Q 14. Write a programme to implement queue using an array.

CODE:

```c
#include <stdio.h>

// function to check whether the queue is empty or not
int isEmpty(int front, int rear)
{
    if (front == -1 && rear == -1)
    {
        return 1;
    }
    return 0;
}
// function to insert in a queue
void enqueue(int queue[], int *rear, int *front, int x, int size)
{
    if (*rear == size - 1)
    {
        printf("Overflow");
    }
    else
    {
        if (isEmpty(*front, *rear))
        {
            *front = 0;
        }
        *rear += 1;
        queue[*rear] = x;
    }
}

// function to delete from a queue
int dequeue(int queue[], int *rear, int *front, int size)
{
    int first_ele = queue[0];
    if (isEmpty(*front, *rear))
    {
        printf("Underflow");
    }
    else if (*front == *rear)
    {
        *front = -1;
        *rear = -1;
}
```

```
    else
    {
        (*front) += 1;
    }
    return first_ele;
}

// printing the queue

void printing(int queue[], int rear, int front)
{

    for (int i = front; i <= rear; i++)
    {
        printf("%d ", queue[i]);
    }
}

int main()
{
    printf("Enter size of the queue\n");
    int size;
    scanf("%d", &size);
    int queue[size];
    int rear = -1;
    int front = -1;
    enqueue(queue, &rear, &front, 12, size);
    enqueue(queue, &rear, &front, 13, size);
    enqueue(queue, &rear, &front, 14, size);
    enqueue(queue, &rear, &front, 15, size);
    enqueue(queue, &rear, &front, 16, size);
    printing(queue, rear, front);
    dequeue(queue, &rear, &front, size);
    printf("\n");
    printf("Printing after deleting\n");
    printing(queue, rear, front);

    return 0;
}
```

OUTPUT:

PS C:\Users\abhip\Desktop\practical c> gcc Queue.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Enter size of the queue

6

12 13 14 15 16

Printing after deleting

13 14 15 16

Q 15. Write a programme to implement circular queue using an array.

Code:

```
#include <stdio.h>
#include <limits.h>
// function to check either the queue is empty or not
int isEmpty(int front, int rear)
{
    if (front == -1 && rear == -1)
    {
        return 1;
    }
    return 0;
}
// function for insertion in circular queue
void enqueue(int queue[], int *rear, int *front, int size, int x)
{
    if (*front == (*rear + 1) % size)
    {
        printf("Overflow");
    }
    else
    {
        if (isEmpty(*front, *rear))
        {
            *front = 0;
        }
        *rear = (*rear + 1) % size;
        queue[*rear] = x;
    }
}

// function for deletion in circular queue

void dequeue(int queue[], int *rear, int *front, int size)
{
    int result = queue[*front];
    if (isEmpty(*front, *rear))
    {
        printf("Underflow");
    }
```

```
  // single element check
    else if (*rear == *front)
    {
        *front = -1;
        *rear = -1;
    }
    else
    {
        *front = (*front + 1) % size;
    }
}

// printing the queue

void print(int queue[], int rear, int front)
{
    if (isEmpty(front, rear))
    {
        printf("Nothing to print");
    }
    else
    {
        for (int i = front; i <= rear; i++)
        {
            printf("%d ", queue[i]);
        }
    }
}
// Function to peek
int peek(int queue[], int rear, int front)
{
    if (isEmpty(front, rear))
    {
        return INT_MIN;
    }
    else
    {
        return queue[front];
    }
}
```

```c
int main()
{
    int rear = -1;
    int front = -1;
    int queue[10];
    int size = sizeof(queue) / sizeof(queue[0]);
    // enqueue elements
    enqueue(queue, &rear, &front, size, 20);
    enqueue(queue, &rear, &front, size, 21);
    enqueue(queue, &rear, &front, size, 22);
    enqueue(queue, &rear, &front, size, 23);
    enqueue(queue, &rear, &front, size, 24);
    print(queue, rear, front);
    // deleting element
    printf("\n");
    printf("Deleting element from queue\n");
    dequeue(queue, &rear, &front, size);
    print(queue, rear, front);
    printf("\n");
    printf("Peeking the queue\n");
    printf("%d", peek(queue, rear, front));
}
```

OUTPUT:

PS C:\Users\abhip\Desktop\practical c> gcc Circular_Queue.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

20 21 22 23 24

Deleting element from queue

21 22 23 24

Peeking the queue

21

Q 16. Write a programme to implement input-restricted queue using an array.

Code:

```
#include <stdio.h>

// function to check whether the queue is empty or not
int isEmpty(int front, int rear)
{
    if (front == -1 && rear == -1)
    {
        return 1;
    }
    return 0;
}
// function to insert in input restricted queue
void enqueue(int queue[], int *front, int *rear, int x, int size)
{
    if (*rear == size - 1)
    {
        printf("Overflow");
    }
    else
    {
        if (isEmpty(*front, *rear))
        {
            *front = 0;
        }
        *rear += 1;
        queue[*rear] = x;
    }
}
// Function to dequeue
void dequeue(int queue[], int *rear, int *front, int size, int choice)
{
    // check whether the queue is empty or not
    if (isEmpty(*front, *rear))
    {
        printf("Underflow");
    }

    else if (*front == *rear)
    {
        *front = -1;
        *rear = -1;
    }
```

```
    else if (choice == 0)
    {
        (*front) += 1;
    }
    else if (choice == 1)
    {
        (*rear) -= 1;
    }
}

// function for printing queue
void printing(int queue[], int front, int rear)
{
    if (isEmpty(front, rear))
    {
        printf("Nothing to print");
    }
    for (int i = front; i <= rear; i++)
    {
        printf("%d ", queue[i]);
    }
}

int main()
{
    printf("Enter size of the queue\n");
    int size;
    scanf("%d", &size);
    int queue[size];
    int front = -1;
    int rear = -1;
    enqueue(queue, &front, &rear, 100, size);
    enqueue(queue, &front, &rear, 101, size);
    enqueue(queue, &front, &rear, 102, size);
    enqueue(queue, &front, &rear, 103, size);
    // printing queue
    printing(queue, front, rear);
    printf("\n");
    // Asking for choice of the user
    printf("Enter 0 to delete from front ans 1 to delete from rear: ");
    int choice;
    scanf("%d", &choice);
    // deleting element
    dequeue(queue, &rear, &front, size, choice);
    printf("Queue after deleting the desired element\n");
    printing(queue, front, rear);
}
```

Output 1:

```
PS C:\Users\abhip\Desktop\practical c> gcc Ip_res_q.c
PS C:\Users\abhip\Desktop\practical c> ./a.exe
Enter size of the queue
5
100 101 102 103
Enter 0 to delete from front ans 1 to delete from rear: 0
Queue after deleting the desired element
101 102 103
```

Output 2:

```
PS C:\Users\abhip\Desktop\practical c> gcc Ip_res_q.c
PS C:\Users\abhip\Desktop\practical c> ./a.exe
Enter size of the queue
5
100 101 102 103
Enter 0 to delete from front ans 1 to delete from rear: 1
Queue after deleting the desired element
100 101 102
```

Q 17. Write a programme to implement Output-restricted queue using an array.

Code:

```
#include <stdio.h>
// function to check whether the queue is empty or not
int isEmpty(int front, int rear)
{
    if (front == -1 && rear == -1)
    {
        return 1;
    }
    return 0;
}

// Function to insert into output restricted queue
void insertion(int queue[], int *front, int *rear, int size, int x, int
choice)
{
    if (choice == 0)
    {
        if (*rear == size - 1)
        {
            printf("Overflow");
            return;
        }
        else if (isEmpty(*front, *rear))
        {
            *front = 0;
            *rear = 0;
            queue[*front] = x;
        }
        else
        {
            for (int i = size - 1; i >= *front; i--)
            {
                queue[i + 1] = queue[i];
            }
            queue[*front] = x;
            *rear += 1;
        }
    }
```

```
  else if (choice == 1)
    {
        if (*rear == size - 1)
        {
            printf("overflow");
            return;
        }
        else if (isEmpty(*front, *rear))
        {
            *front = 0;
            *rear = 0;
            queue[*rear] = x;
        }
        *rear += 1;
        queue[*rear] = x;
    }
}
// function to delete
int deletion(int queue[], int *front, int *rear, int size)
{
    int first_ele = queue[0];
    if (isEmpty(*front, *rear))
    {
        printf("Underflow");
    }
    else if (*front == *rear)
    {
        *front = -1;
        *rear = -1;
    }
    else
    {
        (*front) += 1;
    }
    return first_ele;
}
// function to print the queue
void printing(int queue[], int front, int rear)
{
    if (isEmpty(front, rear))
    {
        printf("Nothing to print");
    }
    for (int i = front; i <= rear; i++)
    {
        printf("%d ", queue[i]);
    }
}
```

```c
int main()
{
    printf("Enter size of the queue\n");
    int size;
    scanf("%d", &size);
    int queue[size];
    printf("Enter 0 to enter from front or 1 to enter from behind\n");
    int choice;
    scanf("%d", &choice);
    int front = -1;
    int rear = -1;
    // entering into queue
    insertion(queue, &front, &rear, size, 100, choice);
    insertion(queue, &front, &rear, size, 101, choice);
    insertion(queue, &front, &rear, size, 102, choice);
    insertion(queue, &front, &rear, size, 103, choice);
    // printing
    printing(queue, front, rear);
}
```

Output 1:

PS C:\Users\abhip\Desktop\practical c> gcc Op_res_q.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Enter size of the queue

5

Enter 0 to enter from front or 1 to enter from behind

0

103 102 101 100

Output 2:

PS C:\Users\abhip\Desktop\practical c> gcc Op_res_q.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Enter size of the queue

5

Enter 0 to enter from front or 1 to enter from behind

1

100 100 101 102 103

Q 18. Write a programme to perform Quick Sort.

Code:

```c
#include <stdio.h>

void quicksort(int array[], int beg, int end)
{
    if (beg < end)
    {
        int pivot = array[beg];
        int left = beg + 1;
        int right = end;
        while (left <= right)
        {
            while (left <= right && array[left] <= pivot)
            {
                left++;
            }
            while (left <= right && array[right] > pivot)
            {
                right--;
            }

            if (left < right)
            {
                // Swap
                int temp = array[left];
                array[left] = array[right];
                array[right] = temp;
            }
        }

        // Swap pivot with array[right]
        int temp2 = array[beg];
        array[beg] = array[right];
        array[right] = temp2;

        // Recursively call for left and right subarrays
        quicksort(array, beg, right - 1);
        quicksort(array, right + 1, end);
    }
}
```

```c
int main()
{
    int array[] = {12, 21, 34, 90, 98, 89, 76, 56};
    int size = sizeof(array) / sizeof(array[0]);

    // Sorting the array
    quicksort(array, 0, size - 1);

    printf("Sorted array is:\n");
    for (int i = 0; i < size; i++)
    {
        printf("%d ", array[i]);
    }

    return 0;
}
```

Output:

PS C:\Users\abhip\Desktop\practical c> gcc quicksort.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Sorted array is:

12 21 34 56 76 89 90 98

Q 19. Write a programme to implement Tower of Hanoi.

Code:

```c
#include <stdio.h>
#include <stdlib.h>

// function for tower of hanoi

void tower(int n, char beg, char aux, char end)
{
    if (n <= 0)
    {
        printf("Invalid number");
    }
    else if (n == 1)
    {
        printf("Move disc from %c to %c\n", beg, end);
    }
    else
    {
        tower(n - 1, beg, end, aux);
        tower(1, beg, aux, end);
        tower(n - 1, aux, beg, end);
    }
}

int main()
{
    int n;
    char a = 'A', b = 'B', c = 'C';
    printf("Enter the number of discs \n");
    scanf("%d", &n);
    printf("Tower od hanoi for %d discs is: \n", n);
    tower(n, a, b, c);
}
```

Output:

PS C:\Users\abhip\Desktop\practical c> gcc Tower_of_hanoi.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Enter the number of discs

3

Tower od hanoi for 3 discs is:

Move disc from A to C

Move disc from A to B

Move disc from C to B

Move disc from A to C

Move disc from B to A

Move disc from B to C

Move disc from A to C

Q 20. Write a programme to implement Singly linked list.

Code:

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
// Structure for Node
typedef struct Node {
  int data;
  struct Node *next;
} Node;
// Structure for head
typedef struct linkedlist {
  Node *head;
} linkedlist;
// function to add a new node at the beginning of the list
void addNode(int data, linkedlist *list) {
  // creating new node
  Node *newNode = (Node *)malloc(sizeof(Node));
  if (list->head == NULL) {
    list->head = newNode;
  } else {
    newNode->next = list->head;
    list->head = newNode;
  }
  newNode->data = data;
}
```

```
// function to insert at any position in linked list
```

```
void addAtpos(int data, int pos, linkedlist *list) {
  Node *newNode = (Node *)malloc(sizeof(Node));
  newNode->data = data;
  if (list->head == NULL || pos == 0) {
    // If the list is empty or insertion at the beginning is explicitly
    // requested
    newNode->next = list->head;
    list->head = newNode;
  } else {
    Node *pointer = list->head;
    int i = 0;
    while (pointer->next != NULL && i < pos - 1) {
      pointer = pointer->next;
      i++;
    }
    newNode->next = pointer->next;
    pointer->next = newNode;
  }
}
// function to delete from beginning of the linked list
int deleteFromBeg(linkedlist *list) {
  if (list->head == NULL) {
    printf("Underflow");
    return INT_MIN;
  }
  Node *firstelement = list->head;
  int data = firstelement->data;
  list->head = list->head->next;
  free(firstelement);
  return data;
}
```

```
// function to delete from any position in the linked list
int delete_specific(linkedlist *list, int pos) {
  if (list->head == NULL) {
    printf("Underflow");
    return INT_MIN;
  }
  if (pos == 0) {
    Node *firstelement = list->head;
    int data = firstelement->data;
    list->head = list->head->next;
    free(firstelement);
    free(list->head);
    return data;
  }
  else {
    Node *pointer = list->head;
    int i = 0;
    while (!(i == pos - 1)) {
      pointer = pointer->next;
      i++;
    }
    Node *tempNode = pointer->next;
    pointer->next = pointer->next->next;
    tempNode->next = NULL;
    return tempNode->data;
    free(tempNode);
  }
}
```

```
// Function to delete if the element data is given
int delete_given_data(linkedlist *list, int data) {
  if (list->head == NULL) {
    printf("Underflow");
    return INT_MIN;
  }
  Node *pointer = list->head;
  // If the element is at the first position
  if (pointer->data == data) {
    list->head = list->head->next;
    int deletedData = pointer->data;
    free(pointer);
    return deletedData;
  }
  // if element is at another position
  Node *tempNode = list->head->next;
  while (!(tempNode->data == data)) {
    pointer = pointer->next;
    tempNode = tempNode->next;
  }
  pointer->next = tempNode->next;
  tempNode->next = NULL;
  return tempNode->data;
  free(tempNode);
}
// printing list
void printList(linkedlist list) {
  Node *current = list.head;
  while (current != NULL) {
    printf("%d ", current->data);
    current = current->next;
```

```
  }
  printf("\n");
}
int main() {
  linkedlist list;
  list.head = NULL;
  addNode(12, &list);
  addNode(11, &list);
  addNode(10, &list);
  addNode(9, &list);
  addNode(7,&list);
  addNode(6, &list);
  printList(list);
 // adding at specific position in linked list
  printf("Enter the position you want to enter the data\n");
  int pos;
  scanf("%d", &pos);
  addAtpos(13, pos, &list);
  printList(list);
  // deleting node
  printf("Deleted element is %d \n", deleteFromBeg(&list));
  printf("Linked list after deleting first element is: ");
  printList(list);
  // deleting from specified position
  printf("Enter position from which you want to delete the Node\n");
  int pos2;
  scanf("%d", &pos2);
  printf("Deleted element is %d \n", delete_specific(&list, pos2));
  printf("Linked list after deleting  element is: ");
  printList(list);
  // deleting given element
```

```
  printf("Enter the element you want to delete: ");
  int element;
  scanf("%d", &element);
  printf("Deleted element is %d\n", delete_given_data(&list, element));
  printf("Linked list after deleting the element is: ");
  printList(list);
  return 0;
}
```

Output:

PS C:\Users\abhip\Desktop\practical c> gcc Linked_list.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

6 7 9 10 11 12

Enter the position you want to enter the data

6

6 7 9 10 11 12 13

Deleted element is 6

Linked list after deleting first element is: 7 9 10 11 12 13

Enter position from which you want to delete the Node

5

Deleted element is 13

Linked list after deleting element is: 7 9 10 11 12

Enter the element you want to delete: 12

Deleted element is 12

Linked list after deleting the element is: 7 9 10 11

Q 21. Write a programme to implement two stacks in one array.

Code:

```c
#include <stdio.h>
#include <stdlib.h>

int twostacks[11];
int top1 = -1;
int top2 = sizeof(twostacks) / sizeof(twostacks[0]);

// function to check whether the stack is full or not
int isFull()
{
    return top1 + 1 == top2;
}

// function to check whether the stack1 is empty or not
int isEmpty1()
{
    return top1 == -1;
}
// function to check whether the stack2 is empty or not
int isEmpty2()
{
    return top2 == sizeof(twostacks) / sizeof(twostacks[0]);
}

// Function to push in stack 1
void push1(int value)
{
    if (isFull())
    {
        printf("Stack 1 Overflow\n");
    }
    top1 += 1;
    twostacks[top1] = value;
}

// Function to push in stack 2
void push2(int value)
{
    if (isFull())
    {
        printf("Stack 2 Overflow\n");
    }
```

```
    top2 -= 1;
    twostacks[top2] = value;
}

// function to pop from stack 1

void pop1()
{
    if (isEmpty1())
    {
        printf("Underflow stack 1\n");
    }
    top1--;
}
// Function to pop from stack 2
void pop2()
{
    if (isEmpty2())
    {
        printf("Underflow stack 2\n");
    }
    top2++;
}

// function to print stack1
void print1()
{
    if (isEmpty1())
    {
        printf("Underflow stack 1");
    }
    for (int i = 0; i <= top1; i++)
    {
        printf("%d ", twostacks[i]);
    }
}
// function to print stack2
void print2()
{
    if (isEmpty2())
    {
        printf("Underflow stack 2");
    }
    for (int i = 10; i >= top2; i--)
    {
        printf("%d ", twostacks[i]);
    }
}
```

```c
int main()
{
    // pushing in stack1
    push1(10);
    push1(20);
    push1(30);
    push1(40);
    push1(50);
    // printing
    print1();
    printf("\n");
    printf("Stack after deleting one element\n");
    // deleting
    pop1();
    print1();
    printf("\n");
    // pushing in stack2
    push2(1);
    push2(2);
    push2(3);
    push2(4);
    push2(5);
    push2(6);
    // printing
    print2();
    printf("\n");
    // deleting
    pop2();
    print2();
}
```

Output:

PS C:\Users\abhip\Desktop\practical c> gcc Onearray_twostacks.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Stack 1 is: 10 20 30 40 50

Stack 1 after deleting one element

10 20 30 40

Stack 2 is: 1 2 3 4 5 6

Stack 2 after deleting one element

1 2 3 4 5

Q 22. Write a programme to implement queue using linked list.

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
  int data;
  struct Node *next;
} Node;
Node *front = NULL;
Node *rear = NULL;
// Function to check if the queue is empty
int isEmpty() { return (front == NULL && rear == NULL); }
// insertion in queue
void enqueue(int x) {
  // create new node
  Node *newNode = (Node *)malloc(sizeof(Node));
  newNode->data = x;
  newNode->next = NULL;
  if (isEmpty()) {
    front = rear = newNode;
  }
  else{
    rear->next = newNode;
    rear = newNode;
  }
}
```

```
// function to display queue
void display(){
  Node *temp = front;
  if(isEmpty()){
    printf("Linkedlist is empty");
  }
  else{
    while(temp!=NULL){
      printf("%d ",temp->data);
      temp = temp->next;
    }
  }
}
// function to delete from queue
void dequeue(){
  Node *temp = front;
  if(isEmpty()){
    printf("List is empty");
  }
  else{
    printf("Deleted element is: %d",temp->data);
    front = front->next;
    free(temp);
  }
}
// main function
int main(){
  enqueue(10);
  enqueue(11);
  enqueue(12);
  enqueue(13);
```

```
 enqueue(14);
 printf("Queue created by you is: ");
 display();
 printf("\n");
 // deleting
 dequeue();
 printf("\n");
 printf("Queue after dequeuing an element");
 display();
}
```

Output:

PS C:\Users\abhip\Desktop\practical c> gcc Queue_using_linkedlist.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Queue created by you is: 10 11 12 13 14

Deleted element is: 10

Queue after dequeuing an element11 12 13 14

Q 23. Write a programme to implement stack using linked list.

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
  int data;
  struct Node *next;
} Node;
Node *top = NULL;
// function to check if the stack is empty or not
int isEmpty() { return (top == NULL); }
// function to push
void push(int x) {
  Node *newNode = (Node *)malloc(sizeof(Node));
  newNode->data = x;
  newNode->next = top;
  top = newNode;
}
// function to display
void display() {
    Node *temp = top;
    if (isEmpty()) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements: ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
```

```
        printf("\n");
    }
}
// function to pop
void pop(){
  Node *temp = top;
  printf("Deleted element is:%d\n",temp->data);
  top = top->next;
  free(temp);
}
int main(){
  push(20);
  push(30);
  push(40);
  push(50);
  push(60);
  push(70);
  display();
  pop();
  display();
  }
```

Output:

PS C:\Users\abhip\Desktop\practical c> gcc Stack_using_linkedlist.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Stack elements: 70 60 50 40 30 20

Deleted element is:70

Stack elements: 60 50 40 30 20

Q 24. Write a programme to implement insertion , deletion and displaying a Binary Search Tree.

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *lchild, *rchild;
};
struct node *root;
// Function to insert a node in BST
void InsBST(int x) {
   struct node *newNode = (struct node *)malloc(sizeof(struct node));
   newNode->data = x;
   newNode->lchild = NULL;
   newNode->rchild = NULL;

   struct node *current = root;
   struct node *parent = NULL;

   while (current != NULL) {
      parent = current;
      if (x < current->data) {
         current = current->lchild;
      } else {
         current = current->rchild;
      }
   }
```

```
    if (parent == NULL) {
        root = newNode; // Tree is empty
    } else if (x < parent->data) {
        parent->lchild = newNode;
    } else {
        parent->rchild = newNode;
    }
}
// Function to find the node with the minimum value in a BST
struct node *findMin(struct node *root) {
    while (root->lchild != NULL) {
        root = root->lchild;
    }
    return root;
}
// Function to delete a node from BST
struct node *DeleteNode(struct node *root, int key) {
    if (root == NULL) {
        return root;
    }
  if (key < root->data) {
        root->lchild = DeleteNode(root->lchild, key);
    } else if (key > root->data) {
        root->rchild = DeleteNode(root->rchild, key);
    } else {
        // Node with only one child or no child
        if (root->lchild == NULL) {
            struct node *temp = root->rchild;
            free(root);
            return temp;
        } else if (root->rchild == NULL) {
```

```
    struct node *temp = root->lchild;
        free(root);
        return temp;
      }
       // Node with two children: Get the inorder successor (smallest
      // in the right subtree) or predecessor (largest in the left subtree)
      struct node *temp = findMin(root->rchild);
     // Copy the inorder successor's content to this node
      root->data = temp->data;

      // Delete the inorder successor
      root->rchild = DeleteNode(root->rchild, temp->data);
   }
   return root;
}

// Function to display the BST in-order
void display(struct node *root) {
   if (root != NULL) {
      display(root->lchild);
      printf("%d ", root->data);
      display(root->rchild);
   }
}
```

```
int main() {
    int x, flag = 1;
    char ans;
    printf("Enter a value: ");
    scanf("%d", &x);

    root = (struct node *)malloc(sizeof(struct node));
    root->data = x;
    root->lchild = NULL;
    root->rchild = NULL;
    display(root);
    while (flag) {
        printf("Enter an element: ");
        scanf("%d", &x);
      InsBST(x);
        printf("Do you want to continue (y/n)? ");
        scanf(" %c", &ans);

        if (ans != 'y' && ans != 'Y') {
            flag = 0;
        }
       display(root);
    }
  // Example of deletion
    printf("\nEnter a value to delete: ");
    scanf("%d", &x);
    root = DeleteNode(root, x);
    printf("BST after deletion: ");
    display(root);
   return 0;
}
```

Output:

PS C:\Users\abhip\Desktop\practical c> gcc BST.c

PS C:\Users\abhip\Desktop\practical c> ./a.exe

Enter a value: 45

45

Enter an element: 46

Do you want to continue (y/n)? y

45 46

Enter an element: 44

Do you want to continue (y/n)? n

44 45 46

Enter a value to delete: 45

BST after deletion: 44 46