: -55 -8 0 85 99

9) Heap Sort

```
#include <iostream>
using namespace std;
void swap(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Function to heapify a subtree rooted with node i which is
an index in the array
void heapify(int arr[], int n, int i) {
```

```
    int largest = i; // Initialize the largest as root
    int left = 2 * i + 1; // Left child
    int right = 2 * i + 2; // Right child

    // If left child is larger than root
    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }

    // If right child is larger than largest so far
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// Function to perform Heap Sort on an array
void heapSort(int arr[], int n) {
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
```

```
    // Extract elements one by one from the heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to the end
        swap(arr[0], arr[i]);

        // Call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    heapSort(arr, n);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output:

Enter the number of elements: 8

Enter 8 elements: 15 99 -8 2 -9 1 5 0

Sorted array: -9 -8 0 1 2 5 15 99

10) Balance Paranthesis

```
#include <iostream>
using namespace std;
#define MAX 100

class Stack
{
    int top;

public:
    char a[MAX]; // Maximum size of Stack

    Stack()
    {
        top = -1;
    }
    bool push(char x)
    {
        if (top >= (MAX - 1))
        {
            cout << "Stack Overflow";
            return false;
        }
        else
        {
            a[++top] = x;
```

```
        return true;
    }
}
char pop()
{
    if (top < 0)
    {
        cout << "Stack Underflow";
        return 0;
    }
    else
    {
        int x = a[top--];
        return x;
    }
}
char peek()
{
    if (top < 0)
    {
        cout << "Stack is Empty";
        return 0;
    }
    else
    {
        char x = a[top];
        return x;
    }
}
```

```
    bool isEmpty()
    {
        return (top < 0);
    }
};
int main()
{
    Stack s;
    char exp[50];
    cout << "Enter Expression :";
    cin >> exp;
    int i = 0;
    int c = 0;
    while (exp[i] != '\0')
    {

        if (exp[i] == '(' ||exp[i] == '[' ||exp[i] == '{' )
        {
            s.push(exp[i]);
            c++;
        }
        if (exp[i] == ')' ||exp[i] == ']' ||exp[i] == '}' )
        {
            if (s.isEmpty())
            {
                cout << "More number of closing parenthesis
!" << endl;
                return 0;
            }
```

```
            s.pop();
            c--;
        }
        i++;
    }
    if (c == 0)
        cout << "Valid Expression" << endl;
    else
        cout << "Invalid Expression"<<endl;
    return 0;
}
```

Output:

Enter Expression :(9-2)(a/b){r*s}[2/3+{7-5}]

Valid Expression

11) Sequential / Linear Search

```
#include <iostream>
using namespace std;
int sequentialSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;          }
    }
    return -1;
}
int main() {
    int n, key;
```

```
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Enter the key to search: ";
    cin >> key;
    int index = sequentialSearch(arr, n, key);
    if (index != -1) {
        cout << "Key " << key << " found at index " << index
<< endl;
    } else {
        cout << "Key " << key << " not found in the array."
<< endl;
    }
    return 0;
}
```

Output:

Enter the number of elements: 5

Enter 5 elements: 4 5 8 9 3

Enter the key to search: 5

Key 5 found at index 1

12) Binary Search

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

```
int binarySearch(int arr[], int n, int key)
{
    int left = 0;
    int right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}

int main() {
    int n, key;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " sorted elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Enter the key to search: ";
    cin >> key;
    int index = binarySearch(arr, n, key);
```

```
    if (index != -1) {
        cout << "Key " << key << " found at index " << index
<< endl;
    } else {
        cout << "Key " << key << " not found in the array."
<< endl;
    }
    return 0;
}
```

Output:

Enter the number of elements: 6

Enter 6 sorted elements: 5 9 7 12 21 35

Enter the key to search: 21

Key 21 found at index 4

13) Sparse Matrix Conversion

```
#include <iostream>
using namespace std;
int main()
{
    // Get the number of rows and columns from the user
    int rows, cols;
    cout << "Enter the number of rows in the matrix: ";
    cin >> rows;
    cout << "Enter the number of columns in the matrix: ";
    cin >> cols;

    int sparseMatrix [rows] [cols];
```

```
cout << "Enter the elements of the matrix:" << endl;
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < cols; j++)
    {
        cout<<"Enter "<<i<<" "<<j<<" Element: ";
        cin >> sparseMatrix[i][j];
    }
}
int size = 0;
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < cols; j++)
    {
        if (sparseMatrix[i][j] != 0)
            size++;
    }
}
int compactMatrix[3][size];
// Making of new matrix
int k = 0;
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < cols; j++)
    {
        if (sparseMatrix[i][j] != 0)
        {
            compactMatrix[0][k] = i;
```

```
                compactMatrix[1][k] = j;
                compactMatrix[2][k] = sparseMatrix[i][j];
                k++;
            }
        }
    }
    // Display the compact matrix with row and column labels
    cout << "Compact Matrix:" << endl;
    cout << "Row\tColumn\tValue" << endl;
    for (int i = 0; i < size; i++)
    {
        cout << compactMatrix[0][i] << "\t" <<
compactMatrix[1][i] << "\t" << compactMatrix[2][i] << endl;
    }
    return 0;
}
```

Output:

Enter the number of rows in the matrix: 3

Enter the number of columns in the matrix: 4

Enter the elements of the matrix:

Enter 0 0 Element: 1

Enter 0 1 Element: 0

Enter 0 2 Element: 5

Enter 0 3 Element: 0

Enter 1 0 Element: 6

Enter 1 1 Element: 2

Enter 1 2 Element: 0

Enter 1 3 Element: 0

Enter 2 0 Element: 0

Enter 2 1 Element: 8

Enter 2 2 Element: 1

Enter 2 3 Element: 0

Compact Matrix:

Row Column Value

0 0 1

0 2 5

1 0 6

1 1 2

2 1 8

2 2 1

14) Infix to Postfix

```
#include <iostream>
#include <string>
using namespace std;
#define MAX  100

class Stack
{
private:
    int top;
public:
    char arr[MAX];
    Stack()
    {
        top = -1;
    }

    void push(char val) {
        if (top == MAX - 1)
        {
```

```
            cout << "Stack Overflow: Cannot push elements
onto the stack." << endl;
            return;
        }
        top++;
        arr[top] = val;
    }
    void pop()
    {
        if (isEmpty())
        {
            cout << "Stack Underflow: Cannot pop element
from an empty stack." << endl;
            return;
        }
        top--;
    }
    char peek() {
        if (isEmpty())
        {
            cout << "Stack is empty." << endl;
            return -1;
        }
        return arr[top];
    }
    bool isEmpty()
    {
        return (top == -1);
    }
```

```
};

int prec(char c)
{
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*' || c == '%')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}

void infixToPostfix(string s)
{
    Stack st;
    string result;

    for (int i = 0; i < s.length(); i++)
    {
        char c = s[i];

        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <=
'Z'))
            result += c;
        else if (c == '(')
            st.push('(');
        else if (c == ')')
```

```
        {
            while (st.peek() != '(')
            {
                result += st.peek();
                st.pop();
            }
            st.pop();
        }
        //Here the character is operator
        else
        {
            while (!st.isEmpty() && prec(c) <=
prec(st.peek()))
            {
                result += st.peek();
                st.pop();
            }
            st.push(c);
        }
    }
    while (!st.isEmpty())
    {
        result += st.peek();
        st.pop();
    }
    cout << "Postfix Expression: " << result << endl;
}
int main()
{
```

```
    string exp;
    cout << "Enter the Infix Expression: ";
    getline(cin, exp);

    infixToPostfix(exp);

    return 0;
}
```

Output:

Enter the Infix Expression: (A+B)*C-(D-F)*(F+G)

Postfix Expression: AB+C*DF-FG+*-

15) Infix to Prefix

```
#include <iostream>
// #include <cctype> only if error comes include this
#include <string>
#include <algorithm>
using namespace std;
#define MAX 100
class Stack
{
    private:
        char arr[MAX];
        int top;

    public:
        Stack() {
            top = -1;
        }
        void push(char val) {
            if (top == MAX - 1) {
                cout << "Stack Overflow" << endl;
            } else {
                top++;
                arr[top] = val;
            }
        }
        void pop() {
            if (top == -1) {
                cout << "Stack Underflow" << endl;
```

```
            } else {
                top--;
            }
        }
        char peek() {
            if (top == -1) {
                cout << "Stack is empty" << endl;
                return '\0';
            } else {
                return arr[top];
            }
        }
        bool isEmpty() {
            return (top == -1);
        }
};
// Function to check if a character is an operator
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' ||
c == '%' || c == '^');
}

// Function to get the precedence of an operator
int getPrecedence(char c) {
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/' || c == '%')
        return 2;
    else if (c == '+' || c == '-')
```

```
        return 1;
    return 0;
}
string infixToPrefix(string infix) {
    string prefix;
    Stack stack;
    // Reverse the infix expression to process it from right
to left
    reverse(infix.begin(), infix.end());
    for (char c : infix)
    {
        if (isalnum(c))
                    prefix += c;
        else if (c == ')')
            stack.push(c);
        else if (c == '(')
        {
            while (!stack.isEmpty() && stack.peek() != ')')
            {
                prefix += stack.peek();
                stack.pop();
            }
            stack.pop(); // Pop the corresponding '('
        }
        else if (isOperator(c))
        {
            while (!stack.isEmpty() &&
getPrecedence(stack.peek()) >= getPrecedence(c)) {
                prefix += stack.peek();
```

```
                stack.pop();
            }
            stack.push(c);
        }
    }
    while (!stack.isEmpty())
    {
        prefix += stack.peek();
        stack.pop();
    }
    // Reverse the prefix expression to get the final result
    reverse(prefix.begin(), prefix.end());
    return prefix;
}
int main()
{
    string infix, prefix;
    cout << "Enter the infix expression: ";
    cin >> infix;
    prefix = infixToPrefix(infix);
    cout << "Prefix expression: " << prefix << endl;
    return 0;
}
```

Output:

Enter the infix expression: (A+B)*C-(D-F)*(F+G)

Prefix expression: -*+ABC*-DF+FG

16) Postfix Evaluation

```
#include <iostream>
using namespace std;
#define MAX 100
class Stack
{
    private:
        int arr[MAX];
        int top;
    public:
        Stack()
        {
            top = -1;
        }
        void push(int val)
        {
            if (top == MAX - 1)
                cout << "Stack Overflow" << endl;
            else
            {
                top++;
                arr[top] = val;
            }
        }
        void pop()
        {
            if (top == -1)
                cout << "Stack Underflow" << endl;
            else
```

```
            top--;
        }
        int peek()
        {
            if (top == -1)
            {
                cout << "Stack is empty" << endl;
                return -1;
            }
            else
                return arr[top];
        }
        bool isEmpty()
        {
            return (top == -1);
        }
};
int evaluatePostfix(string postfix)
{
    Stack stack;
    for (char c : postfix)
    {
        if (isdigit(c))
        {
            stack.push(c - '0'); // Convert char to integer
and push to the stack
        }
        else
        {
```

```
            int operand2 = stack.peek();
            stack.pop();
            int operand1 = stack.peek();
            stack.pop();

            switch (c) {
                case '+':
                    stack.push(operand1 + operand2);
                    break;
                case '-':
                    stack.push(operand1 - operand2);
                    break;
                case '*':
                    stack.push(operand1 * operand2);
                    break;
                case '/':
                    stack.push(operand1 / operand2);
                    break;
                case '%':
                    stack.push(operand1 % operand2);
                    break;
                default:
                    cout << "Invalid operator" << endl;
                    return -1;
            }
        }
    }
    return stack.peek();
}
```

```
int main() {
    string postfix;
    cout << "Enter the postfix expression: ";
    cin >> postfix;

    int result = evaluatePostfix(postfix);

    if (result != -1) {
        cout << "Result: " << result << endl;
    }
    return 0;
}
```

Output:

Enter the postfix expression: 46+2/5*7+

Result: 32

17) Hashing

```
#include <iostream>
using namespace std;
const int HASH_TABLE_SIZE = 10;
int hashFunction(int key) {
    return key % HASH_TABLE_SIZE;
}
int main() {
    int hashTable[HASH_TABLE_SIZE] = {0};
    int numKeys;
    cout << "Enter the number of keys to insert: ";
    cin >> numKeys;
    cout << "Enter " << numKeys << " keys: ";
    for (int i = 0; i < numKeys; ++i) {
        int key;
```

```
        cin >> key;
        int index = hashFunction(key);
        // Simple collision resolution: Linear probing
        while (hashTable[index] != 0) {
            index = (index + 1) % HASH_TABLE_SIZE; // Move
to the next slot
        }
        hashTable[index] = key;
    }
    // Print the hash table
    for (int i = 0; i < HASH_TABLE_SIZE; ++i) {
        cout << "Index " << i << ": " << hashTable[i] <<
endl;
    }
    return 0;
}
```

Output:

Enter the number of keys to insert: 6

Enter 6 keys: 18 15 38 44 25 6

Index 0: 0

Index 1: 0

Index 2: 0

Index 3: 0

Index 4: 44

Index 5: 15

Index 6: 25

Index 7: 6

Index 8: 18

Index 9: 38

18) Sparse Matrix using LinkedList

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

```
class Node
{
     public:
          int row;
          int col;
          int data;
          Node* next;
};
void createNode(Node** head, int row, int col, int value)
{
     Node* newNode = new Node();
     newNode->row = row;
     newNode->col = col;
     newNode->data = value;
     newNode->next = nullptr;
     if (*head == nullptr)
     {
          *head = newNode;
     }
     else
     {
          Node* temp = *head;
          while (temp->next != nullptr)
          {
               temp = temp->next;
          }
          temp->next = newNode;
     }
}
void printList(Node* head)
{
     Node* temp = head;
     while (temp != nullptr)
     {
          cout << "Row: " << temp->row << ", Column: " <<
     temp->col << ", Value: " << temp->data << endl;
          temp = temp->next;
     }
```

```
}
int main()
{
     int sparseMatrix[4][5] = {
          {0, 0, 3, 0, 4},
          {0, 0, 5, 7, 0},
          {0, 0, 0, 0, 0},
          {0, 2, 6, 0, 0}
     };
     Node* head = nullptr;
     for (int i = 0; i < 4; i++)
     {
          for (int j = 0; j < 5; j++)
          {
               int value = sparseMatrix[i][j];
               if (value != 0)
               {
                    createNode(&head, i, j, value);
               }
          }
     }
     printList(head);
     return 0;
}
```

Output:

```
Row: 0, Column: 2, Value: 3
Row: 0, Column: 4, Value: 4
Row: 1, Column: 2, Value: 5
Row: 1, Column: 3, Value: 7
Row: 3, Column: 1, Value: 2
Row: 3, Column: 2, Value: 6
```