GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

Data Structure and Algorithms Lab

LAB MANUAL

COURSE CODE: BCS23203P

B.TECH 3RD SEMESTER

Prepared by:-

MRIDUSMITA BARUAH
Laboratory Instructor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEING
GUWAHATI-781017

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

		L	T	P	C
BSC23203P	**DATA STRUCTURE AND ALGORITHMS**	**0**	**0**	**4**	**2**

LIST OF EXPERIMENTS

1. Write a program to implement bubble sort, insertion sort and selection sort in a menu driven program.
2. Write a program to perform linear search and binary search.
3. Write a program to perform operations in an array.
4. Write a program to implement stack using array.
5. Write a program to implement queue using array.
6. Write a program to implement circular queue, priority queue.
7. Write a program to implement singly linked list along with operations.
8. Write a program to implement circular doubly linked list along with operations.
9. Write a program to create a binary search tree with operations of searching and deletion.
10. Write a program to perform traversal of a binary search tree.
11. Write a program to represent graph in memory and perform breadth first search and depth first search on this graph.

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

LIST OF EXPERIMENTS:

1.Write a program to insert an element at a specific position in a given array.

2. Write a program to delete an element from a specific position of a given array.

3. Write a program to check whether a given element is present in the array or not using linear search method.

4. Write a program to check whether a given element is present in the array or not using binary search.

5. Write a program to sort the elements in an array using bubble sort.

6. Write a program to sort the elements in an array using selection sort.

7. Write a program to sort the elements in an array using insertion sort.

8. Write a program to perform the following operations in a stack using array
   a)PUSH operation
   b)POP operation
   c)Display operation

9. Write a program to perform operations in a Linear Queue using array

10.Write a program to perform operations in a circular Queue using array.

11. Write a program to implement Priority Queue using array.

12. Write a program to perform the following operations in a singly linked list.
   1)create and display
   2)count the number of nodes in a linked list
   3)insert a node at the beginning
   4)insert a node at the end
   5)insert a node after a given node
   6)insert a node before a given node
   7)delete the first node
   8)delete the last node
   9)delete a specific node

13. Write a program to perform the following operations in a doubly linked list.
    1)create and display
    2)count the number of nodes in a linked list
    3)insert a node at the beginning
    4)insert a node at the end
    5)insert a node after a given node
    6)insert a node before a given node
    7)delete the first node
    8)delete the last node
    9)delete a specific node

14.Write a program to perform the following operations in a circular linked list.
    1)create and display
    2)count the number of nodes in a linked list
    3)insert a node at the beginning
    4)insert a node at the end
    5)insert a node after a given node
    6)insert a node before a given node
    7)delete the first node
    8)delete the last node
    9)delete a specific node

15. Write a program to implement Breath First Search.

16. Write a program to implement Depth First Search.

17. Write a program to perform In-order,Pre-order and Post–order traversal of binary tree.

18. Write a program to perform In-order,Pre-order and Post–order traversal of binary search tree.

1.AIM:Write a program to insert an element at a specific position in a given array.

PROGRAM:

```
#include<iostream>
using namespace std;
int main()
{
   int arr[50], i, elem, pos, n;
   cout<<"Enter the Size for Array: ";
   cin>>n;
   cout<<"Enter Array Elements: ";
   for(i=0; i<n; i++)
   cin>>arr[i];
   cout<<"\nElement to insert ";
   cin>>elem;
   cout<<" Position to insert ";
   cin>>pos;
   for(i=n; i>=pos; i--)
   arr[i] = arr[i-1];
   arr[i] = elem;
   n++;
   cout<<"\n Array after insertion:\n";
   for(i=0; i<n; i++)
   cout<<arr[i]<<"  ";
   cout<<endl;
   return 0;
}
```

OUTPUT:

```
Enter the Size for Array:5
Enter Array Elements: 10 20 30 40 50
Element to insert 25
Position to insert 3
Array after insertion:
10 20 25 30 40 50
```

2.AIM: Write a program to delete an element from a specific position of a given array.

PROGRAM:

```
#include<iostream>
using namespace std;
main()
{
	int a[10],pos,i,n;
	cout<<"\nenter the no of array elements";
	cin>>n;
	cout<<"\nenter the array elements";
	for(i=0;i<n;i++)
	cin>>a[i];
	cout<<"\nenter the position from which the no has to be deleted";
	cin>>pos;
	for(i=pos;i<n;i++)
	a[i]=a[i+1];
	n--;
	cout<<"\nThe array after deletion is";
	for(i=0;i<n;i++)
	cout<<a[i]<<" ";
}
```

OUTPUT:

```
enter the no of array elements5
enter the array elements10 20 30 40 50
enter the position from which the no has to be deleted2
The array after deletion is10 20 40 50
```

3.AIM:Write a program to check whether a given element is present in the array or not using linear search method.

PROGRAM:

```
#include<iostream>
using namespace std;
main()
{
int a[20];
int n,i,item;
cout<<"Enter how many numbers";
cin>>n;
cout<<"Enter the number";
for(i=0;i<n;i++)
cin>>a[i];
cout<<"enter the item to search";
cin>>item;
for (i=0;i<n; i++)
{
if (item == a[i])
{
cout<<"Item found at location" <<(i+1);
return;
}
}
if (i == n)
cout<<"Item doesnot exist";
}
}
```

OUTPUT:

```
Enter how many numbers
5
Enter the numbers
10 20 30 40 50
Enter the element to be searched
30
The element 30 is found in 3rd position
```

4. AIM: Write a program to check whether a given element is present in the array or not using binary search.

PROGRAM:

```
#include<iostream>
using namespace std;
main()
{
int a[20];
int n,i,item,loc, beg, mid, end;
cout<<"Enter how many numbers";
cin>>n;
cout<<"Enter the number";
for(i=0;i<n;i++)
cin>>a[i];
}
cout<<"enter the item to search";
cin>>item;
beg = 0; end = n-1;
mid = (beg+end)/2;
while ((beg<=end) && (a[mid]!=item))
{
if (item < a[mid])
end = mid-1;
else
beg = mid+1;
mid = (beg+end)/2;
}
if (a[mid] == item)
cout<<"ITEM found at location"<<(mid+1);
else
cout<<"ITEM doesn't exist";
}
}
```

OUTPUT:

Enter how many numbers
5
Enter the numbers
10 20 30 40 50
Enter the element to be searched
30
The element 30 is found in 3rd position

5.AIM: Write a program to sort the elements in an array using bubble sort.

PROGRAM:

```
#include<iostream>
using namespace std;
main()
{
int i,n,j,temp,a[50];
cout<<"Enter how many number";
cin>>n;
cout<<"Enter the numbers";
for(i=0;i<n;i++)
cin>>a[i];
cout<<"The elements are";
for(i=0;i<n;i++)
cout<<a[i]<<" ";
for(i=0;i<=n-1;i++)
{
 for(j=0;j<=n-1;j++)
  {
   if(a[j]>a[j+1])
    {
     temp=a[j];
     a[j]=a[j+1];
     a[j+1]=temp;
    }
   }
 }
 cout<<"after sorting";
for(i=0;i<n;i++)
cout<<a[i]<<" ";
```

OUTPUT:

```
Enter how many numbers
4
Enter the numbers 12 5 45 8
The elements are
12 5 45 8
After sorting
5 8 12 4
```

6. AIM:Write a program to sort the elements in an array using selection sort.

PROGRAM:

```
#include<iostream>
using namespace std;
main()
{
int i,n,j,temp,a[50];
cout<<"Enter how many number";
cin>>n;
cout<<"Enter the numbers";
for(i=0;i<n;i++)
cin>>a[i];
cout<<"the elements are";
for(i=0;i<n;i++)
cout<<a[i]<<" ";
int min;
for (i=0;i<n;i++)
{
 min=i;
 for(j=i+1;j<n;j++)
  {
   if(a[j]<a[min])
   min=j;
  }
  temp=a[i];
  a[i]=a[min];
  a[min]=temp;
}
}
cout<<"The array after sorting";
for(i=0;i<n;i++)
cout<<a[i]<<" ";
}
```

OUTPUT:

```
Enter how many numbers 4
Enter the numbers 12 5 45 8
The elements are   12 5 45 8
After sorting          5 8 12 45
```

7. AIM:Write a program to sort the elements in an array using insertion sort.

PROGRAM:

```
#include<iostream>
using namespace std;
main()
{
int i,n,j,k,temp,a[50];
cout<<"Enter how many number";
cin>>n;
cout<<"the elements are";
for(i=0;i<n;i++)
cin>>a[i];
for(i=0;i<n;i++)
cout<<a[i]<<" ";
}
for(j=1; j<n; j++)
{
temp = a[j];
k = j-1;
while (k>=0 && a[k]>temp)
{
a[k+1] = a[k];
k--;
}
a[k+1] = temp;
}
}
cout<<"After sorting";
for(i=0;i<n;i++)
cin>>a[i];
}
```

OUTPUT:

```
Enter how many numbers 4
Enter the numbers 12 5 45 8
The elements are   12 5 45 8
After sorting          5 8 12 45
```

8.AIM:Write a program to perform the following operations in a stack using array
a)PUSH operation
b)POP operation
c)Display operation

PROGRAM:

```
#include<iostream>
using namespace std;
#define max 5
class stack
{
private:
int a[max];
int top,i;
public:
stack();
void push();
void pop();
void display();
};
stack::stack()
{
top=-1;
}
void stack::push()
{
int item;
if(top==max-1)
{
cout<<"The stack is full";
}
else
{
cout<<"Enter the elements";
cin>>item;
top++;
a[top]=item;
}
}
```

```
void stack::pop()
{
int data;
if(top==-1)
{
cout<<"The stack is empty";
//return NULL;
}
data=a[top];
top--;
cout<<"The deleted item "<<data;
}
void stack::display()
{
for(i=top;i>=0;i--)
cout<<a[i]<<" ";
}
main()
{
int choice;
char ch;
stack s;
do
{
cout<<"1.push"<<endl;
cout<<"2.pop"<<endl;
cout<<"3.Display"<<endl;
cout<<"enter your choice"<<endl;
cin>>choice;
switch(choice)
{
case 1: s.push();
        break;
case 2: s.pop();
        break;
case 3: s.display();
        break;
default: cout<<"Wrong choice";
}
cout<<"do u wish to continue";
```

```
cin>>ch;
}
while(ch=='y'||ch=='Y');
}
```

OUTPUT:

```
1.push
2.pop
3.Display
Enter your choice 1
Enter the elements 10
do u wish to continuey
1.push
2.pop
3.Display
Enter your choice 1
Enter the elements 15
do u wish to continuey
1.push
2.pop
3.Display
Enter your choice 1
Enter the elements 14
do u wish to continuey
1.push
2.pop
3.Display
Enter your choice 3
 14 15 10
do u wish to continuey
1.push
2.pop
3.Display
Enter your choice 2
The deleted item 14
```

do u wish to continuey
1.push
2.pop
3.Display
Enter your choice 3
15 10
do u wish to continue

9.AIM:Write a program to perform operations in a Linear Queue using array

PROGRAM:

```
#include<iostream>
using namespace std;
#define max 3
class queue
{
 private:
       int a[max],i,rear,front;
 public:
       queue();
void add();
       void del();
       void display();
};
queue::queue()
{
  rear=front=-1;
  for(i=0;i<max;i++)
  {
   a[i]=0;
  }
}
void queue::add()
{
 int item;
 if((rear==max-1&&front==0)||(rear+1==front))
 cout<<"the queue is full";
 if(rear==max-1)
 rear=0;
 else
 {
 cout<<"enter the elements";
 cin>>item;
 rear++;
 a[rear]=item;
 if(front==-1)
 front=0;
```

```
 }
}
void queue::del()
{
 int data;
 if(front==-1)
{
 cout<<"the queue is empty";
}
 else
 data=a[front];
 a[front]=0;
 if(front==rear)
front=rear=-1;
 else
 {
 if(front==max-1)
 front=0;
 else
 front++;
 }
 cout<<"the deleted item is"<<data;
}
void queue::display()
{
 cout<<"the elements are";
 for(i=0;i<max;i++)
 cout<<a[i]<<" ";
}
main()
{
 queue q;
 int choice;
 char ch;
 do
  {
   cout<<"1.add"<<endl;
   cout<<"2.del"<<endl;
   cout<<"3.display"<<endl;
   cout<<"enter your choice";
```

```
  cin>>choice;
  switch(choice)
  {
   case 1: q.add();
           break;
   case 2: q.del();
           break;
   case 3: q.display();
           break;
   default: cout<<"wrong choice";
  }
  cout<<"do u wish to continue";
  cin>>ch;
 }
while(ch=='y'||ch=='Y');
}
```

OUTPUT:

```
1.add
2.del
3.display
Enter your choice 1
10
do u wish to continuey
1.add
2.del
3.display
Enter your choice 1
12
do u wish to continuey
1.add
2.del
3.display
Enter your choice 1
13
do u wish to continuey
1.add
2.del
3.display
```

Enter your choice 3
The elements are
10 12 13
do u wish to continuey
1.add
2.del
3.display
Enter your choice 2
the deleted item is 10
do u wish to continuey
1.add
2.del
3.display
Enter your choice 3
The elements are
12 13
do u wish to continuen

10.AIM:Write a program to perform operations in a circular Queue using array.

PROGRAM:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
#define MAX  5
class cqueue
{
   int a[MAX],i,front,rear;
   public :
            cqueue()
            void insert(int );
            void deletion();
            void display();
 };
cqueue::cqueue()
{
        rear=front=-1;
        for(i=0;i<MAX;i++)
        {
                a[i]=0;
        }
}
void cqueue :: insert(int item)
         {
                if((front==0 && rear==MAX-1) || (rear+1==front))
                        cout<<" Circular Queue is Full";
                else
                {
                  if(rear==MAX-1)
                          rear=0;
                  else
                         rear++;
                  a[rear]=item;
                }
                if(front==-1)
                  front=0;
         }
```

```
 void cqueue :: deletion()
 {
        int data;
        if(front==-1)
              cout<<"Circular Queue is Empty";
        else
        {
              data=a[front];
                  a[fornt]=0;
              if(front==rear)
                front=rear=-1;
              else
              {
                if(front==MAX-1)
                     front=0;
                       else
                         front++;
                              cout<<””the deleted item is”<<data<<endl;
              }
        }
}
 void cqueue :: display()
 {
        cout<<”The elements are”;
        for(i=0;i<MAX;i++)
        {
              cout<<a[i]<<” ”;
        }
}
main()
{
        cqueue q;
        q.insert(10);
        q.insert(20);
        q.insert(30);
        q.insert(40);
        q.insert(50);
        q.display();
        q.deletion();
        q.display();
```

```
        q.insert(70);
        q.display();
    }
```

OUTPUT:

```
The elements are
10 20 30 40 50
the deleted item is 10
The elements are
20 30 40 50
The elements are
70 20 30 40 50
```

11.AIM:Write a program to implement Priority Queue using array.

PROGRAM:

```
#include <iostream>
using namespace std;
#define MAX 5   // Maximum size of the priority queue
class PriorityQueue
{
private:
   int data[MAX];
   int priority[MAX];
   int size;
public:
   PriorityQueue()
   {
      size = 0;
   }
   void enqueue(int value, int p)
   {
      if (size == MAX)
      {
         cout << "Queue Overflow!\n";
         return;
      }
        int i;
      // Find position for new element based on priority
      for (i = size - 1; i >= 0; i--) {
         if (p > priority[i]) {
            data[i + 1] = data[i];
            priority[i + 1] = priority[i];
         } else {
            break;
         }
      }
      data[i + 1] = value;
      priority[i + 1] = p;
      size++;
      cout << "Inserted " << value << " with priority " << p << endl;
   }
```

```
  void dequeue()
  {
     if (size == 0)
    {
        cout << "Queue Underflow!\n";
        return;
     }
     cout << "Deleted element: " << data[0] << " (priority " << priority[0] << ")\n";
     for (int i = 0; i < size - 1; i++)
     {
        data[i] = data[i + 1];
        priority[i] = priority[i + 1];
     }
     size--;
  }
   void display()
   {
     if (size == 0)
     {
        cout << "Queue is empty!\n";
        return;
     }
     cout << "\nCurrent Priority Queue:\n";
     cout << "Value\tPriority\n";
     for (int i = 0; i < size; i++)
     {
        cout << data[i] << "\t" << priority[i] << endl;
     }
     cout << endl;
  }
};
int main()
 {
  PriorityQueue pq;
  pq.enqueue(10, 2);
  pq.enqueue(5, 1);
  pq.enqueue(20, 3);
  pq.enqueue(15, 2);
  pq.display();
```

```
    pq.dequeue();
    pq.display();
    return 0;
}
```

OUTPUT:

```
Inserted 10 with priority 2
Inserted 5 with priority 1
Inserted 20 with priority 3
Inserted 15 with priority 2

Current Priority Queue:
Value   Priority
20      3
10      2
15      2
5       1

Deleted element: 20 (priority 3)

Current Priority Queue:
Value   Priority
10      2
15      2
5       1
```

12. AIM: Write a program to perform the following operations in a singly linked list.

1)create and display
2)count the number of nodes in a linked list
3)insert a node at the beginning
4)insert a node at the end
5)insert a node after a given node
6)insert a node before a given node
7)delete the first node
8)delete the last node
9)delete a specific node

PROGRAM:

```
#include<iostream>
using namespace std;
struct Node {
   int data;
   Node* next;
};
Node* head = nullptr;
// 1) Create and Display
void create(int n)
{
   Node* temp, *newNode;
   for(int i = 0; i < n; i++)
{
      newNode = new Node();
      cout << "Enter data for node " << i + 1 << ": ";
      cin >> newNode->data;
      newNode->next = nullptr;
      if(head == nullptr)
      {
         head = newNode;
      }
      else
      {
         temp = head;
         while(temp->next != nullptr)
            temp = temp->next;
         temp->next = newNode;
```

```
        }
    }
}
void display() {
    Node* temp = head;
    cout << "Linked list: ";
    while(temp != nullptr)
  {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}

// 2) Count nodes
int countNodes()
{
    Node* temp = head;
    int count = 0;
    while(temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}
// 3) Insert at beginning
void insertAtBeginning(int val)
{
    Node* newNode = new Node();
    newNode->data = val;
    newNode->next = head;
    head = newNode;
}
// 4) Insert at end
void insertAtEnd(int val)
{
    Node* newNode = new Node();
    newNode->data = val;
    newNode->next = nullptr;
    if(head == nullptr) {
```

```
        head = newNode;
        return;
    }
    Node* temp = head;
    while(temp->next != nullptr)
        temp = temp->next;
    temp->next = newNode;
}
// 5) Insert after a given node
void insertAfter(int key, int val)
{
    Node* temp = head;
    while(temp != nullptr && temp->data != key)
        temp = temp->next;
    if(temp == nullptr) {
        cout << "Node not found.\n";
        return;
    }
    Node* newNode = new Node();
    newNode->data = val;
    newNode->next = temp->next;
    temp->next = newNode;
}
// 6) Insert before a given node
void insertBefore(int key, int val)
  {
    if(head == nullptr) return;
    if(head->data == key) {
        insertAtBeginning(val);
        return;
    }
    Node* temp = head;
    while(temp->next != nullptr && temp->next->data != key)
        temp = temp->next;
    if(temp->next == nullptr) {
        cout << "Node not found.\n";
        return;
    }
    Node* newNode = new Node();
    newNode->data = val;
```

```
    newNode->next = temp->next;
    temp->next = newNode;
}
// 7) Delete first node
void deleteFirst()
{
    if(head == nullptr) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}
// 8) Delete last node
void deleteLast()
  {
    if(head == nullptr) return;
    if(head->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while(temp->next->next != nullptr)
        temp = temp->next;
    delete temp->next;
    temp->next = nullptr;}
// 9) Delete specific node
void deleteNode(int key) {
    if(head == nullptr) return;
    if(head->data == key) {
        deleteFirst();
        return;
    }
    Node* temp = head;
    while(temp->next != nullptr && temp->next->data != key)
        temp = temp->next;
    if(temp->next == nullptr) {
        cout << "Node not found.\n";
        return;
    }
    Node* toDelete = temp->next;
```

```
   temp->next = temp->next->next;
   delete toDelete;
}
int main()
{
   int choice, val, key, n;
   while(true)
   {
      cout << "\nMenu:\n";
      cout << "1. Create & Display\n2. Count Nodes\n3. Insert at Beginning\n4. Insert at End\n5. Insert After\n";
      cout << "6. Insert Before\n7. Delete First\n8. Delete Last\n9. Delete Node\n10. Exit\nEnter your choice: ";
      cin >> choice;
      switch(choice)
      {
         case 1:
            cout << "Enter number of nodes: ";
            cin >> n;
            create(n);
            display();
            break;
         case 2:
            cout << "Total nodes: " << countNodes() << endl;
            break;
         case 3:
            cout << "Enter value to insert: ";
            cin >> val;
            insertAtBeginning(val);
            display();
            break;
         case 4:
            cout << "Enter value to insert: ";
            cin >> val;
            insertAtEnd(val);
            display();
            break;
         case 5:
            cout << "Enter node value after which to insert: ";
            cin >> key;
```

```
            cout << "Enter new value: ";
            cin >> val;
            insertAfter(key, val);
            display();
            break;
        case 6:
            cout << "Enter node value before which to insert: ";
            cin >> key;
            cout << "Enter new value: ";
            cin >> val;
            insertBefore(key, val);
            display();
            break;
        case 7:
            deleteFirst();
            display();
            break;
        case 8:
            deleteLast();
            display();
            break;
        case 9:
            cout << "Enter node value to delete: ";
            cin >> key;
            deleteNode(key);
            display();
            break;
        case 10:
            return 0;
        default:
            cout << "Invalid choice.\n";
      }
   }
   return 0;
}
```

OUTPUT:

```
Menu:
1. Create & Display
2. Count Nodes
3. Insert at Beginning
4. Insert at End
5. Insert After
6. Insert Before
7. Delete First
8. Delete Last
9. Delete Node
10. Exit
Enter your choice: 1
Enter number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Linked list: 10 -> 20 -> 30 -> NULL

Menu:
1. Create & Display
...
Enter your choice: 2
Total nodes: 3

Menu:
Enter your choice: 3
Enter value to insert: 5
Linked list: 5 -> 10 -> 20 -> 30 -> NULL

Menu:
Enter your choice: 4
Enter value to insert: 40
Linked list: 5 -> 10 -> 20 -> 30 -> 40 -> NULL

Menu:
Enter your choice: 5
Enter node value after which to insert: 20
```

Enter new value: 25
Linked list: 5 -> 10 -> 20 -> 25 -> 30 -> 40 -> NULL

Menu:
Enter your choice: 6
Enter node value before which to insert: 30
Enter new value: 28
Linked list: 5 -> 10 -> 20 -> 25 -> 28 -> 30 -> 40 -> NULL

Menu:
Enter your choice: 7
Linked list: 10 -> 20 -> 25 -> 28 -> 30 -> 40 -> NULL

Menu:
Enter your choice: 8
Linked list: 10 -> 20 -> 25 -> 28 -> 30 -> NULL

Menu:
Enter your choice: 9
Enter node value to delete: 25
Linked list: 10 -> 20 -> 28 -> 30 -> NULL

Menu:
Enter your choice: 10

13. AIM: Write a program to perform the following operations in a doubly linked list.

1)create and display
2)count the number of nodes in a linked list
3)insert a node at the beginning
4)insert a node at the end
5)insert a node after a given node
6)insert a node before a given node
7)delete the first node
8)delete the last node
9)delete a specific node

PROGRAM:

```
#include<iostream>
using namespace std;
struct Node
{
   int data;
   Node* prev;
   Node* next;
};
Node* head = nullptr;
// 1) Create and Display
void create(int n)
{
   Node *newNode, *temp;
   for(int i = 0; i < n; i++) {
      newNode = new Node();
      cout << "Enter data for node " << i+1 << ": ";
      cin >> newNode->data;
      newNode->prev = nullptr;
      newNode->next = nullptr;
      if(head == nullptr) {
         head = newNode;
      } else {
         temp = head;
         while(temp->next != nullptr)
            temp = temp->next;
         temp->next = newNode;
         newNode->prev = temp;
```

```
        }
    }
}
void display()
 {
    Node* temp = head;
    cout << "Doubly Linked List: ";
    while(temp != nullptr) {
        cout << temp->data << " <-> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}
// 2) Count Nodes
int countNodes()
{
    int count = 0;
    Node* temp = head;
    while(temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}
// 3) Insert at Beginning
void insertAtBeginning(int val)
{
    Node* newNode = new Node();
    newNode->data = val;
    newNode->prev = nullptr;
    newNode->next = head;
    if(head != nullptr)
        head->prev = newNode;
    head = newNode;
}
// 4) Insert at End
void insertAtEnd(int val)
{
    Node* newNode = new Node();
    newNode->data = val;
```

```
    newNode->next = nullptr;
    if(head == nullptr) {
        newNode->prev = nullptr;
        head = newNode;
        return;
    }
    Node* temp = head;
    while(temp->next != nullptr)
        temp = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
}
// 5) Insert After a Given Node
void insertAfter(int key, int val)
{
    Node* temp = head;
    while(temp != nullptr && temp->data != key)
        temp = temp->next;
    if(temp == nullptr) {
        cout << "Node not found.\n";
        return;
    }
    Node* newNode = new Node();
    newNode->data = val;
    newNode->next = temp->next;
    newNode->prev = temp;
    if(temp->next != nullptr)
        temp->next->prev = newNode;
    temp->next = newNode;
}
// 6) Insert Before a Given Node
void insertBefore(int key, int val)
 {
    if(head == nullptr) return;
    if(head->data == key) {
        insertAtBeginning(val);
        return;
    }
    Node* temp = head;
    while(temp != nullptr && temp->data != key)
```

```
        temp = temp->next;
    if(temp == nullptr) {
        cout << "Node not found.\n";
        return;
    }
    Node* newNode = new Node();
    newNode->data = val;
    newNode->next = temp;
    newNode->prev = temp->prev;
    temp->prev->next = newNode;
    temp->prev = newNode;
}
// 7) Delete First Node
void deleteFirst()
 {
    if(head == nullptr) return;
    Node* temp = head;
    head = head->next;
    if(head != nullptr)
        head->prev = nullptr;
    delete temp;
}
// 8) Delete Last Node
void deleteLast()
{
    if(head == nullptr) return;
    Node* temp = head;
    if(temp->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }
    while(temp->next != nullptr)
        temp = temp->next;
    temp->prev->next = nullptr;
    delete temp;
}
// 9) Delete a Specific Node
void deleteNode(int key)
 {
```

```
   if(head == nullptr) return;
   Node* temp = head;
   if(head->data == key) {
      deleteFirst();
      return;
   }
    while(temp != nullptr && temp->data != key)
      temp = temp->next;
     if(temp == nullptr) {
      cout << "Node not found.\n";
      return;
   }
   if(temp->next != nullptr)
      temp->next->prev = temp->prev;
   if(temp->prev != nullptr)
      temp->prev->next = temp->next;
   delete temp;
}
int main()
{
   int choice, val, key, n;
   while(true) {
   cout << "\nMenu:\n";cout << "1. Create & Display\n2. Count Nodes\n3. Insert at Beginning\n4. Insert at End\n5. Insert After\n";
      cout << "6. Insert Before\n7. Delete First\n8. Delete Last\n9. Delete Node\n10. Exit\nEnter your choice: ";
      cin >> choice;
      switch(choice) {
         case 1:
            cout << "Enter number of nodes: ";
            cin >> n;
            create(n);
            display();
            break;
         case 2:
            cout << "Total nodes: " << countNodes() << endl;
            break;
         case 3:
            cout << "Enter value to insert: ";
            cin >> val;
```

```
        insertAtBeginning(val);
        display();
        break;
    case 4:
        cout << "Enter value to insert: ";
        cin >> val;
        insertAtEnd(val);
        display();
        break;
    case 5:
        cout << "Enter node value after which to insert: ";
        cin >> key;
        cout << "Enter new value: ";
        cin >> val;
        insertAfter(key, val);
        display();
        break;
    case 6:
        cout << "Enter node value before which to insert: ";
        cin >> key;
        cout << "Enter new value: ";
        cin >> val;
        insertBefore(key, val);
        display();
        break;
    case 7:
        deleteFirst();
        display();
        break;
    case 8:
        deleteLast();
        display();
        break;
    case 9:
        cout << "Enter node value to delete: ";
        cin >> key;
        deleteNode(key);
        display();
        break;
    case 10:
```

```
            return 0;
        default:
            cout << "Invalid choice.\n";
      }
   }
   return 0;
}
```

OUTPUT:

```
Menu:
1. Create & Display
2. Count Nodes
3. Insert at Beginning
4. Insert at End
5. Insert After
6. Insert Before
7. Delete First
8. Delete Last
9. Delete Node
10. Exit
Enter your choice: 1
Enter number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL

Menu:
Enter your choice: 2
Total nodes: 3

Menu:
Enter your choice: 3
Enter value to insert: 5
Doubly Linked List: 5 <-> 10 <-> 20 <-> 30 <-> NULL

Menu:
Enter your choice: 4
```

Enter value to insert: 40
Doubly Linked List: 5 <-> 10 <-> 20 <-> 30 <-> 40 <-> NULL

Menu:
Enter your choice: 5
Enter node value after which to insert: 20
Enter new value: 25
Doubly Linked List: 5 <-> 10 <-> 20 <-> 25 <-> 30 <-> 40 <-> NULL

Menu:
Enter your choice: 6
Enter node value before which to insert: 30
Enter new value: 28
Doubly Linked List: 5 <-> 10 <-> 20 <-> 25 <-> 28 <-> 30 <-> 40 <-> NULL

Menu:
Enter your choice: 7
Doubly Linked List: 10 <-> 20 <-> 25 <-> 28 <-> 30 <-> 40 <-> NULL

Menu:
Enter your choice: 8
Doubly Linked List: 10 <-> 20 <-> 25 <-> 28 <-> 30 <-> NULL

Menu:
Enter your choice: 9
Enter node value to delete: 25
Doubly Linked List: 10 <-> 20 <-> 28 <-> 30 <-> NULL

Menu:
Enter your choice: 10

14.AIM: Write a program to perform the following operations in a circular linked list.

1)create and display
2)count the number of nodes in a linked list
3)insert a node at the beginning
4)insert a node at the end
5)insert a node after a given node
6)insert a node before a given node
7)delete the first node
8)delete the last node
9)delete a specific node

PROGRAM:

```
#include<iostream>
using namespace std;
struct Node
{
   int data;
   Node* next;
};
Node* last = nullptr;
// 1) Create
void create(int n) {
   int val;
   for(int i = 0; i < n; i++) {
      cout << "Enter data for node " << i+1 << ": ";
      cin >> val;
      Node* newNode = new Node();
      newNode->data = val;
      if(last == nullptr) {
         newNode->next = newNode;
         last = newNode;
      } else {
         newNode->next = last->next;
         last->next = newNode;
         last = newNode;
      }
   }
}
// 1) Display
```

```
void display()
{
   if(last == nullptr) {
      cout << "List is empty.\n";
      return;
   }
   Node* temp = last->next;
   cout << "Circular Linked List: ";
   do {
      cout << temp->data << " -> ";
      temp = temp->next;
   } while(temp != last->next);
   cout << "(back to head)\n";
}
// 2) Count Nodes
int countNodes()
{
   if(last == nullptr) return 0;
   int count = 0;
   Node* temp = last->next;
   do {
      count++;
      temp = temp->next;
   } while(temp != last->next);
   return count;
}
// 3) Insert at Beginning
void insertAtBeginning(int val)
{
   Node* newNode = new Node();
   newNode->data = val;
   if(last == nullptr) {
      newNode->next = newNode;
      last = newNode;
   } else {
      newNode->next = last->next;
      last->next = newNode;
   }
}
// 4) Insert at End
```

```
void insertAtEnd(int val)
{
   Node* newNode = new Node();
   newNode->data = val;
   if(last == nullptr) {
      newNode->next = newNode;
      last = newNode;
   } else {
      newNode->next = last->next;
      last->next = newNode;
      last = newNode;
   }
}
// 5) Insert After a Given Node
void insertAfter(int key, int val)
{
   if(last == nullptr) return;
   Node* temp = last->next;
   do {
      if(temp->data == key) {
         Node* newNode = new Node();
         newNode->data = val;
         newNode->next = temp->next;
         temp->next = newNode;
         if(temp == last)
            last = newNode;
         return;
      }
      temp = temp->next;
   } while(temp != last->next);
   cout << "Node not found.\n";
}
// 6) Insert Before a Given Node
void insertBefore(int key, int val)
{
   if(last == nullptr) return;

   Node* curr = last->next;
   Node* prev = last;
   do {
```

```
        if(curr->data == key) {
            Node* newNode = new Node();
            newNode->data = val;
            newNode->next = curr;
            prev->next = newNode;
            if(curr == last->next)
                last->next = newNode;
            return;
        }
        prev = curr;
        curr = curr->next;
    } while(curr != last->next);
    cout << "Node not found.\n";
}
// 7) Delete First Node
void deleteFirst()
{
    if(last == nullptr) return;
    Node* temp = last->next;
    if(last == last->next) {
        delete last;
        last = nullptr;
    } else {
        last->next = temp->next;
        delete temp;
    }
}
// 8) Delete Last Node
void deleteLast() {
    if(last == nullptr) return;
    Node* temp = last->next;
    if(last == last->next) {
        delete last;
        last = nullptr;
        return;
    }
    while(temp->next != last)
        temp = temp->next;
    temp->next = last->next;
    delete last;
```

```
    last = temp;
}
// 9) Delete a Specific Node
void deleteNode(int key)
{
    if(last == nullptr) return;
    Node* curr = last->next;
    Node* prev = last;
    do {
        if(curr->data == key) {
            if(curr == last && curr == last->next) {
                delete last;
                last = nullptr;
            } else {
                prev->next = curr->next;
                if(curr == last)
                    last = prev;
                if(curr == last->next)
                    last->next = curr->next;
                delete curr;
            }
            return;
        }
        prev = curr;
        curr = curr->next;
    } while(curr != last->next);
    cout << "Node not found.\n";
}
int main()
{
    int choice, val, key, n;
    while(true)
     {
        cout << "\nMenu:\n";
        cout << "1. Create & Display\n2. Count Nodes\n3. Insert at Beginning\n4. Insert at End\n5.
Insert After\n";
        cout << "6. Insert Before\n7. Delete First\n8. Delete Last\n9. Delete Node\n10. Exit\nEnter
your choice: ";
        cin >> choice;
        switch(choice)
```

```
{
   case 1:
      cout << "Enter number of nodes: ";
      cin >> n;
      create(n);
      display();
      break;
   case 2:
      cout << "Total nodes: " << countNodes() << endl;
      break;
   case 3:
      cout << "Enter value to insert: ";
      cin >> val;
      insertAtBeginning(val);
      display();
      break;
   case 4:
      cout << "Enter value to insert: ";
      cin >> val;
      insertAtEnd(val);
      display();
      break;
   case 5:
      cout << "Enter node value after which to insert: ";
      cin >> key;
      cout << "Enter new value: ";
      cin >> val;
      insertAfter(key, val);
      display();
      break;
   case 6:
      cout << "Enter node value before which to insert: ";
      cin >> key;
      cout << "Enter new value: ";
      cin >> val;
      insertBefore(key, val);
      display();
      break;
   case 7:
      deleteFirst();
```

```
            display();
            break;
        case 8:
            deleteLast();
            display();
            break;
        case 9:
            cout << "Enter node value to delete: ";
            cin >> key;
            deleteNode(key);
            display();
            break;
        case 10:
            return 0;
        default:
            cout << "Invalid choice.\n";
      }
   }
   return 0;
}
```

OUTPUT:

```
Menu:
1. Create & Display
2. Count Nodes
3. Insert at Beginning
4. Insert at End
5. Insert After
6. Insert Before
7. Delete First
8. Delete Last
9. Delete Node
10. Exit
Enter your choice: 1
Enter number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Circular Linked List: 10 -> 20 -> 30 -> (back to head)
Menu:
```

Enter your choice: 2
Total nodes: 3

Menu:
Enter your choice: 3
Enter value to insert: 5
Circular Linked List: 5 -> 10 -> 20 -> 30 -> (back to head)

Menu:
Enter your choice: 4
Enter value to insert: 40
Circular Linked List: 5 -> 10 -> 20 -> 30 -> 40 -> (back to head)

Menu:
Enter your choice: 5
Enter node value after which to insert: 20
Enter new value: 25
Circular Linked List: 5 -> 10 -> 20 -> 25 -> 30 -> 40 -> (back to head)

Menu:
Enter your choice: 6
Enter node value before which to insert: 30
Enter new value: 28
Circular Linked List: 5 -> 10 -> 20 -> 25 -> 28 -> 30 -> 40 -> (back to head)

Menu:
Enter your choice: 7
Circular Linked List: 10 -> 20 -> 25 -> 28 -> 30 -> 40 -> (back to head)

Menu:
Enter your choice: 8
Circular Linked List: 10 -> 20 -> 25 -> 28 -> 30 -> (back to head)

Menu:
Enter your choice: 9
Enter node value to delete: 25
Circular Linked List: 10 -> 20 -> 28 -> 30 -> (back to head)
Menu:
Enter your choice: 10

15.AIM:Write a program to implement Breath First Search.

PROGRAM:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
// Graph class using adjacency list representation
class Graph
{
   int V; // Number of vertices
   vector<vector<int>> adj; // Adjacency list
public:
   Graph(int V); // Constructor
   void addEdge(int v, int w); // Function to add an edge
   void BFS(int start); // BFS traversal from a given source
};
Graph::Graph(int V) {
   this->V = V;
   adj.resize(V);
}
void Graph::addEdge(int v, int w) {
   adj[v].push_back(w); // Add w to v's list.
   // For undirected graph, also add: adj[w].push_back(v);
}
void Graph::BFS(int start) {
   vector<bool> visited(V, false); // Mark all vertices as not visited
   queue<int> q;
   // Mark the current node as visited and enqueue it
   visited[start] = true;
   q.push(start);
   while (!q.empty()) {
      int current = q.front();
      q.pop();
      cout << current << " ";
    // Get all adjacent vertices of the dequeued vertex
      for (int neighbor : adj[current]) {
         if (!visited[neighbor]) {
            visited[neighbor] = true;
```

```
            q.push(neighbor);
          }
        }
    }
}
int main() {
    Graph g(6); // Create a graph with 6 vertices
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    cout << "Breadth First Search starting from vertex 0:\n";
    g.BFS(0);
    return 0;
}
```

OUTPUT:

```
Breadth First Search starting from vertex 0:
0 1 2 3 4 5
```

16.AIM: Write a program to implement Depth First Search.

PROGRAM:

```
#include <iostream>
#include <vector>
using namespace std;
class Graph {
   int vertices;
   vector<vector<int>> adjList;
public:
   Graph(int v) {
      vertices = v;
      adjList.resize(v);
   }
    void addEdge(int u, int v) {
      adjList[u].push_back(v); // Add edge from u to v
      adjList[v].push_back(u); // For undirected graph (remove for directed)
   }
   void DFSUtil(int v, vector<bool> &visited) {
      visited[v] = true;
      cout << v << " ";
      for (int neighbor : adjList[v]) {
         if (!visited[neighbor]) {
            DFSUtil(neighbor, visited);
         }
      }
   }
   void DFS(int start) {
      vector<bool> visited(vertices, false);
      cout << "Depth First Search starting from vertex " << start << ": ";
      DFSUtil(start, visited);
      cout << endl;
   }
};
int main()
{
   int V, E;
   cout << "Enter number of vertices: ";
   cin >> V;
```

```
    Graph g(V);
    cout << "Enter number of edges: ";
    cin >> E;
    cout << "Enter " << E << " edges (u v):" << endl;
    for (int i = 0; i < E; i++) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }
    int start;
    cout << "Enter starting vertex for DFS: ";
    cin >> start;
    g.DFS(start)
    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 6
Enter number of edges: 7
Enter 7 edges (u v):
0 1
0 2
1 3
1 4
2 4
3 5
4 5
Enter starting vertex for DFS: 0
Depth First Search starting from vertex 0: 0 1 3 5 4 2
```

17.AIM: Write a program to perform In-order,Pre-order and Post–order traversal of binary tree.

PROGRAM:

```
#include <iostream>
using namespace std;
// Define a node structure for the binary tree
struct Node
{
   int data;
   Node* left;
   Node* right;
   Node(int value) {
      data = value;
      left = right = nullptr;
   }
};
// In-order Traversal: Left -> Root -> Right
void inOrderTraversal(Node* root) {
   if (root == nullptr) return;
   inOrderTraversal(root->left);
   cout << root->data << " ";
   inOrderTraversal(root->right);
}
// Pre-order Traversal: Root -> Left -> Right
void preOrderTraversal(Node* root) {
   if (root == nullptr) return;
   cout << root->data << " ";
   preOrderTraversal(root->left);
   preOrderTraversal(root->right);
}
// Post-order Traversal: Left -> Right -> Root
void postOrderTraversal(Node* root) {
   if (root == nullptr) return;
   postOrderTraversal(root->left);
   postOrderTraversal(root->right);
   cout << root->data << " ";
}
```

```
int main()
{
   Node* root = new Node(1);
   root->left = new Node(2);
   root->right = new Node(3);
   root->left->left = new Node(4);
   root->left->right = new Node(5);
   root->right->right = new Node(6);
   cout << "In-order Traversal: ";
   inOrderTraversal(root);
   cout << endl;
   cout << "Pre-order Traversal: ";
   preOrderTraversal(root);
   cout << endl;
   cout << "Post-order Traversal: ";
   postOrderTraversal(root);
   cout << endl;
   return 0;
}
```

OUTPUT:

```
In-order Traversal: 4 2 5 1 3 6
Pre-order Traversal: 1 2 4 5 3 6
Post-order Traversal: 4 5 2 6 3 1
```

18.AIM:Write a program to perform In-order,Pre-order and Post–order traversal of binary search tree.

PROGRAM:

```
#include <iostream>
using namespace std;
struct Node
{
   int data;
   Node* left;
   Node* right;
   Node(int val)
  {
      data = val;
      left = right = nullptr;
   }
};
// Function to insert a node in the BST
Node* insert(Node* root, int key)
{
   if (root == nullptr)
      return new Node(key);
   if (key < root->data)
      root->left = insert(root->left, key);
   else if (key > root->data)
      root->right = insert(root->right, key);
    return root;
}
// In-order Traversal: Left -> Root -> Right
void inOrderTraversal(Node* root) {
   if (root == nullptr) return;
   inOrderTraversal(root->left);
   cout << root->data << " ";
   inOrderTraversal(root->right);
}
// Pre-order Traversal: Root -> Left -> Right
void preOrderTraversal(Node* root) {
   if (root == nullptr) return;
   cout << root->data << " ";
```

```
    preOrderTraversal(root->left);
    preOrderTraversal(root->right);
}
// Post-order Traversal: Left -> Right -> Root
void postOrderTraversal(Node* root) {
    if (root == nullptr) return;
    postOrderTraversal(root->left);
    postOrderTraversal(root->right);
    cout << root->data << " ";
}
int main()
{
    Node* root = nullptr;
    int n, val;
    cout << "Enter number of nodes to insert: ";
    cin >> n;
    cout << "Enter values to insert into BST:\n";
    for (int i = 0; i < n; i++)
   {
        cin >> val;
        root = insert(root, val);
    }
    cout << "\nIn-order Traversal: ";
    inOrderTraversal(root);
    cout << "\nPre-order Traversal: ";
    preOrderTraversal(root);
    cout << "\nPost-order Traversal: ";
    postOrderTraversal(root);
    cout << endl;
    return 0;
}
```

OUTPUT:

```
Enter number of nodes to insert: 6
Enter values to insert into BST:
50 30 70 20 40 60
In-order Traversal: 20 30 40 50 60 70
Pre-order Traversal: 50 30 20 40 70 60
Post-order Traversal: 20 40 30 60 70 50
```