Program 1

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow

```
 #include <stdio.h>
#define MAX 5

int stack[MAX];
int top = -1;

void push(int element) {
    if (top == MAX - 1) {
        printf("Stack Overflow! Unable to push %d\n", element);
    } else {
        stack[++top] = element;
        printf("Pushed: %d\n", element);
    }
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow! The stack is empty.\n");
    } else {
        printf("Popped: %d\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("The stack is empty.\n");
    } else {
        printf("Stack elements are: ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
```

```
int main() {
    int choice, element;

    while (1) {
        printf("\n--- Stack Operations ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to push: ");
                scanf("%d", &element);
                push(element);
                break;

            case 2:
                pop();
                break;

            case 3:
                display();
                break;

            case 4:
                printf("Exiting program.\n");
                return 0;

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}
```

```
--- Stack Operations ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 1
Pushed: 1

--- Stack Operations ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped: 1

--- Stack Operations ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
Exiting program.
```

Program 2

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int precedence(char c) {
    if (c == '^') return 3;
    else if (c == '*' || c == '/') return 2;
    else if (c == '+' || c == '-') return 1;
    else return -1;
}

char associativity(char c) {
    if (c == '^') return 'R';  // Right-to-left associativity
    return 'L';  // Left-to-right associativity
}

void infixToPostfix(const char *expr) {
    int len = strlen(expr);

        char *result = (char *)malloc(len + 1);
    char *stack = (char *)malloc(len);
    int resultIndex = 0;
    int stackIndex = -1;

    if (!result || !stack) {
        printf("Memory allocation failed\n");
        return;
    }

    for (int i = 0; i < len; i++) {
        char c = expr[i];

            if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
            result[resultIndex++] = c;
        }
             else if (c == '(') {
            stack[++stackIndex] = c;
        }

        else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
```

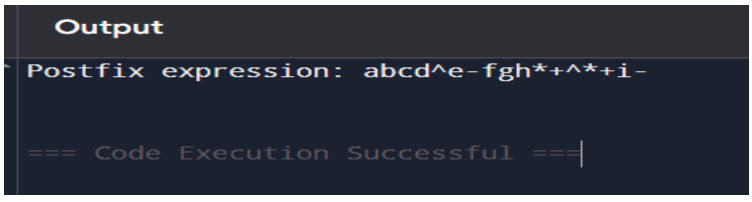
```
            result[resultIndex++] = stack[stackIndex--];
        }
        stackIndex--;  // Pop the '(' from the stack
    }
            else {
        while (stackIndex >= 0 && precedence(c) <= precedence(stack[stackIndex])) {
            if (precedence(c) == precedence(stack[stackIndex]) && associativity(c) ==
'R') break;
            result[resultIndex++] = stack[stackIndex--];
        }
        stack[++stackIndex] = c;
    }
  }


  while (stackIndex >= 0) {
    result[resultIndex++] = stack[stackIndex--];
  }

  result[resultIndex] = '\0';  // Null-terminate the result
  printf("Postfix expression: %s\n", result);


  free(result);
  free(stack);
}

int main() {
  char expr[] = "a+b*(c^d-e)^(f+g*h)-i";
  infixToPostfix(expr);
  return 0;
}
```



Program 3

a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>
#define SIZE 5

int queue[SIZE];
int front = -1, rear = -1;

void insert(int value) {
   if (rear == SIZE - 1) {
      printf("Queue Overflow\n");
      return;
   }
   if (front == -1) front = 0;
   queue[++rear] = value;
}

void delete() {
   if (front == -1 || front > rear) {
      printf("Queue Underflow\n");
      return;
   }
   front++;
}

void display() {
   if (front == -1 || front > rear) {
      printf("Queue is Empty\n");
      return;
   }
   for (int i = front; i <= rear; i++) {
      printf("%d ", queue[i]);
   }
   printf("\n");
}

int main() {
   insert(10);
   insert(20);
   insert(30);
   display();
   delete();
```

```
    display();
    delete();
    delete();
    delete();
    return 0;
}
```

```
Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 5
Inserted 5 into the queue.

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 5

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

Program 3

b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

void insert(int value) {
   if ((front == 0 && rear == MAX - 1) || (rear == (front - 1) % (MAX - 1))) {
      printf("Queue Overflow\n");
      return;
   }
   if (front == -1) {
      front = rear = 0;
   } else if (rear == MAX - 1 && front != 0) {
      rear = 0;
   } else {
      rear++;
   }
   queue[rear] = value;
}

void delete() {
   if (front == -1) {
      printf("Queue Underflow\n");
      return;
   }
   if (front == rear) {
      front = rear = -1;
   } else if (front == MAX - 1) {
      front = 0;
   } else {
      front++;
   }
}

void display() {
   if (front == -1) {
      printf("Queue is Empty\n");
      return;
   }
```

```
    if (rear >= front) {
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    } else {
        for (int i = front; i < MAX; i++) {
            printf("%d ", queue[i]);
        }
        for (int i = 0; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}

int main() {
    int choice, value;
    do {
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 4);
    return 0;
```

```
Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 8
Inserted 8 into the queue.

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 8

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

Program 4

WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* head = NULL;

void createList(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   if (head == NULL) {
      head = newNode;
   } else {
      struct Node* temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
}

void insertFirst(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
}

void insertAtPosition(int data, int position) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   if (position == 1) {
```

```
        newNode->next = head;
        head = newNode;
        return;
    }
    struct Node* temp = head;
    for (int i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position out of range\n");
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

void insertLast(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void displayList() {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
int main() {
    int choice, data, position;

    while (1) {
        printf("1. Create List\n");
        printf("2. Insert at First Position\n");
        printf("3. Insert at Any Position\n");
        printf("4. Insert at Last Position\n");
        printf("5. Display List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to create list: ");
                scanf("%d", &data);
                createList(data);
                break;
            case 2:
                printf("Enter data to insert at first position: ");
                scanf("%d", &data);
                insertFirst(data);
                break;
            case 3:
                printf("Enter data and position to insert: ");
                scanf("%d %d", &data, &position);
                insertAtPosition(data, position);
                break;
            case 4:
                printf("Enter data to insert at last position: ");
                scanf("%d", &data);
                insertLast(data);
                break;
            case 5:
                displayList();
                break;
            case 6:
                exit(0);
            default:
```

```
            printf("Invalid choice\n");
        }
    }
```

```
Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter the value: 87
Do you want to add another node? (1 for Yes, 0 for No): 0

Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter the value: 89
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 95
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 65
Do you want to add another node? (1 for Yes, 0 for No): 0
```

```
Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 5
The elements of the linked list are: 87 89 95 65

Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 5
The elements of the linked list are: 87 89 95 65
```

Program 5

WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* head = NULL;

void createList(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   if (head == NULL) {
      head = newNode;
   } else {
      struct Node* temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
}

void deleteFirst() {
   if (head != NULL) {
      struct Node* temp = head;
      head = head->next;
      free(temp);
   }
}

void deleteLast() {
   if (head != NULL) {
      if (head->next == NULL) {
         free(head);
         head = NULL;
      } else {
         struct Node* temp = head;
```

```
            while (temp->next != NULL && temp->next->next != NULL) {
                temp = temp->next;
            }
            free(temp->next);
            temp->next = NULL;
        }
    }
}

void deleteSpecified(int value) {
    if (head != NULL) {
        if (head->data == value) {
            struct Node* temp = head;
            head = head->next;
            free(temp);
        } else {
            struct Node* temp = head;
            while (temp->next != NULL && temp->next->data != value) {
                temp = temp->next;
            }
            if (temp->next != NULL) {
                struct Node* toDelete = temp->next;
                temp->next = temp->next->next;
                free(toDelete);
            }
        }
    }
}

void displayList() {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, data, value;
```

```
while (1) {
    printf("1. Create List\n");
    printf("2. Delete First Element\n");
    printf("3. Delete Last Element\n");
    printf("4. Delete Specified Element\n");
    printf("5. Display List\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter data to create list: ");
            scanf("%d", &data);
            createList(data);
            break;
        case 2:
            deleteFirst();
            printf("First element deleted.\n");
            break;
        case 3:
            deleteLast();
            printf("Last element deleted.\n");
            break;
        case 4:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteSpecified(value);
            printf("Specified element deleted.\n");
            break;
        case 5:
            displayList();
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
}
```

```
Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter the value: 98
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 89
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 54
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 98
Do you want to add another node? (1 for Yes, 0 for No): 0

Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 2
```

```
Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 4
Last element deleted.

Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 3
Enter the element to delete: 54
Element 54 deleted.
```

Program 6

a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;
struct Node* head2 = NULL;

void createList(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void sortList(struct Node* head) {
    struct Node *i, *j;
```

```
    int temp;
    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

void reverseList(struct Node** head) {
    struct Node *prev = NULL, *current = *head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenateLists(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) {
        head1 = head2;
        return;
    }
    struct Node* temp = head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
}

int main() {
    int choice, data;

    while (1) {
        printf("1. Create List 1\n");
```

```
printf("2. Create List 2\n");
printf("3. Display List 1\n");
printf("4. Display List 2\n");
printf("5. Sort List 1\n");
printf("6. Reverse List 1\n");
printf("7. Concatenate Lists\n");
printf("8. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter data to create List 1: ");
        scanf("%d", &data);
        createList(&head, data);
        break;
    case 2:
        printf("Enter data to create List 2: ");
        scanf("%d", &data);
        createList(&head2, data);
        break;
    case 3:
        printf("List 1: ");
        displayList(head);
        break;
    case 4:
        printf("List 2: ");
        displayList(head2);
        break;
    case 5:
        sortList(head);
        printf("List 1 sorted.\n");
        break;
    case 6:
        reverseList(&head);
        printf("List 1 reversed.\n");
        break;
    case 7:
        concatenateLists(head, head2);
        printf("Lists concatenated.\n");
```

```
            break;
        case 8:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
  }

  return 0;
}
```

```
Menu:
1. Create Linked List 1
2. Create Linked List 2
3. Display Linked List 1
4. Sort Linked List 1
5. Reverse Linked List 1
6. Concatenate Linked List 2 to Linked List 1
7. Display Linked List 2
8. Exit
Enter your choice: 1
Enter the value: 89
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 98
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 87
Do you want to add another node? (1 for Yes, 0 for No): 3
Enter the value: 85
Do you want to add another node? (1 for Yes, 0 for No): 0

Menu:
1. Create Linked List 1
2. Create Linked List 2
3. Display Linked List 1
4. Sort Linked List 1
5. Reverse Linked List 1
```

```
5. Reverse Linked List 1
6. Concatenate Linked List 2 to Linked List 1
7. Display Linked List 2
8. Exit
Enter your choice: 3
The elements of the linked list are: 89 98 87 85

Menu:
1. Create Linked List 1
2. Create Linked List 2
3. Display Linked List 1
4. Sort Linked List 1
5. Reverse Linked List 1
6. Concatenate Linked List 2 to Linked List 1
7. Display Linked List 2
8. Exit
Enter your choice: 6
List 2 concatenated to List 1.

Menu:
1. Create Linked List 1
2. Create Linked List 2
3. Display Linked List 1
4. Sort Linked List 1
5. Reverse Linked List 1
6. Concatenate Linked List 2 to Linked List 1
```

```
Menu:
1. Create Linked List 1
2. Create Linked List 2
3. Display Linked List 1
4. Sort Linked List 1
5. Reverse Linked List 1
6. Concatenate Linked List 2 to Linked List 1
7. Display Linked List 2
8. Exit
Enter your choice: 8
Exiting the program.
```

Program 6

b) WAP to Implement Single Link List to simulate Stack & Queue Operations

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* stackTop = NULL;
struct Node* queueFront = NULL;
struct Node* queueRear = NULL;

// Stack Operations
void push(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = stackTop;
    stackTop = newNode;
}

int pop() {
    if (stackTop == NULL) {
        printf("Stack is empty.\n");
        return -1;
    }
    struct Node* temp = stackTop;
    int data = temp->data;
    stackTop = stackTop->next;
    free(temp);
    return data;
}

void displayStack() {
    struct Node* temp = stackTop;
    if (temp == NULL) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack: ");
        while (temp != NULL) {
            printf("%d ", temp->data);
```

```
            temp = temp->next;
        }
        printf("\n");
    }
}

// Queue Operations
void enqueue(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (queueRear == NULL) {
        queueFront = queueRear = newNode;
        return;
    }
    queueRear->next = newNode;
    queueRear = newNode;
}

int dequeue() {
    if (queueFront == NULL) {
        printf("Queue is empty.\n");
        return -1;
    }
    struct Node* temp = queueFront;
    int data = temp->data;
    queueFront = queueFront->next;
    if (queueFront == NULL) {
        queueRear = NULL;
    }
    free(temp);
    return data;
}

void displayQueue() {
    struct Node* temp = queueFront;
    if (temp == NULL) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue: ");
```

```
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    int choice, data;

    while (1) {
        printf("1. Push to Stack\n");
        printf("2. Pop from Stack\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue to Queue\n");
        printf("5. Dequeue from Queue\n");
        printf("6. Display Queue\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push to stack: ");
                scanf("%d", &data);
                push(data);
                break;
            case 2:
                data = pop();
                if (data != -1) {
                    printf("Popped from stack: %d\n", data);
                }
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Enter data to enqueue to queue: ");
                scanf("%d", &data);
```

```
            enqueue(data);
            break;
        case 5:
            data = dequeue();
            if (data != -1) {
                printf("Dequeued from queue: %d\n", data);
            }
            break;
        case 6:
            displayQueue();
            break;
        case 7:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
  }

  return 0;
}
```

```
Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 3
Stack is empty.

Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 7
Exiting the program.
```

Program 7

WAP to Implement doubly link list with primitive operations a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) Display the contents of the list

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

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

struct Node* head = NULL;

void createList(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

void insertLeft(int newData, int existingData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    struct Node* temp = head;

    while (temp != NULL && temp->data != existingData) {
        temp = temp->next;
    }

    if (temp != NULL) {
        newNode->next = temp;
```

```
        newNode->prev = temp->prev;
        if (temp->prev != NULL) {
            temp->prev->next = newNode;
        } else {
            head = newNode;
        }
        temp->prev = newNode;
    } else {
        printf("Node with data %d not found.\n", existingData);
    }
}

void deleteNode(int value) {
    struct Node* temp = head;

    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

    if (temp != NULL) {
        if (temp->prev != NULL) {
            temp->prev->next = temp->next;
        } else {
            head = temp->next;
        }
        if (temp->next != NULL) {
            temp->next->prev = temp->prev;
        }
        free(temp);
        printf("Node with value %d deleted.\n", value);
    } else {
        printf("Node with value %d not found.\n", value);
    }
}

void displayList() {
    struct Node* temp = head;

    if (temp == NULL) {
        printf("List is empty.\n");
```

```
    } else {
        printf("Doubly Linked List: ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    int choice, data, existingData;

    while (1) {
        printf("1. Create List\n");
        printf("2. Insert Node to the Left\n");
        printf("3. Delete Node\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to create list: ");
                scanf("%d", &data);
                createList(data);
                break;
            case 2:
                printf("Enter new node data: ");
                scanf("%d", &data);
                printf("Enter the existing node data to insert left of: ");
                scanf("%d", &existingData);
                insertLeft(data, existingData);
                break;
            case 3:
                printf("Enter the node value to delete: ");
                scanf("%d", &data);
                deleteNode(data);
                break;
```

```
            case 4:
                displayList();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

```
Menu:
1. Create Doubly Linked List
2. Display Doubly Linked List
3. Insert a node to the left of a specific node
4. Delete a node by value
5. Exit
Enter your choice: 1
Enter the value: 87
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 98
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 56
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter the value: 48
Do you want to add another node? (1 for Yes, 0 for No): 0

Menu:
1. Create Doubly Linked List
2. Display Doubly Linked List
3. Insert a node to the left of a specific node
4. Delete a node by value
5. Exit
Enter your choice: 2
The elements of the doubly linked list are: 87 98 56 48
```

```
Menu:
1. Create Doubly Linked List
2. Display Doubly Linked List
3. Insert a node to the left of a specific node
4. Delete a node by value
5. Exit
Enter your choice: 3
Enter the value to insert: 58
Enter the target node value to insert left of: 2
Node with value 2 not found.

Menu:
1. Create Doubly Linked List
2. Display Doubly Linked List
3. Insert a node to the left of a specific node
4. Delete a node by value
5. Exit
Enter your choice: 5
Exiting the program.
```

Program 8

Write a program a) ToconstructabinarySearchtree. b) To traverse the tree using all the methods i.e., inorder, preorder and post order c) To display the elements in the tree

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

// Structure for a node in the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a node in the binary search tree
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else {
        root->right = insert(root->right, data);
    }
    return root;
}

// In-order traversal (Left, Root, Right)
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
```

```
// Pre-order traversal (Root, Left, Right)
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

// Post-order traversal (Left, Right, Root)
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

// Function to display the elements in the tree using in-order traversal
void display(struct Node* root) {
    printf("In-order traversal: ");
    inorder(root);
    printf("\n");
}

int main() {
    struct Node* root = NULL;
    int choice, data;

    while (1) {
        printf("1. Insert Node\n");
        printf("2. In-order Traversal\n");
        printf("3. Pre-order Traversal\n");
        printf("4. Post-order Traversal\n");
        printf("5. Display In-order Traversal\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order Traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
                printf("Post-order Traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                display(root);
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

```
Menu:
1. Insert a node into the Binary Search Tree
2. Display tree elements (Inorder Traversal)
3. Traverse tree (Inorder Traversal)
4. Traverse tree (Preorder Traversal)
5. Traverse tree (Postorder Traversal)
6. Exit
Enter your choice: 1
Enter value to insert into the tree: 98
98 inserted into the tree.

Menu:
1. Insert a node into the Binary Search Tree
2. Display tree elements (Inorder Traversal)
3. Traverse tree (Inorder Traversal)
4. Traverse tree (Preorder Traversal)
5. Traverse tree (Postorder Traversal)
6. Exit
Enter your choice: 2
The elements in the tree (Inorder Traversal): 98

Menu:
1. Insert a node into the Binary Search Tree
2. Display tree elements (Inorder Traversal)
3. Traverse tree (Inorder Traversal)
```

```
4. Traverse tree (Preorder Traversal)
5. Traverse tree (Postorder Traversal)
6. Exit
Enter your choice: 3
Inorder Traversal: 98

Menu:
1. Insert a node into the Binary Search Tree
2. Display tree elements (Inorder Traversal)
3. Traverse tree (Inorder Traversal)
4. Traverse tree (Preorder Traversal)
5. Traverse tree (Postorder Traversal)
6. Exit
Enter your choice: 5
Postorder Traversal: 98

Menu:
1. Insert a node into the Binary Search Tree
2. Display tree elements (Inorder Traversal)
3. Traverse tree (Inorder Traversal)
4. Traverse tree (Preorder Traversal)
5. Traverse tree (Postorder Traversal)
6. Exit
```

Program 9

a) Write a program to traverse a graph using BFS method.

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

#define MAX_VERTICES 100

struct Queue {
   int items[MAX_VERTICES];
   int front, rear;
};

struct Graph {
   int adj[MAX_VERTICES][MAX_VERTICES];
   int vertices;
};

void initQueue(struct Queue* q) {
   q->front = -1;
   q->rear = -1;
}

int isQueueEmpty(struct Queue* q) {
   return q->front == -1;
}

void enqueue(struct Queue* q, int value) {
   if (q->rear == MAX_VERTICES - 1) {
      printf("Queue Overflow\n");
      return;
   }
   if (q->front == -1)
      q->front = 0;
   q->rear++;
   q->items[q->rear] = value;
}

int dequeue(struct Queue* q) {
   if (isQueueEmpty(q)) {
      printf("Queue Underflow\n");
```

```
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}

void initGraph(struct Graph* g, int vertices) {
    g->vertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}

void addEdge(struct Graph* g, int u, int v) {
    g->adj[u][v] = 1;
    g->adj[v][u] = 1;
}

void bfs(struct Graph* g, int startVertex) {
    int visited[MAX_VERTICES] = {0};
    struct Queue q;
    initQueue(&q);

    visited[startVertex] = 1;
    enqueue(&q, startVertex);

    printf("BFS Traversal starting from vertex %d: ", startVertex);

    while (!isQueueEmpty(&q)) {
        int currentVertex = dequeue(&q);
        printf("%d ", currentVertex);

        for (int i = 0; i < g->vertices; i++) {
            if (g->adj[currentVertex][i] == 1 && !visited[i]) {
```

```
            enqueue(&q, i);
            visited[i] = 1;
        }
    }
  }
  printf("\n");
}

int main() {
  struct Graph g;
  int vertices, edges, u, v, startVertex;

  printf("Enter number of vertices: ");
  scanf("%d", &vertices);
  initGraph(&g, vertices);

  printf("Enter number of edges: ");
  scanf("%d", &edges);

  for (int i = 0; i < edges; i++) {
    printf("Enter edge (u v): ");
    scanf("%d %d", &u, &v);
    addEdge(&g, u, v);
  }

  printf("Enter the starting vertex for BFS: ");
  scanf("%d", &startVertex);

  bfs(&g, startVertex);

  return 0;
}
```

```
Enter the number of vertices: 2
Enter the number of edges: 2
Enter edge (u, v) where u and v are the vertex numbers: 2
5
Enter edge (u, v) where u and v are the vertex numbers: 5
2
Enter the starting vertex for BFS: 5
BFS Traversal starting from vertex 5: 5
```

Program 9

b) Write a program to check whether given graph is connected or not using DFS method.

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

#define MAX_VERTICES 100

struct Graph {
    int adj[MAX_VERTICES][MAX_VERTICES];
    int vertices;
};

void initGraph(struct Graph* g, int vertices) {
    g->vertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}

void addEdge(struct Graph* g, int u, int v) {
    g->adj[u][v] = 1;
    g->adj[v][u] = 1;
}

void dfs(struct Graph* g, int vertex, int visited[]) {
    visited[vertex] = 1;
    for (int i = 0; i < g->vertices; i++) {
        if (g->adj[vertex][i] == 1 && !visited[i]) {
            dfs(g, i, visited);
        }
    }
}

int isConnected(struct Graph* g) {
    int visited[MAX_VERTICES] = {0};
    dfs(g, 0, visited);
    for (int i = 0; i < g->vertices; i++) {
        if (!visited[i]) {
            return 0;
        }
```

```
    }
    return 1;
}

int main() {
    struct Graph g;
    int vertices, edges, u, v;

    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    initGraph(&g, vertices);

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    for (int i = 0; i < edges; i++) {
        printf("Enter edge (u v): ");
        scanf("%d %d", &u, &v);
        addEdge(&g, u, v);
    }

    if (isConnected(&g)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

```
Enter the number of vertices: 2
Enter the number of edges: 2
Enter edge (u, v) where u and v are the vertex numbers: 2
2
Enter edge (u, v) where u and v are the vertex numbers: 2
4
0 The graph is not connected.
```