**Linear Search**

public static void LinearSearch(String[,] Arr, String item) {

for (int i = 0; i < Arr.Length / 2; i++) {

if (Arr[i, 0].ToLower() == item.ToLower()) {

Console.WriteLine("Item Found at index {0}", i);

Console.WriteLine(Arr[i, 0] + " " + Arr[i, 1]);

}

}

}

**Selection Sort**

static void selectionSort(String[,] arr, int n) {

int smallest; string temp;

for (int i = 0; i < n - 1; i++) { smallest = i;

for (int j = i + 1; j < n; j++) {

if (String.Compare(arr[j, 0], arr[smallest, 0]) < 0) {

smallest = j;

}

}

temp = arr[smallest, 0];

arr[smallest, 0] = arr[i, 0];

arr[i, 0] = temp;

}

}

**Singly Linked List**

**Public Class sll**{

public Node head;

public void printList() { Node n = head;

while (n != null) { Console.Write(n.data + " ");

n = n.next;

}

}

**public void push(int new\_data)** {

Node new\_node = new Node(new\_data);

new\_node.next = head;

head = new\_node; }

public void insertAfter(Node prev\_node, int new\_data) {

if (prev\_node == null) {

Console.WriteLine("The given previous"+" node cannot be null");

return;

}

Node new\_node = new Node(new\_data);

new\_node.next = prev\_node.next;

prev\_node.next = new\_node;

}

**public void append(int new\_data)** {

Node new\_node = new Node(new\_data);

if (head == null) {

head = new Node(new\_data);

return;

}

new\_node.next = null;

Node last = head;

while (last.next != null) last = last.next;

last.next = new\_node;

return;

}

**public void deleteNode(int key)** {

Node temp = head, prev = null;

if (temp != null && temp.data == key {

head = temp.next;

return; }

while (temp != null && temp.data != key) {

prev = temp;

temp = temp.next;

}

if (temp == null)

return;

prev.next = temp.next;

}

}

**Doubly Linked List**

**public class Node** {

public int data;

public Node next;

public Node prev;

public Node(int d) { data = d;

prev = null; next = null; } }

**public class DLL** { public Node head;

**public void printList(Node node)** {

Node last = node;

Console.Write("\nTraversing in forward direction: ");

while (node != null) {

Console.Write(node.data + " ");

node = node.next; }

Console.Write("\nTraversing in reverse direction: ");

while (last != null) { Console.Write(last.data + " ");

last = last.prev; } }

**public void push(int new\_data)** {

Node new\_Node = new Node(new\_data);

new\_Node.next = head;

new\_Node.prev = null;

if (head != null) head.prev = new\_Node;

head = new\_Node; }

**public void InsertAfter(Node prev\_Node, int new\_data)** {

if (prev\_Node == null) {

Console.WriteLine("The given previous node cannot be NULL ");

return; }

Node new\_node = new Node(new\_data);

new\_node.next = prev\_Node.next;

prev\_Node.next = new\_node;

new\_node.prev = prev\_Node;

if (new\_node.next != null) new\_node.next.prev = new\_node; }

**public void insertBefore(Node next\_node, int new\_data)** {

Node new\_Node = new Node(new\_data);

if (next\_node == null) {

Console.WriteLine("Node must not be null!");

return; }

new\_Node.prev = next\_node.prev;

next\_node.prev = new\_Node;

new\_Node.next = next\_node;

if (new\_Node.prev != null) {

new\_Node.prev.next = new\_Node; }

else { head = new\_Node; } }

**public void deleteNode(Node del)** {

if (head == null || del == null) { return; }

if (head == del) { head = del.next; }

if (del.next != null) { del.next.prev = del.prev; }

if (del.prev != null) { del.prev.next = del.next; }

return; }

**public void deleteNodeAtGivenPos(Node head, int n)** {

if (head == null || n <= 0) return;

Node current = head; int i;

for (i = 1; current != null && i < n; i++) { current = current.next; }

if (current == null) return;

deleteNode(current); } }

**STACK**

internal class STACK {

int top;

static readonly int max = 1000;

int[] stacck = new int[max];

**public bool isempty()** {

return (top < 0);

}

**public STACK()** {

top = - 1;

}

internal bool push(int data) {

if (top>=max) {

Console.WriteLine("stack over flow");

return false; }

else {

stacck[++top]=data; return true;

}

}

internal bool isfull() {

if (top==max) {

Console.WriteLine("Stack is full");

return true; }

else {

Console.WriteLine("Stack is under Flow");

return true; }

}

**internal void printstack()** {

if (top<0) {

Console.WriteLine("Stack is underflow");

return; }

else {

Console.WriteLine("Elements of Stack");

for (int i = top; i >=0 ; i--) {

Console.WriteLine(stacck[i]+" "); }

}

}

**internal int pop()** {

if (top<0) {

Console.WriteLine("underflow :");

return 0; }

else {

int value = stacck[top--];

return value; }

}

**internal void peek()** {

if (top<0) {

Console.WriteLine("underflow :"); }

else { Console.WriteLine("The Peek Value Of Stack Is: {0}",stacck[top]);

}

}

**internal int count()** { int con = 0;

if (top<0) { Console.WriteLine("underflow :"); return 0; }

else {

for (int i = top; i >=0; i--) { con++; }

Console.WriteLine("total element In Stack Are :{0}",stacck[con]); }

return con; }

}