DSA FINAL PRACTICAL

BREADTH FIRST SEARCH

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

const int MAX_NODES = 100;

bool visited[MAX_NODES] = {false};

int queue[MAX_NODES];
int front = 0, rear = -1;

void add(int value)
{
     queue[++rear] = value;
}
int remove()
{
     int frontValue = queue[front];
     front++;
     return frontValue;
}
bool isEmpty()
{
     return front > rear;
}
void bfs(int source, int n, int A[MAX_NODES][MAX_NODES])
{
     int u, v;
     cout << source;
     visited[source] = true;
     add(source);


     while (!isEmpty())
     {
          u = remove();
          for (v = 1; v <= n; v++)
          {
               if (A[u][v] == 1 && !visited[v])
               {
                    cout << " " << v;
                    visited[v] = true;
                    add(v);
               }
          }
     }
}
int main()
{
     int n, m;
     cout << "Enter the number of nodes and edges: ";
     cin >> n >> m;

     int A[MAX_NODES][MAX_NODES] = {0};
     int u, v;

     cout << "Enter the edges:" << endl;
     for (int i = 0; i < m; ++i)
     {
          cin >> u >> v;
          A[u][v] = 1;
     }

    int sourceNode;
    cout << "Enter the source node: ";
    cin >> sourceNode;

    cout << "BFS traversal starting from node " <<
sourceNode << ": ";
    bfs(sourceNode, n, A);

    return 0;
```

```
}
```

<u>**OUTPUT:**</u>

```
Enter the number of nodes and edges: 7 9
Enter the edges:
1 2
1 3
1 4
2 3
3 4
3 5
4 5
5 6
5 7
Enter the source node: 1
BFS traversal starting from node 1: 1 2 3 4 5 6 7
```

DEPTH FIRST SEARCH

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

const int MAX_NODES = 100;

int A[MAX_NODES][MAX_NODES] = {0};
bool visited[MAX_NODES] = {false};

void dfs(int u, int n)
{
     cout << u << " ";
     visited[u] = true;

     for (int v = 1; v <= n; v++)
     {
          if (A[u][v] == 1 && visited[v] == 0)
          {
               dfs(v, n);
          }
     }
}
int main()
{
     int n, m;
     cout << "Enter the number of nodes and edges: ";
     cin >> n >> m;

     cout << "Enter the edges:" << endl;
     for (int i = 0; i < m; ++i)
     {
          int u, v;
          cin >> u >> v;
          A[u][v] = 1;
     }

     int sourceNode;
     cout << "Enter the source node: ";
     cin >> sourceNode;

     cout << "DFS traversal starting from node " <<
sourceNode << ": ";
     dfs(sourceNode, n);

     return 0;
}
```

<u>**OUTPUT:**</u>

```
Enter the number of nodes and edges: 7 9
Enter the edges:
1 2
1 3
1 4
2 3
3 4
3 5
4 5
```

```
5 6
5 7
Enter the source node: 1
DFS traversal starting from node 1: 1 2 3 4 5 6 7
```

MERGE SORT

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

void mergeArrays(int a[], int n, int b[], int m, int c[])
{
    int i = 0, j = 0, k = 0;

    while ((i < n) && (j < m))
    {
        if (a[i] <= b[j])
        {
            c[k] = a[i];
            i++;
            k++;
        }
        else
        {
            c[k] = b[j];
            j++;
            k++;
        }
    }

    while (i < n)
    {
        c[k] = a[i];
        k++;
        i++;
    }

    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;
    mergeArrays(a, n, b, m, c);
    cout << "Merged Array: ";
    for (i = 0; i < p; i++)
    {
        cout << c[i] << " ";
    }

    return 0;
}
```

```
OUTPUT:
Enter the number of elements in the first array: 4
Enter 4 numbers in sorted order: 1 2 8 6 16
Enter the number of elements in the second array: 5
Enter 5 numbers in sorted order: 9 16 27 65 98
Merged Array: 1 2 8 9 16 16 27 65 98
```

SELECTION SORT

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

void selectionSort(int arr[], int n)
{
     for (int i = 0; i < n - 1; i++)
     {
          int minIndex = i;

          for (int j = i + 1; j < n; j++)
          {
               if (arr[j] < arr[minIndex])
               {
                    minIndex = j;
               }
          }

          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: 5
Enter 5 elements: - 8 0 -5 9 -6
Sorted array: -6 -5 0 8 9
```

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: 5
Enter 5 elements: 98 -52 0 77 54
Sorted array: -52 0 54 77 98
```

QUICK SORT

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

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

int partition(int arr[], int low, int high)
{
     int pivot = arr[low];
     int p = low + 1;
     int q = high;

     while (p <= q)
     {
               while (arr[p] < pivot)
                    p++;
               while (arr[q] > pivot)
                    q--;
               if (p < q)
               swap(arr[p], arr[q]);
     }
     swap(arr[low], arr[q]);
     return q;
}

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 the 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: 8
Enter the elements: 15 26 0 -9 100 3 6 9
Sorted array: -9 0 3 6 9 15 26 100
```

HASHING

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

const int HASH_TABLE_SIZE = 10;

int hashFunction(int key)
{
     return key % HASH_TABLE_SIZE;
}

void insert(int hashTable[], int key)
{
     int index = hashFunction(key);
     while (hashTable[index] != 0)
     {
          index = (index + 1) % HASH_TABLE_SIZE;
     }
     hashTable[index] = key;
}

int search(int hashTable[], int key)
{
     int index = hashFunction(key);
     while (hashTable[index] != 0)
     {
          if (hashTable[index] == key)
          {
                    return index;
          }
          index = (index + 1) % HASH_TABLE_SIZE;
     }
     return -1;
}

void display(int hashTable[])
{
     for (int i = 0; i < HASH_TABLE_SIZE; ++i)
     {
          cout << "Index " << i << ": " << hashTable[i]
<<endl;
     }
}

void remove(int hashTable[],int key)
{
     int index = hashFunction(key);
     while(hashTable[index] != 0)
     {
          if (hashTable[index] == key)
          {
               cout<<hashTable[index]<<"is deleted"<<endl;
               hashTable[index] = 0;
          }
          index = (index + 1) % 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;
          insert(hashTable, key);
     }

     display(hashTable);
     int ele;
     cout<<"Enter element to search :";
     cin>>ele;
     int index = search(hashTable, ele);
     if (index != -1)
          cout << "The key "<<ele<<" is found at index " <<
index << endl;
    else
               cout<< "The key "<<ele<<" is not found" <<
endl;

     int x;
     cout<<"Enter element to delete :";
     cin>>x;
     remove(hashTable,x);

     display(hashTable);

     return 0;
}
```

```
OUTPUT:
Enter the number of keys to insert: 5
Enter 5 keys: 2 15 25 35 16
Index 0: 0
Index 1: 0
Index 2: 2
Index 3: 0
Index 4: 0
Index 5: 15
Index 6: 25
Index 7: 35
Index 8: 16
Index 9: 0
enter element to search :35
The key 35 is found at index 7
enter element to delete :1 35
35is deleted
```

```
Index 0: 0
Index 1: 0
Index 2: 2
Index 3: 0
Index 4: 0
Index 5: 15
Index 6: 25
Index 7: 0
Index 8: 16
Index 9: 0
```

BINARY SEARCH TREE

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

class BST
{
    private:
        class Node
        {
            public:
                int val;
                Node *left;
                Node *right;

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

        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;
        }

        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);
        }

        Node *findMin(Node *node)
        {
            while (node->left != nullptr)
            {
                node = node->left;
            }
            return node;
        }

        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;
        }

        //inorder: left,middle,right
        void inOrderTraversal(Node *node)
        {
            if (node != nullptr)
            {
                inOrderTraversal(node->left);
                cout << node->val << " ";
                inOrderTraversal(node->right);
            }
        }
    public:
        BST()
        {
            root = nullptr;
        }
        void insert(int val)
        {
            root = insert(root, val);
        }
        bool search(int val)
        {
            return search(root, val);
        }
        void remove(int val)
        {
            root = deleteNode(root, val);
        }
        void printInOrder()
        {
            inOrderTraversal(root);
            cout << endl;
        }
};

int main()
{
    BST bst;

    int choice;
    int key;

    while (true)
    {
        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:
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: 5
Value 5 has been inserted into the BST.

Binary Search Tree Operations:
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: 10
Value 10 has been inserted into the BST.

Binary Search Tree Operations:
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: 15
Value 15 has been inserted into the BST.

Binary Search Tree Operations:
1. Insert a node
2. Search for a key
3. Delete a node
4. Print in-order traversal
5. Exit
Enter your choice: 4
In-order traversal of the BST: 5 10 15

Binary Search Tree Operations:
1. Insert a node
2. Search for a key
3. Delete a node
4. Print in-order traversal
5. Exit
Enter your choice: 2
Enter the key to search: 15
Key 15 is found in the BST.

Binary Search Tree Operations:
1. Insert a node
2. Search for a key
3. Delete a node
4. Print in-order traversal
5. Exit
Enter your choice: 2
Enter the key to search: 6
Key 6 is not found in the BST.

Binary Search Tree Operations:
1. Insert a node
2. Search for a key
3. Delete a node
4. Print in-order traversal
5. Exit
```

```
Enter your choice: 3
Enter the key to delete: 5
Key 5 has been deleted from the BST.

Binary Search Tree Operations:
1. Insert a node
2. Search for a key
3. Delete a node
4. Print in-order traversal
5. Exit
Enter your choice: 4
In-order traversal of the BST: 10 15

Binary Search Tree Operations:
1. Insert a node
2. Search for a key
3. Delete a node
4. Print in-order traversal
5. Exit
Enter your choice: 5
Exiting program.
```

SINGLE LINKED LIST

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

class linkedList
{
     struct node
     {
               int data;
               node *ptr;
     } *head;
     public:
          linkedList()
          {
               head = NULL;
```

```
}
void InsBeg(int x)
{
     node *q;
     q = new node;
     q->data = x;
     q->ptr = head;
     head = q;
}
void InsEnd(int x)
{
     node *q, *r;
     if (head == NULL)
     {
          head = new node;
          head->data = x;
          head->ptr = NULL;
     }
     else
     {
          q = head;
          while (q->ptr != NULL)
          {
               q = q->ptr;
          }
          r = new node;
          r->data = x;
          r->ptr = NULL;
          q->ptr = r;
     }
}
void insertAfter(int x, int z)
{
     node *q, *r;
     q = head;
     while (q != NULL)
     {
          if (q->data == z)
          {
                    r = new node;
                    r->data = x;
                    r->ptr = q->ptr;
                    q->ptr = r;
                    q = NULL;
          }
          else
                    q = q->ptr;
     }
}
void insByPos(int x, int c)
{
     node *q, *r;
     q = head;
     for (int i = 1; i < c; i++)
     {
          q = q->ptr;
          if (q == NULL)
          {
               cout <<"\nLess Position";
               return;
          }
     }
     r = new node;
     r->data = x;
     r->ptr = q->ptr;
     q->ptr = r;
}
void deleteBegin()
{
     node *q;
     if (head == NULL)
          cout<<"\nList is empty";
     else
     {
          q = head;
          head = head->ptr;
```

```
                    cout <<"\nThe deleted element is: "<<
q->data;
                    delete q;
               }
          }
          void deleteEnd()
          {
               node *q, *r;
               q = head;
               if (head == NULL)
                    cout<<"\nList is empty";
               else
               {
                    while (q != NULL)
                    {
                         if ((q->ptr)->ptr == NULL)
                         {
                              r = q->ptr;
                              cout <<"\nThe deleted element
is: "<< r->data;
                              delete r;
                              q->ptr = NULL;
                              q = NULL;
                         }
                         else
                              q = q->ptr;
                    }
               }
          }
          void delMid(int z)
          {
               node *q, *r;
               q = head;
               if (head == NULL)
                    cout<<"\nList is empty";
               else
               {
                    if (q->data == z)
```

```
                {
                     head = head->ptr;
                     cout <<"\nThe deleted element is: 
"<< q->data;
                     delete q;
                }
                else
                {
                     while (q != NULL)
                     {
                          if (q->data == z)
                          {
                               r->ptr = q->ptr;
                               cout <<"\nThe deleted 
element is: "<< q->data;
                               delete q;
                               q = NULL;
                          }
                          else
                          {
                               r = q;
                               q = q->ptr;
                          }
                     }
                }
           }
      }
      void display()
      {
           node *q;
           if (head == NULL)
                cout<<"\nList is empty";
           else
           {
                cout <<"\nThe element of the list are: ";
                q = head;
                while (q != NULL)
                {
                        cout <<"\nThe element is: "<<
q->data;
                        q = q->ptr;
                    }
                }
            }
        };
        main()
        {
            linkedList l;
            int x, choice, z;
            char ch = 'Y';
            while (ch == 'Y' || ch == 'y')
            {
                cout << "1.Insert Begin"<<endl;
                cout << "2.Insert End"<<endl;
                cout << "3.Insert After"<<endl;
                cout << "4.Insert By Pos"<<endl;
                cout << "5.Delete Begin"<<endl;
                cout << "6.Delete End"<<endl;
                cout << "7.Delete Middle"<<endl;
                cout << "8.Display"<<endl;
                cout << "9.Exit"<<endl;
                cout << "Enter your choice: ";
                cin >> choice;
                switch (choice)
                {
                    case 1:
                        cout <<"\nEnter element to insert:
";
                        cin >> x;
                        l.InsBeg(x);
                        break;
                    case 2:
                        cout <<"\nEnter element to insert:
";
                        cin >> x;
                        l.InsEnd(x);
```

```
                              break;
                         case 3:
                              cout <<"\nEnter element to insert:
";
                              cin >> x;
                              cout <<"\nEnter after which element
to make the insertion: ";
                              cin >> z;
                              l.insertAfter(x, z);
                              break;
                         case 4:
                              cout <<"\nEnter element to insert:
";
                              cin >> x;
                              cout <<"\nEnter position: ";
                              cin >> z;
                              l.insByPos(x, z);
                              break;
                         case 5:
                              cout <<"\nDelete at begin: ";
                              l.deleteBegin();
                              break;
                         case 6:
                              cout <<"\nDelete at end: ";
                              l.deleteEnd();
                              break;
                         case 7:
                              cout <<"\nDelete middle";
                              cout <<"\nEnter value to delete: ";
                              cin >> z;
                              l.delMid(z);
                              break;
                         case 8:
                              cout <<"\nLinkedList is: ";
                              l.display();
                              break;
                         case 9:
                              exit(0);
```

```
                    default:
                         cout <<"\nInvalid choice";
                         break;
               }
               cout <<"\nDo you want to continue? (Y/N): ";
               cin >> ch;
          }
}
```

<u>OUTPUT</u>:

```

1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 1
Enter element to insert: 10

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 1
Enter element to insert: 20

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
```

```
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 2
Enter element to insert: 30

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 2
Enter element to insert: 40

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 3
Enter element to insert: 50
Enter after which element to make the insertion: 10

Do you want to continue? (Y/N): y
```

```
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 4
Enter element to insert: 60
Enter position: 3

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 8
LinkedList is:
The element of the list are:
The element is: 20
The element is: 10
The element is: 50
The element is: 60
The element is: 30
The element is: 40

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
```

```
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 5
Delete at begin:
The deleted element is: 20

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 6
Delete at end:
The deleted element is: 40

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 7
Delete middle
Enter value to delete: 60
The deleted element is: 60
```

```
Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 8
LinkedList is:
The element of the list are:
The element is: 10
The element is: 50
The element is: 30

Do you want to continue? (Y/N): n
```

DOUBLE LINKED LIST

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

class DoubleLinkedList
{
     struct Node
     {
          int data;
          Node *lptr;
          Node *rptr;
     }*head;
     public:
          DoubleLinkedList()
          {
               head = NULL;
          }
          void insertBeginning(int x)
```

```
{
     Node *q;
     if (head == NULL)
     {
          head = new Node;
          head->lptr = NULL;
          head->data = x;
          head->rptr = NULL;
     }
     else
     {
          q = new Node;
          q->lptr = NULL;
          q->data = x;
          q->rptr = head;
          head->lptr = q;
          head = q;
     }
}
void insertEnd(int x)
{
     Node *q, *r;
     if (head == NULL)
     {
          head = new Node;
          head->data = x;
          head->lptr = NULL;
          head->rptr = NULL;
     }
     else
     {
          q = head;
          while (q->rptr != NULL)
          {
               q = q->rptr;
          }
          r = new Node;
          r->data = x;
```

```
                    r->lptr = q;
                    r->rptr = NULL;
                    q->rptr = r;
               }
          }
          void insertAfter(int x, int element)
          {
               Node *q, *r;
               q = head;
               if (p == NULL)
               {
                    cout << "List is empty." << endl;
                    return;
               }
               while (q != NULL)
               {
                    if (q->data == element)
                    {
                         r = new Node;
                         r->data = x;
                         r->lptr = q;
                         r->rptr = q->rptr;
                         if (q->rptr != NULL)
                         {
                              (q->rptr)->lptr = r;
                         }
                         q->rptr = r;
                         return;
                    }
                    q = q->rptr;
               }
               cout << "Element " << element << " not found
in the list." << endl;
          }
          void insertByPosition(int x, int position)
          {
               Node *q, *r;
               q = head;
```

```
                for (int i = 1; i < position; i++)
                {
                     q = q->rptr;
                     if (q == NULL)
                     {
                          cout << "\nLess positions." << endl;
                          return;
                     }
                }
                r = new Node;
                r->data = x;
                r->lptr = q;
                r->rptr = q->rptr;
                if (q->rptr != NULL)
                {
                     (q->rptr)->lptr = r;
                }
                q->rptr = r;
           }
           void deleteBeginning()
           {
                Node *q;
                if (head == NULL)
                     cout << "List is empty." << endl;
                q = head;
                head = head->rptr;
                if (head != NULL)
                {
                     p->lptr = NULL;
                }
                cout << "Deleted element is: " << q->data <<
endl;
                delete q;
           }
           void deleteEnd()
           {
                Node *q, *r;
                q = head;
```

```
               if (head == NULL)
                    cout << "List is empty." << endl;
               while (q->rptr != NULL)
               {
                    q = q->rptr;
               }
               r = q;
               q = q->lptr;
               if (q != NULL)
               {
                    q->rptr = NULL;
               }
               cout << "Deleted element is: " << r->data <<
endl;
               delete r;
          }
          void deleteMiddle(int x)
          {
               Node *q, *r;
               q = head;
               if (head == NULL)
                    cout << "List is empty." << endl;
               while (q != NULL)
               {
                    if (q->data == x)
                    {
                         if (q == head)
                         {
                              head = head->rptr;
                              if (head != NULL)
                              {
                                   head->lptr = NULL;
                              }
                              cout << "Deleted element is: "
<< q->data << endl;
                              delete q;
                         }
                         else
```

```
                        {
                            r = q;
                            q = q->rptr;
                            (r->lptr)->rptr = q;
                            if (q != NULL)
                            {
                                q->lptr = r->lptr;
                            }
                            cout << "Deleted element is: "
<< r->data << endl;
                            delete r;
                        }
                    }
                    q = q->rptr;
                }
                cout << "Element " << x << " not found in the
list." << endl;
            }
            void display()
            {
                Node *q;
                if (head == NULL)
                    cout << "List is empty." << endl;
                q = head;
                cout << "Elements of the list are: ";
                while (q != NULL)
                {
                    cout << q->data << " ";
                    q = q->rptr;
                }
                cout << endl;
            }
        };
int main()
{
    DoubleLinkedList list;
    int element, elementAfter, position, choice;
    while (true)
        {
            cout << "Linked List Operations:\n";
            cout << "1. Insert at Beginning\n";
            cout << "2. Insert at End\n";
            cout << "3. Insert after an Element\n";
            cout << "4. Insert by Position\n";
            cout << "5. Delete Beginning Element\n";
            cout << "6. Delete End Element\n";
            cout << "7. Delete an Element From the List\n";
            cout << "8. Display the List\n";
            cout << "9. Exit\n";
            cout << "Enter your choice: ";
            cin >> choice;
            switch (choice)
            {
               case 1:
                    cout << "Enter an element to insert: ";
                    cin >> element;
                    list.insertBeginning(element);
                    break;
               case 2:
                    cout << "Enter an element to insert: ";
                    cin >> element;
                    list.insertEnd(element);
                    break;
               case 3:
                    cout << "Enter the element to insert: ";
                    cin >> element;
                    cout << "Enter the element after which to
insert: ";
                    cin >> elementAfter;
                    list.insertAfter(element, elementAfter);
                    break;
               case 4:
                    cout << "Enter the position and element to
insert: ";
                    cin >> position >> element;
                    list.insertByPosition(element, position);
                    break;
               case 5:
                    list.deleteBeginning();
                    break;
               case 6:
                    list.deleteEnd();
                    break;
               case 7:
                    cout << "Enter the element to delete from 
the List: ";
                    cin >> element;
                    list.deleteMiddle(element);
                    break;
               case 8:
                    list.display();
                    break;
               case 9:
                    cout << "Linked list is closed." << endl;
                    exit(0);
               default:
                    cout << "Invalid choice." << endl;
                    break;
            }
        }
        return 0;
}
```

OUTPUT:

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
```

```
Enter your choice: 1
Enter an element to insert: 10
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 1
Enter an element to insert: 20
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 2
Enter an element to insert: 30
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 2
Enter an element to insert: 40
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 3
Enter the element to insert: 50
Enter the element after which to insert: 10
Element 10 not found in the list.
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 4
Enter the position and element to insert: 3 60
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 8
Elements of the list are: 20 10 50 60 30 40
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 5
Deleted element is: 20
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 6
Deleted element is: 40
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to delete from the List: 60
Deleted element is: 60
Element 60 not found in the list.
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to delete from the List: 100
Element 100 not found in the list.
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 8
Elements of the list are: 10 50 30
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 9
Linked list is closed.
```

CIRCULAR LINKED LIST

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

class CircularLinkedList
{
     struct node
     {
          int data;
          node *ptr;
     }*p;
     public:
          CircularLinkedList()
          {
               p = NULL;
          }
          void insertion(int x)
          {
               node *q, *r;
               if (p == NULL)
               {
                    p = new node;
                    p->data = x;
                    p->ptr = p;
               }
               else
               {
                    q = p;
                    while (q->ptr != p)
                    {
                         q = q->ptr;
                    }
                    r = new node;
                    r->data = x;
                    r->ptr = p;
                    q->ptr = r;
               }
          }
          void deletion(int z)
          {
               node *q, *r, *k;
               q = p;
               if (q->data == z)
               {
                    if (p->ptr == p)
                    {
                         cout << "\nThe deleted element is "
<< p->data;
                         delete p;
                         p = NULL;
                    }
                    else
                    {
                         while (q->ptr != p)
                         {
                              q = q->ptr;
                         }
                         r = q->ptr;
                         q->ptr = p->ptr;
                         p = p->ptr;
                         cout << "\nThe deleted element is "
<< r->data;
                         delete r;
                    }
               }
               else
               {
                    r = q;
                    q = q->ptr;
                    while (q != p)
                    {
                         if (q->data == z)
                         {
                              cout << "\nThe deleted element
is " << q->data;
```

```
                        r->ptr = q->ptr;
                        delete q;
                        q = p;
                    }
                    else
                    {
                        r = q;
                        q = q->ptr;
                    }
                }
            }
        }
        void display()
        {
            node *q;
            if (p == NULL)
                cout << "List is empty." << endl;
            else
            {
                q = p;
                cout << "Elements of the list are:" <<
endl;
                do
                {
                    cout << q->data << " ";
                    q = q->ptr;
                } while (q != p);
                cout << endl;
            }
        }
};
int main()
{
    CircularLinkedList l;
    int choice, x, z;
    do
    {
        cout << "Linked List Operations:\n";
        cout << "1. Insertion\n";
        cout << "2. Deletion\n";
        cout << "3. Display\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "Enter an element to insert: ";
            cin >> x;
            l.insertion(x);
            break;
        case 2:
            cout << "Enter the element you want to delete:
";
            cin >> z;
            l.deletion(z);
            break;
        case 3:
            l.display();
            break;
        case 4:
            cout << "linked list closed" << endl;
            exit(0);
            break;
        }
    } while (choice <= 4);
    return 0;
}
```

OUTPUT:

```
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter an element to insert: 10
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter an element to insert: 20
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter an element to insert: 30
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2
Enter the element you want to delete: 80
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Elements of the list are:
10 20 30
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2
Enter the element you want to delete: 30
The deleted element is 30
```

```
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Elements of the list are:
10 20
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 4
linked list closed
```

STACK USING ARRAY

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

#define max 10
class Stack
{
    private:
        int a[max];
        int top;
    public:
        Stack()
        {
            top = -1;
        }
        void push(int x)
        {
            if (top < max-1)
            {
                a[++top] = x;
```

```
                    cout << "Pushed element " << x << " into
the stack" << endl;
               }
               else
                    cout<<"\nStack is full";
          }
          void pop()
          {
               if (top == -1)
                    cout << "Stack is empty";
               else
               {
                    int temp = a[top];
                    a[top]= NULL;
                    top--;
                    cout << "Popped element " << temp << "
from the stack" << endl;
               }
          }
          void display()
          {
               if (top == -1)
                    cout << "Stack is empty";
               else
               {
                    cout<<"\nStack Elements are: "<<endl;
                    for(int i = top; i >= 0;--i)
                              cout<<a[i]<<" ";
               }
          }
};
int main()
{
    Stack S;
    int choice,ele;
    while(true)
    {
        cout<<"\n\nStack Operations are:
\n(1)Push\n(2)Pop\n(3)Display\n(4)Exit\n";
        cout<<"Enter your choice: ";
        cin>>choice;
        if(choice==1)
        {
            cout<<"\nEnter an element to push into the
stack: ";
            cin>>ele;
            S.push(ele);
        }
        if(choice==2)
           S.pop();
        if(choice==3)
            S.display();
        if(choice==4)
        {
            cout<<"\nOut of the Stack";
            exit(0);
        }
        if(choice<1 || choice>4)
            cout<<"Invalid Input";
    }
    return 0;
}
```

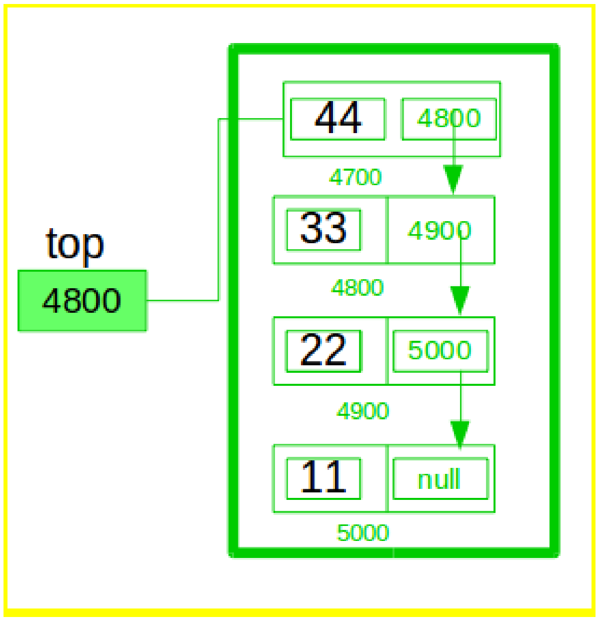
OUTPUT:

```
Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 1

Enter an element to push into the stack: 584
Pushed element 584 into the stack
```

```
Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 1

Enter an element to push into the stack: 652
Pushed element 652 into the stack


Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 1

Enter an element to push into the stack: 952
Pushed element 952 into the stack


Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 3

Stack Elements are:
952 652 584

Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 2
```

```
Popped element 952 from the stack


Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 4

Out of the Stack
```

STACK USING LINKED LIST



```
#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 = top->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
```

QUEUE USING ARRAY

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

#define max 10
class Queue
{
     private:
          int q[max];
          int front;
          int rear;
     public:
          Queue()
          {
               front = rear = -1;
          }
          void insertion(int x)
          {
               if (rear < max-1)
               {
                    rear++;
                    q[rear]=x;
                    cout<<"\nThe inserted element is:
"<<q[rear];
                    if(front == -1)
                         front = 0;
               }
               else
                    cout<<"\nQueue is full";
          }
          void deletion()
          {
               if (front == -1)
                    cout << "Queue is empty";
               else
               {
                    cout<<"\nThe deleted element is:
"<<q[front];
                    q[front] = NULL;
                    front++;
                    if(front > rear)
                         front = rear = -1;
               }
          }
          void display()
          {
               if (front == -1)
                    cout << "Queue is empty";
               else
               {
                    cout << "Elements in the queue: ";
                    for (int i = front; i <= rear; ++i)
                    {
                         cout << q[i] << " ";
                    }
                    cout << endl;
               }
          }
```

```
};
int main()
{
    Queue q;
    int choice,ele;
    while(true)
    {
        cout<<"\nQueue Operations are:
\n(1)Insertion\n(2)Deletion\n(3)Display\n(4)Exit\n";
        cout<<"Enter your choice: ";
        cin>>choice;
        if(choice==1)
        {
            cout<<"\nEnter an element: ";
            cin>>ele;
            q.insertion(ele);
        }
        if(choice==2)
           q.deletion();
        if(choice==3)
            q.display();
        if(choice==4)
        {
            cout<<"\nOut of the Queue";
            exit(0);
        }
        if(choice<1 || choice>4)
            cout<<"Invalid Input";
    }
    return 0;
}
```

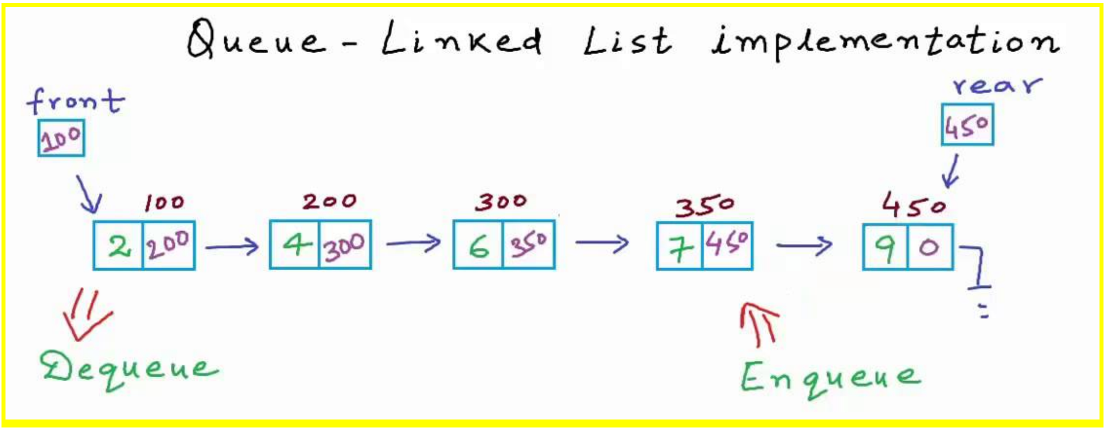
OUTPUT:

```
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
```

```
Enter your choice: 1

Enter an element: 10

The inserted element is: 10
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 1

Enter an element: 20

The inserted element is: 20
Queue is full
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 1

Enter an element: 30

The inserted element is: 30
Queue is full
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 2

The deleted element is: 10
Queue Operations are:
(1)Insertion
(2)Deletion
```

```
(3)Display
(4)Exit
Enter your choice: 3
Elements in the queue: 20 30

Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 4

Out of the Queue
```

QUEUE USING LINKED LIST



```
#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 = temp->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
```

SIMPLE INHERITANCE

```
#include<iostream>
using namespace std;
class student
{
    private:
        int rno;
        char name[20];
    public:
        void get_details()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void put_details()
        {
            cout<<"\nName: "<<name<<"\nrno: "<<rno;
        }
};
class marks:public student{
    private:
        float marks;
    public:
        void get_marks()
        {
            cout<<"\nEnter marks: ";
            cin>>marks;
        }
        void put_marks()
        {
            cout<<"\nMarks: "<<marks;
        }
};
int main()
{
    marks obj;
    obj.get_details();
    obj.get_marks();
    obj.put_details();
    obj.put_marks();
    return 0;
}
```

```
OUTPUT
Enter rno and name: 19 Sushma

Enter marks: 43

Name: Sushma
rno: 19
Marks: 43
```

MULTILEVEL INHERITANCE

```
#include<iostream>
using namespace std;
class student
{
    private:
        int rno;
        char name[20];
    public:
        void get_details()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void put_details()
        {
            cout<<"\nName: "<<name<<"\nrno: "<<rno;
        }
};
class marks:public student
{
    protected:
        float m1,m2;
    public:
        void get_marks()
        {
            cout<<"\nEnter marks: ";
            cin>>m1>>m2;
        }
        void put_marks()
        {
            cout<<"\nMarks: "<<m1<<" "<<m2;
        }
};
class test:public marks
{
    private:
    float total;
    public:
        void display()
        {
            total=m1+m2;
            cout<<"\nTotal: "<<total;
        }
};
int main()
{
    test obj;
    obj.get_details();
    obj.get_marks();
    obj.put_details();
    obj.put_marks();
    obj.display();
    return 0;
}
```

OUTPUT

```
Enter rno and name: 19 Sushma

Enter marks: 23 33
```

```
Name: Sushma
rno: 19
Marks: 23 33
Total: 56
```

MULTIPLE INHERITANCE

```
#include<iostream>
using namespace std;
class student
{
    protected:
        int rno;
        char name[20];
    public:
        void get_details()
        {
            cout<<"\nEnter  rno and name: ";
            cin>>rno>>name;
        }
};
class marks {
   protected:
        float m;
    public:
        void get_marks()
        {
            cout<<"\nEnter marks: ";
            cin>>m;
        }

};
class displaying:public student,public marks
{
    public:
        void display()
        {
            cout<<"\Student details are: ";
            cout<<"\nName: "<<name<<"\nRno:
"<<rno<<"\nMarks: "<<m;

        }
} ;
int main()
{
    displaying obj;
    obj.get_details();
    obj.get_marks();
    obj.display();
    return 0;
}
```

```
OUTPUT
Enter  rno and name: 19 Sushma

Enter marks: 13

Student details are:
Name: Sushma
Rno: 19
Marks: 13
```

HIERARCHICAL INHERITANCE

```
#include<iostream>
using namespace std;
class student
{
    protected:
        int rno;
        char name[30];
    public:
        void get_details()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void put_details()
        {
            cout<<"\nThe student details are: ";
            cout<<"\nName: "<<name<<"\nrno : "<<rno;
        }
};
class science: public student
{
    protected:
        float s1;
    public:
        void get_scmarks()
        {
            cout<<"\nEnter science marks: ";
            cin>>s1;
        }
        void put_scmarks()
        {
            cout<<"\nScience marks are: "<<s1;
        }
};
class arts:public student{
    protected:
        float a1;
    public:
        void get_artsmarks()
        {
            cout<<"\nEnter arts marks: ";
            cin>>a1;
        }
        void put_artsmarks()
        {
            cout<<"\nArts marks are: "<<a1;
        }
} ;
int main()
```

```
{
    science obj1;
    obj1.get_details();
    obj1.get_scmarks();
    obj1.put_details();
    obj1.put_scmarks();
    arts obj2;
    obj2.get_details();
    obj2.get_artsmarks();
    obj2.put_details();
    obj2.put_artsmarks();
    return 0;
}
```

OUTPUT

```
Enter rno and name: 19 Sushma

Enter science marks: 54

The student details are:
Name: Sushma
rno : 19
Science marks are: 54

Enter rno and name: 16 Laxmi

Enter arts marks: 60

The student details are:
Name: Laxmi
rno : 16
Arts marks are: 60
```

HYBRID INHERITANCE

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

```
{
    protected:
        int rno;
        char name[20];
    public:
        void getdata()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void putdata()
        {
            cout<<"\nThe student details are: ";
            cout<<"\nName: "<<name<<"\n rno: "<<rno;
        }
};
class marks:public student
{
    protected:
        float m1,m2;
    public:
        void get_marks()
        {
            cout<<"\nEnter two subject marks: ";
            cin>>m1>>m2;
        }
        void put_marks()
        {
            cout<<"\nMarks are: "<<m1<<"\t"<<m2;
        }
};
class sports
{
    protected:
        float score;
    public:
        void get_score()
        {
            cout<<"\nEnter score: ";
            cin>>score;
        }
        void put_score()
        {
            cout<<"\nScore is: "<<score;
        }
};
class result:public marks,public sports
{
    protected:
        float res;
    public:
        void display()
        {
            res=m1+m2+score;
            cout<<"\nThe result is: "<<res;
        }
};
int main()
{
    result obj;
    obj.getdata();
    obj.get_marks();
    obj.get_score();
    obj.putdata();
    obj.put_marks();
    obj.put_score();
    obj.display();
}
```

OUTPUT

```
Enter rno and name: 19 Sushma

Enter two subject marks: 34 33

Enter score: 9.8
```

```
The student details are:
Name: Sushma
rno: 19
Marks are: 34   33
Score is: 9.8
The result is: 76.8
```

INFIX TO POSTFIX

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

#define MAX  100
class Stack
{
     private:
          int top;
          char arr[MAX];
     public:
          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(char c: s)
    {
        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: ";
    cin >> exp;
    infixToPostfix(exp);
    return 0;
}
```

Output

```
Enter the Infix Expression: (A+B)*C-(D-F)*(F+G)
Postfix Expression: AB+C*DF-FG+*-
```

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 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
        {
            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
```

SWAP TWO NUMBERS USING FUNCTION OVERLOADING

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

void swap(int& a, int& b)
```

```
{
    int temp = a;
    a = b;
    b = temp;
}
void swap(float& a, float& b)
{
    float temp = a;
    a = b;
    b = temp;
}
void swap(char& a, char& b)
{
    char temp = a;
    a = b;
    b = temp;
}
int main()
{
    int num1, num2;
    cout<<"Enter 2 Numbers: ";
    cin>>num1>>num2;
    cout << "Before swapping:" << endl;
    cout << "num1: " << num1 << " num2: " << num2 << endl;
    swap(num1, num2);
    cout << "After swapping:" << endl;
    cout << "num1: " << num1 << " num2: " << num2 << endl;

    float float1 , float2 ;
    cout<<"Enter 2 Decimal Numbers: ";
    cin>>float1>>float2;
    cout << "Before swapping:" << endl;
    cout << "float1: " << float1 << " float2: " << float2 <<
endl;
    swap(float1, float2);
    cout << "After swapping:" << endl;
    cout << "float1: " << float1 << " float2: " << float2 <<
endl;
```

```
    char char1 , char2 ;
    cout<<"Enter 2 Characters: ";
    cin>>char1>>char2;
    cout << "Before swapping:" << endl;
    cout << "char1: " << char1 << " char2: " << char2 <<
endl;
    swap(char1, char2);
    cout << "After swapping:" << endl;
    cout << "char1: " << char1 << " char2: " << char2 <<
endl;

    return 0;
}
```

```
OUTPUT
Enter 2 Numbers: 2 3
Before swapping:
num1: 2 num2: 3
After swapping:
num1: 3 num2: 2
Enter 2 Decimal Numbers: 1.2 4.5
Before swapping:
float1: 1.2 float2: 4.5
After swapping:
float1: 4.5 float2: 1.2
Enter 2 Characters: a f
Before swapping:
char1: a char2: f
After swapping:
char1: f char2: a
```

DEMONSTRATE DEFAULT ARGUMENTS

```
#include<iostream>
using namespace std;
int mul(int x,int y=5,int z=8)
```

```
{
    return x*y*z;
}
int main()
{
    int a,b,c;
    cout<<"\nEnter 3 nos: ";
    cin>>a>>b>>c;
    cout<<"Result is: "<<mul(a,b,c)<<endl;
    cout<<"Result is: "<<mul(a,b)<<endl;
    cout<<"Result is: "<<mul(a)<<endl;
    cout<<"Result is: "<<mul(b)<<endl;
    cout<<"Result is: "<<mul(a,c)<<endl;
    return 0;
}
```

OUTPUT

```
Enter 3 nos: 1 2 5
Result is: 10
Result is: 16
Result is: 40
Result is: 80
Result is: 40
```

DYNAMIC MEMORY ALLOCATION AND DEALLOCATION

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

int main()
{
    int *a = new int;
    int *b = new int;
    *a = 5;
    *b = 6;
    *a = (*a)*(*a);
    cout<<"\na: "<<*a;
    delete a;
    *b = (*b)*(*b);
    cout<<"\nb: "<<*b;
    delete b;
}
```

```
OUTPUT
a: 25
b: 36
```

DYNAMIC MEMORY ALLOCATION AND DEALLOCATION USING ARRAYS

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

int main()
{
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    int* arr = new int[size];
    cout << "Enter the elements of the array:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    cout << "Elements of the array:" << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    delete[] arr;
    return 0;
}
```

```
OUTPUT
Enter the size of the array: 3
Enter the elements of the array:
15 44 87
Elements of the array:
15 44 87
```

CONCEPT OF CLASS AND MEMBER FUNCTIONS

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

class Student
{
    private:
        int rno;
        char name[50];
        float cgpa;
    public:
        void getdata()
        {
            cout<<"\nEnter student details rno,name,cgpa: ";
            cin>>rno>>name>>cgpa;
        }
        void putdata()
        {
            cout<<"\nThe students details are: ";
            cout<<"\nRno: "<<rno<<"\nName: "<<name<<"\nCGPA:
"<<cgpa;
        }
};
int main()
{
    Student s;
    s.getdata();
    s.putdata();
}
```

OUTPUT

```
Enter student details rno,name,cgpa: 19 Sushma 9.54
```

```
The students details are:
Rno: 19
Name: Sushma
CGPA: 9.54
```

DEMONSTRATE CONSTRUCTORS

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

class Integer
{
    private:
        int a,b;
    public:
        Integer()
        {
            a=b=10;
        }
        Integer(int x,int y)
        {
            a=x;
            b=y;
        }
        Integer(Integer &i)
        {
            a=i.a;
            b=i.b;
        }
        void display()
        {
            cout<<"\n The values of a and b are: "<<a<<"
"<<b<<endl;
        }
};
int main()
{
    Integer i1;
    i1.display();
    Integer i2(4,5);
    i2.display();
    Integer i3(i2);
    i3.display();
    return 0;
}
```

```
OUTPUT
 The values of a and b are: 10 10
 The values of a and b are: 4 5
 The values of a and b are: 4 5
```

DEMONSTRATE DESTRUCTORS

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

int c=0;
class test
{
    public:
        test()
        {
            c++;
            cout<<"\nobject created : "<<c;
        }
        ~test()
        {
            cout<<"\nobject destroyed: "<<c;
            c--;
        }
};
int main()
{
    cout<<"\nEntered into main()";
    test t1,t2,t3;
    {
        cout<<"\nEntered into block 1";
        test t4;
    }
    {
    cout<<"\nEntered into block 2: ";
    test t5;
    }
    cout<<"\nReentered into main";
    return 0;
}
```

OUTPUT

```
Entered into main()
object created : 1
object created : 2
object created : 3
Entered into block 1
object created : 4
object destroyed: 4
Entered into block 2:
object created : 4
object destroyed: 4
Reentered into main
object destroyed: 3
object destroyed: 2
object destroyed: 1
```