1. Aim: Program to find largest and smallest element in an array.

Program code:

```
#include <stdio.h>
void main()
{
   int a[10], i, n, max, min;
   printf("Enter the size : ");
   scanf("%d", &n);
   printf("Enter the elements:\n");
   for (i = 0; i < n; i++)
   {
      scanf("%d", &a[i]);
   }
   max = a[0];
   for (i = 0; i < n; i++)
   {
      if (a[i] > max)
      {
         max = a[i];
      }
   }
   min = a[0];
   for (i = 0; i < n; i++)
   {
      if (a[i] < min)
      {
         min = a[i];
      }
   }
   printf("The maximum number in the array is %d\n", max);
   printf("The minimum number in the array is %d\n", min);
}
```

Output:

Enter the size : 4

Enter the elements:

15 40 81 5

The maximum number in the array is 81

The minimum number in the array is 5

2. Aim: Program to Merge two sorted arrays and store in a third array.

Program code:

```
#include <stdio.h>
void main()
{
   int a1[10], a2[10], a3[20], s1, s2, s3, i, j, k;
   printf("Enter the size of array 1 : ");
   scanf("%d", &s1);
   printf("Enter elements of array 1:\n");
   for (i = 0; i < s1; i++)
      scanf("%d", &a1[i]);
   printf("Enter the size of array 2 : ");
   scanf("%d", &s2);
   printf("Enter elements of array 2:\n");
   for (i = 0; i < s2; i++)
      scanf("%d", &a2[i]);
   s3 = s1 + s2;
   i = j = k = 0;
   while (i < s1 && j < s2)
   {
      if (a1[i] < a2[j])
      {
         a3[k] = a1[i];
         i++;
      }
      else
      {
         a3[k] = a2[j];
         j++;
      }
      k++;
   }
```

```
    while (i < s1)
    {
        a3[k] = a1[i];
        i++;
        k++;
    }
    while (j < s2)
    {
        a3[k] = a2[j];
        j++;
        k++;
    }
    printf("Merged array is : ");
    for (i = 0; i < s3; i++)
        printf("%d   ", a3[i]);
    printf("\n");
}
```

Output:

Enter the size of array 1 : 4

Enter elements of array 1:

2 5 7 9

Enter the size of array 2 : 3

Enter elements of array 2:

1 4 6

Merged array is : 1   2   4   5   6   7   9

3. Aim: Program to implement stack operations using array.

Program code:

```
#include <stdlib.h>
#include <stdio.h>
int top = -1;
int stack[5], data, i, size;
void push()
{
    if (top == size - 1)
    {
        printf("Stack is full\n");
    }
    else
    {
        printf("Enter the data : ");
        scanf("%d", &data);
        top++;
        stack[top] = data;
        printf("The element is inserted\n");
    }
}
void pop()
{
    if (top == -1)
    {
        printf("\nStack underflow\n");
    }
    else
    {
        printf("Popped element is %d\n", stack[top]);
        top--;
    }
```

```
}
void display()
{
   if (top == -1)
   {
      printf("\nStack is empty\n");
   }
   else
   {
      printf("The elements in stack are:\n");
      for (i = top; i >= 0; i--)
      {
         printf("%d\n", stack[i]);
      }
   }
}
int main()
{
   int ch;
   printf("Enter the array size : ");
   scanf("%d", &size);
   do
   {
      printf("--OPTIONS--\n1. For push\n2. For pop\n3. For display\n4. For exit\n");
      printf("Enter the choice: ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         push();
         break;
      case 2:
```

```
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\nINVALID CHOICE\n");
        }
    } while (1);
    return 0;
}
```

Output:

Enter the array size : 3

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 1

Enter the data : 11

The element is inserted

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 1

Enter the data : 12

The element is inserted

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 1

Enter the data : 13

The element is inserted

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 1

Stack is full

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 3

The elements in stack are:

13

12

11

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 2

Popped element is 13

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 3

The elements in stack are:

12

11

--OPTIONS--

1. For push

2. For pop

3. For display

4. For exit

Enter the choice: 4

4. Aim: Program to implement queue operations using array.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int front = -1, rear = -1;
int queue[15], data, i, size;
void enqueue()
{
   if (rear == size - 1)
   {
      printf("The queue is full\n");
   }
   else
   {
      printf("Enter the data to be inserted: ");
      scanf("%d", &data);
      rear++;
      queue[rear] = data;
      printf("The element is inserted\n");
      if (front == -1)
      {
         front = 0;
      }
   }
}
void dequeue()
{
   if ((front == -1) || (front > rear))
   {
      printf("\nThe queue is empty\n");
   }
   else
```

```
    {
        printf("%d is deleted\n", queue[front]);
        front++;
    }
}
void display()
{
    if (front == -1 || front > rear)
    {
        printf("\nThe queue is empty\n");
    }
    else
    {
        printf("The elements are:\n");
        for (i = front; i <= rear; i++)
        {
            printf("%d\n", queue[i]);
        }
    }
}
int main()
{
    int ch;
    printf("Enter the array size: ");
    scanf("%d", &size);
    do
    {
        printf("--OPTIONS--\n1. For enqueue\n2. For dequeue\n3. For display\n4. Exit\n");
        printf("Enter the choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
```

```
        case 1:
          enqueue();
          break;
        case 2:
          dequeue();
          break;
        case 3:
          display();
          break;
        case 4:
          exit(0);
        default:
          printf("Invalid Choice\n");
      }
    } while (1);
    return 0;
}
```

Output:

```
Enter the array size: 3
--OPTIONS--
1. For enqueue
2. For dequeue
3. For display
4. Exit
Enter the choice: 1
Enter the data to be inserted: 31
The element is inserted
--OPTIONS--
1. For enqueue
2. For dequeue
3. For display
4. Exit
```

Enter the choice: 1

Enter the data to be inserted: 32

The element is inserted

--OPTIONS--

1. For enqueue

2. For dequeue

3. For display

4. Exit

Enter the choice: 1

Enter the data to be inserted: 33

The element is inserted

--OPTIONS--

1. For enqueue

2. For dequeue

3. For display

4. Exit

Enter the choice: 1

The queue is full

--OPTIONS--

1. For enqueue

2. For dequeue

3. For display

4. Exit

Enter the choice: 3

The elements are:

31

32

33

--OPTIONS--

1. For enqueue

2. For dequeue

3. For display

4. Exit

Enter the choice: 2

31 is deleted

--OPTIONS--

1. For enqueue

2. For dequeue

3. For display

4. Exit

Enter the choice: 3

The elements are:

32

33

--OPTIONS--

1. For enqueue

2. For dequeue

3. For display

4. Exit

Enter the choice: 4

5. Aim: Program to implement Circular queue operations.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int rear = -1, front = -1;
int data, queue[10], size, i;
void enqueue()
{
    if ((front == 0 && rear == size - 1) || (rear == (front - 1) % (size - 1)))
    {
        printf("Queue is full\n");
    }
    else
    {
        rear = (rear + 1) % size;
        printf("Enter the data: ");
        scanf("%d", &data);
        queue[rear] = data;
        printf("The element is inserted\n");

        if (front == -1)
            front = 0;
    }
}
void dequeue()
{
    if (front == -1)
    {
        printf("\nQueue is empty\n");
    }
    else
    {
```

```
        printf("The deleted element is %d\n", queue[front]);
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
        {
            front = (front + 1) % size;
        }
    }
}
void display()
{
    if (front == -1)
    {
        printf("\nQueue is empty\n");
    }
    else
    {
        printf("Elements are:");
        if (front <= rear)
        {
            for (i = front; i <= rear; i++)
            {
                printf("\t%d", queue[i]);
            }
        }
        else
        {
            for (i = front; i < size; i++)
            {
```

```
            printf("\t%d", queue[i]);
        }
        for (i = 0; i <= rear; i++)
        {
            printf("\t%d", queue[i]);
        }
    }
    printf("\n");
  }
}
int main()
{
  int ch;
  printf("Enter the array size: ");
  scanf("%d", &size);
  do
  {
    printf("--OPTIONS--\n1. For insertion\n2. For deletion\n3. For displaying\n4. For
exiting\n");
    printf("Enter your choice: ");
    scanf("%d", &ch);
    switch (ch)
    {
    case 1:
        enqueue();
        break;
    case 2:
        dequeue();
        break;
    case 3:
        display();
        break;
```

```
        case 4:
            exit(0);
        default:
            printf("INVALID CHOICE\n");
        }
    } while (1);
    return 0;
}
```

Output:

```
Enter the array size: 3
--OPTIONS--
1. For insertion
2. For deletion
3. For displaying
4. For exiting
Enter your choice: 1
Enter the data: 40
The element is inserted
--OPTIONS--
1. For insertion
2. For deletion
3. For displaying
4. For exiting
Enter your choice: 1
Enter the data: 41
The element is inserted
--OPTIONS--
1. For insertion
2. For deletion
3. For displaying
4. For exiting
```

Enter your choice: 1

Enter the data: 42

The element is inserted

--OPTIONS--

1. For insertion

2. For deletion

3. For displaying

4. For exiting

Enter your choice: 1

Queue is full

--OPTIONS--

1. For insertion

2. For deletion

3. For displaying

4. For exiting

Enter your choice: 3

Elements are: 40 41 42

--OPTIONS--

1. For insertion

2. For deletion

3. For displaying

4. For exiting

Enter your choice: 2

The deleted element is 40

--OPTIONS--

1. For insertion

2. For deletion

3. For displaying

4. For exiting

Enter your choice: 1

Enter the data: 46

The element is inserted

--OPTIONS--

1. For insertion

2. For deletion

3. For displaying

4. For exiting

Enter your choice: 3

Elements are: 41 42 46

--OPTIONS--

1. For insertion

2. For deletion

3. For displaying

4. For exiting

Enter your choice: 4

6. Aim: Program for implementing Singly Linked List.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int count = 0;
struct node
{
   int data;
   struct node* next;
} *new, *head = NULL, *h, *l, *sl;
void create()
{
   new = (struct node*)malloc(sizeof(struct node));
   printf("\nEnter the data to the node: ");
   scanf("%d", &new->data);
   new->next = NULL;
}
void ins_beg()
{
   create();
   if (head == NULL)
   {
      head = new;
   }
   else
   {
      new->next = head;
      head = new;
   }
   count++;
}
void ins_end()
```

```
{
    create();
    if (head == NULL)
    {
        head = new;
    }
    else
    {
        h = head;
        while (h->next != NULL)
        {
            h = h->next;
        }
        h->next = new;
    }
    count++;
}
void ins_pos()
{
    int pos, i;
    printf("\nEnter the position: ");
    scanf("%d", &pos);
    if (pos == 1)
        ins_beg();
    else if (pos == count + 1)
        ins_end();
    else if (pos > count + 1 || pos < 1)
        printf("\nInvalid position\n");
    else
    {
        create();
        h = head;
```

```
        for (i = 0; i < pos - 2; i++)
            h = h->next;
        new->next = h->next;
        h->next = new;
        count++;
    }
}
void display()
{
    if (head == NULL)
        printf("\nList is empty\n");
    else
    {
        h = head;
        printf("\nLinked list elements are: ");
        while (h != NULL)
        {
            printf("%d -> ", h->data);
            h = h->next;
        }
        printf("NULL\n");
    }
}
void del_beg()
{
    if (head == NULL)
        printf("\nList is empty\n");
    else
    {
        h = head;
        head = head->next;
        printf("\nDeleted element is %d\n", h->data);
```

```
        free(h);
        count--;
    }
}
void del_end()
{
    if (head == NULL)
        printf("\nList is empty\n");
    else if (head->next == NULL)
    {
        printf("\nDeleted element is %d\n", head->data);
        free(head);
        head = NULL;
        count--;
    }
    else
    {
        l = head;
        sl = NULL;
        while (l->next != NULL)
        {
            sl = l;
            l = l->next;
        }
        sl->next = NULL;
        printf("\nDeleted element is %d\n", l->data);
        free(l);
        count--;
    }
}
void del_pos()
{
```

```
    int pos, i;
    if (head == NULL)
        printf("\nList is empty\n");
    else
    {
        printf("\nEnter the position to delete: ");
        scanf("%d", &pos);
        if (pos < 1 || pos > count)
        {
            printf("\nInvalid position\n");
            return;
        }
        if (pos == 1)
            del_beg();
        else if (pos == count)
            del_end();
        else
        {
            l = head;
            sl = NULL;
            for (i = 1; i < pos; i++)
            {
                sl = l;
                l = l->next;
            }
            sl->next = l->next;
            printf("\nDeleted element is %d\n", l->data);
            free(l);
            count--;
        }
    }
}
```

```
int main()
{
   int ch = 0;
   do
   {
      printf("\n--Options--\n");
      printf("1) Insert at start\n2) Insert at end\n3) Insert at a position\n4) Delete at start\n5)
Delete at end\n6) Delete at a position\n7) Display\n8) Exit\n");
      printf("\nChoose option: ");
      scanf("%d", &ch);
      switch (ch)
      {
         case 1:
            ins_beg();
            break;
         case 2:
            ins_end();
            break;
         case 3:
            ins_pos();
            break;
         case 4:
            del_beg();
            break;
         case 5:
            del_end();
            break;
         case 6:
            del_pos();
            break;
         case 7:
            display();
```

```
            break;
        case 8:
            printf("Exiting\n");
            break;
        default:
            printf("Wrong Choice\n");
      }
   } while (ch != 8);
   return 0;
}
```

Output:

```
--Options--
1) Insert at start
2) Insert at end
3) Insert at a position
4) Delete at start
5) Delete at end
6) Delete at a position
7) Display
8) Exit
Choose option: 1
Enter the data to the node: 13
--Options--
1) Insert at start
2) Insert at end
3) Insert at a position
4) Delete at start
5) Delete at end
6) Delete at a position
7) Display
8) Exit
```

Choose option: 1

Enter the data to the node: 11

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 1

Enter the data to the node: 15

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 7

Linked list elements are: 15 -> 11 -> 13 -> NULL

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 2

Enter the data to the node: 16

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 3

Enter the position: 2

Enter the data to the node: 31

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 7

Linked list elements are: 15 -> 31 -> 11 -> 13 -> 16 -> NULL

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 4

Deleted element is 15

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 5

Deleted element is 16

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 6

Enter the position to delete: 3

Deleted element is 13

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 7

Linked list elements are: 31 -> 11 -> NULL

--Options--

1) Insert at start

2) Insert at end

3) Insert at a position

4) Delete at start

5) Delete at end

6) Delete at a position

7) Display

8) Exit

Choose option: 8

Exiting

7. Aim: Program for implementing stack using Singly Linked- Push, Pop, Linear Search.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int count = 0;
struct node {
    int data;
    struct node* next;
} *new, *head = NULL, *h, *l, *sl;
void create() {
    new = (struct node*)malloc(sizeof(struct node));
    printf("Enter the data for the node: ");
    scanf("%d", &new->data);
    new->next = NULL;
}
void push() {
    create();
    if (head == NULL) {
        head = new;
    } else {
        h = head;
        while (h->next != NULL) {
            h = h->next;
        }
        h->next = new;
    }
    count++;
}
void pop() {
    if (head == NULL) {
        printf("List is empty\n");
```

```
    } else if (head->next == NULL) {
        printf("Deleted element is %d\n", head->data);
        free(head);
        head = NULL;
        count--;
    } else {
        l = head;
        sl = head;
        while (l->next != NULL) {
            sl = l;
            l = l->next;
        }
        sl->next = NULL;
        printf("Deleted element is %d\n", l->data);
        free(l);
        count--;
    }
}
void display() {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        h = head;
        printf("Linked list elements are: ");
        while (h != NULL) {
            printf("%d -> ", h->data);
            h = h->next;
        }
        printf("NULL\n");
    }
}
void search() {
```

```
    int se, i = 0, f = 0;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Enter the data to be searched: ");
    scanf("%d", &se);
    h = head;
    while (h != NULL) {
        i++;
        if (h->data == se) {
            printf("Element %d found at position %d\n", se, i);
            f = 1;
        }
        h = h->next;
    }
    if (!f)
        printf("Element %d not found\n", se);
}
void main() {
    int ch;
    while (1) {
        printf("\nChoose an option:\n");
        printf("1) Push\n");
        printf("2) Pop\n");
        printf("3) Search\n");
        printf("4) Display\n");
        printf("5) Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
```

```
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            search();
            break;
        case 4:
            display();
            break;
        case 5:
            exit(0);
        default:
            printf("Wrong choice! Please try again.\n");
      }
   }
}
```

Output:

```
Choose an option:
1) Push
2) Pop
3) Search
4) Display
5) Exit
Enter your choice: 1
Enter the data for the node: 22
Choose an option:
1) Push
2) Pop
3) Search
4) Display
```

5) Exit

Enter your choice: 1

Enter the data for the node: 23

Choose an option:

1) Push

2) Pop

3) Search

4) Display

5) Exit

Enter your choice: 1

Enter the data for the node: 24

Choose an option:

1) Push

2) Pop

3) Search

4) Display

5) Exit

Enter your choice: 4

Linked list elements are: 22 -> 23 -> 24 -> NULL

Choose an option:

1) Push

2) Pop

3) Search

4) Display

5) Exit

Enter your choice: 2

Deleted element is 24

Choose an option:

1) Push

2) Pop

3) Search

4) Display

5) Exit

Enter your choice: 3

Enter the data to be searched: 22

Element 22 found at position 1

Choose an option:

1) Push

2) Pop

3) Search

4) Display

5) Exit

Enter your choice: 4

Linked list elements are: 22 -> 23 -> NULL

Choose an option:

1) Push

2) Pop

3) Search

4) Display

5) Exit

Enter your choice: 5

8. Aim: Program for implementing Queue using Singly Linked.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int count = 0;
struct node {
    int data;
    struct node *next;
} *new, *head = NULL, *h;
void create() {
    new = (struct node *)malloc(sizeof(struct node));
    if (new == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    printf("Enter the data for the node: ");
    scanf("%d", &new->data);
    new->next = NULL;
}
void enqueue() {
    create();
    if (head == NULL) {
        head = new;
    } else {
        h = head;
        while (h->next != NULL) {
            h = h->next;
        }
        h->next = new;
    }
    count++;
}
```

```
void dequeue() {
    if (head == NULL) {
        printf("Queue is empty\n");
    } else {
        h = head;
        head = head->next;
        printf("Deleted element is %d\n", h->data);
        free(h);
        count--;
    }
}
void display() {
    if (head == NULL) {
        printf("Queue is empty\n");
    } else {
        h = head;
        printf("Queue elements are: ");
        while (h != NULL) {
            printf("%d -> ", h->data);
            h = h->next;
        }
        printf("NULL\n");
    }
}
void main() {
    int ch = 0;
    do {
        printf("\n--- Queue Operations ---\n");
        printf("1) Enqueue\n");
        printf("2) Dequeue\n");
        printf("3) Display\n");
        printf("4) Exit\n");
```

```
        printf("Choose an option: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid choice, please try again.\n");
        }
    } while (ch != 4);
}
```

Output:

```
--- Queue Operations ---
1) Enqueue
2) Dequeue
3) Display
4) Exit
Choose an option: 1
Enter the data for the node: 15
--- Queue Operations ---
1) Enqueue
2) Dequeue
3) Display
```

4) Exit

Choose an option: 1

Enter the data for the node: 16

--- Queue Operations ---

1) Enqueue

2) Dequeue

3) Display

4) Exit

Choose an option: 1

Enter the data for the node: 17

--- Queue Operations ---

1) Enqueue

2) Dequeue

3) Display

4) Exit

Choose an option: 3

Queue elements are: 15 -> 16 -> 17 -> NULL

--- Queue Operations ---

1) Enqueue

2) Dequeue

3) Display

4) Exit

Choose an option: 2

Deleted element is 15

--- Queue Operations ---

1) Enqueue

2) Dequeue

3) Display

4) Exit

Choose an option: 3

Queue elements are: 16 -> 17 -> NULL

--- Queue Operations ---

1) Enqueue

2) Dequeue

3) Display

4) Exit

Choose an option: 4

Exiting program...

9. Aim: Program for Doubly linked list - Insertion, Deletion, Display, search.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int count = 0;
struct node {
   struct node* prev;
   int data;
   struct node* next;
} *new, *head = NULL, *h;
void create() {
   new = (struct node*)malloc(sizeof(struct node));
   if (new == NULL) {
      printf("Memory allocation failed!\n");
      exit(0);
   }
   printf("Enter the data for the node: ");
   scanf("%d", &new->data);
   new->next = NULL;
   new->prev = NULL;
}
void ins_beg() {
   create();
   if (head == NULL) {
      head = new;
   } else {
      new->next = head;
      head->prev = new;
      head = new;
   }
   count++;
```

```
}
void ins_end() {
    create();
    if (head == NULL) {
        head = new;
    } else {
        h = head;
        while (h->next != NULL) {
            h = h->next;
        }
        h->next = new;
        new->prev = h;
    }
    count++;
}
void ins_pos() {
    int pos, i;
    printf("Enter the position: ");
    scanf("%d", &pos);
    if (pos == 1) {
        ins_beg();
    } else if (pos == count + 1) {
        ins_end();
    } else if (pos < 1 || pos > count + 1) {
        printf("Invalid position\n");
    } else {
        create();
        h = head;
        for (i = 1; i < pos - 1; i++) {
            h = h->next;
        }
        new->next = h->next;
```

```
        new->prev = h;
        h->next->prev = new;
        h->next = new;
        count++;
    }
}
void display() {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        h = head;
        printf("Linked list elements are: ");
        while (h != NULL) {
            printf("%d -> ", h->data);
            h = h->next;
        }
        printf("NULL\n");
    }
}
void search() {
    int sc, i = 0, f = 0;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Enter the element to search: ");
    scanf("%d", &sc);
    h = head;
    while (h != NULL) {
        i++;
        if (h->data == sc) {
            printf("Element %d found at position %d\n", sc, i);
```

```
            f = 1;
        }
        h = h->next;
    }
    if (!f)
        printf("Element not found\n");
}
void del_beg() {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (head->next == NULL) {
        printf("Deleted element is %d\n", head->data);
        free(head);
        head = NULL;
        count--;
    } else {
        h = head;
        head = head->next;
        head->prev = NULL;
        printf("Deleted element is %d\n", h->data);
        free(h);
        count--;
    }
}
void del_end() {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (head->next == NULL) {
        printf("Deleted element is %d\n", head->data);
        free(head);
        head = NULL;
        count--;
```

```
    } else {
        h = head;
        while (h->next != NULL) {
            h = h->next;
        }
        printf("Deleted element is %d\n", h->data);
        h->prev->next = NULL;
        free(h);
        count--;
    }
}
void del_pos() {
    int pos, i;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Enter the position to delete: ");
    scanf("%d", &pos);
    if (pos == 1) {
        del_beg();
    } else if (pos == count) {
        del_end();
    } else if (pos < 1 || pos > count) {
        printf("Invalid position\n");
    } else {
        h = head;
        for (i = 1; i < pos; i++) {
            h = h->next;
        }
        printf("Deleted element is %d\n", h->data);
        h->prev->next = h->next;
```

```
        h->next->prev = h->prev;
        free(h);
        count--;
    }
}
void main() {
    int ch = 0;
    do {
        printf("\n--- Doubly Linked List Operations ---\n");
        printf("1) Insert at beginning\n");
        printf("2) Insert at end\n");
        printf("3) Insert at a position\n");
        printf("4) Display\n");
        printf("5) Delete at beginning\n");
        printf("6) Delete at end\n");
        printf("7) Delete at a position\n");
        printf("8) Search\n");
        printf("9) Exit\n");
        printf("Choose option: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                ins_beg();
                break;
            case 2:
                ins_end();
                break;
            case 3:
                ins_pos();
                break;
            case 4:
                display();
```

```
            break;
        case 5:
            del_beg();
            break;
        case 6:
            del_end();
            break;
        case 7:
            del_pos();
            break;
        case 8:
            search();
            break;
        case 9:
            printf("Exiting program...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
      }
   } while (ch != 9);
}
```

Output:

```
--- Doubly Linked List Operations ---
1) Insert at beginning
2) Insert at end
3) Insert at a position
4) Display
5) Delete at beginning
6) Delete at end
7) Delete at a position
8) Search
9) Exit
```

Choose option: 1

Enter the data for the node: 33

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 1

Enter the data for the node: 34

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 1

Enter the data for the node: 35

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 4

Linked list elements are: 35 -> 34 -> 33 -> NULL

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 2

Enter the data for the node: 36

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 4

Linked list elements are: 35 -> 34 -> 33 -> 36 -> NULL

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 3

Enter the position: 2

Enter the data for the node: 37

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 4

Linked list elements are: 35 -> 37 -> 34 -> 33 -> 36 -> NULL

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 5

Deleted element is 35

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 6

Deleted element is 36

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 2

Enter the data for the node: 39

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 7

Enter the position to delete: 2

Deleted element is 34

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 8

Enter the element to search: 37

Element 37 found at position 1

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 4

Linked list elements are: 37 -> 33 -> 39 -> NULL

--- Doubly Linked List Operations ---

1) Insert at beginning

2) Insert at end

3) Insert at a position

4) Display

5) Delete at beginning

6) Delete at end

7) Delete at a position

8) Search

9) Exit

Choose option: 9

Exiting program...

10. Aim: Program for binary search tree operations.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   struct node* lchild;
   int data;
   struct node* rchild;
} *root = NULL, *new, *p;
void create()
{
   new = (struct node*)malloc(sizeof(struct node));
   printf("Enter the data to the node: ");
   scanf("%d", &new->data);
   new->lchild = NULL;
   new->rchild = NULL;
}
void search(struct node* rt)
{
   if (new->data < rt->data)
   {
      if (rt->lchild == NULL)
         rt->lchild = new;
      else
         search(rt->lchild);
   }
   else if (new->data > rt->data)
   {
      if (rt->rchild == NULL)
         rt->rchild = new;
      else
```

```
        search(rt->rchild);
    }
    else
    {
        printf("Duplicate data not allowed.\n");
        free(new);
    }
}
void insert()
{
    create();
    if (root == NULL)
        root = new;
    else
        search(root);
}
void preorder(struct node* rt)
{
    if (rt != NULL)
    {
        printf("%d -> ", rt->data);
        preorder(rt->lchild);
        preorder(rt->rchild);
    }
}
void postorder(struct node* rt)
{
    if (rt != NULL)
    {
        postorder(rt->lchild);
        postorder(rt->rchild);
        printf("%d -> ", rt->data);
```

```
    }
}
void inorder(struct node* rt)
{
    if (rt != NULL)
    {
        inorder(rt->lchild);
        printf("%d -> ", rt->data);
        inorder(rt->rchild);
    }
}
void deletenode(struct node* rt)
{
    if (rt->lchild == NULL && rt->rchild == NULL)
    {
        if (rt == root)
        {
            free(rt);
            root = NULL;
        }
        else if (rt == p->lchild)
            p->lchild = NULL;
        else
            p->rchild = NULL;

        if (rt != root)
            free(rt);
    }
    else if (rt->lchild != NULL && rt->rchild == NULL)
    {
        if (rt == root)
        {
```

```
        root = rt->lchild;
        free(rt);
    }
    else if (rt == p->lchild)
    {
        p->lchild = rt->lchild;
        free(rt);
    }
    else
    {
        p->rchild = rt->lchild;
        free(rt);
    }
}
else if (rt->lchild == NULL && rt->rchild != NULL)
{
    if (rt == root)
    {
        root = rt->rchild;
        free(rt);
    }
    else if (rt == p->lchild)
    {
        p->lchild = rt->rchild;
        free(rt);
    }
    else
    {
        p->rchild = rt->rchild;
        free(rt);
    }
}
```

```
    else
    {
        struct node* succParent = rt;
        struct node* succ = rt->rchild;
        while (succ->lchild != NULL)
        {
            succParent = succ;
            succ = succ->lchild;
        }
        rt->data = succ->data;
        if (succParent != rt)
            succParent->lchild = succ->rchild;
        else
            succParent->rchild = succ->rchild;

        free(succ);
    }
}
void dsearch(struct node* rt, int val)
{
    if (rt == NULL)
    {
        printf("Element not found\n");
        return;
    }
    if (val < rt->data)
    {
        p = rt;
        if (rt->lchild != NULL)
            dsearch(rt->lchild, val);
        else
            printf("Element not found\n");
```

```
    }
    else if (val > rt->data)
    {
        p = rt;
        if (rt->rchild != NULL)
            dsearch(rt->rchild, val);
        else
            printf("Element not found\n");
    }
    else
    {
        deletenode(rt);
    }
}
void delete()
{
    int val;
    printf("Enter the value to be deleted: ");
    scanf("%d", &val);
    dsearch(root, val);
}
int main()
{
    int ch;
    do
    {
        printf("\n1) Insert\n2) Preorder\n3) Postorder\n4) Inorder\n5) Delete\n6) Exit");
        printf("\nChoose Option: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
```

```
        insert();
        break;
    case 2:
        if (root == NULL)
            printf("Tree is empty\n");
        else
        {
            preorder(root);
            printf("\n");
        }
        break;
    case 3:
        if (root == NULL)
            printf("Tree is empty\n");
        else
        {
            postorder(root);
            printf("\n");
        }
        break;
    case 4:
        if (root == NULL)
            printf("Tree is empty\n");
        else
        {
            inorder(root);
            printf("\n");
        }
        break;
    case 5:
        delete();
        break;
```

```
        case 6:
            exit(0);
        default:
            printf("Wrong Choice\n");
        }
    } while (1);
    return 0;
}
```

Output:

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 1

Enter the data to the node: 23

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 1

Enter the data to the node: 54

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 1

Enter the data to the node: 45

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 1

Enter the data to the node: 29

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 2

23 -> 54 -> 45 -> 29 ->

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 3

29 -> 45 -> 54 -> 23 ->

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 4

23 -> 29 -> 45 -> 54 ->

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 5

Enter the value to be deleted: 45

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 2

23 -> 54 -> 29 ->

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Delete

6) Exit

Choose Option: 6

11. Aim: Program to implement set data structure and its operations using bit string.

Program code:

```
#include <stdio.h>
int u[10], a[10], b[10], n;
void display(int x[]) {
    int i;
    printf("{");
    for (i = 0; i < n; i++)
        printf("%d,", x[i]);
    printf("}");
}
void bitdis(int x[]) {
    int i;
    printf("{");
    for (i = 0; i < n; i++) {
        if (x[i] == 1)
            printf("%d,", u[i]);
    }
    printf("}");
}
int pos(int x) {
    int i, f = -1;
    for (i = 0; i < n; i++) {
        if (u[i] == x)
            f = i;
    }
    return f;
}
void setunion() {
    int i;
    printf("\nUnion : {");
```

```
    for (i = 0; i < n; i++) {
        if ((a[i] | b[i]) == 1)
            printf("%d,", u[i]);
    }
    printf("}\n");
}
void intersect() {
    int i;
    printf("\nIntersection : {");
    for (i = 0; i < n; i++) {
        if ((a[i] & b[i]) == 1)
            printf("%d,", u[i]);
    }
    printf("}\n");
}
void setdiff() {
    int i;
    printf("\nDifference : {");
    for (i = 0; i < n; i++) {
        if ((a[i] & (!b[i])) == 1)
            printf("%d,", u[i]);
    }
    printf("}\n");
}
void main() {
    int i, p, x;
    printf("Enter size of universal set : ");
    scanf("%d", &n);
    printf("Enter elements : ");
    for (i = 0; i < n; i++) {
        scanf("%d", &u[i]);
        a[i] = b[i] = 0;
```

```
    }
    printf("\nEnter size of set 1 : ");
    scanf("%d", &p);
    printf("\nEnter elements : ");
    for (i = 0; i < p; i++) {
        scanf("%d", &x);
        if (pos(x) != -1)
            a[pos(x)] = 1;
    }
    printf("\nEnter size of set 2 : ");
    scanf("%d", &p);
    printf("\nEnter elements : ");
    for (i = 0; i < p; i++) {
        scanf("%d", &x);
        if (pos(x) != -1)
            b[pos(x)] = 1;
    }
    printf("\nUniversal set : ");
    display(u);
    printf("\nSet 1 bit string : ");
    display(a);
    printf("\nSet 2 bit string : ");
    display(b);
    printf("\nSet 1 : ");
    bitdis(a);
    printf("\nSet 2 : ");
    bitdis(b);
    setunion();
    intersect();
    setdiff();
}
```

Output:

Enter size of universal set : 5

Enter elements : 2 6 3 8 4

Enter size of set 1 : 3

Enter elements : 2 7 4

Enter size of set 2 : 2

Enter elements : 8 2

Universal set : {2,6,3,8,4,}

Set 1 bit string : {1,0,0,0,1,}

Set 2 bit string : {1,0,0,1,0,}

Set 1 : {2,4,}

Set 2 : {2,8,}

Union : {2,8,4,}

Intersection : {2,}

Difference : {4,}

12. Aim: Disjoint Sets and the associated operations (create,union, find)

Program code:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   struct node *rep;
   struct node *next;
   int data;
} *heads[50], *tails[50];
static int countroot = 0;
void makeset(int x) {
   struct node *new = (struct node *)malloc(sizeof(struct node));
   if (new == NULL) {
      printf("Memory allocation failed!\n");
      return;
   }
   new->rep = new;
   new->next = NULL;
   new->data = x;
   heads[countroot] = new;
   tails[countroot] = new;
   countroot++;
   printf("Set created for element %d\n", x);
}
struct node* find(int a) {
   int i;
   struct node *tmp;

   for (i = 0; i < countroot; i++) {
      tmp = heads[i];
      while (tmp != NULL) {
         if (tmp->data == a)
```

```
            return tmp->rep;
        tmp = tmp->next;
    }
  }
  return NULL;
}
void unionsets(int a, int b) {
  int i, j, pos = -1, flag = 0;
  struct node *tail2 = NULL;
  struct node *rep1 = find(a);
  struct node *rep2 = find(b);
  if (rep1 == NULL || rep2 == NULL) {
    printf("\nOne or both elements are not present.\n");
    return;
  }
  if (rep1 == rep2) {
    printf("\nBoth elements are in the same set.\n");
    return;
  }
  for (j = 0; j < countroot; j++) {
    if (heads[j] == rep2) {
      pos = j;
      flag = 1;
      tail2 = tails[j];
      countroot--;
      for (i = pos; i < countroot; i++) {
        heads[i] = heads[i + 1];
        tails[i] = tails[i + 1];
      }
      break;
    }
  }
```

```
    for (j = 0; j < countroot; j++) {
        if (heads[j] == rep1) {
            tails[j]->next = rep2;
            tails[j] = tail2;
            break;
        }
    }
    while (rep2 != NULL) {
        rep2->rep = rep1;
        rep2 = rep2->next;
    }
    printf("\nUnion completed.\n");
}
int search(int x) {
    int i;
    struct node *tmp;

    for (i = 0; i < countroot; i++) {
        tmp = heads[i];
        while (tmp != NULL) {
            if (tmp->data == x)
                return 1;
            tmp = tmp->next;
        }
    }
    return 0;
}
void display() {
    int i;
    struct node *tmp;

    if (countroot == 0) {
```

```
        printf("\nNo sets available.\n");
        return;
    }
    printf("\nThe current sets are:\n");
    for (i = 0; i < countroot; i++) {
        tmp = heads[i];
        printf("Set %d: ", i + 1);
        while (tmp != NULL) {
            printf("%d -> ", tmp->data);
            tmp = tmp->next;
        }
        printf("NULL\n");
    }
}
int main() {
    int c, x, y;
    struct node *rep;
    while (1) {
        printf("\n\n--- Disjoint Set Operations ---\n");
        printf("1: Make Set\n");
        printf("2: Display Sets\n");
        printf("3: Union\n");
        printf("4: Find Set Representative\n");
        printf("5: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &c);
        switch (c) {
            case 1:
                printf("Enter element: ");
                scanf("%d", &x);
                if (search(x))
                    printf("Element already present.\n");
```

```
            else
                makeset(x);
            break;
        case 2:
            display();
            break;
        case 3:
            printf("Enter the two elements to union:\n");
            printf("First element: ");
            scanf("%d", &x);
            printf("Second element: ");
            scanf("%d", &y);
            unionsets(x, y);
            break;
        case 4:
            printf("Enter the element to find: ");
            scanf("%d", &x);
            rep = find(x);
            if (rep == NULL)
                printf("Element not present.\n");
            else
                printf("Element %d belongs to the set with representative %d\n", x, rep->data);
            break;
        case 5:
            printf("Exiting program...\n");
            exit(0);
        default:
            printf("Invalid choice. Try again.\n");
      }
    }
  return 0;
}
```

Output:

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 1

Enter element: 10

Set created for element 10

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 1

Enter element: 20

Set created for element 20

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 1

Enter element: 30

Set created for element 30

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 2

The current sets are:

Set 1: 10 -> NULL

Set 2: 20 -> NULL

Set 3: 30 -> NULL

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 3

Enter the two elements to union:

First element: 10

Second element: 20

Union completed.

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 2

The current sets are:

Set 1: 10 -> 20 -> NULL

Set 2: 30 -> NULL

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 3

Enter the two elements to union:

First element: 30

Second element: 20

Union completed.

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 2

The current sets are:

Set 1: 30 -> 10 -> 20 -> NULL

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 4

Enter the element to find: 30

Element 30 belongs to the set with representative 30

--- Disjoint Set Operations ---

1: Make Set

2: Display Sets

3: Union

4: Find Set Representative

5: Exit

Enter your choice: 5

13. Aim: Program for breadth first search.

Program code:

```
#include<stdio.h>
int visited[10] = {0,0,0,0,0,0,0,0,0,0};
int adj[10][10];
int queue[10];
int front = -1, rear = -1;
void insert(int item)
{
   if (rear == 11)
   {
      printf("full");
   }
   else
   {
      rear++;
      queue[rear] = item;
      if (front == -1)
         front++;
   }
}
int delete()
{
   int p;
   if (front == -1)
   {
      return 0;
   }
   else
   {
      p = queue[front];
      front++;
```

```
        return (p);
    }
}
void bfs(int s, int v)
{
    int p, i;
    insert(s);
    visited[s] = 1;
    p = delete();
    if (p != 0)
    {
        printf("%d ", p);
    }
    while (p != 0)
    {
        for (i = 1; i <= v; i++)
        {
            if ((adj[p][i] == 1) && (visited[i] == 0))
            {
                insert(i);
                visited[i] = 1;
            }
        }
        p = delete();
        if (p != 0)
        {
            printf("%d ", p);
        }
    }
}
int main()
{
```

```
    int i, j, v, s;
    printf("Enter the number of vertices: ");
    scanf("%d", &v);
    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= v; i++)
    {
        for (j = 1; j <= v; j++)
        {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Enter the starting vertex: ");
    scanf("%d", &s);
    bfs(s, v);
}
```

Output:

```
Enter the number of vertices: 5
Enter the adjacency matrix:
0 1 1 0 0
1 0 0 1 0
1 0 0 0 1
0 1 0 0 0
0 0 1 0 0
Enter the starting vertex: 2
2 1 4 3 5
```

14. Aim: Program for depth first search.

Program code:

```
#include<stdio.h>
int adj[10][10], stack[10], top = -1;
int visited[10] = {0,0,0,0,0,0,0,0,0,0};
void push(int item)
{
   if (top == 11)
      printf("Stack is full\n");
   else
   {
      top++;
      stack[top] = item;
   }
}
int pop()
{
   int p;
   if (top == -1)
      return 0;
   else
   {
      p = stack[top];
      top--;
      return p;
   }
}
void dfs(int s, int v)
{
   int p, i;
   push(s);
   visited[s] = 1;
```

```
    p = pop();
    if (p != 0)
        printf("%d ",  p);
    while (p != 0)
    {
        for (i = 1; i <= v; i++)
        {
            if (adj[p][i] == 1 && visited[i] == 0)
            {
                push(i);
                visited[i] = 1;
            }
        }
        p = pop();
        if (p != 0)
            printf("%d ",  p);
    }
}
int main()
{
    int i, j, v, s;
    printf("Enter the number of vertices: ");
    scanf("%d", &v);
    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= v; i++)
    {
        for (j = 1; j <= v; j++)
        {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Enter the starting vertex: ");
```

```
    scanf("%d", &s);
    dfs(s, v);
    return 0;
}
```

Output:

Enter the number of vertices: 5

Enter the adjacency matrix:

0 1 1 0 0

1 0 0 1 0

1 0 0 0 1

0 1 0 0 0

0 0 1 0 0

Enter the starting vertex: 2

2 4 1 3 5

15. Aim: Program to implement Prim's algorithm.

Program code:

```
#include <stdio.h>
#define INF 999
void main() {
    int n, i, j, adj[10][10], tot = 0, no_edge = 0;
    int visited[10] = {0};
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &adj[i][j]);
            if (adj[i][j] == 0)
                adj[i][j] = INF;
        }
    }
    visited[1] = 1;
    printf("\nEdges in the Minimum Spanning Tree are:\n");
    while (no_edge < n - 1) {
        int min = INF;
        int a = 0, b = 0;
        for (i = 1; i <= n; i++) {
            if (visited[i] == 1) {
                for (j = 1; j <= n; j++) {
                    if (adj[i][j] != INF && visited[j] == 0) {
                        if (adj[i][j] < min) {
                            min = adj[i][j];
                            a = i;
                            b = j;
                        }
                    }
```

```
            }
         }
      }
      printf("Edge %d -> %d cost = %d\n", a, b, min);
      visited[b] = 1;
      tot += min;
      no_edge++;
   }
   printf("Total Cost of Minimum Spanning Tree = %d\n", tot);
}
```

Output:

Enter the number of vertices: 4

Enter the adjacency matrix:

0 5 8 0

5 0 10 15

8 10 0 20

0 15 20 0

Edges in the Minimum Spanning Tree are:

Edge 1 -> 2 cost = 5

Edge 1 -> 3 cost = 8

Edge 2 -> 4 cost = 15

Total Cost of Minimum Spanning Tree = 28

16. Aim: Program to implement Kruskal's algorithm.

Program code:

```
#include <stdio.h>
#include <stdlib.h>
#define INF 999
int parent[20];
int find(int i) {
    if (parent[i] != i)
        parent[i] = find(parent[i]);
    return parent[i];
}
void union_set(int a, int b) {
    int parentA = find(a);
    int parentB = find(b);
    parent[parentB] = parentA;
}
int main() {
    int v, i, j;
    int adj[10][10];
    int edge_count = 0, mincost = 0;
    printf("Enter the number of vertices: ");
    scanf("%d", &v);
    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= v; i++) {
        for (j = 1; j <= v; j++) {
            scanf("%d", &adj[i][j]);
            if (adj[i][j] == 0)
                adj[i][j] = INF;
        }
    }
    for (i = 1; i <= v; i++)
        parent[i] = i;
```

```
    printf("\nEdges in the Minimum Spanning Tree are:\n");
    while (edge_count < v - 1) {
        int a = -1, b = -1, min = INF;
        for (i = 1; i <= v; i++) {
            for (j = 1; j <= v; j++) {
                if (adj[i][j] < min) {
                    min = adj[i][j];
                    a = i;
                    b = j;
                }
            }
        }
        if (a == -1 || b == -1)
            break;
        int u = find(a);
        int v_set = find(b);
        if (u != v_set) {
            printf("%d -> %d cost = %d\n", a, b, min);
            union_set(u, v_set);
            mincost += min;
            edge_count++;
        }
        adj[a][b] = adj[b][a] = INF;
    }
    printf("Total cost of Minimum Spanning Tree = %d\n", mincost);
    return 0;
}
```

Output:

Enter the number of vertices: 4

Enter the adjacency matrix:

0 5 8 0

5 0 10 15

8 10 0 20

0 15 20 0

Edges in the Minimum Spanning Tree are:

1 -> 2 cost = 5

1 -> 3 cost = 8

2 -> 4 cost = 15

Total cost of Minimum Spanning Tree = 28