Data Structure Lab

Manav Rachna International Institute of Research and Studies

School of Computer Applications

Submitted By	
Student Name	SAKSHI
Roll No	25/SCA/MCAN/066
Programme	Masters of Computer Applications
Semester	1th Semester
Section	B
Department	School of Computer Applications
Batch	2025-27
Submitted To	
Faculty Name	Dr. Sachin Sharma

Manav Rachna Campus Rd, Gadakhor Basti Village, Sector 43, Faridabad, Haryana 121004

1. Write a program to perform various operations in the list: - Insertion - Deletion – Display

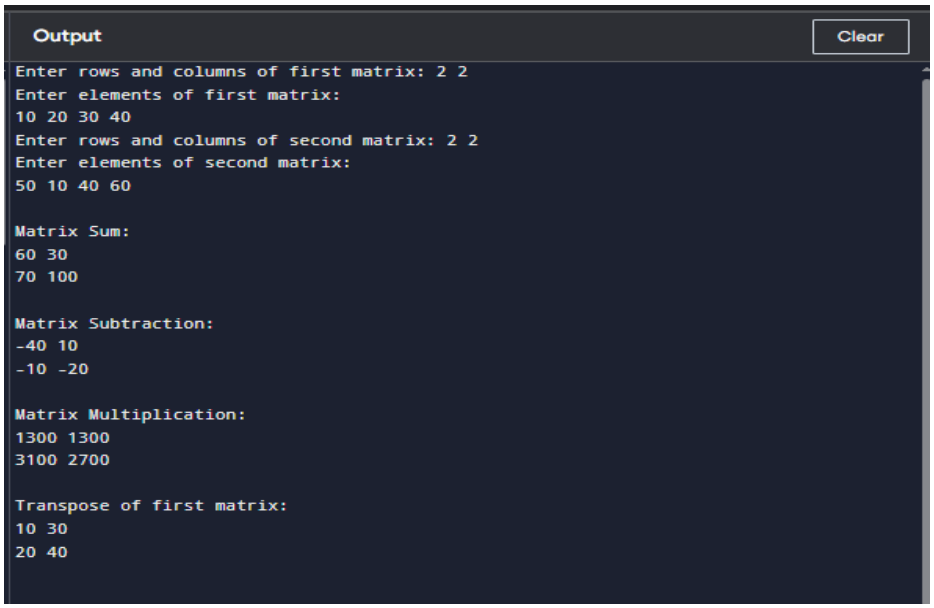
```
#include <stdio.h>

int main() {
   int arr[100], n = 0, choice, pos, value, i;

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

      switch (choice) {

         case 1: // Insertion
            printf("Enter the position (0 to %d): ", n);
            scanf("%d", &pos);
            if (pos < 0 || pos > n) {
               printf("Invalid position!\n");
            } else {
               printf("Enter the value: ");
               scanf("%d", &value);

               // Shift elements right
               for (i = n - 1; i >= pos; i--) {
                  arr[i + 1] = arr[i];
               }

               arr[pos] = value;
               n++;
               printf("Element inserted successfully!\n");
            }
            break;

         case 2: // Deletion
            if (n == 0) {
               printf("List is empty! Nothing to delete.\n");
            } else {
               printf("Enter position to delete (0 to %d): ", n - 1);
               scanf("%d", &pos);

               if (pos < 0 || pos >= n) {
                  printf("Invalid position!\n");
               } else {
```

```
                // Shift elements left
                for (i = pos; i < n - 1; i++) {
                    arr[i] = arr[i + 1];
                }

                n--;
                printf("Element deleted successfully!\n");
            }
        }
        break;

    case 3: // Display
        if (n == 0) {
            printf("List is empty.\n");
        } else {
            printf("List elements: ");
            for (i = 0; i < n; i++) {
                printf("%d ", arr[i]);
            }
            printf("\n");
        }
        break;
    case 4:
        printf("Exiting program...\n");
        return 0;
    default:
        printf("Invalid choice! Please enter again.\n");
    }
  }

  return 0;
}
```

2. **Write a program to find the arithmetic operations on matrices: - Sum and Subtraction - Product of 2 matrices - Transpose of a matrix**

```
#include <stdio.h>

int main() {
  int a[10][10], b[10][10], sum[10][10], sub[10][10], mul[10][10], trans[10][10];
  int r1, c1, r2, c2, i, j, k;

  printf("Enter rows and columns of first matrix: ");
  scanf("%d%d", &r1, &c1);

  printf("Enter elements of first matrix:\n");
  for(i = 0; i < r1; i++)
    for(j = 0; j < c1; j++)
      scanf("%d", &a[i][j]);

  printf("Enter rows and columns of second matrix: ");
  scanf("%d%d", &r2, &c2);

  printf("Enter elements of second matrix:\n");
  for(i = 0; i < r2; i++)
    for(j = 0; j < c2; j++)
      scanf("%d", &b[i][j]);


  if(r1 == r2 && c1 == c2) {
    for(i = 0; i < r1; i++)
      for(j = 0; j < c1; j++) {
        sum[i][j] = a[i][j] + b[i][j];
        sub[i][j] = a[i][j] - b[i][j];
      }

    printf("\nMatrix Sum:\n");
    for(i = 0; i < r1; i++) {
      for(j = 0; j < c1; j++)
        printf("%d ", sum[i][j]);
      printf("\n");
    }

    printf("\nMatrix Subtraction:\n");
    for(i = 0; i < r1; i++) {
      for(j = 0; j < c1; j++)
        printf("%d ", sub[i][j]);
      printf("\n");
    }
  } else {
    printf("\nSum and Subtraction not possible\n");
```

```
    }


    if(c1 == r2) {
       for(i = 0; i < r1; i++)
          for(j = 0; j < c2; j++) {
             mul[i][j] = 0;
             for(k = 0; k < c1; k++)
                mul[i][j] += a[i][k] * b[k][j];
          }

       printf("\nMatrix Multiplication:\n");
       for(i = 0; i < r1; i++) {
          for(j = 0; j < c2; j++)
             printf("%d ", mul[i][j]);
          printf("\n");
       }
    } else {
       printf("\nMultiplication not possible\n");
    }

    printf("\nTranspose of first matrix:\n");
    for(i = 0; i < c1; i++) {
       for(j = 0; j < r1; j++)
          printf("%d ", a[j][i]);
       printf("\n");
    }

    return 0;
}
```



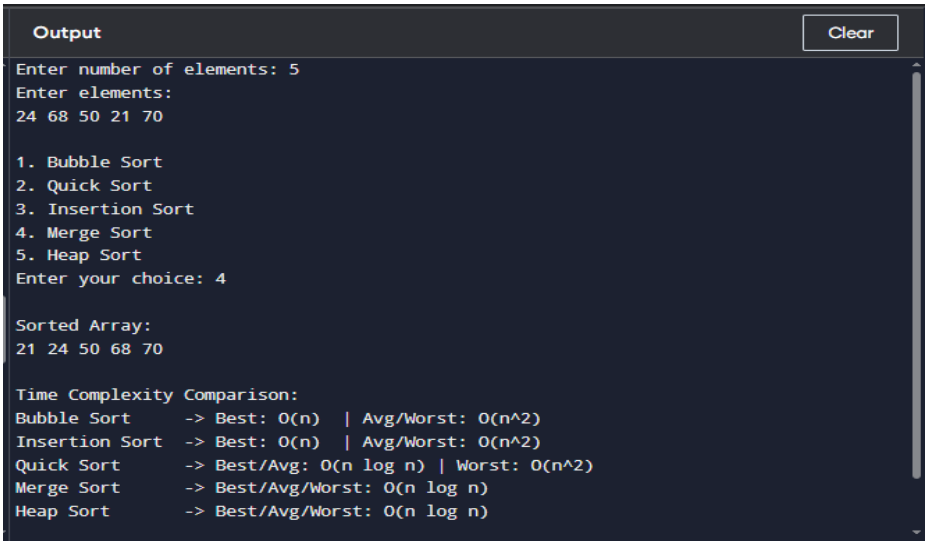
3. **Write a Program to sort the list using: - Bubble Sort - Quick Sort - Insertion sort - Merge Sort - Heap Sort Also, find the comparison on the basis of time complexity**

```
#include <stdio.h>

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

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

int partition(int arr[], int low, int high) {
   int pivot = arr[high], i = low - 1, temp;
   for (int j = low; j < high; j++) {
      if (arr[j] < pivot) {
         i++;
         temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
      }
   }

   temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp;
   return i + 1;

}

void quickSort(int arr[], int low, int high) {

   if (low < high) {
      int pi = partition(arr, low, high);
      quickSort(arr, low, pi - 1);
```

```
      quickSort(arr, pi + 1, high);
   }
}

void merge(int arr[], int l, int m, int r) {

   int i, j, k;
   int n1 = m - l + 1, n2 = r - m;

   int L[n1], R[n2];

   for (i = 0; i < n1; i++) L[i] = arr[l + i];
   for (j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

   i = 0; j = 0; k = l;

   while (i < n1 && j < n2)
      arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

   while (i < n1) arr[k++] = L[i++];
   while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {

   if (l < r) {
      int m = (l + r) / 2;
      mergeSort(arr, l, m);
      mergeSort(arr, m + 1, r);
      merge(arr, l, m, r);
   }
}

void heapify(int arr[], int n, int i) {

   int largest = i, left = 2 * i + 1, right = 2 * i + 2, temp;

   if (left < n && arr[left] > arr[largest]) largest = left;
   if (right < n && arr[right] > arr[largest]) largest = right;

   if (largest != i) {
      temp = arr[i]; arr[i] = arr[largest]; arr[largest] = temp;
      heapify(arr, n, largest);
   }
}

void heapSort(int arr[], int n) {
   int temp;
```

```
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {
        temp = arr[0]; arr[0] = arr[i]; arr[i] = temp;
        heapify(arr, i, 0);
    }
}

void display(int arr[], int n) {

    for(int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[50], n, choice;

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

    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("\n1. Bubble Sort\n2. Quick Sort\n3. Insertion Sort\n4. Merge Sort\n5. Heap
Sort\n");

    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch(choice) {
        case 1: bubbleSort(arr, n); break;
        case 2: quickSort(arr, 0, n - 1); break;
        case 3: insertionSort(arr, n); break;
        case 4: mergeSort(arr, 0, n - 1); break;
        case 5: heapSort(arr, n); break;
        default: printf("Invalid Choice"); return 0;
    }

    printf("\nSorted Array:\n");
    display(arr, n);

    printf("\nTime Complexity Comparison:\n");
    printf("Bubble Sort     -> Best: O(n)  | Avg/Worst: O(n^2)\n");
    printf("Insertion Sort  -> Best: O(n)  | Avg/Worst: O(n^2)\n");
    printf("Quick Sort      -> Best/Avg: O(n log n) | Worst: O(n^2)\n");
    printf("Merge Sort      -> Best/Avg/Worst: O(n log n)\n");
```

```
    printf("Heap Sort      -> Best/Avg/Worst: O(n log n)\n");

    return 0;
}
```



4. **Write a program to search the element using: - Linear Search - Binary Search Also, find the comparison on the basis of time complexity**

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
    for(int i = 0; i < n; i++)
        if(arr[i] == key)
            return i;
    return -1;
}

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1, mid;
    while(low <= high) {
        mid = (low + high) / 2;
        if(arr[mid] == key)
            return mid;
        else if(arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

int main() {
```

```
    int arr[50], n, key, choice, result;

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

    printf("Enter elements (sorted for binary search):\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("\nEnter element to search: ");
    scanf("%d", &key);

    printf("\n1. Linear Search\n2. Binary Search\nEnter choice: ");
    scanf("%d", &choice);

    if(choice == 1)
        result = linearSearch(arr, n, key);
    else if(choice == 2)
        result = binarySearch(arr, n, key);
    else {
        printf("Invalid Choice\n");
        return 0;
    }
    if(result != -1)
        printf("\nElement found at index %d\n", result);
    else
        printf("\nElement not found\n");

    printf("\nTime Complexity Comparison:\n");
    printf("Linear Search  -> Best: O(1), Average: O(n), Worst: O(n)\n");
    printf("Binary Search  -> Best: O(1), Average/Worst: O(log n) (Only works on sorted
data)\n");

    return 0;
}
```

Output Clear

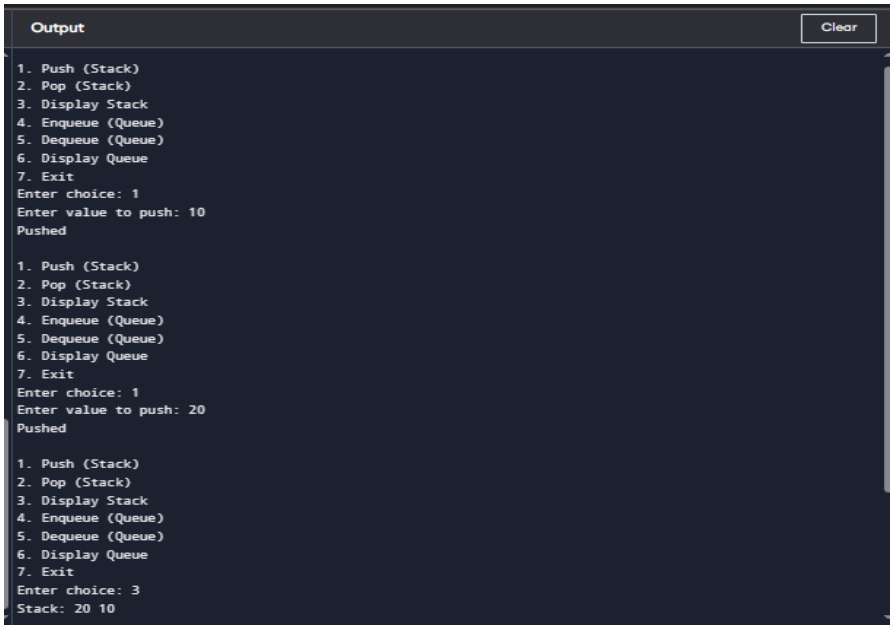
```
Enter number of elements: 5
Enter elements (sorted for binary search):
10 20 30 40 50

Enter element to search: 40

1. Linear Search
2. Binary Search
Enter choice: 2

Element found at index 3

Time Complexity Comparison:
Linear Search  -> Best: O(1), Average: O(n), Worst: O(n)
Binary Search  -> Best: O(1), Average/Worst: O(log n) (Only works on sorted
    data)
```

5. **Write a program to perform all operations: - For the Stack using an Array. - For the Queue using an Array**.

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

#define SIZE 5

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

void push() {

   int value;
   if(top == SIZE - 1)
      printf("Stack Overflow\n");
   else {
      printf("Enter value to push: ");
      scanf("%d", &value);
      stack[++top] = value;
      printf("Pushed\n");
   }
}

void pop() {

   if(top == -1)
      printf("Stack Underflow\n");
   else
      printf("Popped: %d\n", stack[top--]);
}

void displayStack() {

   if(top == -1)
      printf("Stack Empty\n");
   else {
      printf("Stack: ");
      for(int i = top; i >= 0; i--)
         printf("%d ", stack[i]);
      printf("\n");
   }
}

void enqueue() {

   int value;
   if(rear == SIZE - 1)
```

```
        printf("Queue Overflow\n");
    else {
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        if(front == -1) front = 0;
        queue[++rear] = value;
        printf("Enqueued\n");
    }
}

void dequeue() {

    if(front == -1 || front > rear)
        printf("Queue Underflow\n");
    else
        printf("Dequeued: %d\n", queue[front++]);
}

void displayQueue() {

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

int main() {

    int choice;

    while(1) {

        printf("\n1. Push (Stack)\n2. Pop (Stack)\n3. Display Stack\n");
        printf("4. Enqueue (Queue)\n5. Dequeue (Queue)\n6. Display Queue\n7. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch(choice) {

            case 1: push(); break;
            case 2: pop(); break;
            case 3: displayStack(); break;
            case 4: enqueue(); break;
            case 5: dequeue(); break;
```

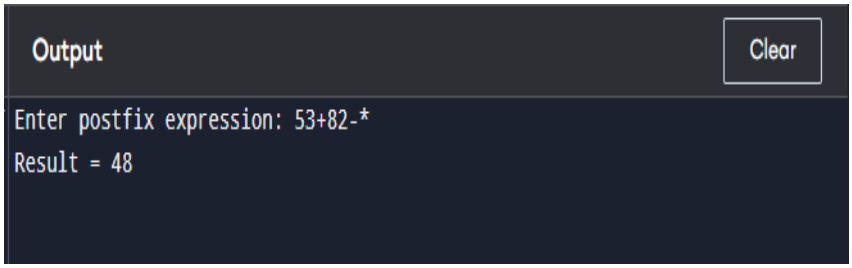
```
            case 6: displayQueue(); break;
            case 7: exit(0);
            default: printf("Invalid Choice\n");
        }
    }
    return 0;
}
```



6. Write a program to evaluate the Postfix Notation

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

#define SIZE 50

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

void push(int value) {
    stack[++top] = value;
}

int pop() {
    return stack[top--];
}

int main() {
    char postfix[50];
    int i, a, b, result;

    printf("Enter postfix expression: ");
    scanf("%s", postfix);
```

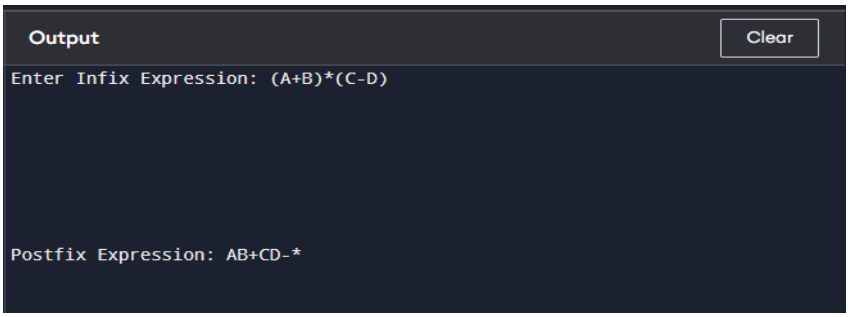
```
    for(i = 0; postfix[i] != '\0'; i++) {
        if(isdigit(postfix[i])) {
            push(postfix[i] - '0');
        } else {
            b = pop();
            a = pop();

            switch(postfix[i]) {
                case '+': result = a + b; break;
                case '-': result = a - b; break;
                case '*': result = a * b; break;
                case '/': result = a / b; break;
            }
            push(result);
        }
    }

    printf("Result = %d\n", pop());

    return 0;
}
```



7. Write a program to convert infix notation into Postfix Notation.

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

#define SIZE 50

char stack[SIZE];
int top = -1;

void push(char x) {
    stack[++top] = x;
}

char pop() {
    return stack[top--];
```

```
}
int precedence(char x) {
   if(x == '(')
      return 0;
   if(x == '+' || x == '-')
      return 1;
   if(x == '*' || x == '/' || x == '%')
      return 2;
   return 0;
}
int main() {
   char infix[50], postfix[50], ch;
   int i = 0, j = 0;
   printf("Enter Infix Expression: ");
   scanf("%s", infix);
   while((ch = infix[i++]) != '\0') {
      if(isalnum(ch)) {
         postfix[j++] = ch;
      }
      else if(ch == '(') {
         push(ch);
      }
      else if(ch == ')') {
         while(stack[top] != '(')
            postfix[j++] = pop();
         pop();
      }
      else {
         while(top != -1 && precedence(stack[top]) >= precedence(ch))
            postfix[j++] = pop();
         push(ch);
      }
   }
   while(top != -1)
      postfix[j++] = pop();
   postfix[j] = '\0';
   printf("Postfix Expression: %s\n", postfix);

   return 0;
}
```



8. Write a program to perform the operations of the linked list: - Insertion - Deletion – Display

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

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

struct Node *head = NULL;

void insert() {

   int value;
   struct Node *newnode, *temp;
   newnode = (struct Node*)malloc(sizeof(struct Node));
   printf("Enter value to insert: ");
   scanf("%d", &value);
   newnode->data = value;
   newnode->next = NULL;

   if(head == NULL) {

      head = newnode;
   }
else {
      temp = head;
      while(temp->next != NULL)
         temp = temp->next;
      temp->next = newnode;
   }
}

void delete() {

   int value;
   struct Node *temp = head, *prev = NULL;
   if(head == NULL) {
      printf("List is empty\n");
      return;
   }

   printf("Enter value to delete: ");
   scanf("%d", &value);

   if(temp != NULL && temp->data == value) {
```

```
        head = temp->next;
        free(temp);
        return;
    }

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

        prev = temp;
        temp = temp->next;
    }

    if(temp == NULL) {
        printf("Value not found\n");
        return;
    }

    prev->next = temp->next;
    free(temp);
}

void display() {

    struct Node *temp = head;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while(temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {

    int choice;
    while(1) {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: insert(); break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("Invalid choice\n");
```

```
    }
  }
  return 0;
}
```

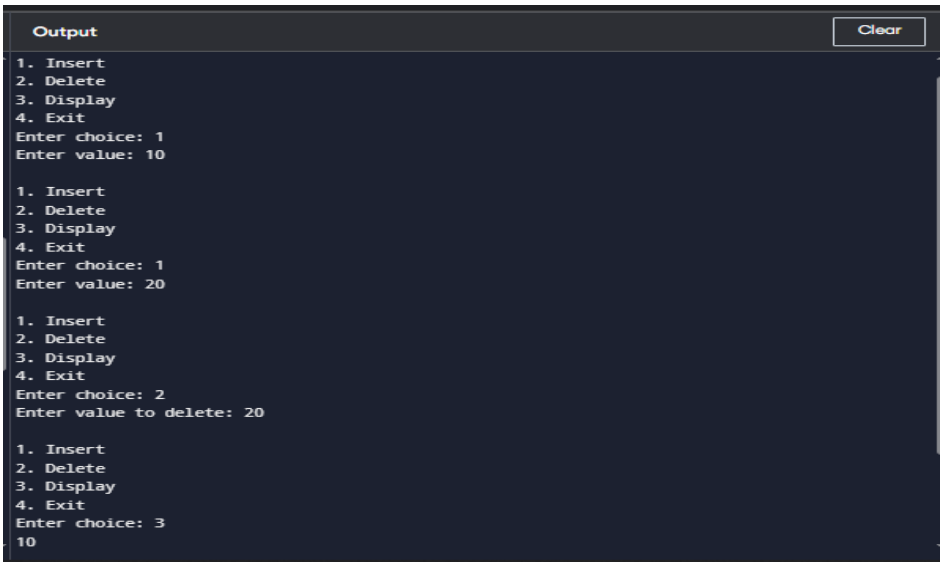
```
Output                                                    Clear

---- Menu ----
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 10
Node inserted.

---- Menu ----
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 20
Node inserted.

---- Menu ----
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Linked List: 10 -> 20 -> NULL
```

9. **Write a program to perform the operations of the Circular linked list: - Insertion - Deletion – Display**

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

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

struct Node* head = NULL;

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

```
        newNode->next = head;
    }
}

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

void display() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("\n");
}

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

```
    while (1) {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                delete(value);
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```



10. Write a program to perform all operations: - For Stack using Linked List. - For Queue using Linked List - For Circular Queue using Linked List

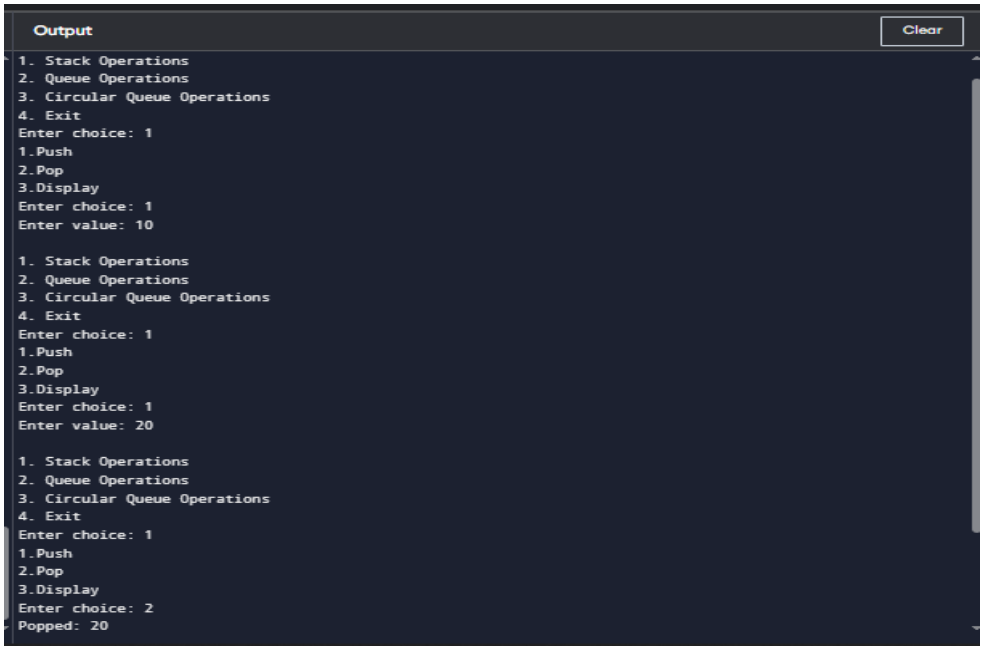
```
#include <stdio.h>
#include <stdlib.h>

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

// ------------------- STACK USING LINKED LIST -------------------
struct Node* top = NULL;

void push(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = top;
   top = newNode;
}

void pop() {
   if (top == NULL) {
      printf("Stack is empty\n");
      return;
   }
   struct Node* temp = top;
   printf("Popped: %d\n", top->data);
   top = top->next;
   free(temp);
}

void displayStack() {
   if (top == NULL) {
      printf("Stack is empty\n");
      return;
   }
   struct Node* temp = top;
   printf("Stack elements: ");
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}

// ------------------- QUEUE USING LINKED LIST -------------------
struct Node* front = NULL;
```

```
struct Node* rear = NULL;

void enqueue(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;
   if (rear == NULL) {
      front = rear = newNode;
      return;
   }
   rear->next = newNode;
   rear = newNode;
}

void dequeue() {
   if (front == NULL) {
      printf("Queue is empty\n");
      return;
   }
   struct Node* temp = front;
   printf("Dequeued: %d\n", front->data);
   front = front->next;
   if (front == NULL) rear = NULL;
   free(temp);
}

void displayQueue() {
   if (front == NULL) {
      printf("Queue is empty\n");
      return;
   }
   struct Node* temp = front;
   printf("Queue elements: ");
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}

// ------------------- CIRCULAR QUEUE USING LINKED LIST -------------------
struct Node* cfront = NULL;
struct Node* crear = NULL;

void cEnqueue(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if (cfront == NULL) {
```

```
        cfront = crear = newNode;
        newNode->next = cfront;
    } else {
        crear->next = newNode;
        crear = newNode;
        crear->next = cfront;
    }
}

void cDequeue() {
    if (cfront == NULL) {
        printf("Circular Queue is empty\n");
        return;
    }
    struct Node* temp = cfront;
    printf("Dequeued: %d\n", cfront->data);
    if (cfront == crear) {
        cfront = crear = NULL;
    } else {
        cfront = cfront->next;
        crear->next = cfront;
    }
    free(temp);
}

void displayCQueue() {
    if (cfront == NULL) {
        printf("Circular Queue is empty\n");
        return;
    }
    struct Node* temp = cfront;
    printf("Circular Queue elements: ");
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != cfront);
    printf("\n");
}

// ------------------- MAIN MENU -------------------
int main() {
    int choice, subChoice, value;

    while (1) {
        printf("\n1. Stack Operations\n2. Queue Operations\n3. Circular Queue Operations\n4.
Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
```

```
case 1:
   printf("1.Push\n2.Pop\n3.Display\nEnter choice: ");
   scanf("%d", &subChoice);

   if (subChoice == 1) {
      printf("Enter value: ");
      scanf("%d", &value);
      push(value);
   } else if (subChoice == 2) {
      pop();
   } else if (subChoice == 3) {
      displayStack();
   }
   break;

case 2:
   printf("1.Enqueue\n2.Dequeue\n3.Display\nEnter choice: ");
   scanf("%d", &subChoice);

   if (subChoice == 1) {
      printf("Enter value: ");
      scanf("%d", &value);
      enqueue(value);
   } else if (subChoice == 2) {
      dequeue();
   } else if (subChoice == 3) {
      displayQueue();
   }
   break;

case 3:
   printf("1.Enqueue\n2.Dequeue\n3.Display\nEnter choice: ");
   scanf("%d", &subChoice);

   if (subChoice == 1) {
      printf("Enter value: ");
      scanf("%d", &value);
      cEnqueue(value);
   } else if (subChoice == 2) {
      cDequeue();
   } else if (subChoice == 3) {
      displayCQueue();
   }
   break;

case 4:
   exit(0);
default:
```

```
            printf("Invalid choice\n");
        }
    }
    return 0;
}
```

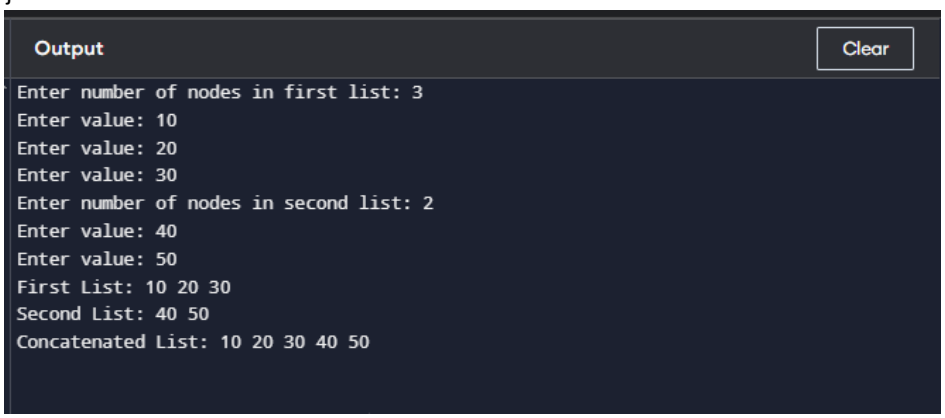


11. Write a program to perform concatenation of two linked lists.

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

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

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

// Function to display a linked list
void display(struct Node* head) {
    if (!head) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
```

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

// Function to concatenate list2 at the end of list1
struct Node* concatenate(struct Node* list1, struct Node* list2) {
    if (!list1) return list2;
    struct Node* temp = list1;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = list2;
    return list1;
}

int main() {
    struct Node *head1 = NULL, *head2 = NULL, *temp;
    int n1, n2, value, i;

    printf("Enter number of nodes in first list: ");
    scanf("%d", &n1);
    for (i = 0; i < n1; i++) {
        printf("Enter value: ");
        scanf("%d", &value);
        struct Node* newNode = createNode(value);
        if (!head1) {
            head1 = newNode;
            temp = head1;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }

    printf("Enter number of nodes in second list: ");
    scanf("%d", &n2);
    for (i = 0; i < n2; i++) {
        printf("Enter value: ");
        scanf("%d", &value);
        struct Node* newNode = createNode(value);
        if (!head2) {
            head2 = newNode;
            temp = head2;
        } else {
            temp->next = newNode;
```

```
            temp = newNode;
        }
    }

    printf("First List: ");
    display(head1);
    printf("Second List: ");
    display(head2);

    head1 = concatenate(head1, head2);
    printf("Concatenated List: ");
    display(head1);

    return 0;
}
```



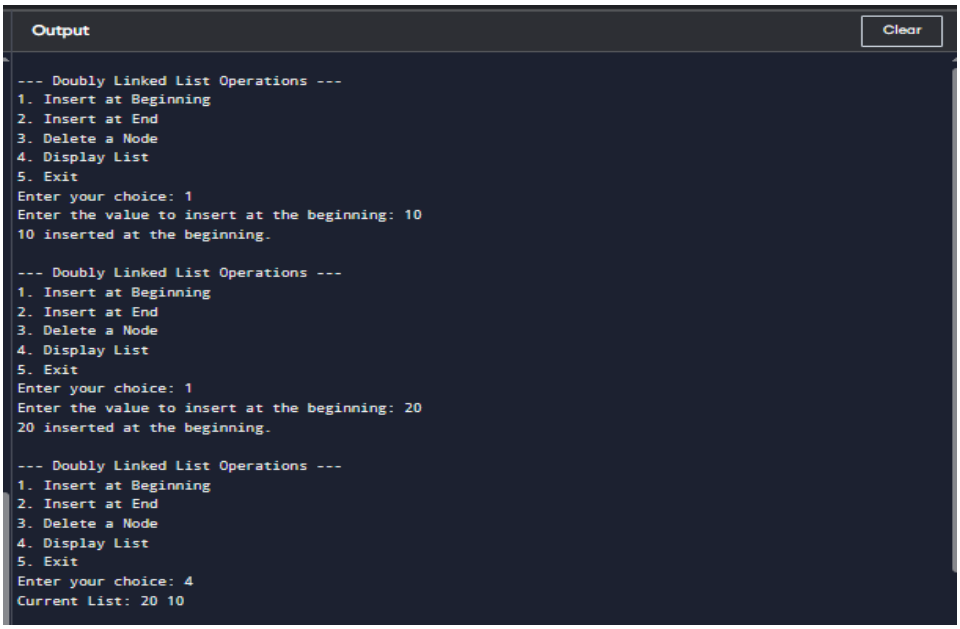
12. Write a program to perform the operations of the Double linked list: - Insertion - Deletion – Display

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

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

struct Node* head = NULL;

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
```

```
void insertAtBeginning(int value) {
   struct Node* newNode = createNode(value);
   if (!head) {
      head = newNode;
   } else {
      newNode->next = head;
      head->prev = newNode;
      head = newNode;
   }
   printf("%d inserted at the beginning.\n", value);
}

void insertAtEnd(int value) {
   struct Node* newNode = createNode(value);
   if (!head) {
      head = newNode;
      printf("%d inserted as the first node.\n", value);
      return;
   }
   struct Node* temp = head;
   while (temp->next) temp = temp->next;
   temp->next = newNode;
   newNode->prev = temp;
   printf("%d inserted at the end.\n", value);
}

void deleteNode(int value) {
   if (!head) {
      printf("List is empty. Cannot delete.\n");
      return;
   }
   struct Node* temp = head;
   while (temp && temp->data != value) temp = temp->next;
   if (!temp) {
      printf("Value %d not found in the list.\n", value);
      return;
   }
   if (temp->prev) temp->prev->next = temp->next;
   else head = temp->next;
   if (temp->next) temp->next->prev = temp->prev;
   free(temp);
   printf("Value %d deleted from the list.\n", value);
}

void display() {
   if (!head) {
      printf("List is empty.\n");
```

```
        return;
    }
    printf("Current List: ");
    struct Node* temp = head;
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, value;
    while (1) {
        printf("\n--- Doubly Linked List Operations ---\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Delete a Node\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert at the beginning: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;

            case 2:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;

            case 3:
                printf("Enter the value to delete: ");
                scanf("%d", &value);
                deleteNode(value);
                break;

            case 4:
                display();
                break;

            case 5:
                printf("Exiting program.\n");
```

```
            exit(0);
        default:
            printf("Invalid choice! Please enter again.\n");
    }
  }
  return 0;
}
```



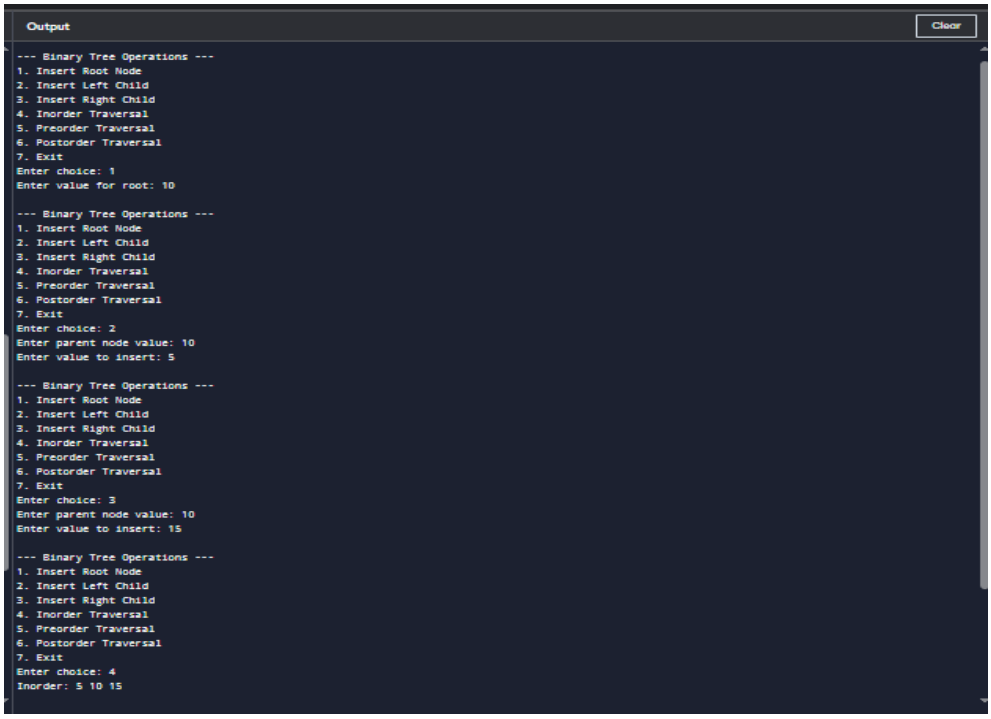
13. Write a program to perform tree traversal methods

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

struct Node {
  int data;
  struct Node* left;
  struct Node* right;
};

struct Node* createNode(int value) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}

void inorder(struct Node* root) {
  if (root == NULL) return;
  inorder(root->left);
  printf("%d ", root->data);
```

```
    inorder(root->right);
}

void preorder(struct Node* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void postorder(struct Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

int main() {
    struct Node* root = NULL;
    int choice, value;
    while (1) {
        printf("\n--- Binary Tree Operations ---\n");
        printf("1. Insert Root Node\n");
        printf("2. Insert Left Child\n");
        printf("3. Insert Right Child\n");
        printf("4. Inorder Traversal\n");
        printf("5. Preorder Traversal\n");
        printf("6. Postorder Traversal\n");
        printf("7. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (root != NULL) {
                    printf("Root already exists.\n");
                    break;
                }
                printf("Enter value for root: ");
                scanf("%d", &value);
                root = createNode(value);
                break;
            case 2:
            case 3:
                printf("Enter parent node value: ");
                int parent;
                scanf("%d", &parent);
                struct Node* temp = root;
```

```
    struct Node* parentNode = NULL;
    struct Node* queue[100];
    int front = 0, rear = 0;
    if (temp != NULL) queue[rear++] = temp;
    while (front < rear) {
        temp = queue[front++];
        if (temp->data == parent) {
            parentNode = temp;
            break;
        }
        if (temp->left) queue[rear++] = temp->left;
        if (temp->right) queue[rear++] = temp->right;
    }
    if (parentNode == NULL) {
        printf("Parent not found.\n");
        break;
    }
    printf("Enter value to insert: ");
    scanf("%d", &value);
    struct Node* newNode = createNode(value);
    if (choice == 2) {
        if (parentNode->left != NULL) {
            printf("Left child already exists.\n");
            free(newNode);
        } else parentNode->left = newNode;
    } else {
        if (parentNode->right != NULL) {
            printf("Right child already exists.\n");
            free(newNode);
        } else parentNode->right = newNode;
    }
    break;

case 4:
    printf("Inorder: ");
    inorder(root);
    printf("\n");
    break;

case 5:
    printf("Preorder: ");
    preorder(root);
    printf("\n");
    break;

case 6:
    printf("Postorder: ");
    postorder(root);
```

```
            printf("\n");
            break;

        case 7:
            exit(0);
        default:
            printf("Invalid choice.\n");
      }
   }
   return 0;
}
```

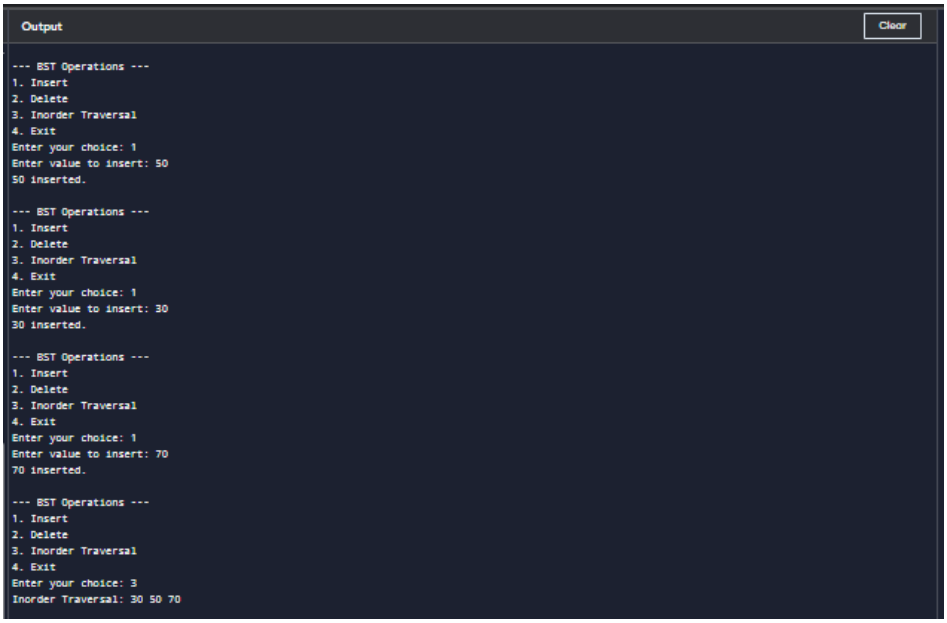


14. Write a program to perform insertion and deletion in the Binary Search Tree

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

struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};

struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->left = NULL;
   newNode->right = NULL;
```

```
    return newNode;
}

struct Node* insert(struct Node* root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    return root;
}
struct Node* findMin(struct Node* root) {
    while (root && root->left) root = root->left;
    return root;
}
struct Node* deleteNode(struct Node* root, int value) {
    if (!root) return NULL;
    if (value < root->data)
        root->left = deleteNode(root->left, value);
    else if (value > root->data)
        root->right = deleteNode(root->right, value);
    else {
        if (!root->left) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (!root->right) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        } else {
            struct Node* temp = findMin(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }
    return root;
}
void inorder(struct Node* root) {
    if (!root) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

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

```
    while (1) {
        printf("\n--- BST Operations ---\n");
        printf("1. Insert\n2. Delete\n3. Inorder Traversal\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                printf("%d inserted.\n", value);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                root = deleteNode(root, value);
                printf("%d deleted if it existed.\n", value);
                break;
            case 3:
                printf("Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 4:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}
```



15. **Write a program to represent an undirected graph using the adjacency matrix to implement the graph and perform the operations with menu-driven options for the following tasks: 1. Create graph 2. Insert an edge 3. Print Adjacency Matrix 4. List all vertices that are adjacent to a specified vertex. 6. Exit program**

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

int adjacencyMatrix[100][100];
int numVertices = 0;

void createGraph() {
   printf("Enter the number of vertices in the graph: ");
   scanf("%d", &numVertices);
   for (int i = 0; i < numVertices; i++)
      for (int j = 0; j < numVertices; j++)
         adjacencyMatrix[i][j] = 0;
   printf("Graph with %d vertices created.\n", numVertices);
}

void insertEdge() {
   if (numVertices == 0) {
      printf("Graph is not created yet.\n");
      return;
   }
   int u, v;
   printf("Enter the two vertices to connect (0 to %d): ", numVertices - 1);
   scanf("%d %d", &u, &v);
   if (u < 0 || u >= numVertices || v < 0 || v >= numVertices) {
      printf("Invalid vertices.\n");
      return;
   }
   adjacencyMatrix[u][v] = 1;
   adjacencyMatrix[v][u] = 1;
   printf("Edge inserted between %d and %d.\n", u, v);
}

void printAdjacencyMatrix() {
   if (numVertices == 0) {
      printf("Graph is not created yet.\n");
      return;
   }
   printf("Adjacency Matrix:\n   ");
   for (int i = 0; i < numVertices; i++)
      printf("%d ", i);
   printf("\n");
   for (int i = 0; i < numVertices; i++) {
      printf("%d: ", i);
```

```
        for (int j = 0; j < numVertices; j++)
            printf("%d ", adjacencyMatrix[i][j]);
        printf("\n");
    }
}

void listAdjacentVertices() {
    if (numVertices == 0) {
        printf("Graph is not created yet.\n");
        return;
    }
    int v;
    printf("Enter the vertex to list adjacent vertices (0 to %d): ", numVertices - 1);
    scanf("%d", &v);
    if (v < 0 || v >= numVertices) {
        printf("Invalid vertex.\n");
        return;
    }
    printf("Vertices adjacent to %d: ", v);
    for (int i = 0; i < numVertices; i++) {
        if (adjacencyMatrix[v][i] == 1)
            printf("%d ", i);
    }
    printf("\n");
}

int main() {
    int choice;
    while (1) {
        printf("\n--- Undirected Graph Menu ---\n");
        printf("1. Create Graph\n");
        printf("2. Insert an Edge\n");
        printf("3. Print Adjacency Matrix\n");
        printf("4. List Adjacent Vertices\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createGraph();
                break;
            case 2:
                insertEdge();
                break;
            case 3:
                printAdjacencyMatrix();
                break;
```

```
            case 4:
                listAdjacentVertices();
                break;
            case 5:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}
```