# Central university of Haryana

Department of Computer Science & Engineering

(Village Jant-Pali, Mahendergarh, Haryana-123031)


# Data Structures and Algorithms Lab

(BT CS 304)

Submitted by :-

Vikas Singh

Roll No: 241504

Btech (CSE) 2nd year

Submitted to :-

Prof. Anant R. Bara

(Assistant Professor)

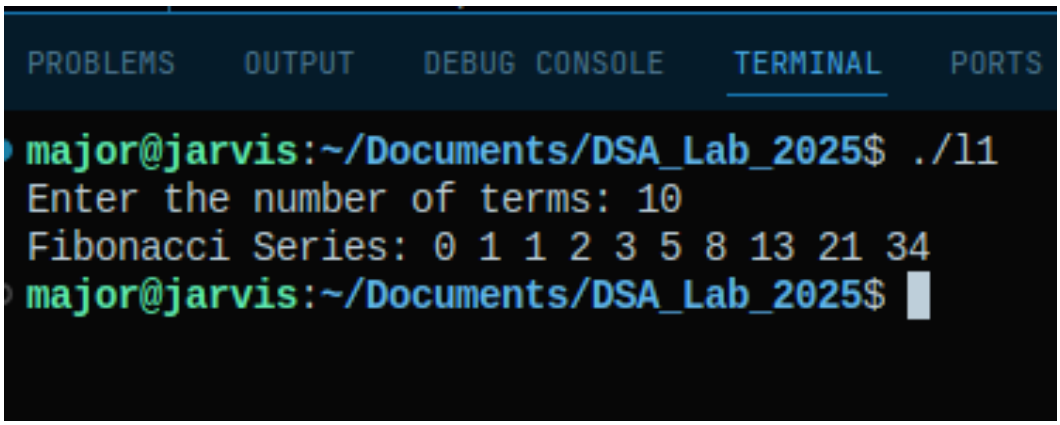
Department of CSE

**Program 1:** Recursive generation of Fibonacci series up to N terms.

**Solution:**

```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Recursive Fibonacci series
int RecursiveFibbonacci(int n)
{
   if (n == 0)  return 0;
   if (n == 1)  return 0;
   return RecursiveFibbonacci(n - 1) + RecursiveFibbonacci(n - 2);
}
// Main Function
int main()
{
   int n;
   cout << "Enter the number of terms: ";
   cin >> n;
   cout << "Fibonacci Series: ";
   // Loop to print the series up to n terms
   for (int i = 0; i < n; i++)
   {
      cout << RecursiveFibbonacci(i) << " ";
   }
   cout << endl;
   return 0;
}
```

**Output:**



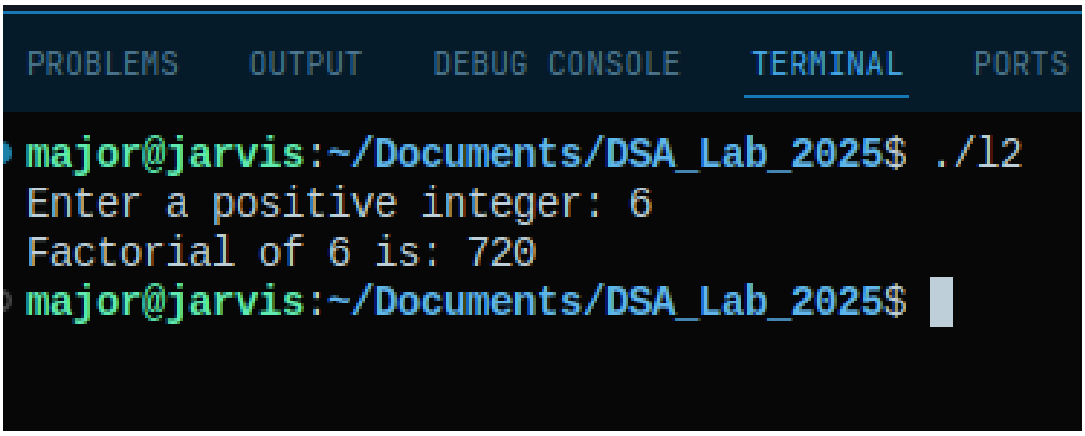

**GitHub Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab1.cpp

**Program 2:** Recursive computation of factorial for a given number.

**Solution:**

```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Recursive Fibonacci series
long long factorial(int n)
{
   if (n <= 1)  return 1;
   else  return n * factorial(n - 1);
}
// Main Function
int main()
{
   int number;
   cout << "Enter a positive integer: ";
   cin >> number;

   if (number < 0) {
      cout << "Error: Factorial of a negative number doesn't exist." << endl;
   }
   else{
      cout << "Factorial of " << number << " is: " << factorial(number) << endl;
   }
   return 0;
}
```

**Output:**




**GitHub Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab2.cpp

**Program 3:** Program to insert, delete, and traverse elements in a one-dimensional array.

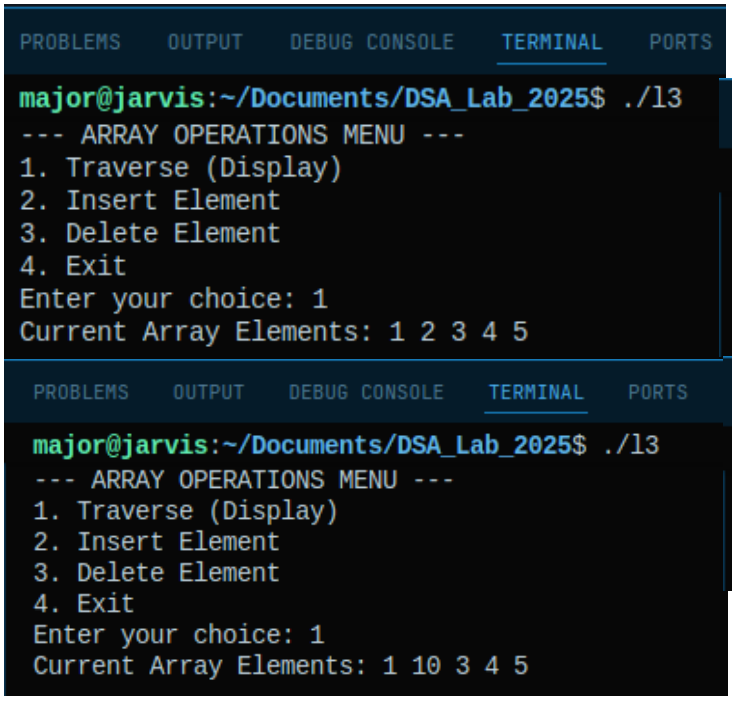
**Solution:**

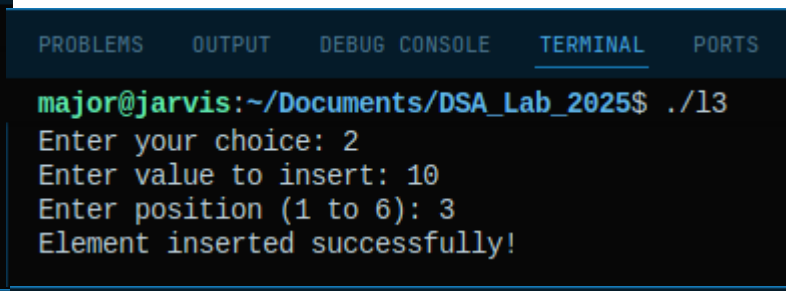
```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Define maximum capacity of the array
const int MAX_SIZE = 100;
// Global variables
int arr[MAX_SIZE];
int currentSize = 0;
// Traverse Function
void traverse(){
    if (currentSize == 0){
        cout << "Array is empty." << endl;
        return;
    }
    cout << "Current Array Elements: ";
    for (int i = 0; i < currentSize; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
}
// Insertion Function
void insertElement(int value, int position){
    if (currentSize >= MAX_SIZE){
        cout << "Error: Array is full (Overflow)." << endl;
        return;
    }
    if (position < 0 || position > currentSize + 1){
        cout << "Error: Invalid position!" << endl;
        return;
    }
    for (int i = currentSize; i >= position; i--){
        arr[i] = arr[i - 1];
    }
    arr[position - 1] = value;
    currentSize++;
    cout << "Element inserted successfully!" << endl;
}
```

```
// Deletion Function
void deleteElement(int position){
    if (currentSize == 0){
        cout << "Error: Array is empty (Underflow)." << endl;
        return;
    }
    if (position < 1 || position > currentSize){
        cout << "Error: Invalid position!" << endl;
        return;
    }
    for (int i = position - 1; i < currentSize - 1; i++){
        arr[i] = arr[i + 1];
    }
    currentSize--;
    cout << "Element deleted successfully!" << endl;
}
// Main Function
int main(){
    int choice, value, position;
    cout << "Enter initial number of elements (max " << MAX_SIZE << "): ";
    cin >> currentSize;
    cout << "Enter " << currentSize << " elements: ";
    for (int i = 0; i < currentSize; i++){
        cin >> arr[i];
    }
    do{
        cout << "\n--- ARRAY OPERATIONS MENU ---" << endl;
        cout << "1. Traverse (Display)\n
                    2. Insert Element\n
                    3. Delete Element\n
                    4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice){
        case 1:
            traverse();
            break;
        case 2:
            cout << "Enter value to insert: ";
```

```
            cin >> value;
            cout << "Enter position (1 to " << currentSize + 1 << "): ";
            cin >> position;
            insertElement(value, position);
            break;
        case 3:
            cout << "Enter position to delete (1 to " << currentSize << "): ";
            cin >> position;
            deleteElement(position);
            break;
        case 4:
            cout << "Exiting program..." << endl;
            break;
        default:
            cout << "Invalid choice! Try again." << endl;
        }
    } while (choice != 4);
    return 0;
}
```

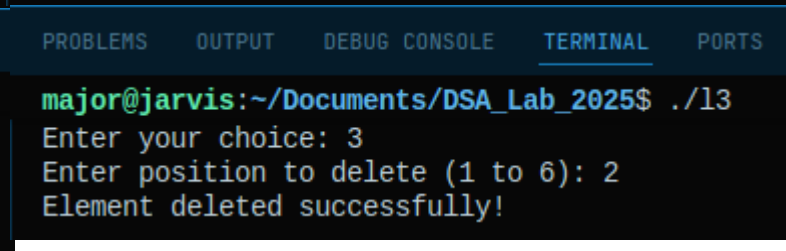
## Output:










**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab3.cpp

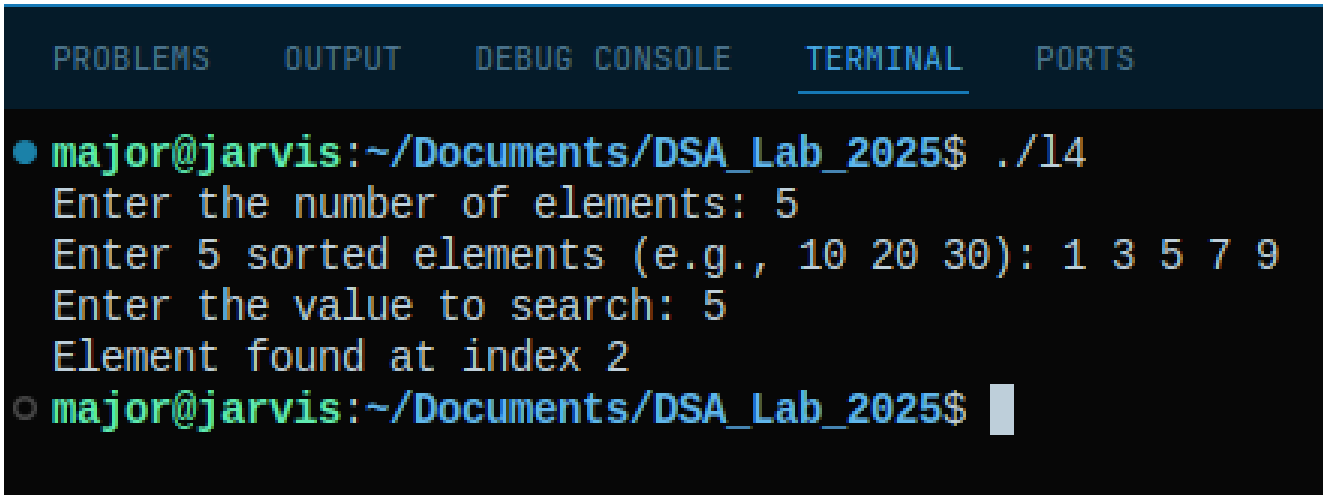
**Program 4:** Program to implement binary search in a sorted array.

**Solution:**

```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Binary Search Funcion
int binarySearch(int arr[], int size, int target){
    int low = 0;
    int high = size - 1;
    while (low <= high){

        int mid = low + (high - low) / 2;

        if (arr[mid] == target){
            return mid;
        }
        if (arr[mid] < target){
            low = mid + 1;
        }
         else{
            high = mid - 1;
        }
    }
    return -1;
}
// Mian Function
int main(){
    int n, target;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];

    cout << "Enter " << n << " sorted elements (e.g., 10 20 30): ";
    for (int i = 0; i < n; i++){
        cin >> arr[i];
    }

    cout << "Enter the value to search: ";
```

```
    cin >> target;
    int result = binarySearch(arr, n, target);

    if (result != -1){
        cout << "Element found at index " << result << endl;
    }
    else{
        cout << "Element not found in the array." << endl;
    }

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab4.cpp

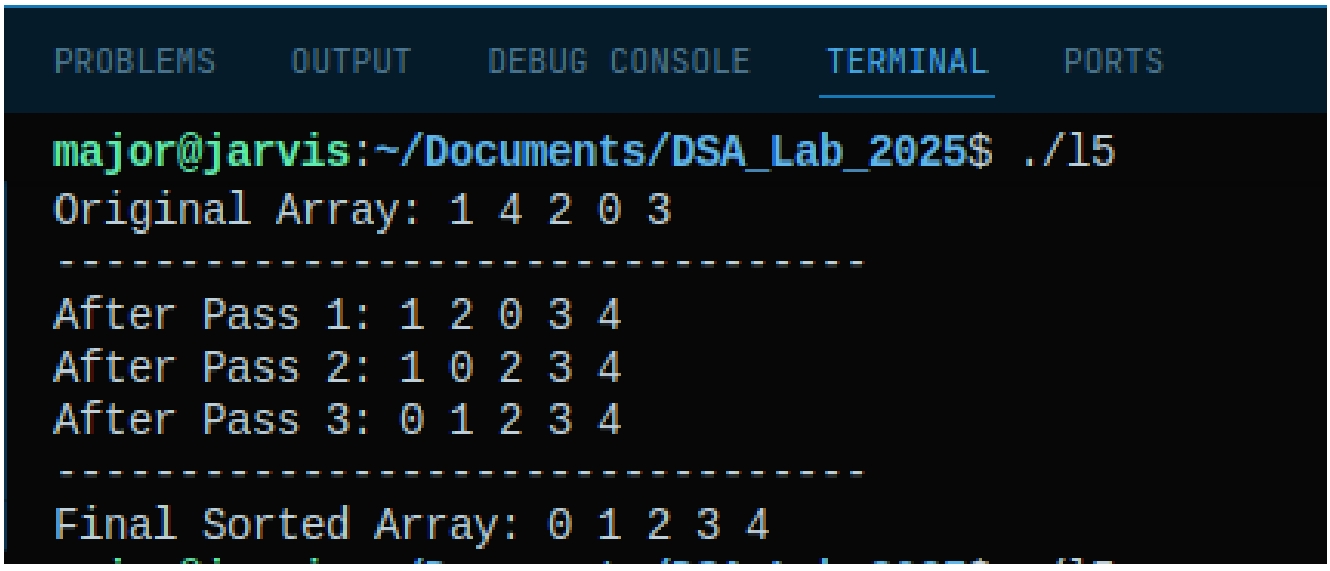
**Program 5:** Program to implement Bubble Sort and display intermediate passes.

**Solution:**

```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Print function
void printArray(int arr[], int size){
    for (int i = 0; i < size; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
}
// Bubble Sort Function
void bubbleSort(int arr[], int n){
    bool swapped;
    for (int i = 0; i < n - 1; i++){
        swapped = false;
        for (int j = 0; j < n - i - 1; j++){
            if (arr[j] > arr[j + 1]){
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        if (swapped == false) return;

        cout << "After Pass " << i + 1 << ": ";
        printArray(arr, n);
    }
}
// Main Function
int main(){
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
```

```
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++){
        cin >> arr[i];
    }

    cout << "\nOriginal Array: ";
    printArray(arr, n);
    cout << "--------------------------------" << endl;

    bubbleSort(arr, n);

    cout << "--------------------------------" << endl;
    cout << "Final Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab5.cpp

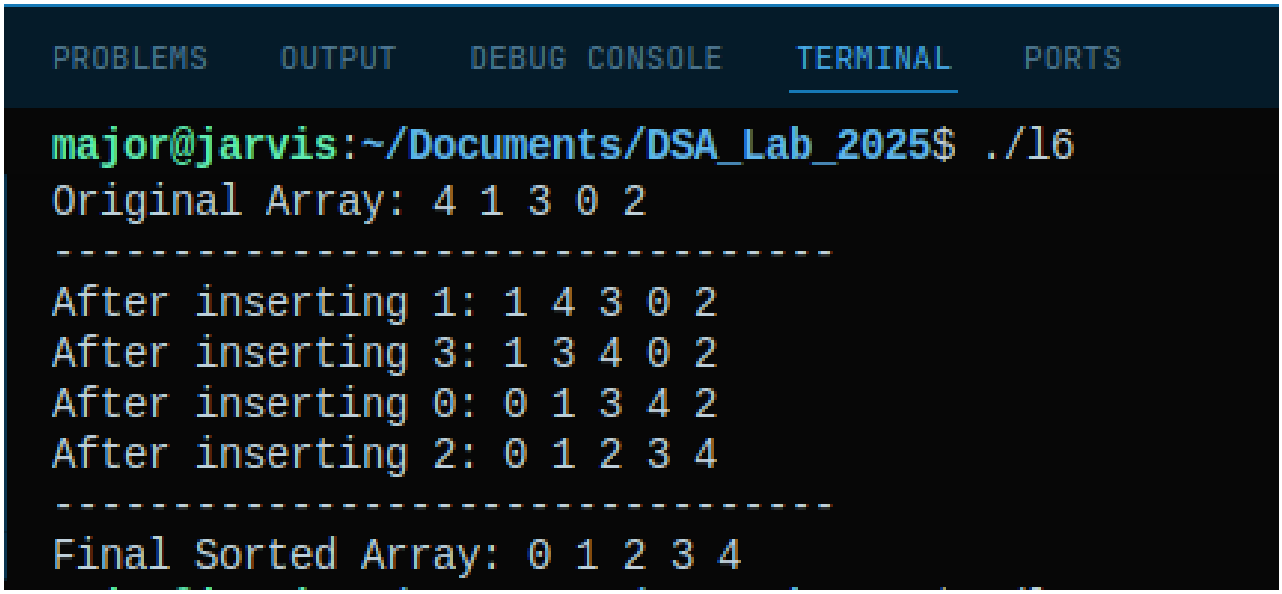
**Program 6:** Program to implement Insertion Sort.

**Solution:**

```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Print Function
void printArray(int arr[], int size){
    for (int i = 0; i < size; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
}
// Insertion Sort Function
void insertionSort(int arr[], int n){
    int key, j;

    for (int i = 1; i < n; i++){
        key = arr[i];
        j = i - 1;

        while ( j >= 0  &&  arr[j] > key ){
            arr[j + 1] = arr[j];
            j = j - 1;
        }

        arr[j + 1] = key;
        cout << "After inserting " << key << ": ";
        printArray(arr, n);
    }
}
// Main Function
int main(){
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++){
```

```
        cin >> arr[i];
    }

    cout << "\nOriginal Array: ";
    printArray(arr, n);
    cout << "---------------------------------" << endl;
    insertionSort(arr, n);
    cout << "---------------------------------" << endl;
    cout << "Final Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab6.cpp

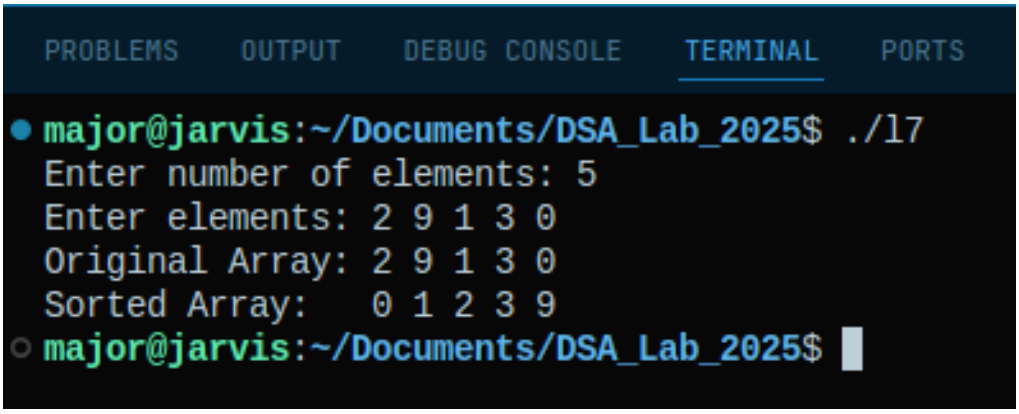
**Program 7:** Program to implement Merge Sort using recursion.

**Solution:**

```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Merge Function
void merge(int arr[], int left, int mid, int right){
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right – mid;
    // Create temporary arrays
    int leftArray[n1], rightArray[n2];
    for (i = 0; i < n1; i++){
        leftArray[i] = arr[left + i];
    }
    for (j = 0; j < n2; j++){
        rightArray[j] = arr[mid + 1 + j];
    }
    i = j = 0; k = left;
    while ( i < n1  &&  j < n2 ){
        if ( leftArray[i] <= rightArray[j] ){
            arr[k] = leftArray[i];
            i++;
        } else{
            arr[k] = rightArray[j];
            j++;
        }  k++;
    }
    while (i < n1){
        arr[k] = leftArray[i];
        i++, k++;
    }
    while (j < n2) {
        arr[k] = rightArray[j];
        j++, k++;
    }
}
// Recursive Merge Sort Function
void mergeSort(int arr[], int left, int right){
```

```
    if (left < right){
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
// Print Function
void printArray(int arr[], int size){
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}
// Main Function
int main(){
    int n; int arr[n];
    cout << "Enter number of elements: ";
    cin >> n;
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++){
        cin >> arr[i];
    }
    cout << "Original Array: ";
    printArray(arr, n);
    mergeSort(arr, 0, n - 1);
    cout << "Sorted Array:   ";
    printArray(arr, n);
    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab7.cpp

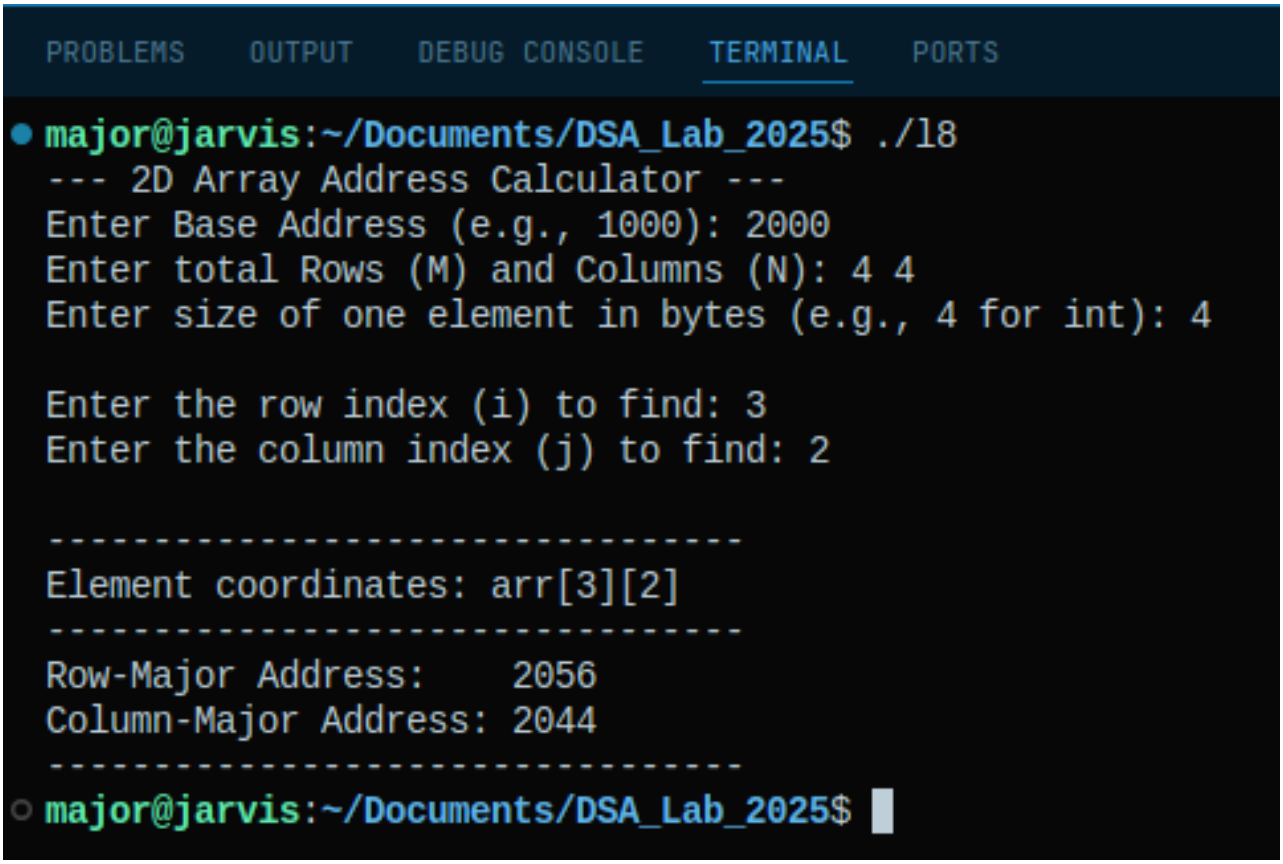
**Program 8:** Program to calculate address of an element in a 2D array using row-major and column-major formulas.

**Solution:**

```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Main Function
int main(){
   long baseAddress;
   int rows, cols, i, j, elementSize;

   // Input 2D Array
   cout << "--- 2D Array Address Calculator ---" << endl;
   cout << "Enter Base Address (e.g., 1000): ";
   cin >> baseAddress;

   cout << "Enter total Rows (M) and Columns (N): ";
   cin >> rows >> cols;
   cout << "Enter size of one element in bytes (e.g., 4 for int): ";
   cin >> elementSize;

   // Input Target Coordinate (i,j)
   cout << "\nEnter the row index (i) to find: ";
   cin >> i;
   cout << "Enter the column index (j) to find: ";
   cin >> j;

   // Check
   if (i >= rows || j >= cols || i < 0 || j < 0){
      cout << "Error: Indices out of bounds!" << endl;
      return 1;
   }

   long rowMajorAddr = baseAddress + elementSize * ((i * cols) + j);
   long colMajorAddr = baseAddress + elementSize * ((j * rows) + i);

   cout << "\n--------------------------------" << endl;
   cout << "Element coordinates: arr[" << i << "][" << j << "]" << endl;
   cout << "--------------------------------" << endl;
```

```
    cout << "Row-Major Address:    " << rowMajorAddr << endl;
    cout << "Column-Major Address: " << colMajorAddr << endl;
    cout << "--------------------------------" << endl;

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab8.cpp

**Program 9:** Program to implement stack using arrays with push, pop, peek, and display operations.

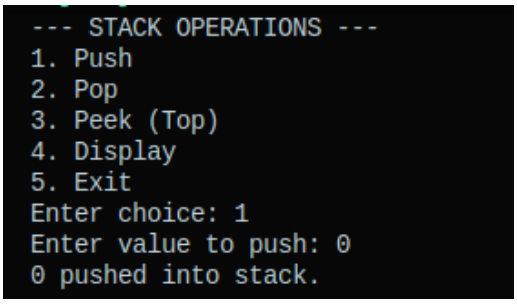
**Solution:**

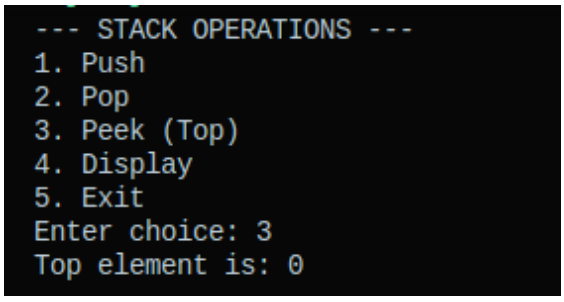
```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Size Defination
#define MAX 100
// Stack class
class Stack{
    int top;
    int arr[MAX];
public:
    // Constructor
    Stack(){
        top = -1;
    }
    // Push Function
    void push(int x){
        if (top >= MAX - 1){
            cout << "Error: Stack Overflow! Cannot add more elements." << endl;
            return;
        }
        arr[++top] = x;
        cout << x << " pushed into stack." << endl;
    }
     // Pop Function
    void pop(){
        if (top < 0){
            cout << "Error: Stack Underflow! Stack is empty." << endl;
            return;
        }
        int x = arr[top--];
        cout << x << " popped from stack." << endl;
    }
    // Peek Function
    void peek(){
        if (top < 0){
            cout << "Stack is Empty." << endl;
            return;
```

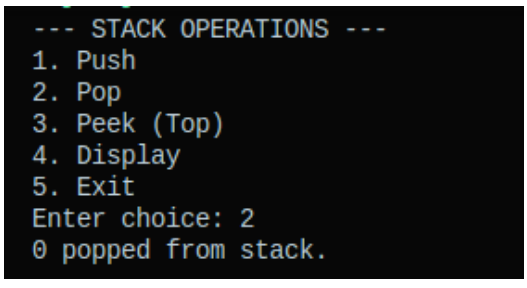
```
        }
        cout << "Top element is: " << arr[top] << endl;
    }
     // Display Function
    void display(){
        if (top < 0){
            cout << "Stack is Empty." << endl;
            return;
        }
        cout << "Stack elements: ";
        for (int i = top; i >= 0; i--){
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};
// Main Function
int main(){
    Stack s;
    int choice, value;
    do{
        cout << "\n--- STACK OPERATIONS ---" << endl;
        cout << "1. Push" << endl;
        cout << "2. Pop" << endl;
        cout << "3. Peek (Top)" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice){
        case 1:
            cout << "Enter value to push: ";
            cin >> value;
            s.push(value);
            break;
        case 2:
            s.pop();
            break;
```

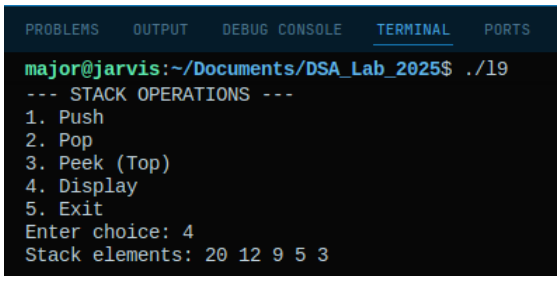
```
        case 3:
            s.peek();
            break;
        case 4:
            s.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid Choice!" << endl;
        }
    } while (choice != 5);

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab9.cpp

**Program 10:** Program to implement stack using linked list with dynamic memory allocation.

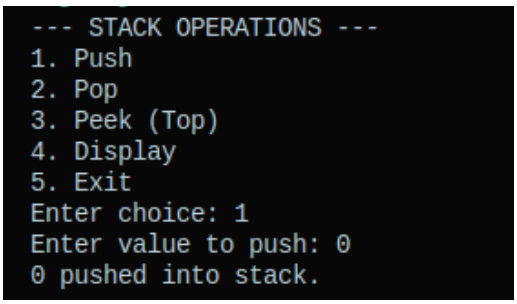
**Solution:** // Author: [Vikas Singh 241504]

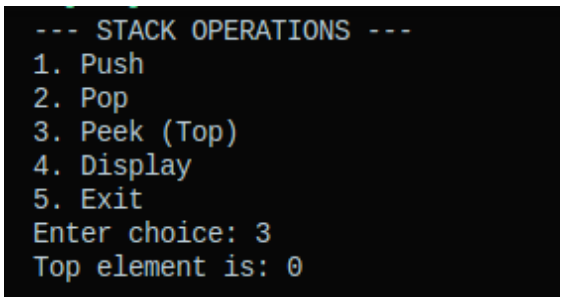
```
#include <iostream>
using namespace std;
// Node Structure
struct Node{
    int data;
    Node *next;
};
// Stack class
class Stack{
private:
    Node *top;
public:
    // Constructor
    Stack(){
        top = NULL;
    }
    // Push Function
    void push(int value){
        Node *newNode = new Node();
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        cout << value << " pushed to stack." << endl;
    }
    // Pop Function
    void pop(){
        if (top == NULL){
            cout << "Error: Stack Underflow (Empty)." << endl;
            return;
        }
        Node *temp = top;
        top = top->next;
        cout << temp->data << " popped from stack." << endl;
        delete temp;
    }
    // Peek Function
```

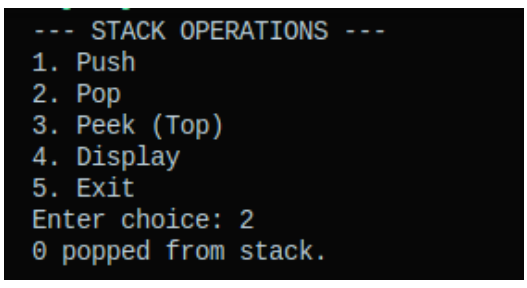
```
    void peek(){
        if (top == NULL){
            cout << "Stack is Empty." << endl;
            return;
        }
        cout << "Top element is: " << top->data << endl;
    }
    // Display Function
    void display(){
        if (top == NULL){
            cout << "Stack is Empty." << endl;
            return;
        }
        Node *temp = top;
        cout << "Stack: ";
        while (temp != NULL){
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};
// Main Function
int main()
{
    Stack s;
    int choice, val;
    do{
        cout << "\n--- DYNAMIC STACK MENU ---" << endl;
        cout << "1. Push" << endl;
        cout << "2. Pop" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter choice: ";
        cin >> choice;
        switch (choice){
        case 1:
            cout << "Enter value: ";
```

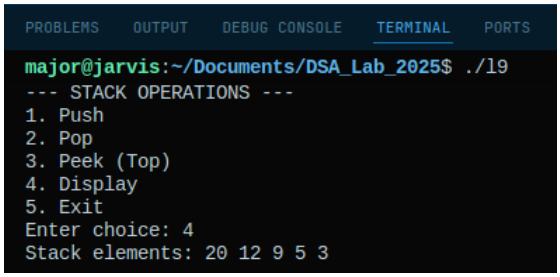
```
            cin >> val;
            s.push(val);
            break;
        case 2:
            s.pop();
            break;
        case 3:
            s.peek();
            break;
        case 4:
            s.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:  cout << "Invalid choice." << endl;
        }
    } while (choice != 5);
    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab10.cpp

**Program 11:** Program to implement queue using arrays with enqueue, dequeue, peek, and display Operations.

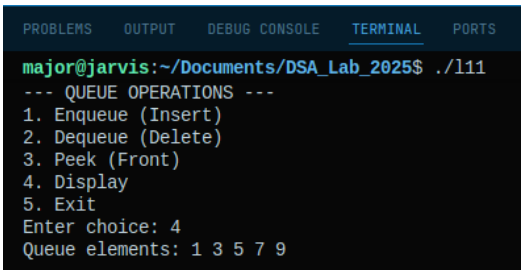
**Solution:**

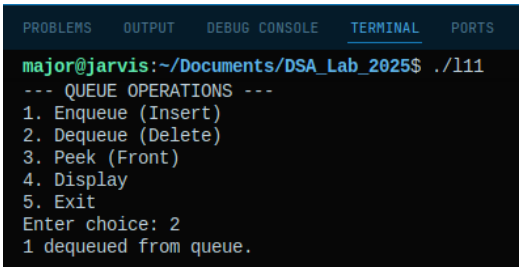
```
// Author: [Vikas Singh 241504]
#include <iostream>
using namespace std;
// Maximum capacity of the queue
#define MAX 100
// Queue Class
class Queue{
    int arr[MAX];
    int front;
    int rear;
public:
    // Constructor
    Queue(){
        front = rear = -1;
    }
    // Enqueue Function
    void enqueue(int value){
        if (rear == MAX - 1){
            cout << "Error: Queue Overflow! (Queue is full)" << endl;
            return;
        }
        if (front == -1){
            front = 0;
        }
        arr[++rear] = value;
        cout << value << " enqueued into queue." << endl;
    }
    // Dequeue Function
    void dequeue(){
        if (front == -1 || front > rear){
            cout << "Error: Queue Underflow! (Queue is empty)" << endl;
            return;
        }
        cout << arr[front] << " dequeued from queue." << endl;
        front++;
        if (front > rear){
            front = rear = -1;
```

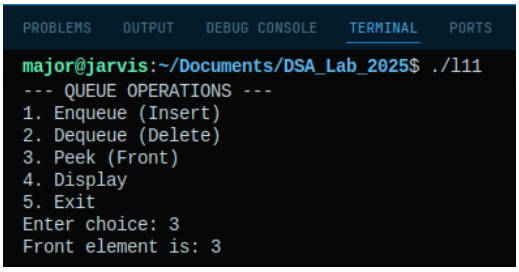
```
    }
  }
   // Peek Function
  void peek(){
    if (front == -1 || front > rear){
      cout << "Queue is Empty." << endl;
      return;
    }
    cout << "Front element is: " << arr[front] << endl;
  }
  // Display Function
  void display(){
    if (front == -1 || front > rear){
      cout << "Queue is Empty." << endl;
      return;
    }
    cout << "Queue elements: ";
    for (int i = front; i <= rear; i++){
      cout << arr[i] << " ";
    }
    cout << endl;
  }
};
// Main Function
int main(){
  Queue q;
  int choice, value;
  do{
    cout << "\n--- QUEUE OPERATIONS ---" << endl;
    cout << "1. Enqueue (Insert)" << endl;
    cout << "2. Dequeue (Delete)" << endl;
    cout << "3. Peek (Front)" << endl;
    cout << "4. Display" << endl;
    cout << "5. Exit" << endl;
    cout << "Enter choice: ";
    cin >> choice;
    switch (choice){
    case 1:
      cout << "Enter value to enqueue: ";
```

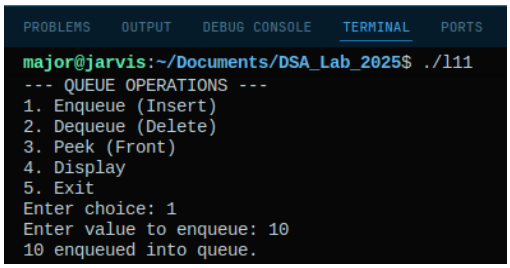
```
            cin >> value;
            q.enqueue(value);
            break;
        case 2:
            q.dequeue();
            break;
        case 3:
            q.peek();
            break;
        case 4:
            q.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:  cout << "Invalid choice!" << endl;
        }
    } while (choice != 5);
    return 0;
}
```

## Output:




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab11.cpp

**Program 12:** Program to implement queue using linked list with dynamic memory allocation.

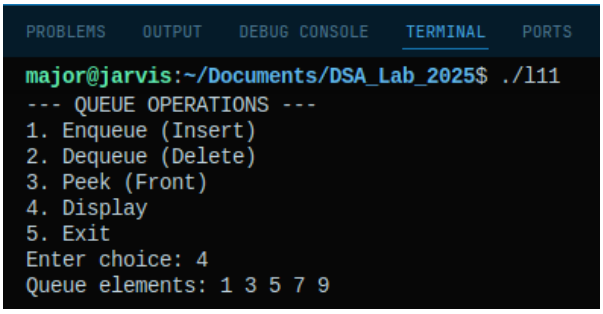
**Solution:** // Author: [Vikas Singh 241504]

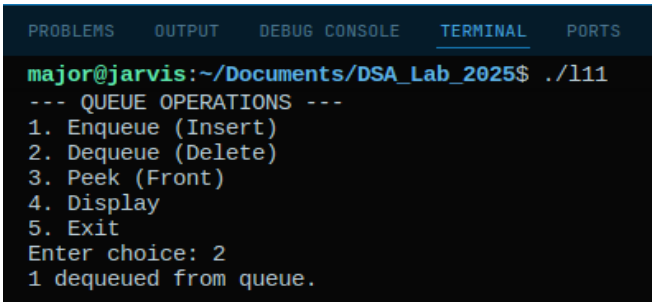
```
#include <iostream>
using namespace std;
// Node Structure
struct Node{
    int data;
    Node *next;
};
// Queue class
class Queue{
private:
    Node *front, *rear;
public:
    // Constructor
    Queue(){
        front = rear = NULL;
    }
    // Enqueue function
    void enqueue(int value){
        Node *newNode = new Node();
        if (!newNode){
            cout << "Heap Overflow! Cannot allocate memory." << endl;
            return;
        }
        newNode->data = value;
        newNode->next = NULL;
        if (front == NULL){
            front = rear = newNode;
        }  else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << value << " enqueued to queue." << endl;
    }
    // Dequeue function
    void dequeue(){
        if (front == NULL){
```

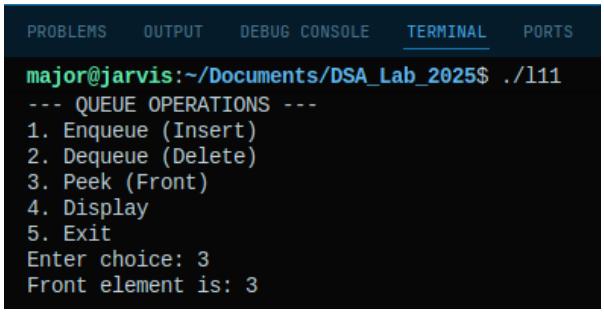
```
            cout << "Error: Queue Underflow (Empty)." << endl;
            return;
        }
        Node *temp = front;
        cout << front->data << " dequeued from queue." << endl;
        front = front->next;
        if (front == NULL){
            rear = NULL;
        } delete temp;
    }
    // Peek function
    void peek(){
        if (front == NULL){
            cout << "Queue is Empty." << endl;
            return;
        }
        cout << "Front element is: " << front->data << endl;
    }
    // Display function
    void display(){
        if (front == NULL) {
            cout << "Queue is Empty." << endl;
            return;
        }
        Node *temp = front;
        cout << "Queue: ";
        while (temp != NULL){
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};
// Main function
int main(){
    Queue q;
    int choice, val;
    do{
        cout << "\n--- LINKED LIST QUEUE MENU ---" << endl;
```

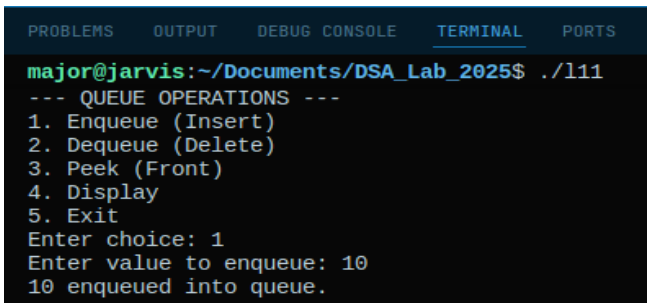
```
    cout << "1. Enqueue" << endl;
    cout << "2. Dequeue" << endl;
    cout << "3. Peek" << endl;
    cout << "4. Display" << endl;
    cout << "5. Exit" << endl;
    cout << "Enter choice: ";
    cin >> choice;
    switch (choice){
    case 1: cout << "Enter value: ";
       cin >> val;
       q.enqueue(val);
       break;
    case 2: q.dequeue();
       break;
    case 3:  q.peek();
       break;
    case 4: q.display();
       break;
    case 5: cout << "Exiting..." << endl; break;
    default:  cout << "Invalid choice." << endl;
    }
  } while (choice != 5);
  return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab12.cpp

**Program 13:** Program to implement queue using two stacks

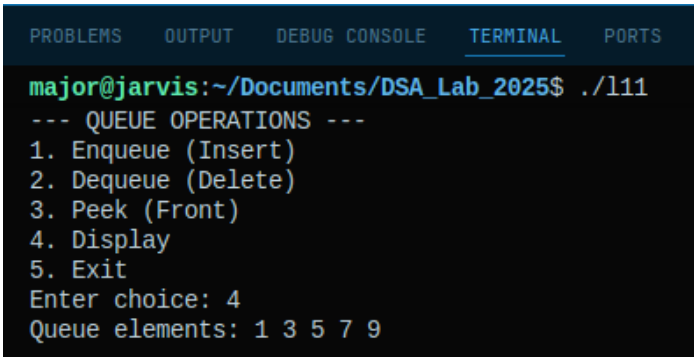
**Solution:** // Author: [Vikas Singh 241504]

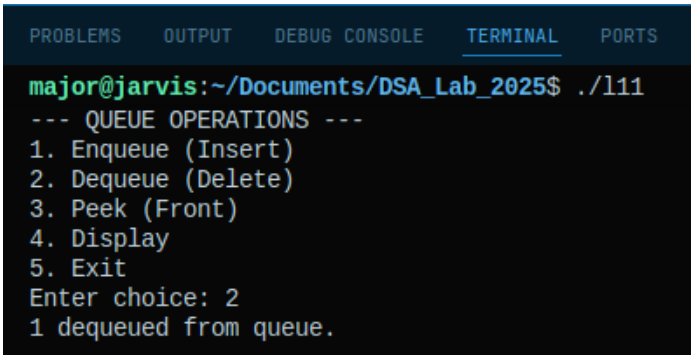
```
#include <iostream>
#include <stack>
using namespace std;
// Queue using Stack class
class QueueUsingStacks{
private:
   stack<int> inputStack;
   stack<int> outputStack;
public:
   // Enqueue Function
   void enqueue(int x){
      inputStack.push(x);
      cout << x << " enqueued." << endl;
   }
   // Dequeue Function
   void dequeue(){
      if ( outputStack.empty()  &&  inputStack.empty() ){
         cout << "Error: Queue Underflow (Empty)." << endl;
         return;
      }
      if (outputStack.empty()){
         while (!inputStack.empty()){
            outputStack.push(inputStack.top());
            inputStack.pop();
         }
      }
      int val = outputStack.top();
      outputStack.pop();
      cout << val << " dequeued." << endl;
   }
   // Peek Function
   void peek(){
      if (outputStack.empty() && inputStack.empty()){
         cout << "Queue is Empty." << endl;
         return;
      }
```

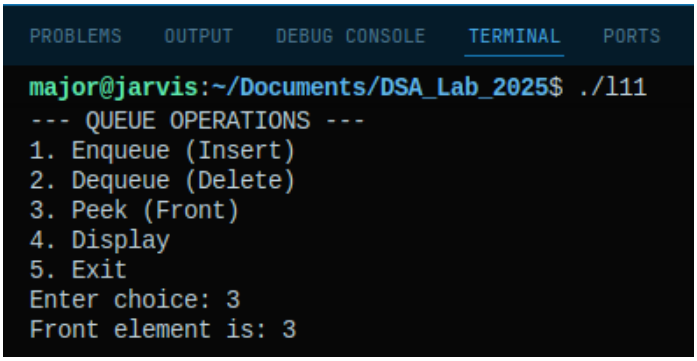
```
        if (outputStack.empty()){
            while (!inputStack.empty()){
                outputStack.push(inputStack.top());
                inputStack.pop();
            }
        }
        cout << "Front element: " << outputStack.top() << endl;
    }
    // Is Empty Function
    bool isEmpty(){
        return inputStack.empty() && outputStack.empty();
    }
};
// Main Function
int main(){
    QueueUsingStacks q;
    int choice, val;

    do{
        cout << "\n--- QUEUE VIA STACKS MENU ---" << endl;
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice){
        case 1:
            cout << "Enter value: ";
            cin >> val;
            q.enqueue(val);
            break;
        case 2:
            q.dequeue();
            break;
        case 3:
            q.peek();
            break;
```

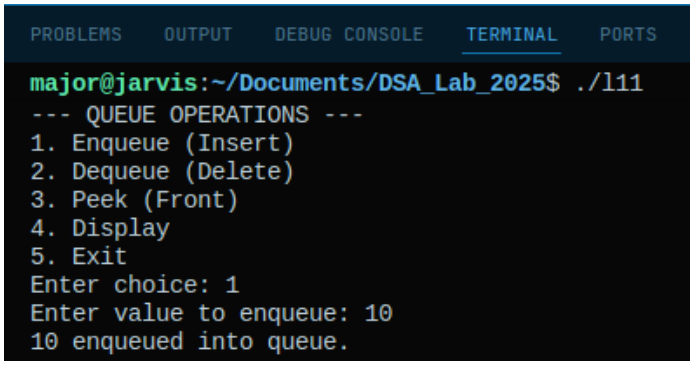
```
        case 4:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice." << endl;
        }
    } while (choice != 4);
    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab13.cpp

**Program 14:** Program to create a singly linked list and perform insertion at beginning, middle, and end.

**Solution:** // Author: [Vikas Singh 241504]

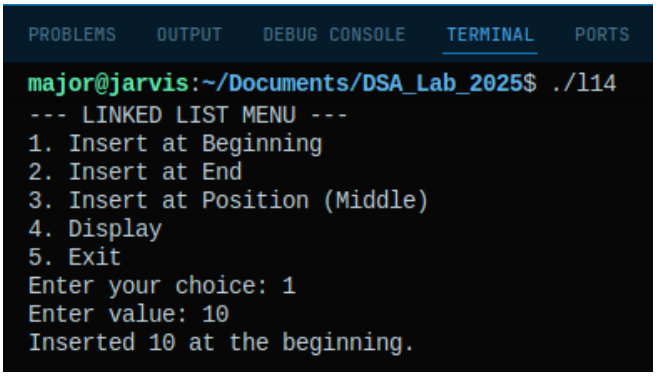
```
#include <iostream>
using namespace std;
// Node Structure
struct Node{
    int data;
    Node *next;
};
// Linked List class
class LinkedList{
private:
    Node *head;
public:
    // Constructor
    LinkedList(){
        head = NULL;
    }
    // Insert Function
    void insertAtBeginning(int value){
        Node *newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
        cout << "Inserted " << value << " at the beginning." << endl;
    }
    // Insert at the End function
    void insertAtEnd(int value){
        Node *newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;
        if (head == NULL){
            head = newNode;
            cout << "Inserted " << value << " as the first element." << endl;
            return;
        }
        Node *temp = head;
        while (temp->next != NULL){
```

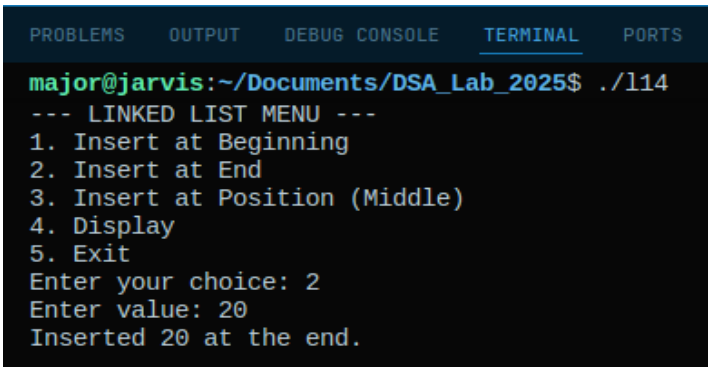
```
        temp = temp->next;
    }
    temp->next = newNode;
    cout << "Inserted " << value << " at the end." << endl;
}
// Insert at any Position
void insertAtPosition(int value, int position){
    if (position < 1){
        cout << "Invalid position!" << endl;
        return;
    }
    if (position == 1){
        insertAtBeginning(value);
        return;
    }
    Node *newNode = new Node();
    newNode->data = value;
    Node *temp = head;
    for (int i = 1; i < position - 1; i++){
        if (temp == NULL){
            cout << "Position out of bounds." << endl;
            return;
        }
        temp = temp->next;
    }
    if (temp == NULL){
        cout << "Position out of bounds." << endl;
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    cout << "Inserted " << value << " at position " << position << "." << endl;
}
// Display Function
void display()
{
    if (head == NULL){
        cout << "List is Empty." << endl;
        return;
```

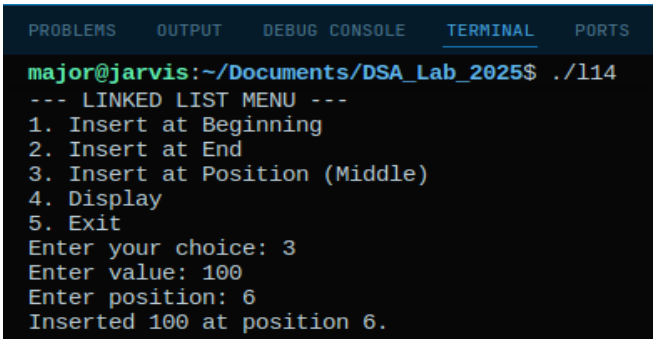
```
        }
        Node *temp = head;
        cout << "Current List: ";
        while (temp != NULL){
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};
// Main function
int main(){
    LinkedList list;
    int choice, value, pos;
    do{
        cout << "\n--- LINKED LIST MENU ---" << endl;
        cout << "1. Insert at Beginning" << endl;
        cout << "2. Insert at End" << endl;
        cout << "3. Insert at Position (Middle)" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice){
        case 1:
            cout << "Enter value: ";
            cin >> value;
            list.insertAtBeginning(value);
            break;
        case 2:
            cout << "Enter value: ";
            cin >> value;
            list.insertAtEnd(value);
            break;
        case 3:
            cout << "Enter value: ";
            cin >> value;
            cout << "Enter position: ";
```

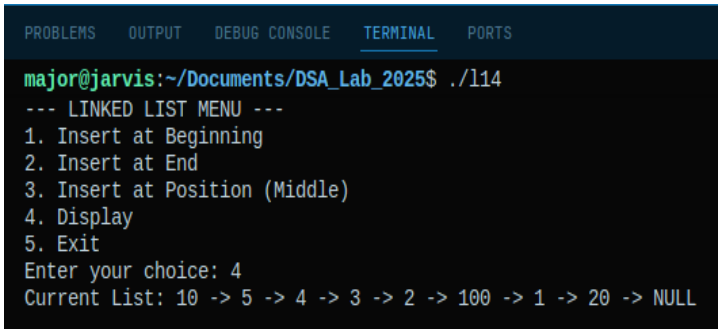
```
            cin >> pos;
            list.insertAtPosition(value, pos);
            break;
        case 4:
            list.display();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:  cout << "Invalid choice." << endl;
        }
    } while (choice != 5);
    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab14.cpp

**Program 15:** Program to delete a node from a singly linked list by value or position.

**Solution:** // Author: [Vikas Singh 241504]

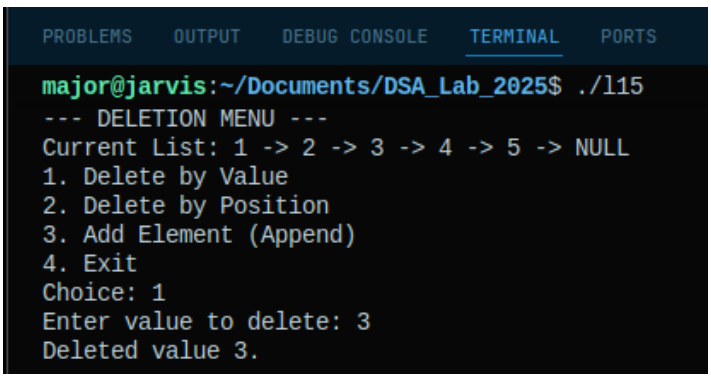
```
#include <iostream>
using namespace std;
// Node Structure
struct Node{
    int data;
    Node *next;
};
// Linked List Class
class LinkedList{
private:
    Node *head;
public:
    // Constructor
    LinkedList(){
        head = NULL;
    }
    // Insert Function
    void insertAtBeginning(int value){
        Node *newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }
    // Display Function
    void display(){
        if (head == NULL){
            cout << "List is Empty." << endl;
            return;
        }
        Node *temp = head;
        cout << "Current List: ";
        while (temp != NULL){
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
```

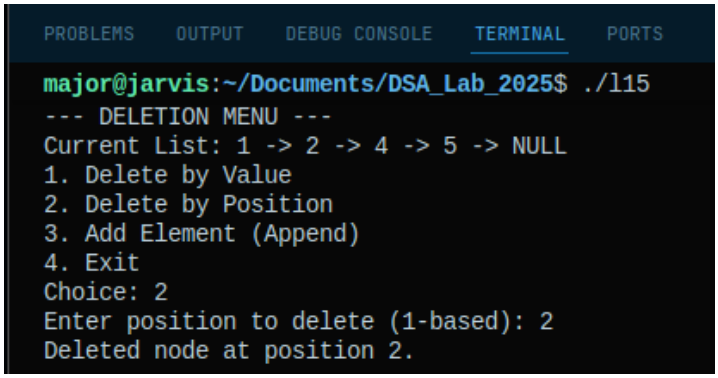
```
}
// Delete by value function
void deleteByValue(int value){
    if (head == NULL){
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    if (head->data == value){
        Node *toDelete = head;
        head = head->next;
        delete toDelete;
        cout << "Deleted value " << value << "." << endl;
        return;
    }
    Node *temp = head;
    while (temp->next != NULL && temp->next->data != value){
        temp = temp->next;
    }
    if (temp->next == NULL){
        cout << "Value " << value << " not found in the list." << endl;
        return;
    }
    Node *toDelete = temp->next;
    temp->next = toDelete->next;
    delete toDelete;
    cout << "Deleted value " << value << "." << endl;
}
// Delete by position function
void deleteByPosition(int position){
    if (head == NULL){
        cout << "List is empty." << endl;
        return;
    }
    if (position < 1){
        cout << "Invalid position." << endl;
        return;
    }
    if (position == 1){
        Node *toDelete = head;
```

```
        head = head->next;
        delete toDelete;
        return;
    }
    Node *temp = head;
    for (int i = 1; i < position - 1; i++){
        if (temp == NULL || temp->next == NULL){
            cout << "Position out of bounds." << endl;
            return;
        }
        temp = temp->next;
    }
    if (temp->next == NULL){
        cout << "Position out of bounds." << endl;
        return;
    }
    Node *toDelete = temp->next;
    temp->next = toDelete->next;
    delete toDelete;
  }
};
// Main Function
int main(){
  LinkedList list;
  int choice, val, pos;
  do{
    cout << "\n--- DELETION MENU ---" << endl;
    list.display();
    cout << "1. Delete by Value" << endl;
    cout << "2. Delete by Position" << endl;
    cout << "3. Add Element (Append)" << endl;
    cout << "4. Exit" << endl;
    cout << "Choice: ";
    cin >> choice;
    switch (choice){
    case 1:
        cout << "Enter value to delete: ";
        cin >> val;
        list.deleteByValue(val);
```

```
            break;
        case 2:
            cout << "Enter position to delete (1-based): ";
            cin >> pos;
            list.deleteByPosition(pos);
            break;
        case 3:
            cout << "Enter value to add: ";
            cin >> val;
            list. insertAtBeginning(val);
            break;
        case 4:
            cout << "Exiting..." << endl;
            break;
        default:  cout << "Invalid choice." << endl;
        }
    } while (choice != 4);
    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab15.cpp

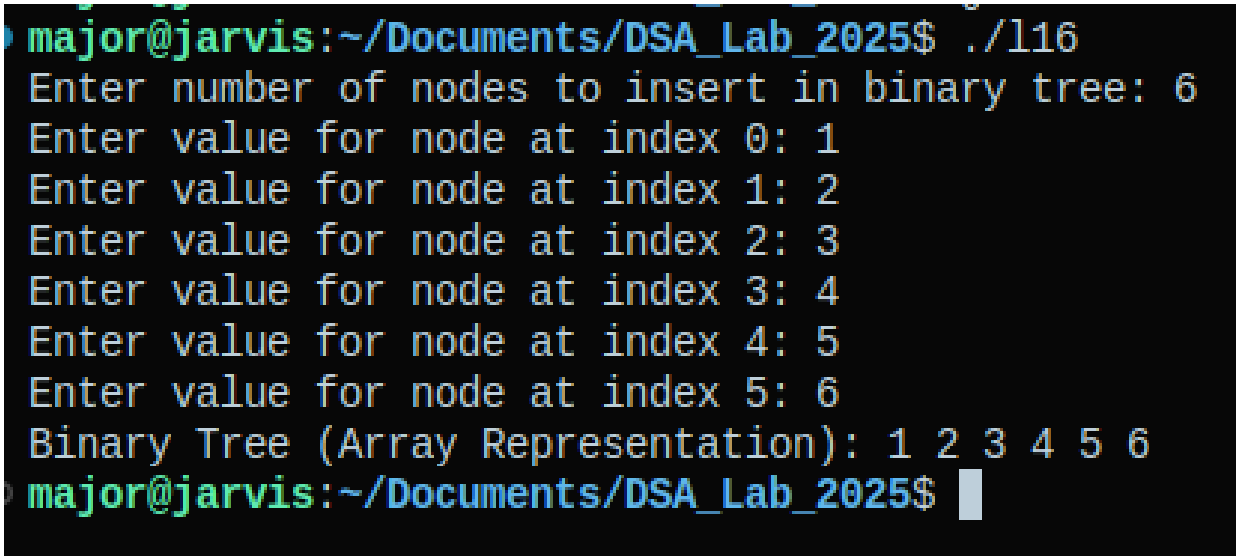
**Program 16:** Program to create a binary tree using array representation.

**Solution:** // Author: [Vikas Singh 241504]

```
#include <iostream>
using namespace std;
// Maximum size of array for tree
const int MAX = 100;
// Insert Node
void insertNode(int tree[], int index, int value){
   if (index >= MAX){
      cout << "Insertion failed! Index out of range.\n";
      return;
   }
   tree[index] = value;
}
// Print Tree
void printTree(int tree[], int size){
   cout << "Binary Tree (Array Representation): ";
   for (int i = 0; i < size; i++){
      cout << tree[i] << " ";
   }
   cout << endl;
}
// Creat Tree
void createTree(int tree[], int size){
   for (int i = 0; i < size; i++){
      int value;
      cout << "Enter value for node at index " << i << ": ";
      cin >> value;
      insertNode(tree, i, value);
   }
}
// Main Function
int main(){
   int tree[MAX];

   for (int i = 0; i < MAX; i++){
      tree[i] = -1;
  }
```

```
    int n;
    cout << "Enter number of nodes to insert in binary tree: ";
    cin >> n;

    createTree(tree, n);
    printTree(tree, n);

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab16.cpp

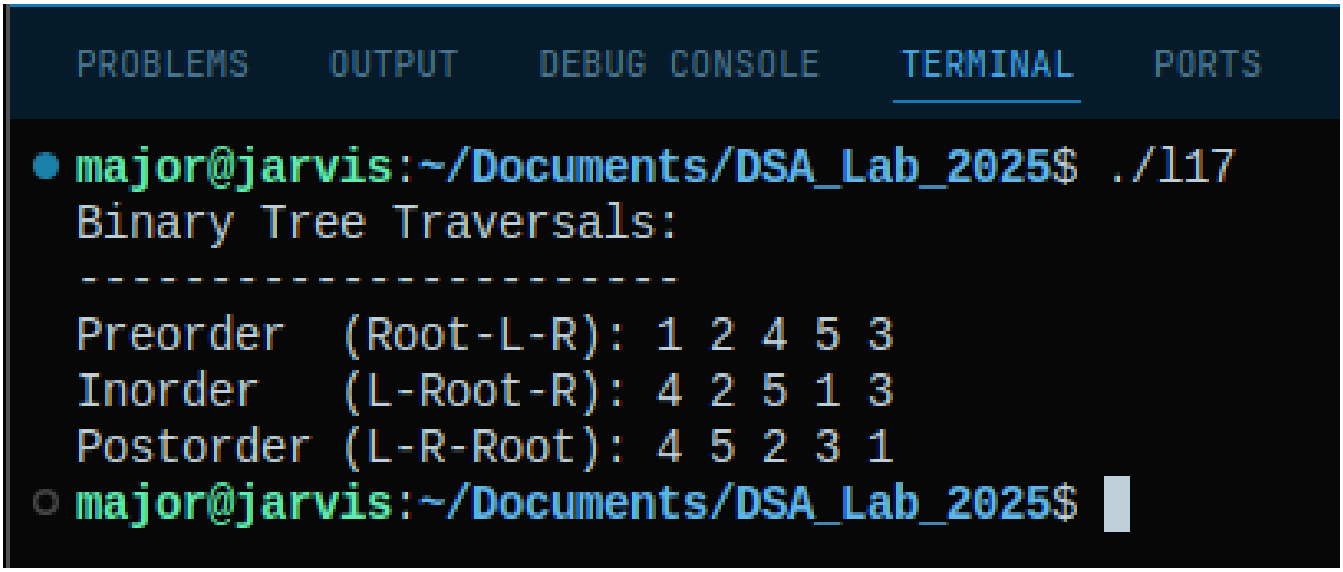
**Program 17:** Program to perform preorder, inorder, and postorder traversal of a binary tree.

**Solution:** // Author: [Vikas Singh 241504]

```
#include <iostream>
using namespace std;
// Node Structure
struct Node{
    int data;
    Node *left;
    Node *right;
};
// Create Node Function
Node *createNode(int value){
    Node *newNode = new Node();
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
// PreOrder Traversal Function
void preorder(Node *root){
    if (root == NULL) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}
// Inorder Traversal Function
void inorder(Node *root){
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}
// Postorder Traversal Function
void postorder(Node *root){
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
```

```
}
// Main Function
int main(){
    Node *root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    cout << "Binary Tree Traversals:" << endl;
    cout << "-----------------------" << endl;
    cout << "Preorder  (Root-L-R): ";
    preorder(root);
    cout << endl;
    cout << "Inorder   (L-Root-R): ";
    inorder(root);
    cout << endl;
    cout << "Postorder (L-R-Root): ";
    postorder(root);
    cout << endl;
    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab17.cpp

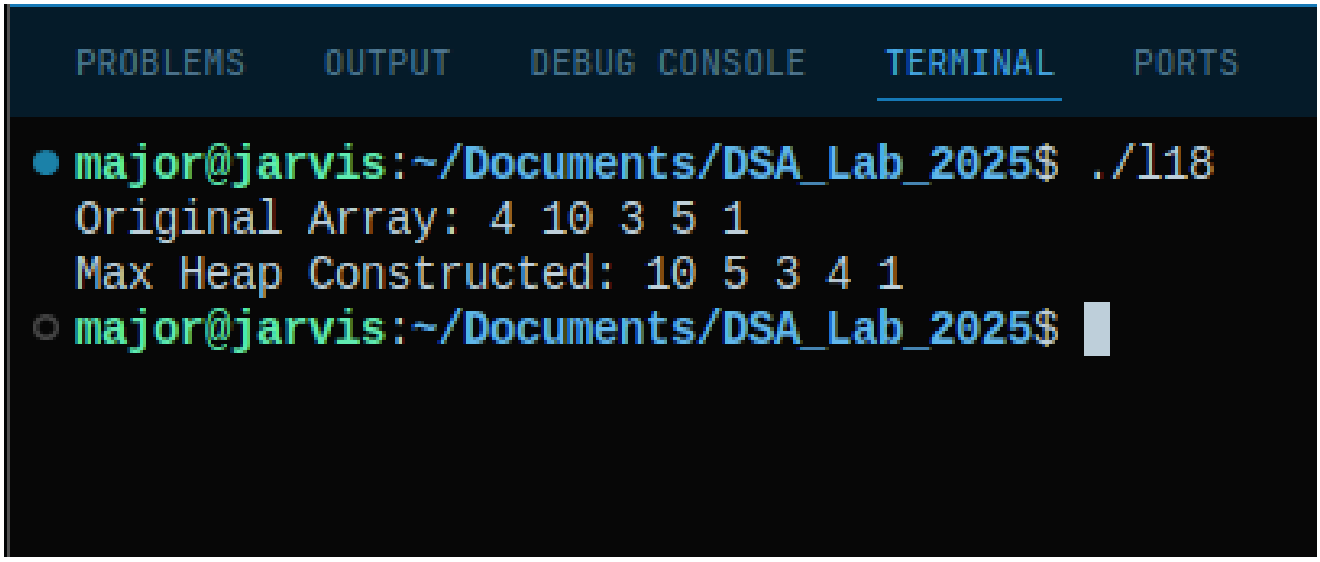
**Program 18:** Program to build a max heap and perform heapify operation.

**Solution:** // Author: [Vikas Singh 241504]

```
#include <iostream>
using namespace std;
// Heapify function
void heapify(int arr[], int n, int i){
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]){
        largest = left;
    }
    if (right < n && arr[right] > arr[largest]){
        largest = right;
    }
    if (largest != i){
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
// Max heap from Array
void buildMaxHeap(int arr[], int n){
    for (int i = n / 2 - 1; i >= 0; i--){
        heapify(arr, n, i);
    }
}
// Print Array function
void printArray(int arr[], int n){
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}
// Main Function
int main(){
    int arr[] = {4, 10, 3, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << "Original Array: ";
    printArray(arr, n);

    buildMaxHeap(arr, n);

    cout << "Max Heap Constructed: ";
    printArray(arr, n);

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab18.cpp

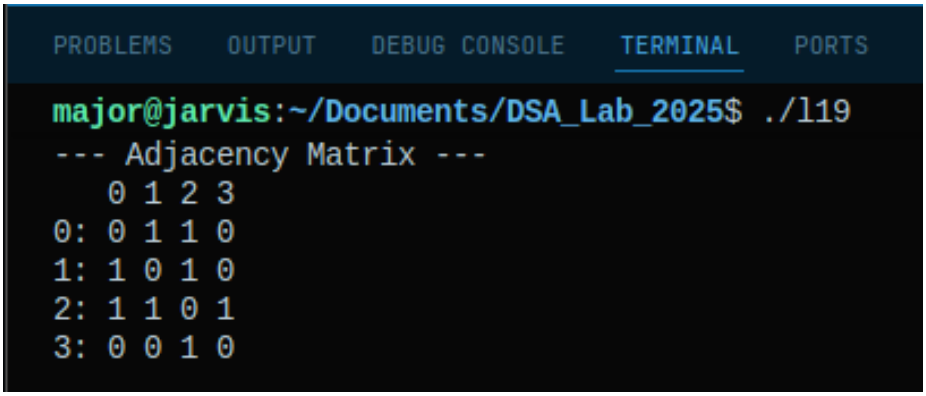
**Program 19:** Program to represent a graph using adjacency matrix.

**Solution:** // Author: [Vikas Singh 241504]

```
#include <iostream>
using namespace std;
// Maximum number of vertices
#define MAX 20
// Graph class
class Graph{
private:
   int adjMatrix[MAX][MAX];
   int numVertices;
public:
   // Constructor
   Graph(int vertices){
      numVertices = vertices;
      for (int i = 0; i < numVertices; i++){
         for (int j = 0; j < numVertices; j++){
            adjMatrix[i][j] = 0;
         }
      }
   }
   // Function to add an edge
   void addEdge(int i, int j){
      if (i >= numVertices || j >= numVertices || i < 0 || j < 0){
         cout << "Invalid Vertex Index!" << endl;
         return;
      }
      adjMatrix[i][j] = 1;
      adjMatrix[j][i] = 1;
   }
   // Function to remove an edge
   void removeEdge(int i, int j){
      if (i >= numVertices || j >= numVertices || i < 0 || j < 0)
         return;
      adjMatrix[i][j] = 0;
      adjMatrix[j][i] = 0;
   }
   // Function to display the matrix
```

```
    void display(){
        cout << "\n--- Adjacency Matrix ---" << endl;
        cout << "   ";
        for (int i = 0; i < numVertices; i++)
            cout << i << " ";
        cout << endl;
        for (int i = 0; i < numVertices; i++){
            cout << i << ": ";
            for (int j = 0; j < numVertices; j++){
                cout << adjMatrix[i][j] << " ";
            }
            cout << endl;
        }
    }
};
// Main Function
int main(){
    int v = 4;
    Graph g(v);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 3);
    g.display();
    cout << "\nRemoving edge between 1 and 2..." << endl;
    g.removeEdge(1, 2);
    g.display();
    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab19.cpp

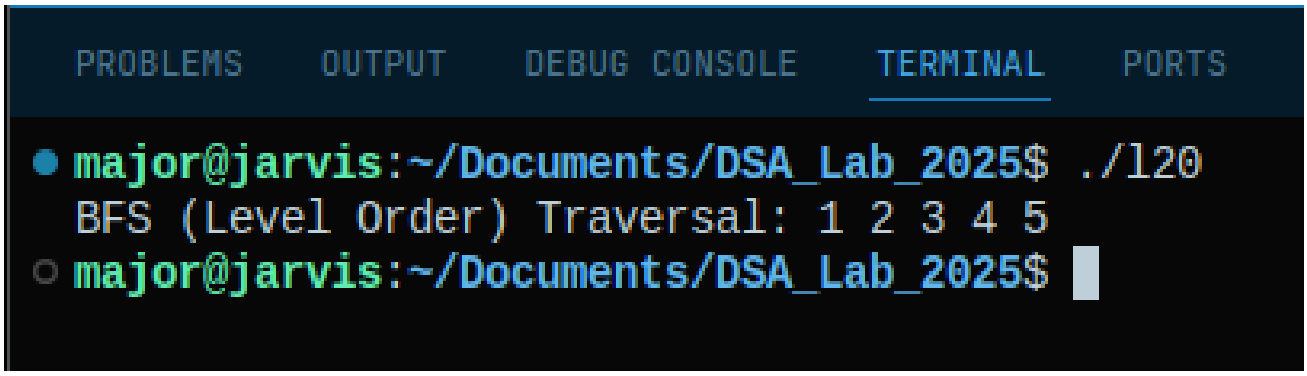
**Program 20:** Program to perform Breadth-First Search (BFS) traversal of a graph.

**Solution:** // Author: [Vikas Singh 241504]

```
#include <iostream>
#include <queue>
using namespace std;
// Node Structure
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};
// BFS Function
void bfsTraversal(Node* root) {
    if (root == nullptr) {
        return;
    }
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        Node* current = q.front();
        q.pop();
        cout << current->data << " ";
        if (current->left != nullptr) {
            q.push(current->left);
        }
        if (current->right != nullptr) {
            q.push(current->right);
        }
    }
}
// Main Function
int main() {
```

```
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);

    cout << "BFS (Level Order) Traversal: ";
    bfsTraversal(root);
    cout << endl;

    return 0;
}
```

**Output:**




**Github Link:** https://github.com/Vikas-Singh-0897/DSA_Practical_2025/blob/main/Lab20.cpp