Central university of Haryana

Department of Computer Science & Engineering

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

Data Structures and Algorithms Lab

(BT CS 301)

Submitted by :-

Name: Sumit Dhakad

Roll No: 241500

Submitted to :-

Mr. Anant R. Bara

(Assistant Professor)

Department of CSE

INDEX

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

int fib(int n) {
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}

int main() {
    int n;
    cout << "Enter number of terms: ";
    cin >> n;

    cout << "Fibonacci Series up to " << n << " terms: ";

    for (int i = 0; i < n; i++) {
        cout << fib(i) << " ";
    }

    return 0;
}
```

OUTPUT:

```
Enter number of terms: 7
Fibonacci Series up to 7 terms: 0 1 1 2 3 5 8
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/01.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

int factorial(int n) {
    if (n == 0 || n == 1)
        return 1;                 // base case
    return n * factorial(n - 1); // recursive call
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    cout << "Factorial of " << n << " = " << factorial(n) << endl;

    return 0;
}
```

OUTPUT:

```
Enter a number: 5
Factorial of 5 = 120
```

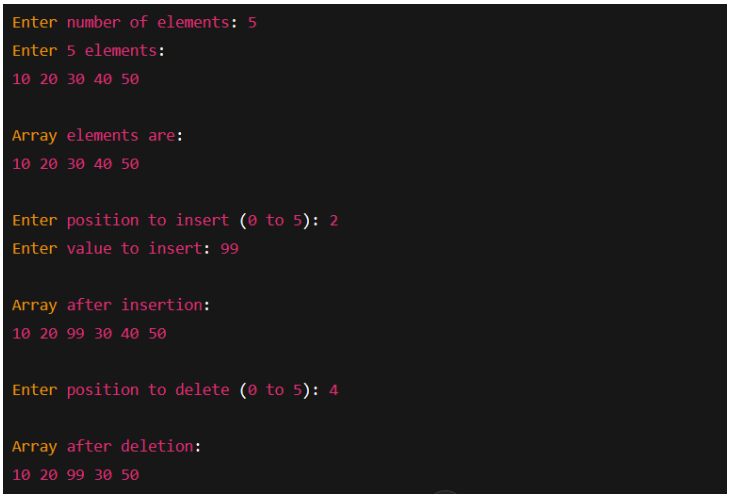
https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/02.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

int main() {
    int arr[100], n, pos, value;

    cout << "Enter number of elements: ";
    cin >> n;

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

    // -------- Traversing --------
    cout << "\nArray elements are:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    // -------- Insertion --------
    cout << "\nEnter position to insert (0 to " << n << "): ";
    cin >> pos;
    cout << "Enter value to insert: ";
    cin >> value;

    for (int i = n; i > pos; i--)
        arr[i] = arr[i - 1];
```

```
    arr[pos] = value;
    n++;

    cout << "\nArray after insertion:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    // -------- Deletion --------
    cout << "\nEnter position to delete (0 to " << n - 1 << "): ";
    cin >> pos;

    for (int i = pos; i < n - 1; i++)
        arr[i] = arr[i + 1];

    n--;

    cout << "\nArray after deletion:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

OUTPUT:



https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/03.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;   // middle index

        if (arr[mid] == key)
            return mid;              // element found
        else if (arr[mid] < key)
            low = mid + 1;           // search right half
        else
            high = mid - 1;          // search left half
    }
    return -1;                       // not found
}

int main() {
    int n, key;
    int arr[100];

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter " << n << " sorted elements:\n";
    for (int i = 0; i < n; i++)
```

```
        cin >> arr[i];

    cout << "Enter element to search: ";
    cin >> key;

    int result = binarySearch(arr, n, key);

    if (result != -1)
        cout << "Element found at index: " << result << endl;
    else
        cout << "Element not found." << endl;

    return 0;
}
```

OUTPUT:

```
Enter 6 sorted elements:
2 5 8 12 15 20
Enter element to search: 12
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/04.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

int main() {
    int arr[100], n;

    cout << "Enter number of elements: ";
    cin >> n;

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

    cout << "\n--- Bubble Sort Process ---\n";

    for (int i = 0; i < n - 1; i++) {
        cout << "Pass " << i + 1 << ": ";

        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // swap
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
```

```
        // display array after each pass
        for (int k = 0; k < n; k++)
            cout << arr[k] << " ";
        cout << endl;
    }

    cout << "\nSorted Array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

OUTPUT:

```
Enter number of elements: 5
Enter 5 elements:
5 3 8 4 2

--- Bubble Sort Process ---
Pass 1: 3 5 4 2 8
Pass 2: 3 4 2 5 8
Pass 3: 3 2 4 5 8
Pass 4: 2 3 4 5 8

Sorted Array: 2 3 4 5 8
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/05.CPP

Program 6: Program to implement Insertion Sort.

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

int main() {
    int arr[100], n;

    cout << "Enter number of elements: ";
    cin >> n;

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

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

        // Move elements greater than key
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }

    cout << "\nSorted array:\n";
    for (int i = 0; i < n; i++)
```

```
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

OUTPUT:

```
Enter number of elements: 5
Enter 5 elements:
9 5 1 4 3

Sorted array:
1 3 4 5 9
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/06.CPP

Program 7: Program to implement Merge Sort using recursion.

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

// Function to merge two halves
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    // Copy data
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;

    // Merge back to array
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
```

```
    }

    // Copy remaining elements
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Recursive Merge Sort
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);        // Sort left half
        mergeSort(arr, mid + 1, right);   // Sort right half
        merge(arr, left, mid, right);     // Merge them
    }
}

int main() {
    int n, arr[100];

    cout << "Enter number of elements: ";
```

```
    cin >> n;

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

    mergeSort(arr, 0, n - 1);

    cout << "\nSorted array:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

OUTPUT:

```
Enter number of elements: 6
Enter 6 elements:
8 3 1 7 0 10

Sorted array:
0 1 3 7 8 10
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/07.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

int main() {
    int base, size, rows, cols;
    int i, j;

    cout << "Enter base address: ";
    cin >> base;

    cout << "Enter size of each element (in bytes): ";
    cin >> size;

    cout << "Enter number of rows: ";
    cin >> rows;

    cout << "Enter number of columns: ";
    cin >> cols;

    cout << "Enter index (i j) of element A[i][j]: ";
    cin >> i >> j;

    // Assuming lower bound is 0 for both dimensions
    int L1 = 0, L2 = 0;

    // Row-major address
    int rowMajor = base + ((i - L1) * cols + (j - L2)) * size;
```

```
    // Column-major address
    int colMajor = base + ((j - L2) * rows + (i - L1)) * size;

    cout << "\nAddress using Row-major formula: " << rowMajor;
    cout << "\nAddress using Column-major formula: " << colMajor << endl;

    return 0;
}
```

OUTPUT:

```
Enter base address: 1000
Enter size of each element (in bytes): 4
Enter number of rows: 3
Enter number of columns: 4
Enter index (i j) of element A[i][j]: 2 1

Address using Row-major formula: 1024
Address using Column-major formula: 1028
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/08.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

class Stack {
    int arr[5];  // fixed size array
    int topIndex; // to track the top element
    int capacity; // maximum size of stack

public:
    Stack() {
        topIndex = -1;
        capacity = 5;  // you can change this as needed
    }

    // push an element onto stack
    void push(int val) {
        if (full()) {
            cout << "Stack Overflow! Cannot push " << val << endl;
            return;
        }
        topIndex++;
        arr[topIndex] = val;
    }

    // pop and print the top element
    void pop() {
        if (empty()) {
```

```
            cout << "\nStack Underflow!" << endl;
            return;
        }
        cout << arr[topIndex] << " " ;
        topIndex--;
    }

    // return top element
    int top() {
        if (empty()) {
            cout << "Stack is empty!" << endl;
            return -1;
        }
        return arr[topIndex];
    }

    // check if stack is empty
    bool empty() {
        return topIndex == -1;
    }

    // check if stack is full
    bool full() {
        return topIndex == capacity - 1;
    }
};

int main() {
    Stack s;
    s.push(10);
```

```
    s.push(20);
    s.push(30);
    s.push(40);
    s.push(50);
    s.push(60);  // should trigger "Stack Overflow"

    while (!s.empty()) {
        s.pop();
    }
    s.pop();  // should trigger "Stack Underflow"

    return 0;
}
```

OUTPUT:

```
Stack Overflow! Cannot push 60
50 40 30 20 10
Stack Underflow!

[Done] exited with code=0 in 1.839 seconds
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/09.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

// Node structure
struct Node {
    int data;
    Node* next;
};
// Stack class using linked list
class Stack {
private:
    Node* top;
public:
    Stack() {
        top = NULL;
    }

    // Push operation
    void push(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        cout << value << " pushed to stack.\n";
    }

    // Pop operation
```

```
void pop() {
    if (top == NULL) {
        cout << "Stack Underflow! Nothing to pop.\n";
        return;
    }
    Node* temp = top;
    cout << temp->data << " popped from stack.\n";
    top = top->next;
    delete temp;
}

// Peek operation
void peek() {
    if (top == NULL) {
        cout << "Stack is empty.\n";
        return;
    }
    cout << "Top element: " << top->data << endl;
}

// Display operation
void display() {
    if (top == NULL) {
        cout << "Stack is empty.\n";
        return;
    }
    cout << "Stack (top -> bottom): ";
    Node* temp = top;
    while (temp != NULL) {
        cout << temp->data << " ";
```

```
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    Stack st;
    int choice, value;

    do {
        cout << "\n--- Stack using Linked List ---\n";
        cout << "1. Push\n2. Pop\n3. Peek\n4. Display\n5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

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

            case 2:
                st.pop();
                break;

            case 3:
                st.peek();
                break;
```

```
            case 4:
                st.display();
                break;

            case 5:
                cout << "Exiting...\n";
                break;

            default:
                cout << "Invalid choice! Try again.\n";
        }

    } while (choice != 5);

    return 0;
}
```

OUTPUT:

```
--- Stack using Linked List ---
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 10
10 pushed to stack.

Enter your choice: 1
Enter value to push: 20
20 pushed to stack.

Enter your choice: 1
Enter value to push: 30
30 pushed to stack.

Enter your choice: 4
Stack (top -> bottom): 30 20 10
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/10.CPP

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

```
//Author – Sumit Dhakad - 241500
#include<iostream>
using namespace std;

class Queue {
    int arr[5];
    int front, rear, size;

public:
    Queue() {
        front = -1;
        rear = -1;
        size = 5; // queue capacity
    }
    void enqueue(int val) {
        if (rear == size - 1) {
            cout << "\nQueue Overflow! Cannot insert " << val << endl;
            return;
        }

        if (front == -1) front = 0;
        rear++;
        arr[rear] = val;
        cout << val << " inserted into queue." << endl;
    }

    void dequeue() {
        if (Isempty()) {
```

```
        cout << "\nQueue Underflow! No element to delete." << endl;
        return;
    }

    cout << "Deleted element: " << arr[front] << endl;

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

int peek() {
    if (Isempty()) {
        cout << "\nQueue is empty! No front element." << endl;
        return 0;
    }
    cout << "Front element: " << arr[front] << endl;
    return arr[front];
}

bool Isempty() {
    return (front == -1);
}

void display() {
    if (Isempty()) {
        cout << "\nQueue is empty!" << endl;
        return;
```

```
        }

        cout << "\nQueue elements: ";
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Queue Q1;
    Q1.enqueue(10);
    Q1.enqueue(20);
    Q1.enqueue(30);
    Q1.enqueue(40);
    Q1.enqueue(50);
    Q1.enqueue(60); // will cause overflow

    Q1.display();

    Q1.peek();

    cout << "\nAfter Deletion:" << endl;
    Q1.dequeue();
    Q1.dequeue();
    Q1.display();

    Q1.dequeue();
    Q1.dequeue();
```

```
    Q1.dequeue();
    Q1.dequeue(); // underflow
}
```

OUTPUT:

```
10 inserted into queue.
20 inserted into queue.
30 inserted into queue.
40 inserted into queue.
50 inserted into queue.

Queue Overflow! Cannot insert 60

Queue elements: 10 20 30 40 50
Front element: 10

After Deletion:
Deleted element: 10
Deleted element: 20

Queue elements: 30 40 50
Deleted element: 30
Deleted element: 40
Deleted element: 50

Queue Underflow! No element to delete.
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/11.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

// Node implemented as a class
class Node {
public:
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

// Queue implemented using linked list (classes only)
class Queue {
private:
    Node* front;
    Node* rear;

public:
    Queue() : front(nullptr), rear(nullptr) {}

    ~Queue() {
        // free all nodes
        while (!isEmpty()) dequeue();
    }

    bool isEmpty() const {
        return front == nullptr;
```

```
}

// Enqueue (insert at rear)
void enqueue(int val) {
    Node* newNode = new Node(val);
    if (rear == nullptr) {      // empty queue
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << val << " enqueued to queue.\n";
}

// Dequeue (remove from front)
int dequeue() {
    if (isEmpty()) {
        cout << "Queue Underflow! Nothing to dequeue.\n";
        return -1; // sentinel
    }
    Node* temp = front;
    int retval = temp->data;
    front = front->next;
    if (front == nullptr) rear = nullptr; // queue became empty
    delete temp;
    cout << retval << " dequeued from queue.\n";
    return retval;
}

// Peek / front element
```

```
    int peek() const {
        if (isEmpty()) {
            cout << "Queue is empty. No front element.\n";
            return -1;
        }
        cout << "Front element: " << front->data << '\n';
        return front->data;
    }

    // Display queue from front to rear
    void display() const {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return;
        }
        cout << "Queue (front -> rear): ";
        Node* temp = front;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next) cout << " ";
            temp = temp->next;
        }
        cout << '\n';
    }
};

int main() {
    // Predefined demonstration (no user input)
    Queue q;
```

```
    // Enqueue some elements
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    // Display current queue
    q.display();

    // Peek front
    q.peek();

    // Dequeue one element
    q.dequeue();

    // Display after dequeue
    q.display();

    // More operations
    q.enqueue(40);
    q.enqueue(50);
    q.display();

    // Dequeue all elements
    while (!q.isEmpty()) q.dequeue();

    // Try dequeue on empty queue (to show underflow handling)
    q.dequeue();

    return 0;
}
```

OUTPUT:

```
10 enqueued to queue.
20 enqueued to queue.
30 enqueued to queue.
Queue (front -> rear): 10 20 30
Front element: 10
10 dequeued from queue.
Queue (front -> rear): 20 30
40 enqueued to queue.
50 enqueued to queue.
Queue (front -> rear): 20 30 40 50
20 dequeued from queue.
30 dequeued from queue.
40 dequeued from queue.
50 dequeued from queue.
Queue Underflow! Nothing to dequeue.
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/12.CPP

Program 13: Program to implement queue using two stacks.

```
//Author – Sumit Dhakad - 241500
#include <iostream>
#include <stack>
using namespace std;

class QueueUsingStacks {
private:
    stack<int> s1, s2;  // Two stacks

public:
    // Enqueue operation
    void enqueue(int value) {
        s1.push(value);
        cout << value << " enqueued.\n";
    }

    // Dequeue operation
    int dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow! Nothing to dequeue.\n";
            return -1;
        }

        // Move all elements from s1 → s2 only if s2 is empty
        if (s2.empty()) {
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
```

```
    }

    int removed = s2.top();
    s2.pop();
    cout << removed << " dequeued.\n";
    return removed;
}

// Peek front element
int peek() {
    if (isEmpty()) {
        cout << "Queue is empty.\n";
        return -1;
    }

    if (s2.empty()) {
        while (!s1.empty()) {
            s2.push(s1.top());
            s1.pop();
        }
    }

    cout << "Front element: " << s2.top() << '\n';
    return s2.top();
}

// Check if queue is empty
bool isEmpty() const {
    return s1.empty() && s2.empty();
}
```

```
// Display queue contents (front → rear)
void display() {
    if (isEmpty()) {
        cout << "Queue is empty.\n";
        return;
    }

    // To display, copy stacks (no modification to original)
    stack<int> tempS1 = s1;
    stack<int> tempS2 = s2;

    cout << "Queue (front → rear): ";

    // First print elements of s2 (already in correct order)
    while (!tempS2.empty()) {
        cout << tempS2.top() << " ";
        tempS2.pop();
    }

    // Now print elements of s1 in correct order
    // Reverse s1 into a temp stack
    stack<int> rev;
    while (!tempS1.empty()) {
        rev.push(tempS1.top());
        tempS1.pop();
    }

    while (!rev.empty()) {
        cout << rev.top() << " ";
```

```
            rev.pop();
        }

        cout << "\n";
    }
};

int main() {
    QueueUsingStacks q;

    // Predefined operations
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    q.display();
    q.peek();

    q.dequeue();
    q.display();

    q.enqueue(40);
    q.enqueue(50);
    q.display();

    q.dequeue();
    q.dequeue();
    q.display();

    q.dequeue();
```

```
    q.dequeue();
    q.dequeue();  // Underflow case

    return 0;
}
```

OUTPUT:

```
10 enqueued.
20 enqueued.
30 enqueued.
Queue (front → rear): 10 20 30
Front element: 10
10 dequeued.
Queue (front → rear): 20 30
40 enqueued.
50 enqueued.
Queue (front → rear): 20 30 40 50
20 dequeued.
30 dequeued.
Queue (front → rear): 40 50
40 dequeued.
50 dequeued.
Queue Underflow! Nothing to dequeue.
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/13.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    // Insert at beginning
    void insertAtBeginning(int value) {
        Node* newNode = new Node(value);
        newNode->next = head;
```

```
        head = newNode;
        cout << value << " inserted at beginning.\n";
    }

    // Insert at end
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);

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

        cout << value << " inserted at end.\n";
    }

    // Insert at a given position (middle)
    void insertAtPosition(int value, int position) {
        Node* newNode = new Node(value);

        // Insert at head if position is 1
        if (position == 1) {
            newNode->next = head;
            head = newNode;
            cout << value << " inserted at position " << position <<
".\n";
            return;
```

```
        }

        Node* temp = head;
        for (int i = 1; i < position - 1 && temp != nullptr; i++)
            temp = temp->next;

        if (temp == nullptr) {
            cout << "Position out of range. Cannot insert " << value <<
".\n";
            return;
        }

        newNode->next = temp->next;
        temp->next = newNode;

        cout << value << " inserted at position " << position << ".\n";
    }

    // Display linked list
    void display() {
        if (head == nullptr) {
            cout << "List is empty.\n";
            return;
        }

        cout << "Linked List: ";
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
```

```
        cout << "\n";
    }
};

int main() {
    LinkedList list;

    // Predefined insertions
    list.insertAtBeginning(30);
    list.insertAtBeginning(20);
    list.insertAtBeginning(10);

    list.display();

    list.insertAtEnd(40);
    list.insertAtEnd(50);

    list.display();

    list.insertAtPosition(25, 3);
    list.insertAtPosition(35, 6);
    list.insertAtPosition(5, 1);   // Also works for beginning

    list.display();

    return 0;
}
```

OUTPUT:

```
10 inserted at beginning.
20 inserted at beginning.
30 inserted at beginning.
Linked List: 30 20 10
40 inserted at end.
50 inserted at end.
Linked List: 30 20 10 40 50
25 inserted at position 3.
35 inserted at position 6.
5 inserted at position 1.
Linked List: 5 30 20 25 10 40 35 50
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/14.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    // Insert at end (helper for demo)
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
```

```
    if (head == nullptr) {
        head = newNode;
        return;
    }

    Node* temp = head;
    while (temp->next != nullptr)
        temp = temp->next;
    temp->next = newNode;
}

// DELETE by VALUE
void deleteByValue(int value) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

    // If node to delete is head
    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Node with value " << value << " deleted.\n";
        return;
    }

    Node* temp = head;
    while (temp->next != nullptr && temp->next->data != value)
        temp = temp->next;
```

```
    if (temp->next == nullptr) {
        cout << "Value " << value << " not found.\n";
        return;
    }

    Node* delNode = temp->next;
    temp->next = delNode->next;
    delete delNode;

    cout << "Node with value " << value << " deleted.\n";
}

// DELETE by POSITION (1-based indexing)
void deleteByPosition(int position) {
    if (head == nullptr || position <= 0) {
        cout << "Invalid position.\n";
        return;
    }

    // Deleting head node
    if (position == 1) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Node at position " << position << " deleted.\n";
        return;
    }

    Node* temp = head;
```

```
    for (int i = 1; i < position - 1 && temp != nullptr; i++)
        temp = temp->next;

    if (temp == nullptr || temp->next == nullptr) {
        cout << "Position " << position << " out of range.\n";
        return;
    }

    Node* delNode = temp->next;
    temp->next = delNode->next;
    delete delNode;

    cout << "Node at position " << position << " deleted.\n";
}

// Display list
void display() {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

    cout << "Linked List: ";
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << "\n";
}
```

```
};

int main() {
    LinkedList list;

    // Predefined List
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtEnd(40);
    list.insertAtEnd(50);

    list.display();

    // Delete by Value
    list.deleteByValue(30);
    list.display();

    list.deleteByValue(10);
    list.display();

    // Delete by Position
    list.deleteByPosition(2);
    list.display();

    list.deleteByPosition(10); // out of range example
    list.display();

    return 0;
}
```

OUTPUT:

```
Linked List: 10 20 30 40 50
Node with value 30 deleted.
Linked List: 10 20 40 50
Node with value 10 deleted.
Linked List: 20 40 50
Node at position 2 deleted.
Linked List: 20 50
Position 10 out of range.
Linked List: 20 50
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/15.CPP

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

```
//Author – Sumit Dhakad - 241500
#include <iostream>
using namespace std;
class BinaryTree {
public:
    int tree[100];
    int n;
    BinaryTree() {
        n = 7;  // number of nodes
        int arr[8] = { -1, 10, 20, 30, 40, 50, 60, 70 };

        for (int i = 1; i <= n; i++) {
            tree[i] = arr[i];
        }
    }
    void display() {
        cout << "Binary tree in array form: ";
        for (int i = 1; i <= n; i++) {
            cout << tree[i] << " ";
        }
        cout << endl;
    }
};
int main() {
    BinaryTree bt;
    bt.display();
    return 0; }
```

OUTPUT:

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/16.CPP


```
Binary tree in array form: 10 20 30 40 50 60 70
```

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

```
//Author – Sumit Dhakad - 241500

#include <iostream>
using namespace std;

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

Node* createNode(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

void preorder(Node* root) {
    if (root == NULL) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}

void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

void postorder(Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}

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 << "Preorder: ";
    preorder(root);
    cout << "\nInorder: ";
    inorder(root);
```

```
    cout << "\nPostorder: ";
    postorder(root);
    return 0;
}
```

OUTPUT:

```
Preorder: 1 2 4 5 3
Inorder: 4 2 5 1 3
Postorder: 4 5 2 3 1
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/17.CPP

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

```
//Author – Sumit Dhakad - 241500

#include <iostream>
using namespace std;

void heapify(int arr[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;

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

void buildMaxHeap(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[100];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    buildMaxHeap(arr, n);

    cout << "Array after building max heap: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

OUTPUT:

```
Enter number of elements: 6
Enter 6 elements: 3 5 9 6 8 20
Array after building max heap: 20 8 9 6 5 3
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/18.CPP

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

```
//Author – Sumit Dhakad - 241500

#include <iostream>
using namespace std;

int main() {
    int n, edges;
    cout << "Enter number of vertices: ";
    cin >> n;
    int adj[10][10] = {0};
    cout << "Enter number of edges: ";
    cin >> edges;
    cout << "Enter edges (u v):" << endl;
    for (int k = 0; k < edges; k++) {
        int u, v;
        cin >> u >> v;
        adj[u][v] = 1;
        adj[v][u] = 1; // for undirected graph
    }

    cout << "Adjacency Matrix:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << adj[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:

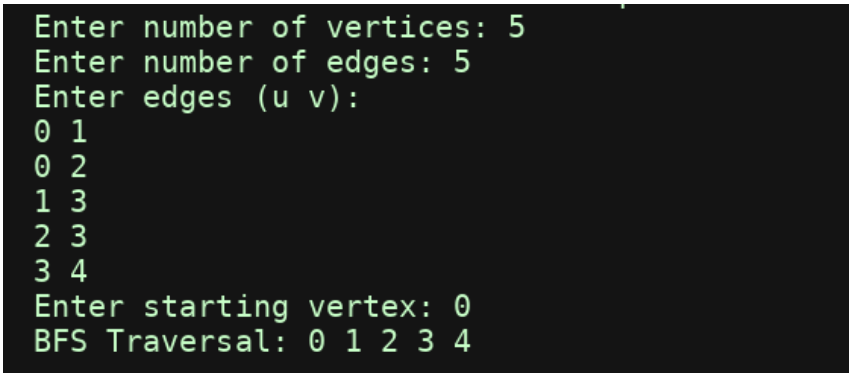
```
Enter number of vertices: 4
Enter number of edges: 4
Enter edges (u v):
0 1
0 2
1 2
2 3
Adjacency Matrix:
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
```

https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/19.CPP

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

```
//Author – Sumit Dhakad - 241500

#include <iostream>
#include <queue>
using namespace std;

int main() {
    int n, edges;
    cout << "Enter number of vertices: ";
    cin >> n;
    int adj[10][10] = {0};
    cout << "Enter number of edges: ";
    cin >> edges;
    cout << "Enter edges (u v):" << endl;
    for (int k = 0; k < edges; k++) {
        int u, v;
        cin >> u >> v;
        adj[u][v] = 1;
        adj[v][u] = 1; // undirected
    }

    int start;
    cout << "Enter starting vertex: ";
    cin >> start;

    bool visited[10] = {false};
    queue<int> q;
    visited[start] = true;
    q.push(start);

    cout << "BFS Traversal: ";
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";
        for (int i = 0; i < n; i++) {
            if (adj[v][i] == 1 && !visited[i]) {
                visited[i] = true;
                q.push(i);
            }
        }
    }
    return 0;
}
```

OUTPUT:



https://github.com/sumit-dhakadd/DSA_PROGRAMS/blob/main/20.CPP

COMPLETE GIT REPOSITORY LINK

GitHub Link - https://github.com/sumit-dhakadd/DSA_PROGRAMS.git