

SR no.	Topic	Date	Page number	Sign
1	Implementation of different sorting techniques		1-18	
2	Implementation of different searching algorithm		19-22	
3	Implementation of stacks (Using Arrays and Linked list)		23-31	
4	Implementation of stack applications like: a) postfix evaluation b) Balancing parenthesis		32-40	
5	Implement all different types of Queues		41-58	
6	Demonstrate Application of queue (e.g., Priority queue, Breadth first Queue)		59-68	
7	Implementation of all types of linked lists a) Singly b) Doubly c) Circular		69-98	
8	Demonstrate application of linked list a) Polynomial Addition b) Sparse Matrix		99-107	
9	Create and perform various operations on binary search operations		108-114	

## Practical 1

Q) Implementation of different sorting techniques

### a) **Bubble sort**

→ Code:

```
#include<iostream>
#include<conio.h>

using namespace std;

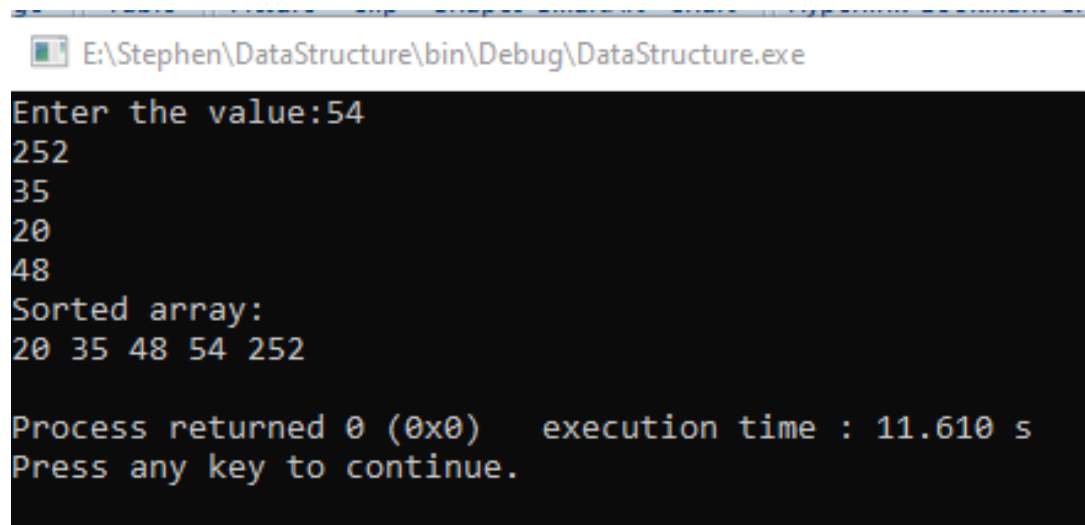
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int bubble[5];
```

```
cout << "Enter the value: \n";  
for (int i = 0; i < 5; ++i) {  
    cin >> bubble[i];  
}  
  
int N = sizeof(bubble) / sizeof(bubble[5]);  
bubbleSort(bubble, N);  
cout << "Sorted array: \n";  
printArray(bubble, N);  
return 0;  
}
```

Output for Bubble Sorting:



The screenshot shows a Windows command prompt window with the title bar "E:\Stephen\DataStructure\bin\Debug\DataStructure.exe". The command prompt displays the following text:

```
Enter the value:54
252
35
20
48
Sorted array:
20 35 48 54 252

Process returned 0 (0x0)   execution time : 11.610 s
Press any key to continue.
```

## b) Insertion Sort

→ Code:

```
#include<iostream>
#include<conio.h>

using namespace std;

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
```

```
    cout << endl;
}
int main()
{
    int number[6];
    cout << "Enter the numbers:\n";
    for (int i = 0; i < 6; ++i) {
        cin >> number[i];
    }
    int N = sizeof(number) / sizeof(number[6]);
    insertionSort(number, N);
    printArray(number, N);
    return 0;
}
```

Output for Insertion Sorting:

```
E:\Stephen\DataStructure\InsertionSort.exe
Enter the numbers:32
54
39
15
28
75
15 28 32 39 54 75

Process returned 0 (0x0)   execution time : 28.685 s
Press any key to continue.
```

## c) Radix Sort

→ Code:

```
#include<iostream>
#include<conio.h>

using namespace std;

int getMax(int arr[], int n)
{
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

void countSort(int arr[], int n, int exp)
{
    int output[n];

    int i, count[10] = {
        0
    };
    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    for (i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    for (i = 0; i < n; i++) arr[i] = output[i];
}
```

```

}

void radixsort(int arr[], int n)
{
    int m = getMax(arr, n);

    for (int exp = 1; m / exp > 0; exp *= 10)
        countSort(arr, n, exp);
}

void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

int main()
{
    int arr[5];

    cout << "Enter the numbers:\n";

    for (int i = 0; i < 5; ++i) {
        cin >> arr[i];
    }

    int n = sizeof(arr) / sizeof(arr[0]);

    radixsort(arr, n);

    cout << "Sorted array:";


    print(arr, n);

    return 0;
}

```

Output for Radix Sorting:

---

 E:\Stephen\DataStructure\RadixSort.exe

```
Enter the numbers:32
```

```
48
```

```
29
```

```
53
```

```
10
```

```
Sorted array:10 29 32 48 53
```

```
Process returned 0 (0x0)   execution time : 35.160 s
```

```
Press any key to continue.
```

## d) Selection Sort

→ Code:

```
#include<iostream>
#include<conio.h>

using namespace std;

void swap(int * xp, int * yp)
{
    int temp = * xp;

    * xp = * yp;

    * yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

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

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

            if (arr[j] < arr[min_idx])

                min_idx = j;

        if (min_idx != i)

            swap( & arr[min_idx], & arr[i]);
    }
}

void printArray(int arr[], int size)
```

```

{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int numbers[5];
    cout << "Enter the numbers:\n";
    for (int i = 0; i < 5; ++i) {
        cin >> numbers[i];
    }

    int n = sizeof(numbers) / sizeof(numbers[5]);
    selectionSort(numbers, n);
    cout << "Sorted array: \n";
    printArray(numbers, n);
    return 0;
}

```

Output for Selection Sorting:

A screenshot of a Windows command prompt window. The title bar at the top reads "E:\Stephen\DataStructure\SelectionSort.exe". The command prompt shows the following text:  
Enter the numbers:32  
95  
20  
34  
73  
Sorted array:  
20 32 34 73 95  
  
Process returned 0 (0x0) execution time : 9.349 s  
Press any key to continue.

## e) Shell Sort

→ Code:

```
#include <iostream>

using namespace std;

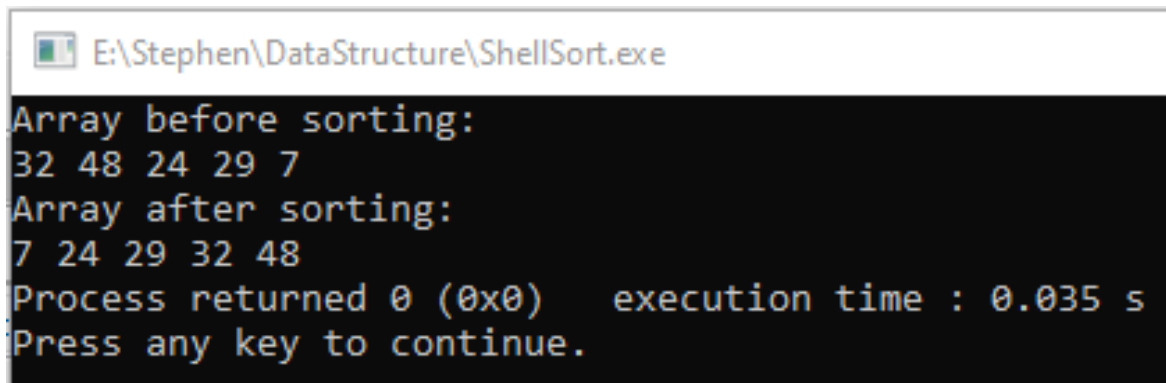
int shellSort(int arr[], int n)
{
    for (int gap = n / 2; gap > 0; gap /= 2)
    {
        for (int i = gap; i < n; i += 1)
        {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];
            arr[j] = temp;
        }
    }
    return 0;
}

void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

int main()
```

```
{  
    int arr[] = { 12, 34, 32, 2, 3 }, i;  
    int n = sizeof(arr) / sizeof(arr[0]);  
    cout << "Array before sorting: \n";  
    printArray(arr, n);  
    shellSort(arr, n);  
    cout << "\nArray after sorting: \n";  
    printArray(arr, n);  
    return 0;  
}
```

Output for Shell Sorting:



```
E:\Stephen\DataStructure\ShellSort.exe
Array before sorting:
32 48 24 29 7
Array after sorting:
7 24 29 32 48
Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

## f) Quick Sort

→ Code:

```
#include <iostream>

using namespace std;

int partition(int arr[], int start, int end)
{
    int pivot = arr[start];

    int count = 0;

    for (int i = start + 1; i <= end; i++) {

        if (arr[i] <= pivot)

            count++;

    }

    int pivotIndex = start + count;

    swap(arr[pivotIndex], arr[start]);

    int i = start, j = end;

    while (i < pivotIndex && j > pivotIndex) {

        while (arr[i] <= pivot) {

            i++;

        }

        while (arr[j] > pivot) {

            j--;

        }

        if (i < pivotIndex && j > pivotIndex) {

            swap(arr[i++], arr[j--]);

        }

    }
```

```

    }

    return pivotIndex;
}

void quickSort(int arr[], int start, int end)
{
    if (start >= end)
        return;

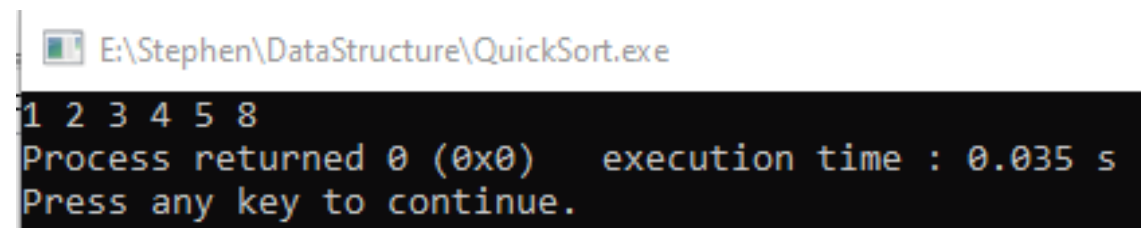
    int p = partition(arr, start, end);
    quickSort(arr, start, p - 1);
    quickSort(arr, p + 1, end);
}

int main()
{
    int arr[] = { 9, 3, 4, 2, 1, 8 };
    int n = 6;
    quickSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}

```

Output for Quick Sorting:



```
E:\Stephen\DataStructure\QuickSort.exe
1 2 3 4 5 8
Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

## Practical 2

Q) Implementation of different searching algorithms:

a) **Binary Search**

b) **Sequential Search**

→ Code:

```
#include<iostream>

#include<conio.h> using namespace std; void binary();

void sequential();
void binary() {

    int i, n, a[10], st = 0, ed = 9, md;

    cout << "Enter 10 elements for array in sorted manner \n";

    for (i = 0; i < 10; i++)

    {

        cin >> a[i];

    }

    cout << "Enter the number you want to search \n";

    cin >> n;

    md = (st + ed) / 2;

    while (n != a[md] && st <= ed)

    {

        if (n > a[md])

            st = md + 1;

        else

            ed = md - 1;

        md = (st + ed) / 2;
```

```

    }

    if (n == a[md])

        cout << n << " found at position " << md;
    if (st > ed)

        cout << "Not found";

}

void sequential()

{

    int i, array[10];

    cout << "Enter 10 elements: \n";

    for (i = 0; i < 10; i++)

    {

        cin >> array[i];

    }

    cout << "Enter the number you want to find (from 10 to 100)...";

    int key;

    cin >> key;

    int flag = 0;

    for (i = 0; i < 10; i++)

    {

        if (array[i] == key)

        {

            flag = 1;

            break;

        }

    }

```

```

    }

    if (flag)
    {
        cout << "Your number is at subscript position " << i << ".\n";
    } else
    {
        cout << "The input number is not present in this array.\n" << endl << endl;
    }
}

int main()
{
    int ch;

    cout << "1-Sequential\n2-Binary\n";

    cout << "Enter your choice\n";

    cin >> ch;

    if (ch == 1)
        sequential();

    else if (ch == 2)
        binary();

    else
        cout << "Invalid choice\n";

    getch();

    return 0;
}

```

Output for Sequential Search:

```
E:\Stephen\DataStructure\BinarySequential.exe
1-sequential
2-Binary
enter your choice
1
enter 10 elements
25
355
15
32
95
53
24
37
17
39
Enter the number you want to find (from 10 to 100)à32
Your number is at subscript position 3.
```

Output for Binary Search:

```
E:\Stephen\DataStructure\BinarySequential.exe
1-sequential
2-Binary
enter your choice
2
Enter 10 elements for array in sorted manner
7
19
24
38
43
52
58
73
81
95
Enter the number you want to search
43
43 found at position 4
Process returned 0 (0x0)   execution time : 127.365 s
Press any key to continue.
```

## Practical 3

Q) Implementation of stacks

### a) Using Arrays

→ Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#define MAX 5

using namespace std;

int top;

class stack
{
public:
    int stack[MAX];

    void push();

    void pop();

    void display();

    int full();

    int empty();

};

int stack::full()

{
    if (top == MAX - 1)

        return 1;

    else

    {

        return 0;
```

```

    }
}
int stack::empty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}
void stack::push()
{
    //int item;

    int push_item;
    if (full() == 1)
        cout << "Overflow \n";
    else
    {
        cout << "Enter the number to push to the stack";
        cin >> push_item;
        top = top + 1;
        stack[top] = push_item;
    }
}
void stack::pop()
{

```

```

if (empty() == 1)

    cout << "Stack Overflow";

else

{

    cout << "Poped Element\n" << stack[top];

    top = top - 1;

}

}

void stack::display()

{

    int a;

    if (top == -1)

        cout << "Stack is Empty \n";

    else

    {

        cout << "Stack Element\n";

        for (a = top; a > 0; a--)

            cout << stack[a] << "\n";

    }

}

int main()

{

    stack s;

    int choice;

    while (choice != 4)

    {

```

```
cout<< "\n1 Push";

cout<< "\n2 p\Pop";

cout<< "\n3 Display";

cout<< "\n4 exit";

cout<< "\nEnter your choice";

cin>> choice;

switch (choice)
{

case 1:
    s.push();

    break;

case 2:
    s.pop();

    break;

case 3:
    s.display();

    break;

case 4:
    exit(1);

default:
    cout << "\nInvalid choice";

}

}

getch();

return 0;

}
```

Output for Implementation of Stacks using Array:

 E:\Stephen\DataStructure\Stack1.exe

```
1 push
2pop
3display
4exit
enter your choice1
enter the number to push to the stack32

1 push
2pop
3display
4exit
enter your choice1
enter the number to push to the stack23

1 push
2pop
3display
4exit
enter your choice3
stack element
23
32

1 push
2pop
3display
4exit
enter your choice2
poped element
23
1 push
2pop
3display
4exit
enter your choice3
stack element
32

1 push
2pop
3display
4exit
enter your choice
```

## b) Using Linked List

→ Code:

```
#include<iostream>
#include<conio.h>
#include<malloc.h>
#include<stdlib.h>

using namespace std;

class nodestack
{
    public: int info;
    nodestack * link;
    int empty();
    void push();
    void pop();
    void display();
} * top = NULL;

void nodestack::push()
{
    nodestack * tmp;
    int pushed_item;
    tmp = new nodestack;
    cout << "Enter element to be pushed\n";
    cin >> pushed_item;
    tmp -> info = pushed_item;
    tmp -> link = top;

    top = tmp;
}

int nodestack::empty()
{
    if (top == NULL)
        return 1;
    else
        return 0;
}

void nodestack::pop()
```

```

{
    nodestack * tmp;
    if (empty())
        cout << "stack empty\n";
    else
    {
        tmp = top;
        cout << "Popped item is\n";
        cout << tmp -> info;
        top = top -> link;
        free(tmp);
    }
}

void nodestack::display()
{
    nodestack * ptr;
    ptr = top;
    cout << "Stack elements are\n";
    while (ptr != NULL)
    {
        cout << ptr -> info << "\n";
        ptr = ptr -> link;
    }
}

int main()
{
    nodestack n;

    int c;

```

```
while (c != 4)
{
    cout << "\n1=push\n2=pop\n3=display\n4=exit\n";
    cout << "Enter choice\n";
    cin >> c;
    switch (c)
    {
        case 1:
            n.push();
            break;
        case 2:
            n.pop();
            break;
        case 3:
            n.display();
            break;
        case 4:
            exit(1);
        default:
            cout << "Incorrect choice\n";
    }
}
return 0;
getch();
}
```

Output for Implementation of Stacks using Linked List:

 E:\Stephen\DataStructure\LinkedList.exe

```
1=push
2=pop
3=display
4=exit
enter choice
1
enter element to be pushed
3

1=push
2=pop
3=display
4=exit
enter choice
1
enter element to be pushed
2

1=push
2=pop
3=display
4=exit
enter choice
3
stack elements are
2
3

1=push
2=pop
3=display
4=exit
enter choice
2
popped item is
2
1=push
2=pop
3=display
4=exit
enter choice
3
stack elements are
3
```

## Practical 4

Q) Implementation of stack applications like

### a) **Postfix Evaluation**

→ Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>

const int MAX=50;
using namespace std;

class postfix
{
private:
    int stack[MAX];
    int top, nn;
    char * s;
public: postfix();

    void setexpr(char * str);void push(int item);int pop();

    void calculate();void show();
};

postfix::postfix()
{
    top = -1;
}

void postfix::setexpr(char * str)
{
    s = str;
}

void postfix::push(int item)
{
    if (top == MAX - 1)
```

```

        cout << endl << "Stack is full";
    else
    {
        top++;
        stack[top] = item;
        cout << "\t" << item;
    }
}

int postfix::pop()
{
    if (top == -1)
    {
        cout << endl << "Stack is empty";
        return 0;
    } else
    {
        int data = stack[top];
        top--;
        cout << "\n" << data;
        return data;
    }
}

void postfix::calculate()
{
    int n1, n2, n3;
    while ( * s)

```

```

{
    if ( * s == ' ' || * s == '\t')
    {
        s++;
        continue;
    }
    if (isdigit( * s))
    {
        nn = * s - '0';
        push(nn);
    } else
    {
        n1 = pop();
        n2 = pop();
        switch ( * s)
        {
            case '+':
                n3 = n2 + n1;
                break;

            case '-':
                n3 = n2 - n1;
                break;

            case '/':
                n3 = n2 / n1;
                break;

            case '*':
                n3 = n2 * n1;
                break;

            default:

```

```

        cout << "\nUnknown operator";
    }
    push(n3);
}

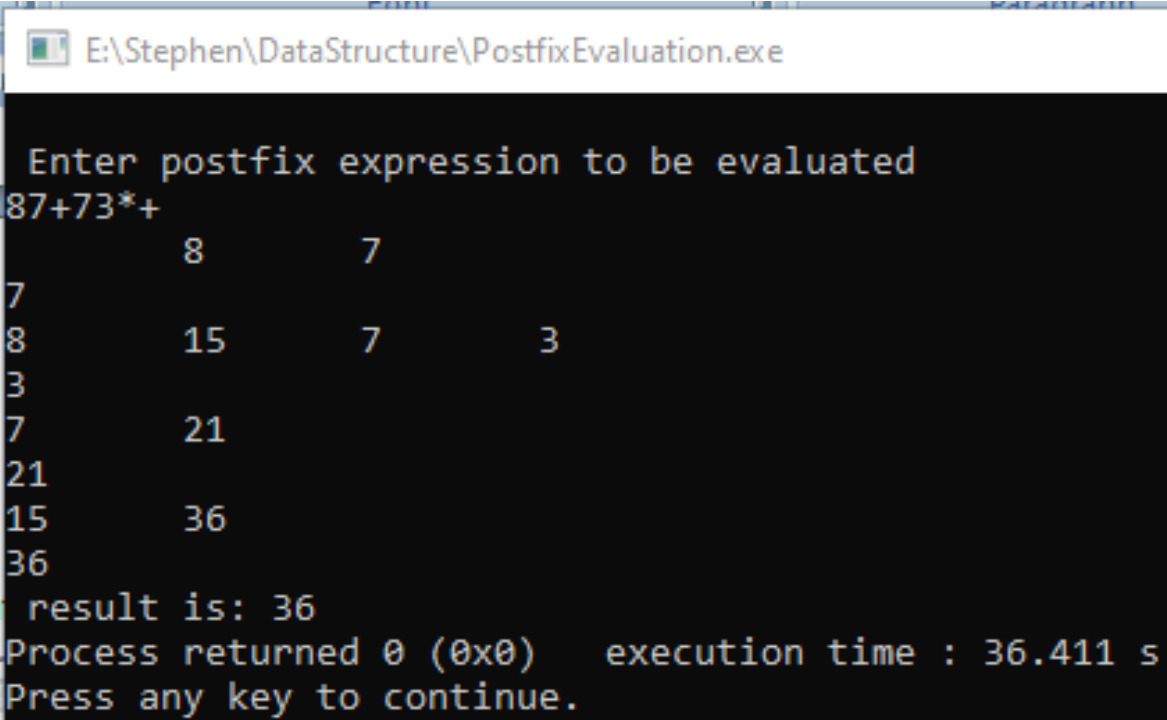
s++;
}
}

void postfix::show()
{
    nn = pop();
    cout << "\n result is: " << nn;
}

int main()
{
    char expr[MAX];
    cout << "\n Enter Postfix Expression to be evaluated \n";
    cin.getline(expr, MAX);
    postfix q;
    q.setexpr(expr);
    q.calculate();
    q.show();
    getch();
}

```

Output Postfix Evaluation:



```
E:\Stephen\DataStructure\PostfixEvaluation.exe

Enter postfix expression to be evaluated
87+73*+
      8      7
7
8      15      7      3
3
7      21
21
15      36
36
result is: 36
Process returned 0 (0x0)   execution time : 36.411 s
Press any key to continue.
```

## b) Balancing Parenthesis

→ Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>

# define MAX 5

using namespace std;
class stack
{
public:

    int count, top;
    char arr[MAX];

    stack() {

        count = 0;
        top = -1;
    }

    void push(char);void pop();
};

void stack::push(char d)
{

    int flag;
    if (count == MAX) {

        cout << "\nStack is full";

    } else

    {

        count++;

        top++;

        arr[top] = d;
```

```

    }
}

void stack::pop()
{
    if (top == -1)
    {
        cout << "\nStack is empty";
    } else
    {
        char d = arr[top];
        cout << d << endl;
        top--;
        count--;
    }
}

int main()
{
    stack s1;
    char exp[20];
    char ch;
    int num, n, i;
    cout << "\nEnter the expression\n";
    cin >> exp;
    num = strlen(exp);
    for (i = 0; i < num; i++)

```

```

{
    if (exp[i] == '(')
    {
        s1.push(exp[i]);
    } else if (exp[i] == ')')
    {
        s1.pop();
    }
}

if (s1.top != -1)
{
    cout << "\nNo matching parenthesis, Wrong expression\n";
} else
{
    cout << "\nMatching parenthesis found expressions is correct\n";
}

getch();

return 0;
}

```

Output Balancing Parenthesis:

---

 E:\Stephen\DataStructure\BalancingParenthesis.exe

```
Enter the expression
```

```
(a*b+C(-d)
```

```
(
```

```
No matching parenthesis,Wrong expression
```

```
Process returned 0 (0x0)   execution time : 369.268 s
```

```
Press any key to continue.
```

## Practical 5

Q) Implementation of different types of queues

### a) Linear Queue

→ Code:

```
#include<stdio.h>
#include<iostream>
#include<malloc.h>

using namespace std;

class node
{
    public: int info;

    node * link;

    void insert();

    void del();

    void display();

};

node * front = NULL, * rear = NULL;

int main()
{
    int choice;

    node n;

    while (1) {

        cout << "1.Insert\n";

        cout << "2.Delete\n";

        cout << "3.Display\n";

        cout << "4.Quit\n";
```

```

    cout << "Enter your choice";

    cin >> choice;

    switch (choice)
    {
    case 1:
        n.insert();

        break;

    case 2:
        n.del();

        break;

    case 3:
        n.display();

        break;

    case 4:
        exit(1);

    default:
        cout << "Wrong choice\n";

    }

}

return 0;

}

void node::insert()

{

    node * tmp;

    int added_item;

```

```

tmp = new node;

cout << "Input the element for adding queue";

cin >> added_item;

tmp -> info = added_item;

tmp -> link = NULL;

if (front == NULL)

    front = tmp;

else

    rear -> link = tmp;

rear = tmp;
}

void node::del()

{

    node * tmp;

    if (front == NULL)

        cout << "Queue UnderFlow\n";

    else

    {

        tmp = front;

        cout << "Deleted element";

        cin >> tmp -> info;

        front = front -> link;

        delete(tmp);

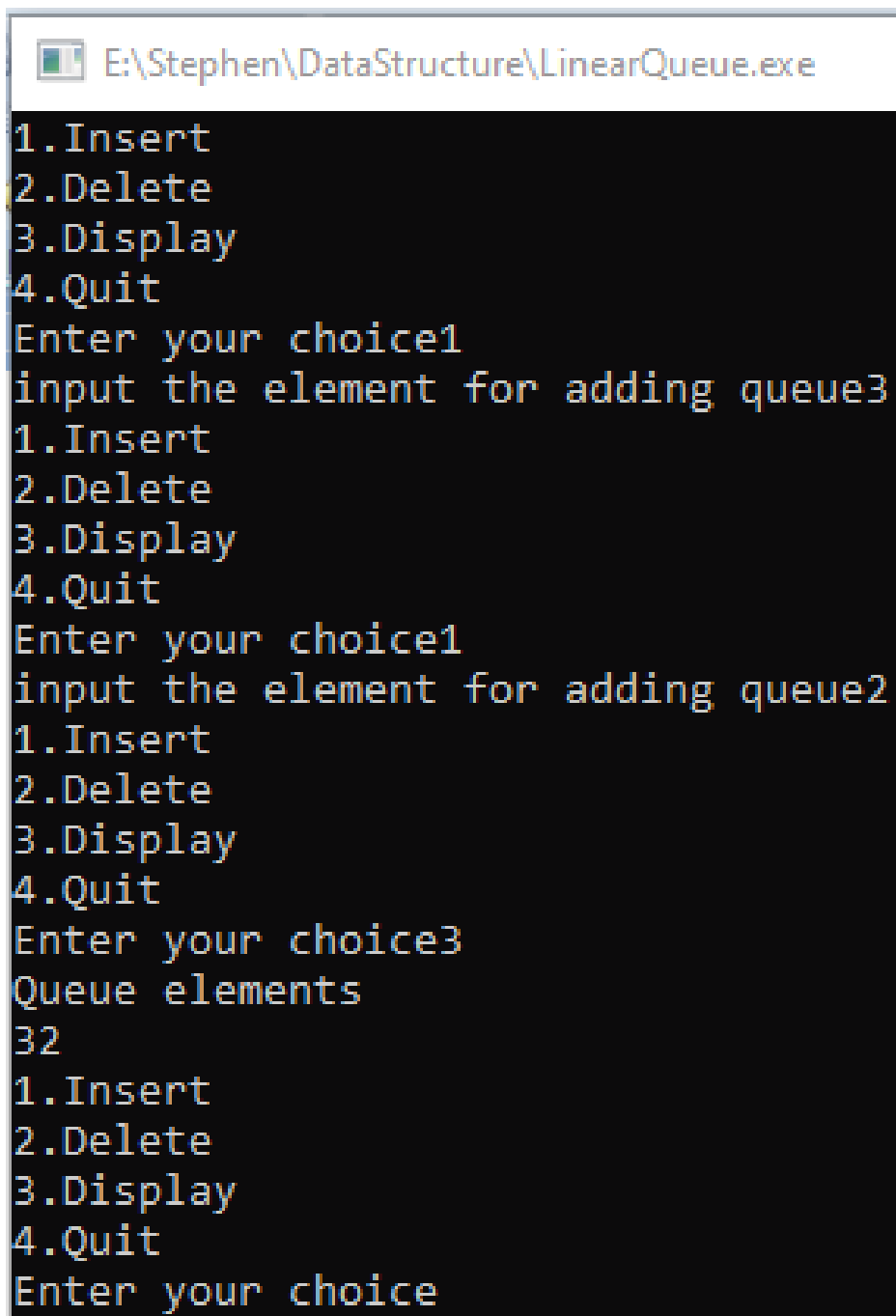
    }

}

```

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

Output for Implementation of Linear Queue:



```
E:\Stephen\DataStructure\LinearQueue.exe
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice1
input the element for adding queue3
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice1
input the element for adding queue2
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice3
Queue elements
32
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice
```

## b) Circular Queue

→ Code:

```
#include<iostream>
#include<stdio.h>
using namespace std;

class cirqueue
{
    int info;

    cirqueue * link;

public:
    void insert();
    void del();
    void display();
};

cirqueue * rear = NULL;

void cirqueue::insert() {
    int num;

    cirqueue * tmp;

    cout << "Enter the element for insertion:";

    cin >> num;

    tmp = new cirqueue;

    tmp -> info = num;

    if (rear == NULL) {
        rear = tmp;

        tmp -> link = rear;
    } else
```

```

{
    tmp -> link = rear -> link;
    rear -> link = tmp;
    rear = tmp;
}
}

void cirqueue::del()
{
    cirqueue * tmp, * q;
    if (rear == NULL)
    {
        cout << "Queue underflow\n";
        return;
    }
    if (rear -> link == rear)
    {
        tmp = rear;
        rear = NULL;
        delete(tmp);
        return;
    }
    q = rear -> link;
    tmp = q;
    rear -> link = q -> link;
    cout << "Deleted elements is" << tmp -> info;

```

```

    delete(tmp);
}

void cirqueue::display()
{
    cirqueue * q;
    if (rear == NULL)
    {
        cout << "Queue is empty\n";
        return;
    }
    q = rear -> link;
    cout << "Queue is:\n";
    while (q != rear)
    {
        cout << q -> info;
        q = q -> link;
    }
    cout << rear -> info;
}

int main()
{
    //clrscr();

    cirqueue s;

    int choice;

    char ch;

```

```

do
{
    cout << "1: Insert\n2: Delete\n3: Display Queue";
    cout << "\nEnter your choice(1-3): ";
    cin >> choice;
    switch (choice)
    {
        case 1:
            s.insert();
            break;

        case 2:
            s.del();
            break;

        case 3:
            s.display();
            break;

        default:
            cout << "Please enter correct choice(1-3)!!";
    }

    cout << "\n Press (y) to continued:";
    cin >> ch;
}

while (ch == 'y' || ch == 'Y');

return 0;
}

```

Output for Implementation of Circular Queue:

 E:\Stephen\DataStructure\CircularQueue.exe

```
1:insert
2:delete
3:display Queue
Enter your choice(1-3): 1
enter the element for insertion:32

press (y) to continued:y
1:insert
2:delete
3:display Queue
Enter your choice(1-3): 1
enter the element for insertion:54

press (y) to continued:y
1:insert
2:delete
3:display Queue
Enter your choice(1-3): 1
enter the element for insertion:23

press (y) to continued:y
1:insert
2:delete
3:display Queue
Enter your choice(1-3): 2
deleted elements is32
press (y) to continued:y
1:insert
2:delete
3:display Queue
Enter your choice(1-3): 3
Queue is:
5423
press (y) to continued:
```

## c) Double Ended Queue

→ Code:

```
#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node

{

    public:

        Node * next;

        int info;

}* front = NULL, * rear = NULL;

void fpush()

{

    int item;

    Node * temp, * q, * p;

    cout << "Enter the data: ";

    cin >> item;

    temp = new Node;

    temp -> info = item;

    temp -> next = NULL;

    if (front == NULL)

    {

        front = temp;

        rear = temp;

    } else
```

```

{
    q = front;
    while (q != NULL)
    {
        if (q -> next == NULL)
        {
            cout << "q->data : " << q -> info;

            p = q;
        }
        q = q -> next;
    }
    p -> next = temp;
    rear = temp;
}
}

void rpush()
{
    int data;

    Node * temp, * q, * p;

    cout << "Enter the data :- ";

    cin >> data;

    temp = new Node;

    temp -> info = data;

    temp -> next = NULL;

    if (front == NULL)
    {
        front = temp;
    }
}

```

```

    rear = temp;
} else
{
    q = front;
    while (q != NULL)
    {
        if (q -> next == NULL)
        {
            cout << "q->data : " << q -> info;

            p = q;
        }

        q = q -> next;
    }

    p -> next = temp;
    rear = temp;
}
}

void fpop()
{
    Node * temp, * p;
    temp = front;
    front = temp -> next;
    delete temp;

    /* while(temp!=NULL)
    {
        if(temp->next==NULL)

```

```

{
    p=temp;
}

temp=temp->next;

}*/

//delete p;
}

void rpop()
{
    Node * temp, * p;

    temp = front;

    front = temp -> next;

    delete temp;
}

void display()
{
    Node * tmp;

    tmp = front;

    cout << "Elements in queue are: " << endl;

    while (tmp != NULL)

    {

        cout << tmp -> info << " | ";

        tmp = tmp -> next;

    }

    cout << endl;
}

int main()

```

```
{  
    int ch;  
    while (1)  
    {  
        cout << "\n Press 1 for insert from front";  
        cout << "\n Press 2 for insert from rear";  
        cout << "\n Press 3 for Delete from front";  
        cout << "\n Press 4 for Delete from rear";  
        cout << "\n Press 5 for Display";  
        cout << "\n Press 6 for exit";  
        cin >> ch;  
        switch (ch)  
        {  
            case 1:  
                fpush();  
                break;  
            case 2:  
                rpush();  
                break;  
            case 3:  
                fpop();  
                break;  
            case 4:  
                rpop();  
                break;  
            case 5:
```

```
    display();  
    break;  
case 6:  
    exit(0);  
    break;  
default:  
    cout << "Please enter correct choice: ";  
}  
}  
getch();  
return 0;  
}
```

Output for Implementation of Double Ended Queue:

Select E:\Stephen\DataStructure\DoubleEndedQueue.exe

```
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit1
Enter the data :- 32
```

```
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit1
Enter the data :- 54
```

```
q->data : 32
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit5
```

```
Elements in queue are :
32 | 54 |
```

```
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit4
```

```
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit3
```

Select E:\Stephen\DataStructure\DoubleEndedQueue.exe

```
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit3
```

```
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit5
```

Elements in queue are :

```
press 1 for insert from front
press 2 for insert from rear
press 3 for Delete from front
press 4 for Delete from rear
press 5 for Display
press 6 for exit
```

## Practical 6

Q) Demonstrate application of queues

### a) Priority Queue

→ Code:

```
#include<iostream>
#include<stdio.h>
using namespace std;
class node
{
    int priority;
    int info;
    node * link;
public:
    void insert();
    void del();
    void display();
}* front = NULL;
void node::insert()
{
    node * tmp, * q;
    int added_item, item_priority;
    tmp = new node;
    cout << "Input the item value to be added in the queue: \n";
    cin >> added_item;
    cout << "enter its priority:";
    cin >> item_priority;
    tmp -> info = added_item;
```

```

tmp -> priority = item_priority;
if (front == NULL || item_priority < front -> priority)
{
    tmp -> link = front;
    front = tmp;
} else
{
    q = front;
    while (q -> link != NULL && q -> link -> priority <= item_priority)
        q = q -> link;
    tmp -> link = q -> link;
    q -> link = tmp;
}
}

void node::del()
{
    node * tmp;
    if (front == NULL)
    {
        cout << "queue underflow";
    } else
    {
        tmp = front;
        cout << "deleted item is" << tmp -> info;
        front = front -> link;
        delete tmp;
    }
}

```

```

    }
}

void node::display()
{
    node * ptr;
    ptr = front;
    if (front == NULL)
        cout << "queue is empty\n";
    else
    {
        cout << "queue is: ";
        cout << "priority item\n";
        while (ptr != NULL)
        {
            cout << "\t" << ptr -> priority << "\t" << ptr -> info;
            ptr = ptr -> link;
        }
    }
}

int main()
{
    node n1;
    int choice;
    while (1)
    {
        cout << "\n1. Insert\n";
        cout << "2. Delete\n";
    }
}

```

```
cout << "3. Display\n";
cout << "Enter your choice: ";
cin >> choice;
switch (choice)
{
case 1:
    n1.insert();
    break;
case 2:
    n1.del();
    break;
case 3:
    n1.display();
    break;
default:
    cout << "Wrong choice \n";
}
}
return 0;
}
```

Output for demonstration on Priority Queue:

E:\Stephen\DataStructure\PriorityQueue.exe

```
1.insert
2.delete
3.display
enter your choice1
input the item value to be added in the queue:32
enter its priority:1

1.insert
2.delete
3.display
enter your choice1
input the item value to be added in the queue:23
enter its priority:2

1.insert
2.delete
3.display
enter your choice1
input the item value to be added in the queue:74
enter its priority:3

1.insert
2.delete
3.display
enter your choice3
queue is:priority item
          1      32      2      23      3      74

1.insert
2.delete
3.display
enter your choice2
deleted item is32
1.insert
2.delete
3.display
enter your choice12
wrong choice:
```

```
1.insert
2.delete
3.display
enter your choice1
input the item value to be added in the queue:32
enter its priority:1

1.insert
2.delete
3.display
enter your choice1
input the item value to be added in the queue:23
enter its priority:2

1.insert
2.delete
3.display
enter your choice1
input the item value to be added in the queue:45
enter its priority:3

1.insert
2.delete
3.display
enter your choice2
deleted item is32
1.insert
2.delete
3.display
enter your choice3
queue is:priority item
          2          23          3          45

1.insert
2.delete
3.display
enter your choice
```

## b) Breadth First Search

→ Code:

```
#include<iostream>
#include<conio.h>
#include<iomanip>
using namespace std;
int adj[10][10], n, visited[10];
void bfs(int v)
{
    int q[10], front = -1, rear = -1, i;
    visited[v] = 1;
    q[++rear] = v;
    cout << "Visiting order...\n";
    while (front != rear)
    {
        v = q[++front];
        cout << v;
        for (i = 0; i < n; i++)
        {
            if (!visited[i] && adj[v][i])
            {
                visited[i] = 1;
                q[++rear] = i;
            }
        }
    }
}
```

```

int main()
{
    int i, j, m, a, b, v;
    char c;
    cout << "\nEnter the no of nodes and no of edges: \n";
    cin >> n >> m;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            adj[i][j] = 0;
        }
    }
    for (i = 1; i <= m; i++)
    {
        cout << "Enter an edge: \n";
        cin >> a >> b;
        adj[a][b] = 1;
        adj[b][a] = 1;
    }
    do
    {
        cout << "Adjacency matrix\n";
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)

```

```
{  
    cout << setw(4) << adj[i][j];  
}  
cout << "\n";  
}  
cout << "Enter initial value: \n";  
cin >> v;  
for (i = 0; i < n; i++)  
{  
    visited[i] = 0;  
}  
bfs(v);  
cout << "Do u wish to continue? (y/n)";  
cin >> c;  
}  
while (c != 'n');  
getch();  
}
```

Output for demonstration on Breadth First Search:

```
E:\Stephen\DataStructure\BreadthFirstSearch.exe

Enter the no of nodes And no of edges:4
4
enter an edge
2
3
enter an edge
3
1
enter an edge
3
4
enter an edge
1
2
Adjacency matrix
  0  0  0  0
  0  0  1  1
  0  1  0  1
  0  1  1  0
enter initial value
2
Visiting order...
213Do u wish to continue(y/n)????
```

## Practical 7

Q) Implementation of all types of linked list

### a) **Singly Linked List**

→ Code:

```
#include<iostream>
#include<stdio.h>
#include<malloc.h>
using namespace std;

class node {
public: int info;
node * link;
void create_list(int data);
void addatbeg(int data);
void addafter(int data, int pos);
void display();
void del(int data);
void search(int data);
void count();
void reverse();
void sort();
};
node * start;
void node::create_list(int data) {
node * q, * tmp;
tmp = new node;
tmp -> info = data;
tmp -> link = NULL;
if (start == NULL)
start = tmp;
else {
q = start;
while (q -> link != NULL)
q = q -> link;

q -> link = tmp;
}
}
void node::addatbeg(int data) {
node * tmp;
tmp = new node;
tmp -> info = data;
tmp -> link = start;
start = tmp;
}
void node::addafter(int data, int pos) {
node * tmp, * q;
```

```

int i;
q = start;
for (i = 0; i < pos - 1; i++) {
    q = q -> link;
    if (q == NULL) {

        cout << "There are less than " << pos << " elements";
        return;

    }
}
tmp = new node;
tmp -> link = q -> link;
tmp -> info = data;
q -> link = tmp;
}
void node::del(int data) {
    node * tmp, * q;
    if (start -> info == data) {
        tmp = start;
        start = start -> link;
        delete tmp;
        return;
    }
    q = start;
    while (q -> link -> link != NULL) {
        if (q -> link -> info == data) {
            tmp = q -> link;
            q -> link = tmp -> link;
            delete tmp;
            return;

        }
        q = q -> link;
    }
    if (q -> link -> info == data) {
        tmp = q -> link;
        delete tmp;
        q -> link = NULL;
        return;
    }
    cout << "Element " << data << " not found\n";
}

void node::search(int data) {
    node * ptr = start;
    int pos = 1;
    while (ptr != NULL) {
        if (ptr -> info == data) {
            cout << "Element " << data << " found at position " << pos << "\n";
            return;
        }
    }
}

```

```

    ptr = ptr -> link;
    pos++;
}
if (ptr == NULL)
    cout << "Element " << data << " not found in list\n";
}

void node::count() {
    node * q = start;
    int cnt = 0;
    while (q != NULL) {
        q = q -> link;
        cnt++;
    }
    cout << "Number of elements are: " << cnt;
}

void node::reverse() {
    node * p1, * p2, * p3;
    if (start -> link == NULL)
        return;
    p1 = start;
    p2 = p1 -> link;
    p3 = p2 -> link;
    p1 -> link = NULL;

    p2 -> link = p1;
    while (p3 != NULL)

    {
        p1 = p2;
        p2 = p3;
        p3 = p3 -> link;
        p2 -> link = p1;
    }
    start = p2;
}

void node::sort() {
    int t;
    node * q, * tmp;
    if (start == NULL)
        cout << "\nThere are no elements in the list";
    else {
        q = start;
        while (q != NULL) {
            tmp = q -> link;
            while (tmp != NULL)

            {
                if (q -> info > tmp -> info) {
                    t = q -> info;
                    q -> info = tmp -> info;
                    tmp -> info = t;
                }
            }
            q = tmp;
        }
    }
}

```

```

    }
    tmp = tmp -> link;
}
q = q -> link;
}
cout << "\n\n\t List sorted successfully\n";
}
}
void node::display() {
    node * q;
    if (start == NULL) {
        cout << "List is empty\n";
        return;
    }
    q = start;
    cout << "List is: \n";

    while (q != NULL)

    {
        cout << "" << q -> info << "\t";
        q = q -> link;
    }
    cout << "\n";
}
int main() {
    node n1;
    int ch, n2, m, pos, i;
    start = NULL;

    while (1) {
        cout << "1. Create list\n";
        cout << "2. Add at beginning\n";
        cout << "3. Add after\n";
        cout << "4. Delete\n";
        cout << "5. Search\n";
        cout << "6. Count\n";
        cout << "7. Reverse\n";
        cout << "8. Sort\n";
        cout << "9. Display\n";
        cout << "10. Quit\n";
        cout << "Enter your choice: ";

        cin >> ch;
        switch (ch) {
            case 1:
                cout << "How many nodes you want: ";
                cin >> n2;
                for (i = 0; i < n2; i++) {
                    cout << "Enter the element: \n";
                    cin >> m;
                    n1.create_list(m);
                }

```

```

        break;
    case 2:
        cout << "Enter the element: \n";
        cin >> m;
        n1.addatbeg(m);
        break;
    case 3:
        cout << "Enter the element: \n";
        cin >> m;

        cout << "Enter the position after which this element is inserted: ";
        cin >> pos;

        n1.addafter(m, pos);
        break;

    case 4:
        if (start == NULL) {
            cout << "List is empty";
            continue;
        }
        cout << "Enter the element for deletion: \n";
        cin >> m;
        n1.del(m);
        break;
    case 5:

        cout << "Enter the element to be searched: \n";
        cin >> m;

        n1.search(m);
        break;
    case 6:
        n1.count();
        break;
    case 7:
        n1.reverse();
        break;
    case 8:
        n1.sort();
        break;

    case 9:
        n1.display();
        break;
    case 10:
        exit(1);
    default:
        cout << "Wrong choice";
    }
}
return 0;
}

```

Output for Implementation of Singly Linked List:

 E:\Stephen\DataStructure\SinglyLinkedList.exe

```
Enter your choice: 1
How many nodes you want:4
Enter the element:10
Enter the element:30
Enter the element:70
Enter the element:100
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice: 9
List is :
10      30      70      100
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice:
```

 E:\Stephen\DataStructure\SinglyLinkedList.exe

```
Enter your choice: 2
Enter the element:20
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice: 9
List is :
20      10      30      70      100
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice:
```

E:\Stephen\DataStructure\SinglyLinkedList.exe

```
Enter your choice: 3
Enter the element: 40
Enter the position after which this element is inserted: 5
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice: 9
List is :
20      10      30      70      100      40
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice:
```

E:\Stephen\DataStructure\SinglyLinkedList.exe

```
Enter your choice: 4
Enter the element for deletion: 70
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice: 9
List is :
20      10      30      100      40
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice:
```

E:\Stephen\DataStructure\SinglyLinkedList.exe

```
Enter your choice: 5
Enter the element to be searched
100
Element 100 found at position
41.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice: 6
Number of elements are:51.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice:
```

E:\Stephen\DataStructure\SinglyLinkedList.exe

```
Enter your choice: 7
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
Enter your choice: 9
List is :
40      100      30      10      20
1.Create list
2.Add at beginning
3.Add after
4.Delete
5.Search
6.Count
7.Revese
8.Sort
9.Display
10.Quit
```

E:\Stephen\DataStructure\SinglyLinkedList.exe

Enter your choice: 8

List sorted successfully

1.Create list

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Count

7.Revese

8.Sort

9.Display

10.Quit

Enter your choice: 9

List is :

10      20      30      40      100

1.Create list

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Count

7.Revese

8.Sort

9.Display

10.Quit

Enter your choice:

## b) Doubly Linked Llist

→Code:

```
#include<process.h>
#include<conio.h>
#include<iostream>
using namespace std;

struct dnode {
    int data;
    struct dnode * prev;
    struct dnode * next;
};

class link {
    dnode * list;

public:
    dnode * head;
    link() {
        list = NULL;

        head = NULL;
    }

    void get_list();
    void display_list();
    void merge(link, link);
    friend void union_list(dnode * , dnode * );
    friend void intersact(dnode * , dnode * );
};

void link::get_list() {
    struct dnode * q, * tmp;
    int d, n;

    tmp = new dnode;

    cout << "Enter how many elements you want to enter: \n";
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cout << "Enter the elements: ";
        cin >> d;
        tmp = new dnode;
        tmp -> data = d;
        tmp -> next = NULL;
        tmp -> prev = NULL;
        if (head == NULL)
            head = tmp;
        else {
```

```

    q = head;
    while (q -> next != NULL)
        q = q -> next;

    q -> next = tmp;
    tmp -> prev = q;
}
}
}

void link::display_list() {
    dnode * q;
    q = head;

    if (q == NULL) {
        cout << "No data is in the List ..";
    }

    while (q != NULL) {
        cout << "" << q -> data;
        q = q -> next;

        cout << "\t";
    }

    cout << "\n";
}

void intersact(dnode * l1, dnode * l2) {
    dnode * h;

    h = l2;
    while ((l1 != NULL) && (l2 != NULL)) {
        if (l1 -> data == l2 -> data) {
            cout << l1 -> data << "\t";
            l1 = l1 -> next;
            l2 = l2 -> next;
        } else if (l1 -> data > l2 -> data)

            l2 = l2 -> next;
        else
            l1 = l1 -> next;
    }
}

void union_list(dnode * l1, dnode * l2)

{
    cout << endl;

    dnode * h;
    h = l1;
    while (l1 != NULL) {

```

```

    cout << l1 -> data << "\t";

    l1 = l1 -> next;
}

int flag = 0;
while (l2 != NULL) {
    l1 = h;
    flag = 0;
    while (l1 != NULL) {
        if (l1 -> data == l2 -> data)

        {

            flag = 1;

            break;
        }

        l1 = l1 -> next;
    }

    if (flag == 0) {
        cout << l2 -> data << "\t";
    }

    l2 = l2 -> next;
}
}

void link::merge(link l1, link l2)

{
    struct dnode * nxt_node, * pre_node, * pptr, * qptr;
    int dat;

    pptr = l1.head;
    qptr = l2.head;
    head = nxt_node = pre_node = NULL;
    while (pptr != NULL && qptr != NULL) {
        if (pptr -> data <= qptr -> data) {
            dat = pptr -> data;
            pptr = pptr -> next;
        } else

        {

            dat = qptr -> data;
            qptr = qptr -> next;
        }

        nxt_node = new dnode;
        nxt_node -> data = dat;

```

```

nxt_node -> next = NULL;
nxt_node -> prev = NULL;
if (head == NULL)

    head = nxt_node;
else {
    pre_node -> next = nxt_node;
    nxt_node -> prev = pre_node;
}

pre_node = nxt_node;
}

if (pptr == NULL) {
    while (qptr != NULL) {
        nxt_node = new dnode;
        nxt_node -> data = qptr -> data;
        nxt_node -> next = NULL;
        nxt_node -> prev = NULL;
        if (head == NULL)

            head = nxt_node;

        else {
            pre_node -> next = nxt_node;
            nxt_node -> prev = pre_node;
        }

        pre_node = nxt_node;
        qptr = qptr -> next;
    }
} else if (qptr == NULL) {
    while (pptr != NULL) {
        nxt_node = new dnode;

        nxt_node -> data = pptr -> data;
        nxt_node -> next = NULL;
        nxt_node -> prev = NULL;

        if (head == NULL)
            head = nxt_node;
        else {
            pre_node -> next = nxt_node;
            nxt_node -> prev = pre_node;
        }

        pre_node = nxt_node;
        pptr = pptr -> next;
    }
}

cout << "The Lists are merged" << endl;

```

```

    return;
}

int main() {
    link l;

    link l1, l2, l3;
    int ch;
    do {
        cout << " Operations on list..." << endl;
        cout << "1. Union" << endl;
        cout << "2. Merge" << endl;
        cout << "3. Intersection" << endl;
        cout << "4. Exit" << endl;
        cout << " Enter your choice: " << endl;
        cin >> ch;
        switch (ch)

        {

        case 1:

            cout << "Enter the first List: \n" << endl;
            l1.get_list(); // to create a first list

            cout << "Enter the second List: \n" << endl;
            l2.get_list(); // to create a second list

            cout << "The first list is " << endl;
            l1.display_list();
            cout << endl;
            cout << "The second list is " << endl;
            l2.display_list();

            cout << endl << endl << "Union of First two List... " << endl;
            union_list(l1.head, l2.head);
            getch();
            break;

        case 2:

            cout << "Enter the First List in ascending order: \n" << endl;
            l1.get_list(); // to create a first list

            cout << "Enter the Second List in ascending order: \n" << endl;
            l2.get_list(); // to create a second list

            cout << "The first list is \n" << endl;
            l1.display_list();
            cout << "The second list is \n" << endl;
            l2.display_list();
            l3.merge(l1, l2);
            l3.display_list();

```

```

    getch();
    break;

case 3:

    cout << "Enter the First List in ascending order. " << endl;
    l1.get_list(); // to create a first list

    cout << "Enter the Second List in ascending order. " << endl;
    l2.get_list(); // to create a second list cout<<"The first list is "<<endl;

    l1.display_list();
    cout << "The second list is \n" << endl;
    l2.display_list();
    cout << endl << endl << "Intersection of first two list... " << endl;
    intersact(l1.head, l2.head);
    getch();
    break;

case 4:
    exit(1);

default:
    cout << " The option is invalid...";
}

    getch();
} while (1);
getch();
}

```

## Output for Implementation of Doubly Linked List:

```
E:\Stephen\DataStructure\DoublyLinkedList.exe
Operations onlist..
1.Union
2.Merge
3.Intersection
4.Exit
Enter ur choice:
1
Enter the First List.
Enter how many elements u want to enter 4
Enter the elements 15
Enter the elements 32
Enter the elements 45
Enter the elements 57
Enter the Second List.
Enter how many elements u want to enter 4
Enter the elements 57
Enter the elements 84
Enter the elements 34
Enter the elements 19
The first list is
15      32      45      57

The second list is
57      84      34      19

Union of First two List...
15      32      45      57      84      34      19
```

```
E:\Stephen\DataStructure\DoublyLinkedList.exe
Operations on list..
1.Union
2.Merge
3.Intersection
4.Exit
Enter ur choice:
3
Enter the First List in ascending order.
Enter how many elements u want to enter 4
Enter the elements 12
Enter the elements 23
Enter the elements 32
Enter the elements 45
Enter the Second List in ascending order.
Enter how many elements u want to enter 4
Enter the elements 21
Enter the elements 32
Enter the elements 47
Enter the elements 58
12      23      32      45
The second list is
21      32      47      58

Intersaction of First two list...
32
```

## c) Circular Linked List

→Code:

```
#include<iostream>

using namespace std;

class singly_circular
{
public:
    int flag = true;
    int pos, i, value, count = 0;

    struct node
    {
        int data;
        struct node * next;
        struct node * prev;
    };

    struct node * tmp = NULL;
    struct node * start = NULL;
    struct node * last = NULL;
    struct node * p = NULL;
    struct node * ptr = NULL;

    void create(int x)
    {
        tmp = new node;
        tmp -> data = x;
        if (last == NULL)
        {
            last = tmp;
```

```

    tmp -> next = last;
} else
{
    tmp -> next = last -> next;
    last -> next = tmp;
    last = tmp;
}
}

void add_atbegin(int x)
{
    if (last == NULL)
    {
        cout << "List is empty\n";
    }

    tmp = new node;
    tmp -> data = x;
    tmp -> next = last -> next;
    last -> next = tmp;
}

void add_after(int x, int pos)
{
    if (last == NULL)
    {
        cout << "List is empty\n";
    }

    p = last -> next;

```

```

for (int i = 0; i < pos - 1; i++)
{
    p = p -> next;
    if (p == last -> next)
    {
        cout << "Position does not exist\n";
        //break;
    }
}

tmp = new node;
tmp -> next = p -> next;
tmp -> data = x;
p -> next = tmp;
if (p == last)
{
    last = tmp;
}
}

void del(int x)
{
    //p=last->next;
    if (last -> next == last && last -> data == x) // for only one node
    {
        tmp = last;
        last = NULL;
        delete(tmp);
        return;
    }
}

```

```

}
p = last -> next;
if (p -> data == x) //first element deleted
{
    tmp = p;
    last -> next = p -> next;
    delete(tmp);
    return;
}
while (p -> next != last)
{
    if (p -> next -> data == x)
    {
        tmp = p -> next;
        p -> next = tmp -> next;
        delete(tmp);
        //cout<<"Deleted item is: "<<x;
        return;
    } //delete element in between
    p = p -> next;
}
if (p -> next -> data == x)
{
    tmp = p -> next;
    p -> next = last -> next;
    delete(tmp);
}

```

```

    last = p;
    return;
} //last element deleted
cout << "Element not found\n";
}
void search1(int x)
{
    int pos = 1;
    while (p -> next != last)
    {
        if (p -> data == x)
        {
            cout << "Element found at position " << pos - 1 << ".\n";
        }
        p = p -> next;
        pos++;
    }
    if (p == NULL)
        cout << "Item not found.\n";
}
void sort()
{
    int x;
    if (last == NULL)
    {
        cout << "List is empty. \n\n";
    }
}

```

```

p = last -> next;
while (p != last)
{
    ptr = p -> next;
    while (ptr != last -> next)
    {
        if (ptr != last -> next)
        {
            if (p -> data > ptr -> data)
            {
                x = p -> data;
                p -> data = ptr -> data;
                ptr -> data = x;
            }
        }
        ptr = ptr -> next;
    }
    p = p -> next;
}
}

int count1()
{
    if (last == NULL)
    {
        cout << "List is empty. \n\n";
    } else

```

```

{
    p = last -> next;
    while (p != last)
    {
        count++;
        p = p -> next;
    }
    count++;
    cout << "Number of elements are " << count << "\n";
}
}

void display()
{
    if (last == NULL)
    {
        cout << "List is empty. \n\n";
        return;
    }
    p = last -> next;
    cout << "\nSingly Circular Linked List: \n";
    while (p != last)
    {
        cout << p -> data << " -> ";
        p = p -> next;
    }
    cout << last -> data << "\n\n";
}

```

```

};

int main()
{
    singly_circular d;
    int x, ch;
    int pos;
    while (ch != 9)
    {
        cout << "1. Create a list\n2. Add at begin\n3. Add after\n4. Search\n";
        cout << "5. Sort\n6. Count\n7. Display\n8. Delete\n9. Exit\n";
        cout << "Enter the choice:\n";
        cin >> ch;
        switch (ch)
        {
            case 1:
                cout << "Enter the value: \n";
                cin >> x;
                d.create(x);
                d.display();
                break;
            case 2:
                cout << "Enter the value: \n";
                cin >> x;
                d.add_atbegin(x);
                d.display();
                break;

```

case 3:

```
cout << "Enter the position: \n";
```

```
cin >> pos;
```

```
cout << "Enter the value:\n";
```

```
cin >> x;
```

```
d.add_after(x, pos);
```

```
d.display();
```

```
break;
```

case 4:

```
cout << "Enter element to be searched: \n";
```

```
cin >> x;
```

```
d.search1(x);
```

```
d.display();
```

```
break;
```

case 5:

```
cout << "Before sorting: ";
```

```
d.display();
```

```
d.sort();
```

```
cout << "After sorting: ";
```

```
d.display();
```

```
break;
```

case 6:

```
d.count1();
```

```
d.display();
```

```
break;
```

case 7:

```
d.display();
```

```
        break;
    case 8:
        cout << "Enter the element to be delete: \n";
        cin >> x;
        d.del(x);
        d.display();
        break;
    case 9:
        break;
    default:
        cout << "Wrong choice\n";
    }
}
return 0;
}
```

## Output for Implementation of Circular Linked List:

E:\Stephen\DataStructure\CircularLinkedList.exe	E:\Stephen\DataStructure\CircularLinkedList.exe
<pre>1.Create a list 2.Add at begin 3.Add after 4.Search 5.Sort 6.Count 7.Display 8.Delete 9.Exit Enter the choice: 1 Enter the value : 12  Singly Circular Linked List : 12  1.Create a list 2.Add at begin 3.Add after 4.Search 5.Sort 6.Count 7.Display 8.Delete 9.Exit Enter the choice:</pre>	<pre>1.Create a list 2.Add at begin 3.Add after 4.Search 5.Sort 6.Count 7.Display 8.Delete 9.Exit Enter the choice: 1 Enter the value : 32  Singly Circular Linked List : 12 -&gt; 32  1.Create a list 2.Add at begin 3.Add after 4.Search 5.Sort 6.Count 7.Display 8.Delete 9.Exit Enter the choice:</pre>

E:\Stephen\DataStructure\CircularLinkedList.exe

```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
3
Enter the position :
4
Enter the value :
57
Singly Circular Linked List :
12 -> 32 -> 98 -> 15 -> 57
```

```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
```

E:\Stephen\DataStructure\CircularLinkedList.exe

```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
5
Before sorting -
Singly Circular Linked List :
12 -> 32 -> 98 -> 15 -> 57
After sorting -
Singly Circular Linked List :
12 -> 15 -> 32 -> 57 -> 98
```

```
1.Create a list
2.Add at begin
3.Add after
4.Search
5.Sort
6.Count
7.Display
8.Delete
9.Exit
Enter the choice:
6
Number of element are 5
Singly Circular Linked List :
12 -> 15 -> 32 -> 57 -> 98
```

## Practical 8

Q) Demonstrate application of linked list

### a) Polynomial addition

→ Code:

```
#include<iostream>

using namespace std;

struct Node {

    int coeff;

    int pow;

    struct Node * next;

};

void create_node(int x, int y, struct Node ** temp)
{
    struct Node * r, * z;

    z = * temp;

    if (z == NULL) {

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

        r -> coeff = x;

        r -> pow = y;

        * temp = r;

        r -> next = (struct Node * ) malloc(sizeof(struct Node));

        r = r -> next;

        r -> next = NULL;

    } else {

        r -> coeff = x;

        r -> pow = y;

        r -> next = (struct Node * ) malloc(sizeof(struct Node));
```

```

    r = r -> next;

    r -> next = NULL;
}
}

void polyadd(struct Node * p1, struct Node * p2, struct Node * result)
{
    while (p1 -> next && p2 -> next) {
        if (p1 -> pow > p2 -> pow) {
            result -> pow = p1 -> pow;
            result -> coeff = p1 -> coeff;
            p1 = p1 -> next;
        } else if (p1 -> pow < p2 -> pow) {
            result -> pow = p2 -> pow;
            result -> coeff = p2 -> coeff;
            p2 = p2 -> next;
        } else {
            result -> pow = p1 -> pow;
            result -> coeff = p1 -> coeff + p2 -> coeff;
            p1 = p1 -> next;
            p2 = p2 -> next;
        }
        result -> next = (struct Node * ) malloc(sizeof(struct Node));
        result = result -> next;
        result -> next = NULL;
    }
    while (p1 -> next || p2 -> next) {
        if (p1 -> next) {
            result -> pow = p1 -> pow;

```

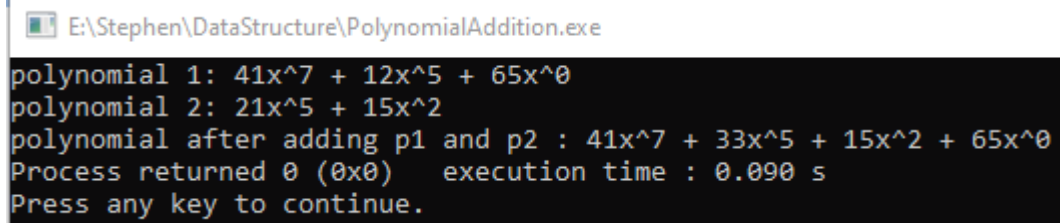
```

    result -> coeff = p1 -> coeff;
    p1 = p1 -> next;
}
if (p2 -> next) {
    result -> pow = p2 -> pow;
    result -> coeff = p2 -> coeff;
    p2 = p2 -> next;
}
result -> next = (struct Node * ) malloc(sizeof(struct Node));
result = result -> next;
result -> next = NULL;
}
}
void printpoly(struct Node * node) {
    while (node -> next != NULL) {
        printf("%dx^%d", node -> coeff, node -> pow);
        node = node -> next;
        if (node -> next != NULL)
            printf(" + ");
    }
}
int main() {
    struct Node * p1 = NULL, * p2 = NULL, * result = NULL;
    create_node(41, 7, & p1);
    create_node(12, 5, & p1);
    create_node(65, 0, & p1);
    create_node(21, 5, & p2);

```

```
create_node(15, 2, & p2);  
printf("polynomial 1: ");  
printpoly(p1);  
printf("\npolynomial 2: ");  
printpoly(p2);  
result = (struct Node * ) malloc(sizeof(struct Node));  
polyadd(p1, p2, result);  
printf("\npolynomial after adding p1 and p2 : ");  
printpoly(result);  
return 0;  
}
```

Output for demonstration on Polynomial Addition:



The screenshot shows a Windows command prompt window with the title bar "E:\Stephen\DataStructure\PolynomialAddition.exe". The output text is as follows:

```
polynomial 1: 41x^7 + 12x^5 + 65x^0  
polynomial 2: 21x^5 + 15x^2  
polynomial after adding p1 and p2 : 41x^7 + 33x^5 + 15x^2 + 65x^0  
Process returned 0 (0x0)   execution time : 0.090 s  
Press any key to continue.
```

## b) Sparse Matrix

→Code:

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

class Node
{
public:
    int row;

    int col;

    int data;

    Node * next;
};

void create_new_node(Node ** p, int row_index, int col_index, int x)
{
    Node * temp = * p;

    Node * r;

    if (temp == NULL)
    {
        temp = new Node();

        temp -> row = row_index;

        temp -> col = col_index;

        temp -> data = x;

        temp -> next = NULL;

        * p = temp;
    } else
    {
        while (temp -> next != NULL)
```

```

        temp = temp -> next;
    r = new Node();
    r -> row = row_index;
    r -> col = col_index;
    r -> data = x;
    r -> next = NULL;
    temp -> next = r;
}
}
void printList(Node * start)
{
    Node * ptr = start;
    cout << "Row_position: ";
    while (ptr != NULL)
    {
        cout << ptr -> row << " ";
        ptr = ptr -> next;
    }
    cout << endl;
    cout << "Column position: ";
    ptr = start;
    while (ptr != NULL)
    {
        cout << ptr -> col << " ";
        ptr = ptr -> next;
    }
    cout << endl;

```

```

cout << "Value: ";

ptr = start;

while (ptr != NULL)

{
    cout << ptr -> data << " ";

    ptr = ptr -> 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 * first = NULL;

    for (int i = 0; i < 4; i++)

    {

        for (int j = 0; j < 5; j++)

        {

            if (sparseMatrix[i][j] != 0)

                create_new_node( & first, i, j,

                    sparseMatrix[i][j]);

        }

    }


    printList(first);

    return 0;

}

```

Output for demonstration on Sparse Matrix:

 E:\Stephen\DataStructure\SparseMatrix.exe

```
row_position:0 0 1 1 3 3
column_position:2 4 2 3 1 2
Value:3 4 5 7 2 6
Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

## Practical 9

Q) Create and perform various operations on

# Binary Search Tree

→ Code:

```
#include<iostream.h>

#include<process.h>

#include<conio.h>

using namespace std;
struct node {
    int data;
    struct node * left;

    struct node * right;
};
class BST {
public: node * tree;
    BST() {
        tree = NULL;
    }
    void createTree(node ** , int item);
    void preorder(node * );
    void inorder(node * );
    void postorder(node * );
    int totalNodes(node * );
    void removeTree(node ** );
    void findsmallestNode(node * );
    void findLargestNode(node * );
    void deleteNode(int);
};
void BST::createTree(node ** tree, int item) {
    if ( * tree == NULL) {
        * tree = new node;
        ( * tree) -> data = item;
        ( * tree) -> left = NULL;
        ( * tree) -> right = NULL;
    } else {
        if (( * tree) -> data > item)
            createTree( & (( * tree) -> left), item);
        else
            createTree( & (( * tree) -> right), item);
    }
}
void BST::preorder(node * tree) {
    if (tree != NULL) {

        cout << " " << tree -> data;
```

```

        preorder(tree -> left);

        preorder(tree -> right);
    }
}

void BST::inorder(node * tree) {
    if (tree != NULL) {
        inorder(tree -> left);
        cout << " " << tree -> data;
        inorder(tree -> right);
    }
}

void BST::postorder(node * tree) {
    if (tree != NULL) {

        postorder(tree -> left);
        postorder(tree -> right);
        cout << " " << tree -> data;
    }
}

int BST::totalNodes(node * tree) {
    if (tree == NULL)
        return 0;
    else
        return (totalNodes(tree -> left) + totalNodes(tree -> right) + 1);
}

void BST::removeTree(node ** tree) {
    if ((* tree) != NULL) {
        removeTree( & (* tree) -> left);
        removeTree( & (* tree) -> right);
        delete( * tree);
    }
}

void BST::findsmallestNode(node * tree) {
    if (tree == NULL || tree -> left == NULL)
        cout << tree -> data;
    else
        findsmallestNode(tree -> left);
}

node * find_Insucc(node * curr) {
    node * succ = curr -> right;
    if (succ != NULL) {
        while (succ -> left != NULL)
            succ = succ -> left;
    }
    return (succ);
}

void BST::findLargestNode(node * tree) {
    if (tree == NULL || tree -> right == NULL)
        cout << tree -> data;
    else
        findLargestNode(tree -> right);
}

```

```

}
void BST::deleteNode(int item) {
    node * curr = tree, * succ, * pred;
    int flag = 0, delcase;
    while (curr != NULL && flag != 1) {
        if (item < curr -> data) {
            pred = curr;
            curr = curr -> left;
        } else if (item > curr -> data) {

            pred = curr;
            curr = curr -> right;
        } else {
            flag = 1;
        }
    }
    if (flag == 0) {
        cout << "\n item does not exist:no deletion\n";
        getch();
    }
    if (curr -> left == NULL && curr -> right == NULL)
        delcase = 1;
    else if (curr -> left != NULL && curr -> right != NULL)
        delcase = 3;
    else
        delcase = 2;
    if (delcase == 1) {

        if (pred -> left == curr)
            pred -> left = NULL;
        else
            pred -> right = NULL;
        delete(curr);
        pred -> right;
    }
    if (delcase == 2) {
        if (pred -> left == curr) {
            if (curr -> left == NULL)
                pred -> left = curr -> right;
            else
                pred -> left = curr -> left;
        } else {

            if (curr -> left == NULL)
                pred -> right = curr -> right;
            else
                pred -> right = curr -> left;
            delete(curr);
        }
    }
    if (delcase == 3) {
        succ = find_Insucc(curr);
        int item1 = succ -> data;
        deleteNode(item1);
    }
}

```

```

        curr -> data = item1;
    }
}
}
int main() {
    BST obj;
    int choice;
    int height = 0, total = 0, n, item;
    node ** tmp;
    while (1) {

        cout << "\n Binary search tree common operation\n";
        cout << "1) Create Tree \n";
        cout << "2) Traversal \n";
        cout << "3) Total Nodes\n";
        cout << "4) Remove Tree\n";
        cout << "5) Insert Nodes\n";
        cout << "6) Find Smallest Nodes \n";
        cout << "7) Find Largest Node \n";
        cout << "8) Delete Node\n";
        cout << "9) Exit\n";

        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {

        case 1:
            cout << "\n Creating Tree----";
            cout << "How many nodes u want to enter :";
            cin >> n;
            for (int i = 0; i < n; i++) {
                cout << "Enter Values: ";
                cin >> item;
                obj.createTree( & obj.tree, item);
            }
            break;

        case 2:
            cout << "\n Inorder Traversal: ";
            obj.inorder(obj.tree);
            cout << "\n Preorder Traversal: ";
            obj.preorder(obj.tree);
            cout << "\n Postorder Traversal: ";
            obj.postorder(obj.tree);
            getch();
            break;

        case 3:
            total = obj.totalNodes(obj.tree);
            cout << "Total nodes : " << total;
            getch();
            break;

```

```

case 4:
    obj.removeTree( & obj.tree);
    cout << "\n Tree is removed from memory";
    getch();
    break;

case 5:
    cout << "\n Insert node in a tree \n";
    cout << "Enter value: ";
    cin >> item;
    obj.createTree( & obj.tree, item);
    cout << "\nItem is inserted\n";
    getch();
    break;

case 6:
    cout << "\n\nSmallest node is: \n";
    obj.findsmallestNode(obj.tree);
    getch();
    break;

case 7:
    cout << "\n\nLargest node is: \n";
    obj.findLargestNode(obj.tree);
    getch();
    break;

case 8:
    cout << "\n\n Deleting a node from a tree--\n";
    cout << "Enter value= ";
    cin >> item;
    obj.deleteNode(item);
    break;

case 9:
    exit(1);
    break;
}
}
}

```

Output for performing various operations on Binary Search Tree:

```
E:\Stephen\DataStructure\BinarySearchTree.exe
Binary search tree common operation
1)Create Tree
2)Traversal
3)Total Nodes
4)Remove Tree
5)Insert Nodes
6)Find Smallest Nodes
7)Find Largest Node
8)Delete Node
9)Exit
Enter your choice :1

Creating Tree----How many nodes u want to enter :4
Enter Values :45
Enter Values :23
Enter Values :32
Enter Values :47

Binary search tree common operation
1)Create Tree
2)Traversal
3)Total Nodes
4)Remove Tree
5)Insert Nodes
6)Find Smallest Nodes
7)Find Largest Node
8)Delete Node
9)Exit
Enter your choice :2

Inorder Traversal : 23 32 45 47
preorder Traversal : 45 23 32 47
Postorder Traversal : 32 23 47 45
Binary search tree common operation
```

```
E:\Stephen\DataStructure\BinarySearchTree.exe
Enter your choice :3
Total nodes :4
Binary search tree common operation
1)Create Tree
2)Traversal
3)Total Nodes
4)Remove Tree
5)Insert Nodes
6)Find Smallest Nodes
7)Find Largest Node
8)Delete Node
9)Exit
Enter your choice :5
Insert node in a tree
Enter value :55
Item is inserted
Binary search tree common operation
1)Create Tree
2)Traversal
3)Total Nodes
4)Remove Tree
5)Insert Nodes
6)Find Smallest Nodes
7)Find Largest Node
8)Delete Node
9)Exit
Enter your choice :6
Smallest node is:
23
Binary search tree common operation

E:\Stephen\DataStructure\BinarySearchTree.exe
Enter your choice :7
Largest node is:
55
Binary search tree common operation
1)Create Tree
2)Traversal
3)Total Nodes
4)Remove Tree
5)Insert Nodes
6)Find Smallest Nodes
7)Find Largest Node
8)Delete Node
9)Exit
Enter your choice :8
Deleting a node from a tree--
Enter value=47
Binary search tree common operation
1)Create Tree
2)Traversal
3)Total Nodes
4)Remove Tree
5)Insert Nodes
6)Find Smallest Nodes
7)Find Largest Node
8)Delete Node
9)Exit
Enter your choice :2
Inorder Traversal : 23 32 45 55
preorder Traversal : 45 23 32 55
Postorder Traversal : 32 23 55 45
```