Subject Code: LPCIT-101

Subject Name: Data Structures Laboratory

Programme: B.Tech. (IT)	**L: 0 T: 0 P: 2**
Semester: 3	**Teaching Hours: 26**
Theory/Practical: Practical	**Credits: 2**
Internal Marks: 30	**Percentage of Numerical/Design Problems: 100%**
External Marks: 20	**Duration of End Semester Exam(ESE): 1.5 hr**
Total Marks: 50	**Elective Status: Compulsory**

Name of Practicals:

1. Program to insert a new element at end as well as at a given position in an array.
2. Program to delete an element from a given whose value is given or whose position is given.
3. Program to find the location of a given element using Linear Search
4. Program to find the location of a given element using Binary Search
5. Program to implement push and pop operations on a stack using linear array.
6. Program to convert an infix expression to a postfix expression using stacks.
7. Program to evaluate a postfix expression using stacks.
8. Implement recursive function for Tower of Hanoi problem.
9. Program to implement insertion and deletion operations in a queue using linear array.
10. Program to implement linked list.
11. Program to implement push and pop operations on a stack using linked list.
12. Program to implement push and pop operations on a queue using linked list.
13. Program to sort an array of integers in ascending order using bubble sort.

14 Program to sort an array of integers in ascending order using selection sort.

15 Program to sort an array of integers in ascending order using insertion sort.

16 Program to sort an array of integers in ascending order using quick sort

17. Program to traverse a Binary search tree in Pre-order, In-order and Post-order.

18 Program to traverse graphs using BFS.

19. Program to traverse graphs using DFS

Experiment No.: 1

Program to insert a new element at end as well as at a given position in an array.

Code:

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

// Function to insert an element at the end of the array
void insertAtEnd(int arr[], int &n, int capacity, int element) {
    if (n < capacity) {
        arr[n] = element;
        n++;
        cout << "Element inserted at the end successfully.\n";
    } else {
        cout << "Array is full, cannot insert at the end.\n";
    }
}

// Function to insert an element at a given position in the array
void insertAtPosition(int arr[], int &n, int capacity, int
element, int position) {
    if (position < 0 || position > n || n >= capacity) {
        cout << "Invalid position or array is full.\n";
        return;
    }
    for (int i = n; i > position; i--) {
        arr[i] = arr[i - 1];
    }
    arr[position] = element;
    n++;
    cout << "Element inserted at position " << position << "
successfully.\n";
}
```

```
// Function to display the array
void displayArray(int arr[], int n) {
    cout << "Array elements: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int capacity = 10, n = 5;
    int arr[capacity] = {1, 2, 3, 4, 5};

    cout << "Initial ";
    displayArray(arr, n);

    // Insert at end
    insertAtEnd(arr, n, capacity, 6);
    displayArray(arr, n);

    // Insert at given position
    insertAtPosition(arr, n, capacity, 99, 2);
    displayArray(arr, n);

    return 0;
}
```

Output:

```
Initial Array elements: 1 2 3 4 5
Element inserted at the end successfully.
Array elements: 1 2 3 4 5 6
Element inserted at position 2 successfully.
Array elements: 1 2 99 3 4 5 6
```

Experiment No.: 2

Program to delete an element from a given whose value is given or whose position is given.

Code:

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

// Function to delete an element by value
void deleteByValue(int arr[], int &n, int value) {
    int pos = -1;
    for (int i = 0; i < n; i++) {
        if (arr[i] == value) {
            pos = i;
            break;
        }
    }
    if (pos == -1) {
        cout << "Element not found.\n";
        return;
    }
    for (int i = pos; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }
    n--;
    cout << "Element " << value << " deleted successfully.\n";
}

// Function to delete an element by position
void deleteByPosition(int arr[], int &n, int position) {
    if (position < 0 || position >= n) {
        cout << "Invalid position.\n";
```

```
        return;
    }
    for (int i = position; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }
    n--;
    cout << "Element at position " << position << " deleted
successfully.\n";
}

// Function to display the array
void displayArray(int arr[], int n) {
    cout << "Array elements: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[10] = {10, 20, 30, 40, 50};
    int n = 5;

    cout << "Initial ";
    displayArray(arr, n);

    // Delete by value
    deleteByValue(arr, n, 30);
    displayArray(arr, n);

    // Delete by position
    deleteByPosition(arr, n, 2);
    displayArray(arr, n);

    return 0;
```

```
}
```

Output:

```
Initial Array elements: 10 20 30 40 50
Element 30 deleted successfully.
Array elements: 10 20 40 50
Element at position 2 deleted successfully.
Array elements: 10 20 50
```

Experiment No.: 3

Program to find the location of a given element using Linear Search

Code:

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

// Function to perform Linear Search
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i; // Return the index of the found element
        }
    }
    return -1; // Element not found
}

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key;

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

    int position = linearSearch(arr, n, key);

    if (position != -1) {
        cout << "Element " << key << " found at index " <<
position << ".\n";
    } else {
        cout << "Element " << key << " not found in the array.\n";
    }
```

```
    return 0;
}
```

Output 1:

```
Enter the element to search: 30
Element 30 found at index 2.
```

Output 2:

```
Enter the element to search: 100
Element 100 not found in the array.
```

Experiment No.: 4

Program to find the location of a given element using Binary Search

Code:

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

// Function to perform Binary Search
int binarySearch(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == key)
            return mid; // Element found at index mid
        else if (arr[mid] < key)
            left = mid + 1; // Search in the right half
        else
            right = mid - 1; // Search in the left half
    }
    return -1; // Element not found
}

int main() {
    int arr[] = {10, 20, 30, 40, 50}; // Sorted array
    int n = sizeof(arr) / sizeof(arr[0]);
    int key;

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

    int position = binarySearch(arr, 0, n - 1, key);
```

```
    if (position != -1) {
        cout << "Element " << key << " found at index " <<
position << ".\n";
    } else {
        cout << "Element " << key << " not found in the array.\n";
    }

    return 0;
}
```

Output 1:

```
Enter the element to search: 40
Element 40 found at index 3.
```

Output 2:

```
Enter the element to search: 25
Element 25 not found in the array.
```

Experiment No.: 5

Program to implement push and pop operations on a stack using linear array.

Code:

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

#define MAX 5 // Maximum size of the stack

class Stack {
private:
    int arr[MAX]; // Stack array
    int top; // Top pointer

public:
    Stack() { top = -1; } // Constructor to initialize top

    // Push operation
    void push(int value) {
        if (top >= MAX - 1) {
            cout << "Stack Overflow! Cannot push " << value << ".
\n";
            return;
        }
        arr[++top] = value;
        cout << "Pushed " << value << " into the stack.\n";
    }

    // Pop operation
    void pop() {
        if (top < 0) {
            cout << "Stack Underflow! Cannot pop.\n";
            return;
```

```
        }
        cout << "Popped " << arr[top] << " from the stack.\n";
        top--;
    }

    // Display stack elements
    void display() {
        if (top < 0) {
            cout << "Stack is empty.\n";
            return;
        }
        cout << "Stack elements: ";
        for (int i = top; i >= 0; i--) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.display();

    s.pop();
    s.display();

    s.pop();
    s.pop();
    s.pop(); // Trying to pop from an empty stack

    return 0;
}
```

Output:

```
Pushed 10 into the stack.
Pushed 20 into the stack.
Pushed 30 into the stack.
Stack elements: 30 20 10
Popped 30 from the stack.
Stack elements: 20 10
Popped 20 from the stack.
Popped 10 from the stack.
Stack Underflow! Cannot pop.
```

Experiment No.: 6

Program to convert an infix expression to a postfix expression using stacks.

Code:

```
#include <iostream>
#include <stack>
using namespace std;

// Function to return precedence of operators
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return 0;
}

// Function to check if character is an operand
bool isOperand(char ch) {
    return (ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9');
}

// Function to convert infix to postfix
string infixToPostfix(string infix) {
    stack<char> st;
    string postfix = "";

    for (char ch : infix) {
        if (isOperand(ch)) {
            postfix += ch; // Append operand to result
        } else if (ch == '(') {
            st.push(ch); // Push '(' to stack
        } else if (ch == ')') {
```

```
            while (!st.empty() && st.top() != '(') {
                postfix += st.top();
                st.pop();
            }
            st.pop(); // Remove '(' from stack
        } else { // Operator encountered
            while (!st.empty() && precedence(st.top()) >=
precedence(ch)) {
                postfix += st.top();
                st.pop();
            }
            st.push(ch);
        }
    }

    // Pop all remaining operators from stack
    while (!st.empty()) {
        postfix += st.top();
        st.pop();
    }

    return postfix;
}

int main() {
    string infix;
    cout << "Enter an infix expression: ";
    cin >> infix;

    string postfix = infixToPostfix(infix);
    cout << "Postfix expression: " << postfix << endl;

    return 0;
}
```

Output:

```
Enter an infix expression: A+B*C
Postfix expression: ABC*+

Enter an infix expression: (A+B)*C
Postfix expression: AB+C*
```

Experiment No.: 7

Program to evaluate a postfix expression using stacks.

Code:

```
#include <iostream>
#include <stack>
#include <cmath>
using namespace std;

// Function to evaluate a postfix expression
int evaluatePostfix(string postfix) {
    stack<int> st;

    for (char ch : postfix) {
        if (isdigit(ch)) {
            st.push(ch - '0'); // Convert char to integer and push
        } else {
            // Pop two operands from stack
            int val2 = st.top(); st.pop();
            int val1 = st.top(); st.pop();

            // Perform operation
            switch (ch) {
                case '+': st.push(val1 + val2); break;
                case '-': st.push(val1 - val2); break;
                case '*': st.push(val1 * val2); break;
                case '/': st.push(val1 / val2); break;
                case '^': st.push(pow(val1, val2)); break;
            }
        }
    }
    return st.top(); // Final result
}
```

```
int main() {
    string postfix;
    cout << "Enter a postfix expression: ";
    cin >> postfix;

    int result = evaluatePostfix(postfix);
    cout << "Evaluated result: " << result << endl;

    return 0;
}
```

Output:

```
Enter a postfix expression: 53+2*
Evaluated result: 16

Enter a postfix expression: 82/4+
Evaluated result: 8
```

Experiment No.: 8

Implement recursive function for Tower of Hanoi problem.

Code:

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


// Recursive function for Tower of Hanoi
void towerOfHanoi(int n, char source, char auxiliary, char
destination) {
    if (n == 1) {
        cout << "Move disk 1 from " << source << " to " <<
destination << endl;
        return;
    }
    towerOfHanoi(n - 1, source, destination, auxiliary); // Move
n-1 disks to auxiliary peg
    cout << "Move disk " << n << " from " << source << " to " <<
destination << endl;
    towerOfHanoi(n - 1, auxiliary, source, destination); // Move
n-1 disks from auxiliary to destination
}


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


    towerOfHanoi(n, 'A', 'B', 'C'); // A = source, B = auxiliary,
C = destination


    return 0;
}
```

Output:

```
Enter number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Experiment No.: 9

Program to implement insertion and deletion operations in a queue using linear array.

Code:

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

#define MAX 5 // Maximum size of the queue

class Queue {
private:
    int arr[MAX];
    int front, rear;

public:
    Queue() {
        front = -1;
        rear = -1;
    }

    // Check if the queue is empty
    bool isEmpty() {
        return (front == -1 || front > rear);
    }

    // Check if the queue is full
    bool isFull() {
        return (rear == MAX - 1);
    }

    // Insert element into the queue (Enqueue)
    void enqueue(int value) {
```

```
        if (isFull()) {
            cout << "Queue Overflow! Cannot insert " << value <<
".\n";
            return;
        }
        if (front == -1) front = 0; // Initialize front
        arr[++rear] = value;
        cout << "Inserted " << value << " into the queue.\n";
    }

    // Delete element from the queue (Dequeue)
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow! Cannot delete.\n";
            return;
        }
        cout << "Deleted " << arr[front] << " from the queue.\n";
        front++; // Move front forward

        // Reset queue if it becomes empty
        if (front > rear) {
            front = -1;
            rear = -1;
        }
    }

    // Display queue elements
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return;
        }
        cout << "Queue elements: ";
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
```

```
        cout << endl;
    }
};

int main() {
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.display();

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

    q.dequeue();
    q.dequeue();
    q.dequeue(); // Trying to dequeue from an empty queue

    return 0;
}
```

Output:

```
Inserted 10 into the queue.
Inserted 20 into the queue.
Inserted 30 into the queue.
Queue elements: 10 20 30
Deleted 10 from the queue.
Queue elements: 20 30
Deleted 20 from the queue.
Deleted 30 from the queue.
Queue Underflow! Cannot delete.
```

Experiment No.: 10

Program to implement linked list.

Code:

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

// Node structure for linked list
struct Node {
    int data;
    Node* next;
};

// Class to handle Linked List operations
class LinkedList {
private:
    Node* head;

public:
    LinkedList() { head = nullptr; }

    // Insert a new node at the end
    void insert(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;

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

```
            }
            temp->next = newNode;
        }
        cout << "Inserted " << value << " into the list.\n";
    }

    // Delete a node with a given value
    void remove(int value) {
        if (head == nullptr) {
            cout << "List is empty.\n";
            return;
        }

        Node* temp = head;
        Node* prev = nullptr;

        // If head node holds the key
        if (temp != nullptr && temp->data == value) {
            head = temp->next;
            delete temp;
            cout << "Deleted " << value << " from the list.\n";
            return;
        }

        // Search for the node to delete
        while (temp != nullptr && temp->data != value) {
            prev = temp;
            temp = temp->next;
        }

        if (temp == nullptr) {
            cout << "Element " << value << " not found in the
list.\n";
            return;
        }
```

```
        prev->next = temp->next;
        delete temp;
        cout << "Deleted " << value << " from the list.\n";
    }

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

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

    // Destructor to free memory
    ~LinkedList() {
        Node* temp;
        while (head != nullptr) {
            temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    LinkedList list;
```

```
    list.insert(10);
    list.insert(20);
    list.insert(30);
    list.display();

    list.remove(20);
    list.display();

    list.remove(40); // Attempt to delete non-existing element

    return 0;
}
```

Output:

```
Inserted 10 into the list.
Inserted 20 into the list.
Inserted 30 into the list.
Linked List: 10 -> 20 -> 30 -> NULL
Deleted 20 from the list.
Linked List: 10 -> 30 -> NULL
Element 40 not found in the list.
```

Experiment No.: 11

Program to implement push and pop operations on a stack using linked list.

Code:

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

// Node structure for stack
struct Node {
    int data;
    Node* next;
};

// Stack class using linked list
class Stack {
private:
    Node* top; // Pointer to top of stack

public:
    Stack() { top = nullptr; }

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

    // Pop operation
    void pop() {
```

```
        if (top == nullptr) {
            cout << "Stack Underflow! Cannot pop.\n";
            return;
        }
        Node* temp = top;
        cout << "Popped " << top->data << " from the stack.\n";
        top = top->next;
        delete temp;
    }

    // Display stack elements
    void display() {
        if (top == nullptr) {
            cout << "Stack is empty.\n";
            return;
        }
        Node* temp = top;
        cout << "Stack elements: ";
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    // Destructor to free memory
    ~Stack() {
        while (top != nullptr) {
            Node* temp = top;
            top = top->next;
            delete temp;
        }
    }
};
```

```
int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.display();

    s.pop();
    s.display();

    s.pop();
    s.pop();
    s.pop(); // Trying to pop from an empty stack

    return 0;
}
```

Output:

```
Pushed 10 into the stack.
Pushed 20 into the stack.
Pushed 30 into the stack.
Stack elements: 30 20 10
Popped 30 from the stack.
Stack elements: 20 10
Popped 20 from the stack.
Popped 10 from the stack.
Stack Underflow! Cannot pop.
```

Experiment No.: 12

Program to implement push and pop operations on a queue using linked list.

Code:

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

// Node structure for queue
struct Node {
    int data;
    Node* next;
};

// Queue class using linked list
class Queue {
private:
    Node* front;
    Node* rear;

public:
    Queue() {
        front = nullptr;
        rear = nullptr;
    }

    // Enqueue operation (Insert at rear)
    void enqueue(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;

        if (rear == nullptr) { // If queue is empty
```

```
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << "Inserted " << value << " into the queue.\n";
}

// Dequeue operation (Delete from front)
void dequeue() {
    if (front == nullptr) {
        cout << "Queue Underflow! Cannot delete.\n";
        return;
    }
    Node* temp = front;
    cout << "Deleted " << front->data << " from the queue.\n";
    front = front->next;
    delete temp;

    if (front == nullptr) { // If queue becomes empty
        rear = nullptr;
    }
}

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

```
        }
        cout << endl;
    }

    // Destructor to free memory
    ~Queue() {
        while (front != nullptr) {
            Node* temp = front;
            front = front->next;
            delete temp;
        }
    }
};

int main() {
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.display();

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

    q.dequeue();
    q.dequeue();
    q.dequeue(); // Trying to dequeue from an empty queue

    return 0;
}
```

Output:

```
Inserted 10 into the queue.
Inserted 20 into the queue.
Inserted 30 into the queue.
Queue elements: 10 20 30
Deleted 10 from the queue.
Queue elements: 20 30
Deleted 20 from the queue.
Deleted 30 from the queue.
Queue Underflow! Cannot delete.
```

Experiment No.: 13

Program to sort an array of integers in ascending order using bubble sort.

Code:

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

// Function to perform Bubble Sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j+1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Function to display the array
void display(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << "Original array: ";
    display(arr, n);

    bubbleSort(arr, n);

    cout << "Sorted array: ";
    display(arr, n);

    return 0;
}
```

Output:

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
```

Experiment No.: 14

Program to sort an array of integers in ascending order using selection sort.

Code:

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

// Function to perform Selection Sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i; // Assume the minimum element is at
index i
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j; // Update the minimum element index
            }
        }
        // Swap the found minimum element with the first element
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

// Function to display the array
void display(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
```

```
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    display(arr, n);

    selectionSort(arr, n);

    cout << "Sorted array: ";
    display(arr, n);

    return 0;
}
```

Output:

```
Original array: 64 25 12 22 11
Sorted array: 11 12 22 25 64
```

Experiment No.: 15

Program to sort an array of integers in ascending order using insertion sort

Code:

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

// Function to perform Insertion Sort
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i]; // Current element to be inserted
        int j = i - 1;

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

// Function to display the array
void display(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {29, 10, 14, 37, 13};
```

```
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    display(arr, n);

    insertionSort(arr, n);

    cout << "Sorted array: ";
    display(arr, n);

    return 0;
}
```

Output:

```
Original array: 29 10 14 37 13
Sorted array: 10 13 14 29 37
```

Experiment No.: 16

Program to sort an array of integers in ascending order using quick sort

Code:

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

// Function to partition the array
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Pivot element
    int i = low - 1;       // Index for smaller element

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) { // If current element is smaller than pivot
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]); // Place pivot at correct position
    return (i + 1);
}

// Function to implement Quick Sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high); // Get partition index

        quickSort(arr, low, pi - 1); // Sort left sub-array
        quickSort(arr, pi + 1, high); // Sort right sub-array
    }
}
```

```
// Function to display the array
void display(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    display(arr, n);

    quickSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    display(arr, n);

    return 0;
}
```

Output:

```
Original array: 10 7 8 9 1 5
Sorted array: 1 5 7 8 9 10
```

Experiment No.: 17

Program to traverse a Binary search tree in Pre-order, In-order and Post-order.

Code:

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

// Node structure for BST
struct Node {
    int data;
    Node* left;
    Node* right;
};

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

// Function to insert a node into BST
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }

    if (value < root->data) {
        root->left = insert(root->left, value);
    } else {
        root->right = insert(root->right, value);
```

```
    }

    return root;
}

// Pre-order traversal (Root -> Left -> Right)
void preOrder(Node* root) {
    if (root != nullptr) {
        cout << root->data << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}

// In-order traversal (Left -> Root -> Right)
void inOrder(Node* root) {
    if (root != nullptr) {
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}

// Post-order traversal (Left -> Right -> Root)
void postOrder(Node* root) {
    if (root != nullptr) {
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}

int main() {
    Node* root = nullptr;
    root = insert(root, 50);
```

```
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    cout << "Pre-order traversal: ";
    preOrder(root);
    cout << endl;

    cout << "In-order traversal: ";
    inOrder(root);
    cout << endl;

    cout << "Post-order traversal: ";
    postOrder(root);
    cout << endl;

    return 0;
}
```

Output:

```
Pre-order traversal: 50 30 20 40 70 60 80
In-order traversal: 20 30 40 50 60 70 80
Post-order traversal: 20 40 30 60 80 70 50
```

Experiment No.: 18

Program to traverse graphs using BFS.

Code:

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

// Graph class to represent an adjacency list
class Graph {
    int V; // Number of vertices
    vector<vector<int>> adj; // Adjacency list

public:
    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }

    // Function to add an edge to the graph
    void addEdge(int v, int w) {
        adj[v].push_back(w);
        adj[w].push_back(v); // For an undirected graph
    }

    // Function to perform BFS traversal
    void BFS(int start) {
        vector<bool> visited(V, false);
        queue<int> q;

        // Mark the starting node as visited and enqueue it
        visited[start] = true;
        q.push(start);
```

```
        cout << "BFS Traversal: ";
        while (!q.empty()) {
            int node = q.front();
            cout << node << " ";
            q.pop();

            // Visit all adjacent vertices of the current node
            for (int neighbor : adj[node]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    q.push(neighbor);
                }
            }
        }
        cout << endl;
    }
};

int main() {
    Graph g(6);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);

    int startNode = 0;
    g.BFS(startNode);

    return 0;
}
```

Output:

```
BFS Traversal: 0 1 2 3 4 5
```

Experiment No.: 19

Program to traverse graphs using DFS.

Code:

```
#include <iostream>
#include <vector>
using namespace std;

// Graph class to represent an adjacency list
class Graph {
    int V; // Number of vertices
    vector<vector<int>> adj; // Adjacency list

public:
    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }

    // Function to add an edge to the graph
    void addEdge(int v, int w) {
        adj[v].push_back(w);
        adj[w].push_back(v); // For an undirected graph
    }

    // Recursive function for DFS traversal
    void DFSUtil(int v, vector<bool> &visited) {
        visited[v] = true;
        cout << v << " ";

        // Visit all adjacent vertices of the current node
        for (int neighbor : adj[v]) {
            if (!visited[neighbor]) {
                DFSUtil(neighbor, visited);
```

```
            }
        }
    }

    // Function to perform DFS traversal
    void DFS(int start) {
        vector<bool> visited(V, false);
        cout << "DFS Traversal: ";
        DFSUtil(start, visited);
        cout << endl;
    }
};

int main() {
    Graph g(6);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);

    int startNode = 0;
    g.DFS(startNode);

    return 0;
}
```

Output:

```
DFS Traversal: 0 1 3 5 4 2
```

Subject Code: LPCIT-102

Subject Name: Object Oriented Programming using C++ Laboratory

Programme: B.Tech. (IT)	**L: 0 T: 0 P: 2**
Semester: 3	**Teaching Hours: 26**
Theory/Practical: Practical	**Credits: 1**
Internal Marks: 30	**Percentage of Numerical/Design Problems: 100%**
External Marks: 20	**Duration of End Semester Exam(ESE): 1.5 Hours**
Total Marks: 50	**Elective Status: Compulsory**

Name of Practicals:

1. Demonstrate the use of data types in C++.
2. Demonstrate the use of unary, binary and special operators in C++.
3. Programming practice on if, if-else, nested if and switch statements in C++.
4. Programming practice on “for”, “do-while”, “while”, “break” and “continue” statements in C++.
5. Demonstrate the use of class, objects, reference variable and scope resolution operator.
6. Programming practice to use objects as arguments and returning objects from functions.
8. Initialization of the member variables using constructors and illustrate the use of destructors.
9. Demonstrate the concept of default constructor, parameterized constructor and copy constructor.
10. Demonstrate the role of “static” keyword C++..
11. Demonstrate the overloading of unary operators and binary operators of C++.
12. Demonstrate the different types of inheritance and illustrate the concept of overriding.
13. Illustrate the concept of Abstract class and abstract functions.
14. Illustrate the concept of virtual functions and pure virtual functions
15. Demonstrate the use of exception handling in C++.
16. Demonstrate string handling functions of C++.
17. Illustrate the concept of streams and file pointers
18. Perform read and write operations on a file.
19. Demonstrate the concept of Templates

Experiment 1:

Demonstrate the use of data types in C++

Code:

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

int main() {
    // Integer data type
    int intNum = 10;
    cout << "Integer: " << intNum << endl;

    // Floating-point data type
    float floatNum = 3.14;
    cout << "Float: " << floatNum << endl;

    // Double data type
    double doubleNum = 3.1415926535;
    cout << "Double: " << doubleNum << endl;

    // Character data type
    char charValue = 'A';
    cout << "Character: " << charValue << endl;

    // Boolean data type
    bool boolValue = true;
    cout << "Boolean: " << boolValue << endl;

    // String data type
    string strValue = "Hello, C++";
    cout << "String: " << strValue << endl;
    return 0; }
```

Output:

```
Integer: 10
Float: 3.14
Double: 3.1415926535
Character: A
Boolean: 1
String: Hello, C++
```

Experiment 2:

Demonstrate the use of unary, binary, and special operators in C++

Code:

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

int main() {
    int a = 10, b = 5, result;

    // Unary Operators
    cout << "Unary Operators:" << endl;
    cout << "Value of a: " << a << endl;
    cout << "Increment (a++): " << a++ << endl;
    cout << "Decrement (--b): " << --b << endl;

    // Binary Operators
    cout << "\nBinary Operators:" << endl;
    cout << "Addition (a + b): " << (a + b) << endl;
    cout << "Multiplication (a * b): " << (a * b) << endl;
    cout << "Division (a / b): " << (a / b) << endl;

    // Special Operators
    cout << "\nSpecial Operators:" << endl;

    // sizeof Operator
    cout << "Size of int: " << sizeof(int) << " bytes" << endl;

    // Address-of Operator (&)
    cout << "Address of a: " << &a << endl;

    // Pointer Operator (*)
    int* ptr = &a;
```

```
    cout << "Value at pointer ptr: " << *ptr << endl;

    return 0;
}
```

Output:

```
Unary Operators:
Value of a: 10
Increment (a++): 10
Decrement (--b): 4

Binary Operators:
Addition (a + b): 15
Multiplication (a * b): 50
Division (a / b): 2

Special Operators:
Size of int: 4 bytes
Address of a: 0x7ffee9a8cbdc  (example address, varies)
Value at pointer ptr: 11
```

Experiment 3:

Programming practice on if, if-else, nested if, and switch statements in C++

Code:

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

int main() {
    int num;

    // If Statement
    cout << "Enter a number: ";
    cin >> num;

    if (num > 0) {
        cout << "The number is positive." << endl;
    }

    // If-Else Statement
    if (num % 2 == 0) {
        cout << "The number is even." << endl;
    } else {
        cout << "The number is odd." << endl;
    }

    // Nested If Statement
    if (num >= 1 && num <= 100) {
        cout << "The number is within the range of 1 to 100." << endl;
        if (num > 50) {
            cout << "The number is greater than 50." << endl;
        } else {
            cout << "The number is 50 or less." << endl;
```

```
        }
    } else {
        cout << "The number is out of range." << endl;
    }

    // Switch Statement
    cout << "\nEnter a day number (1-7): ";
    int day;
    cin >> day;

    switch (day) {
        case 1: cout << "Sunday" << endl; break;
        case 2: cout << "Monday" << endl; break;
        case 3: cout << "Tuesday" << endl; break;
        case 4: cout << "Wednesday" << endl; break;
        case 5: cout << "Thursday" << endl; break;
        case 6: cout << "Friday" << endl; break;
        case 7: cout << "Saturday" << endl; break;
        default: cout << "Invalid day number!" << endl;
    }

    return 0;
}
```

Output:

```
Enter a number: 27
The number is positive.
The number is odd.
The number is within the range of 1 to 100.
The number is 50 or less.

Enter a day number (1-7): 3
Tuesday
```

Experiment 4:

Programming practice on “for”, “do-while”, “while”, “break” and “continue” statements in C++

Code:

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

int main() {
    // For Loop
    cout << "For Loop: Printing numbers from 1 to 5\n";
    for (int i = 1; i <= 5; i++) {
        cout << i << " ";
    }
    cout << endl;

    // While Loop
    cout << "\nWhile Loop: Printing numbers from 5 to 1\n";
    int x = 5;
    while (x >= 1) {
        cout << x << " ";
        x--;
    }
    cout << endl;

    // Do-While Loop
    cout << "\nDo-While Loop: Printing numbers from 1 to 5\n";
    int y = 1;
    do {
        cout << y << " ";
        y++;
    } while (y <= 5);
    cout << endl;
```

```
    // Using Break
    cout << "\nBreak Statement: Printing numbers until 3\n";
    for (int i = 1; i <= 5; i++) {
        if (i == 4) {
            break; // Terminates the loop when i is 4
        }
        cout << i << " ";
    }
    cout << endl;

    // Using Continue
    cout << "\nContinue Statement: Skipping number 3\n";
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue; // Skips the rest of the code for i=3
        }
        cout << i << " ";
    }
    cout << endl;

    return 0;
}
```

Output:

```
For Loop: Printing numbers from 1 to 5
1 2 3 4 5

While Loop: Printing numbers from 5 to 1
5 4 3 2 1

Do-While Loop: Printing numbers from 1 to 5
1 2 3 4 5

Break Statement: Printing numbers until 3
1 2 3

Continue Statement: Skipping number 3
1 2 4 5
```

Experiment 5:

Demonstrate the use of class, objects, reference variable, and scope resolution operator in C++

Code:

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

class Demo {
private:
    int value;

public:
    // Constructor to initialize value
    Demo(int val) {
        value = val;
    }

    // Function to display value using scope resolution operator
    void display();

    // Function to modify value using reference variable
    void modifyValue(int &ref) {
        ref += 10; // Modifying reference variable
    }
};

// Scope resolution operator used to define function outside class
void Demo::display() {
    cout << "Value: " << value << endl;
}

int main() {
```

```
    // Creating object of class Demo
    Demo obj(20);

    // Display initial value
    obj.display();

    // Reference variable
    int num = 30;
    int &refNum = num; // refNum is a reference to num

    cout << "Original num: " << num << endl;

    // Modifying num using reference
    obj.modifyValue(refNum);

    cout << "Modified num using reference: " << num << endl;

    return 0;
}
```

Output:

```
Value: 20
Original num: 30
Modified num using reference: 40
```

Experiment 6:

Programming practice to use objects as arguments and returning objects from functions in C++

Code:

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

class Number {
private:
    int value;

public:
    // Constructor to initialize value
    Number(int val = 0) {
        value = val;
    }

    // Function to display value
    void display() {
        cout << "Value: " << value << endl;
    }

    // Function that takes an object as argument
    void add(Number obj) {
        value += obj.value;
    }

    // Function that returns an object
    Number multiply(Number obj) {
        Number temp;
        temp.value = value * obj.value;
        return temp;
```

```
    }
};

int main() {
    // Creating objects
    Number num1(10), num2(5), result;

    cout << "Initial values:" << endl;
    num1.display();
    num2.display();

    // Passing object as an argument
    num1.add(num2);
    cout << "\nAfter adding num2 to num1:" << endl;
    num1.display();

    // Returning an object from function
    result = num1.multiply(num2);
    cout << "\nAfter multiplying num1 and num2:" << endl;
    result.display();

    return 0;
}
```

Output:

```
Initial values:
Value: 10
Value: 5

After adding num2 to num1:
Value: 15

After multiplying num1 and num2:
Value: 75
```

Experiment 7:

Initialization of the member variables using constructors and illustrate the use of destructors in C++

Code:

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

class Demo {
private:
    int value;

public:
    // Constructor to initialize value
    Demo(int val) {
        value = val;
        cout << "Constructor called! Value initialized to " << value << endl;
    }

    // Function to display value
    void display() {
        cout << "Value: " << value << endl;
    }

    // Destructor
    ~Demo() {
        cout << "Destructor called! Value " << value << " is destroyed." << endl;
    }
};

int main() {
    // Creating objects
```

```
    cout << "Creating Object obj1..." << endl;
    Demo obj1(10);
    obj1.display();

    cout << "\nCreating Object obj2..." << endl;
    Demo obj2(20);
    obj2.display();

    cout << "\nEnd of main function, objects will be destroyed
automatically.\n";

    return 0;
}
```

Output:

```
Creating Object obj1...
Constructor called! Value initialized to 10
Value: 10

Creating Object obj2...
Constructor called! Value initialized to 20
Value: 20

End of main function, objects will be destroyed automatically.
Destructor called! Value 20 is destroyed.
Destructor called! Value 10 is destroyed.
```

Experiment 8:

Demonstrate the concept of default constructor, parameterized constructor, and copy constructor in C++

Code:

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

class Demo {
private:
    int value;

public:
    // Default Constructor
    Demo() {
        value = 0;
        cout << "Default Constructor called! Value initialized to
" << value << endl;
    }

    // Parameterized Constructor
    Demo(int val) {
        value = val;
        cout << "Parameterized Constructor called! Value
initialized to " << value << endl;
    }

    // Copy Constructor
    Demo(const Demo &obj) {
        value = obj.value;
        cout << "Copy Constructor called! Copied value: " << value
<< endl;
    }

    // Function to display value
```

```
    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    // Default Constructor
    cout << "Creating obj1 using Default Constructor..." << endl;
    Demo obj1;
    obj1.display();

    // Parameterized Constructor
    cout << "\nCreating obj2 using Parameterized Constructor..."
<< endl;
    Demo obj2(50);
    obj2.display();

    // Copy Constructor
    cout << "\nCreating obj3 using Copy Constructor (copying
obj2)..." << endl;
    Demo obj3 = obj2;
    obj3.display();

    return 0;
}
```

Output:

```
Creating obj1 using Default Constructor...
Default Constructor called! Value initialized to 0
Value: 0

Creating obj2 using Parameterized Constructor...
Parameterized Constructor called! Value initialized to 50
Value: 50

Creating obj3 using Copy Constructor (copying obj2)...
Copy Constructor called! Copied value: 50
Value: 50
```

Experiment 9:

Demonstrate the role of the “static” keyword in C++

Code:

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

class Demo {
private:
    int normalVar;         // Instance variable
    static int staticVar;  // Static variable

public:
    // Constructor to initialize normalVar
    Demo(int val) {
        normalVar = val;
        staticVar++; // Increment static variable
    }

    // Function to display values
    void display() {
        cout << "Normal Variable: " << normalVar << ", Static
Variable: " << staticVar << endl;
    }

    // Static function to access static variable
    static void showStatic() {
        cout << "Static Variable (accessed via static function): "
<< staticVar << endl;
    }
};

// Initialize static variable outside the class
int Demo::staticVar = 0;
```

```
int main() {
    cout << "Creating obj1..." << endl;
    Demo obj1(10);
    obj1.display();

    cout << "\nCreating obj2..." << endl;
    Demo obj2(20);
    obj2.display();

    cout << "\nCreating obj3..." << endl;
    Demo obj3(30);
    obj3.display();

    // Calling static function
    cout << "\nCalling static function directly using class
name..." << endl;
    Demo::showStatic();

    return 0;
}
```

Output:

```
Creating obj1...
Normal Variable: 10, Static Variable: 1

Creating obj2...
Normal Variable: 20, Static Variable: 2

Creating obj3...
Normal Variable: 30, Static Variable: 3

Calling static function directly using class name...
Static Variable (accessed via static function): 3
```

Experiment 10:

Demonstrate the overloading of unary operators and binary operators in C++

Code:

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

class OperatorOverload {
private:
    int value;

public:
    // Constructor
    OperatorOverload(int val = 0) {
        value = val;
    }

    // Overloading Unary Operator (-)
    OperatorOverload operator-() {
        return OperatorOverload(-value);
    }

    // Overloading Binary Operator (+)
    OperatorOverload operator+(const OperatorOverload &obj) {
        return OperatorOverload(value + obj.value);
    }

    // Display Function
    void display() {
        cout << "Value: " << value << endl;
    }
};
```

```
int main() {
    OperatorOverload obj1(10), obj2(20), obj3;

    cout << "Initial Values:" << endl;
    obj1.display();
    obj2.display();

    // Unary Operator Overloading
    obj3 = -obj1;
    cout << "\nAfter Unary Operator Overloading (-obj1):" << endl;
    obj3.display();

    // Binary Operator Overloading
    obj3 = obj1 + obj2;
    cout << "\nAfter Binary Operator Overloading (obj1 + obj2):"
<< endl;
    obj3.display();

    return 0;
}
```

Output:

```
Initial Values:
Value: 10
Value: 20
After Unary Operator Overloading (-obj1):
Value: -10
After Binary Operator Overloading (obj1 + obj2):
Value: 30
```

Experiment 11:

Demonstrate the different types of inheritance and illustrate the concept of overriding in C++

Code:

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

// Base Class
class Base {
public:
    void show() {
        cout << "Base class show() function" << endl;
    }
};

// Single Inheritance
class DerivedSingle : public Base {
public:
    void show() {
        cout << "DerivedSingle class overriding show() function"
<< endl;
    }
};

// Multiple Inheritance
class Parent1 {
public:
    void display1() {
        cout << "Parent1 class display1() function" << endl;
    }
};

class Parent2 {
```

```
public:
    void display2() {
        cout << "Parent2 class display2() function" << endl;
    }
};

class DerivedMultiple : public Parent1, public Parent2 {
public:
    void show() {
        cout << "DerivedMultiple class show() function" << endl;
    }
};

// Multilevel Inheritance
class Intermediate : public Base {
public:
    void show() {
        cout << "Intermediate class overriding show() function" <<
endl;
    }
};

class DerivedMultilevel : public Intermediate {
public:
    void show() {
        cout << "DerivedMultilevel class overriding show()
function" << endl;
    }
};

// Hierarchical Inheritance
class Derived1 : public Base {
public:
    void show() {
        cout << "Derived1 class overriding show() function" <<
endl;
```

```
    }
};

class Derived2 : public Base {
public:
    void show() {
        cout << "Derived2 class overriding show() function" <<
endl;
    }
};

int main() {
    Base baseObj;
    DerivedSingle singleObj;
    DerivedMultiple multipleObj;
    DerivedMultilevel multilevelObj;
    Derived1 derived1Obj;
    Derived2 derived2Obj;

    cout << "Base class function call:" << endl;
    baseObj.show();

    cout << "\nSingle Inheritance:" << endl;
    singleObj.show();

    cout << "\nMultiple Inheritance:" << endl;
    multipleObj.display1();
    multipleObj.display2();
    multipleObj.show();

    cout << "\nMultilevel Inheritance:" << endl;
    multilevelObj.show();

    cout << "\nHierarchical Inheritance:" << endl;
    derived1Obj.show();
```

```
    derived2Obj.show();

    return 0;
}
```

Output:

```
Base class function call:
Base class show() function

Single Inheritance:
DerivedSingle class overriding show() function

Multiple Inheritance:
Parent1 class display1() function
Parent2 class display2() function
DerivedMultiple class show() function

Multilevel Inheritance:
DerivedMultilevel class overriding show() function

Hierarchical Inheritance:
Derived1 class overriding show() function
Derived2 class overriding show() function
```

Experiment 12:

Illustrate the concept of Abstract class and abstract functions in C++

Code:

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

// Abstract class with a pure virtual function
class Shape {
public:
    virtual void area() = 0; // Pure virtual function (Abstract function)
    void display() {
        cout << "This is a shape." << endl;
    }
};

// Derived class implementing the abstract function
class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) { radius = r; }
    void area() {
        cout << "Area of Circle: " << 3.14 * radius * radius << endl;
    }
};

// Another derived class implementing the abstract function
class Rectangle : public Shape {
private:
    double length, breadth;
public:
```

```
    Rectangle(double l, double b) { length = l; breadth = b; }
    void area() {
        cout << "Area of Rectangle: " << length * breadth << endl;
    }
};

int main() {
    // Shape s;  // This will cause an error because we cannot instantiate an abstract class

    Shape* shape1 = new Circle(5);
    Shape* shape2 = new Rectangle(4, 6);

    cout << "Using Abstract Class Pointer:" << endl;
    shape1->area();
    shape2->area();

    delete shape1;
    delete shape2;

    return 0;
}
```

Output:

```
Using Abstract Class Pointer:
Area of Circle: 78.5
Area of Rectangle: 24
```

Experiment 13:

Illustrate the concept of virtual functions and pure virtual functions in C++

Code:

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

// Base class with a virtual function and a pure virtual function
class Base {
public:
    virtual void show() { // Virtual function
        cout << "Base class show function" << endl;
    }

    virtual void display() = 0; // Pure virtual function (Abstract function)
};

// Derived class implementing the pure virtual function
class Derived : public Base {
public:
    void show() { // Overriding virtual function
        cout << "Derived class show function" << endl;
    }

    void display() { // Implementing pure virtual function
        cout << "Derived class display function" << endl;
    }
};

int main() {
    Base* basePtr;  // Pointer of base class type
    Derived obj;
```

```
    basePtr = &obj;

    cout << "Using base class pointer to call functions:" << endl;
    basePtr->show();   // Calls derived class show() due to
runtime polymorphism
    basePtr->display(); // Calls derived class display()

    return 0;
}
```

Output:

```
Using base class pointer to call functions:
Derived class show function
Derived class display function
```

Experiment 14:

Demonstrate the use of exception handling in C++

Code:

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

void divide(int a, int b) {
    try {
        if (b == 0)
            throw "Division by zero error!";
        cout << "Result: " << a / b << endl;
    }
    catch (const char* msg) {
        cout << "Exception caught: " << msg << endl;
    }
}

int main() {
    int num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    divide(num1, num2);

    return 0;
}
```

Output:

```
Enter two numbers: 10 2
Result: 5

Enter two numbers: 5 0
Exception caught: Division by zero error!
```

Experiment 15:

Demonstrate string handling functions of C++

Code:

```
#include <iostream>
#include <cstring> // For C-style string functions
#include <string>  // For C++ string class
using namespace std;

int main() {
    // C-style strings
    char str1[] = "Hello";
    char str2[] = "World";
    char str3[20];

    // String length
    cout << "Length of str1: " << strlen(str1) << endl;

    // String concatenation
    strcat(str1, str2);
    cout << "Concatenated string (str1 + str2): " << str1 << endl;

    // String copy
    strcpy(str3, str2);
    cout << "Copied string: " << str3 << endl;

    // String comparison
    if (strcmp(str1, str2) == 0)
        cout << "str1 and str2 are equal" << endl;
    else
        cout << "str1 and str2 are not equal" << endl;

    // Using C++ string class
```

```
    string s1 = "Hello";
    string s2 = "World";
    string s3;

    // String concatenation
    s3 = s1 + " " + s2;
    cout << "C++ String concatenation: " << s3 << endl;

    // String length
    cout << "Length of s3: " << s3.length() << endl;

    // String comparison
    if (s1 == s2)
        cout << "s1 and s2 are equal" << endl;
    else
        cout << "s1 and s2 are not equal" << endl;

    return 0;
}
```

Output:

```
Length of str1: 5
Concatenated string (str1 + str2): HelloWorld
Copied string: World
str1 and str2 are not equal
C++ String concatenation: Hello World
Length of s3: 11
s1 and s2 are not equal
```

Experiment 16:

Illustrate the concept of streams and file pointers

Code:

```
#include <iostream>
#include <fstream> // For file handling
using namespace std;

int main() {
    fstream file;
    file.open("example.txt", ios::out); // Open file in write mode

    if (!file) {
        cout << "File creation failed!" << endl;
        return 1;
    }

    cout << "Writing to file..." << endl;
    file << "Hello, this is a file handling example in C++!" << endl;
    file.close(); // Closing file

    // Reading from the file
    file.open("example.txt", ios::in); // Open file in read mode
    if (!file) {
        cout << "Error opening file!" << endl;
        return 1;
    }

    cout << "Reading from file:" << endl;
    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }
```

```
    file.close(); // Closing file

    return 0;
}
```

Output:

```
Writing to file...
Reading from file:
Hello, this is a file handling example in C++!
```

Experiment 17:

Perform read and write operations on a file

Code:

```
#include <iostream>
#include <fstream> // For file handling
using namespace std;

int main() {
    string filename = "data.txt"; // File name

    // Writing to file
    ofstream outFile(filename); // Open file in write mode
    if (!outFile) {
        cout << "Error creating file!" << endl;
        return 1;
    }

    cout << "Writing data to file..." << endl;
    outFile << "This is a test file.\n";
    outFile << "File handling in C++ is interesting!\n";
    outFile.close(); // Closing file

    // Reading from file
    ifstream inFile(filename); // Open file in read mode
    if (!inFile) {
        cout << "Error opening file!" << endl;
        return 1;
    }

    cout << "Reading data from file:" << endl;
    string line;
    while (getline(inFile, line)) {
```

```
        cout << line << endl;
    }

    inFile.close(); // Closing file

    return 0;
}
```

Output:

```
Writing data to file...
Reading data from file:
This is a test file.
File handling in C++ is interesting!
```

Experiment 18:

Demonstrate the concept of Templates

Code:

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

// Template function for swapping two values
template <typename T>
void swapValues(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}

// Template class for a simple Box
template <typename T>
class Box {
private:
    T value;
public:
    Box(T val) : value(val) {} // Constructor
    void show() {
        cout << "Box contains: " << value << endl;
    }
};

int main() {
    // Using function template
    int x = 10, y = 20;
    cout << "Before swapping: x = " << x << ", y = " << y << endl;
    swapValues(x, y);
    cout << "After swapping: x = " << x << ", y = " << y << endl;
```

```
    // Using class template
    Box<int> intBox(100);
    Box<double> doubleBox(10.5);

    intBox.show();
    doubleBox.show();

    return 0;
}
```

Output:

```
Before swapping: x = 10, y = 20
After swapping: x = 20, y = 10
Box contains: 100
Box contains: 10.5
```

Subject Code: LPCIT-103

Subject Name: Data Communication and Computer Networks Laboratory

Programme: B.Tech. (IT)	**L: 0 T: 0 P: 2**
Semester: 3	**Teaching Hours: 26**
Theory/Practical: Practical	**Credits: 1**
Internal Marks: 30	**Percentage of Numerical/Design Problems: 100%**
External Marks: 20	**Duration of End Semester Exam(ESE): 1.5 hr**
Total Marks: 50	**Elective Status: Compulsory**

Detailed Contents Prerequisite: Fundamentals of Computers

1. Study of different types of network cables and practically implement the cross-wired cable and straight through cable using clamping tool

2. Study of network devices in detail

3. Study of network IP

4. Connect the computers in Local Area Network

5. Study the basic network commands and network configuration commands

6. Performing an initial switch configuration

7. Performing an initial routing configuration

8. Configuring and troubleshooting a switched network

Experiment 1:

Study of Different Types of Network Cables and Practical Implementation of Cross-Wired and Straight-Through Cables using a Clamping Tool

Objective:

- Understand different types of network cables.
- Practically implement **Cross-Wired (Crossover) Cable** and **Straight-Through Cable** using a **Clamping Tool**.

Types of Network Cables:

1. **Twisted Pair Cable**: Includes Unshielded Twisted Pair (UTP) and Shielded Twisted Pair (STP).
2. **Coaxial Cable**: Used in older networks and cable TV connections.
3. **Fiber Optic Cable**: Used for high-speed and long-distance communication.

Straight-Through vs Cross-Wired Cable:

Straight-Through Cable is used to connect different devices such as a PC to a switch or a switch to a router. Both ends follow the same wiring standard (T568A to T568A or T568B to T568B).

Cross-Wired Cable is used to connect similar devices such as a PC to another PC or a switch to another switch. One end follows the T568A standard, while the other end follows the T568B standard.

Materials Required:

- **UTP Cable (Cat5e/Cat6)**
- **RJ45 Connectors**
- **Clamping (Crimping) Tool**
- **Cable Tester**
- **Wire Stripper**

Steps to Create a Straight-Through Cable (T568B Standard):

1. Strip about **1 inch of outer insulation** using a **wire stripper**.
2. Arrange the wires in the **T568B order**: White-Orange, Orange, White-Green, Blue, White-Blue, Green, White-Brown, Brown.
3. Insert the wires into an **RJ45 connector** and use a **crimping tool** to secure them.
4. Repeat the same for the other end since Straight-Through uses the **same standard** on both ends.
5. Use a **cable tester** to verify connectivity.

Steps to Create a Cross-Wired Cable (T568A to T568B):

1. Follow the **T568A wiring** on one end: White-Green, Green, White-Orange, Blue, White-Blue, Orange, White-Brown, Brown.
2. Follow the **T568B wiring** on the other end.
3. Insert into **RJ45 connectors** and use the **crimping tool**.
4. Use a **cable tester** to ensure proper connectivity.

Experiment 2:

Study of Network Devices in Detail

Objective:

- To understand different types of **network devices** and their roles in a network.

Network Devices and Their Functions:

1. **Hub**
    - Works at **Layer 1 (Physical Layer)** of the OSI model.
    - **Broadcasts** data to all connected devices.
    - No filtering or intelligent data transfer.
    - Used in **small networks** but replaced by switches.
2. **Switch**
    - Operates at **Layer 2 (Data Link Layer)**.
    - **Forwards data** based on **MAC addresses**.
    - Provides **collision-free communication** using **full-duplex mode**.
    - Used in **modern networks** for efficient data transfer.
3. **Router**
    - Works at **Layer 3 (Network Layer)**.
    - **Routes data** between different networks using **IP addresses**.
    - Uses **routing protocols** (RIP, OSPF, BGP) to find the best path.
    - **Connects LANs, WANs, and the Internet.**
4. **Modem (Modulator-Demodulator)**
    - Converts **digital signals** to **analog** for transmission over telephone lines and vice versa.
    - Enables Internet access via **DSL, cable, or fiber connections**.
5. **Access Point (AP)**
    - Extends a **wired network** to wireless devices.
    - Used in **Wi-Fi networks** for better coverage.
    - Supports multiple users with **SSID broadcasting**.
6. **Gateway**
    - Operates at **Layer 3, 4, or higher**.

- Converts data formats between **different network architectures** (e.g., IPv4 to IPv6, email to SMS).
- Acts as a bridge between two **incompatible networks**.

7. **Bridge**
    - Works at **Layer 2 (Data Link Layer)**.
    - Divides large networks into **smaller segments**.
    - Filters traffic based on **MAC addresses**.
8. **Repeater**
    - Works at **Layer 1 (Physical Layer)**.
    - Amplifies weak signals to extend network distance.
    - Used in **long-distance communications**.

Experiment 3:

Study of Network IP

Objective:

- To understand **Internet Protocol (IP)** and its types (**IPv4 & IPv6**).
- Learn **IP addressing**, **subnetting**, and **network classes**.

Internet Protocol (IP):

IP is a set of rules that governs how data is sent and received over a network.

Types of IP:

1. **IPv4 (Internet Protocol Version 4)**
    - **32-bit address** (e.g., 192.168.1.1).
    - Supports **4.3 billion addresses**.
    - Uses **dot-decimal notation**.
    - Example: `192.168.10.5`
2. **IPv6 (Internet Protocol Version 6)**
    - **128-bit address** (e.g., 2001:db8::ff00:42:8329).
    - Supports **trillions of addresses**.
    - Uses **hexadecimal notation**.
    - Example: `2001:0db8:85a3:0000:0000:8a2e:0370:7334`

IPv4 Address Classes:

Class	Range	Default Subnet Mask	Used For
A	1.0.0.0 – 126.255.255.255	255.0.0.0	Large networks
B	128.0.0.0 – 191.255.255.255	255.255.0.0	Medium networks
C	192.0.0.0 – 223.255.255.255	255.255.255.0	Small networks
D	224.0.0.0 – 239.255.255.255	-	Multicasting
E	240.0.0.0 – 255.255.255.255	-	Experimental

Subnetting in IPv4:

- Subnetting divides a large network into smaller networks.

- Example:
    - **IP Address:** `192.168.1.0/24`
    - **Subnet Mask:** `255.255.255.0`
    - **Usable Hosts:** `254`

Private vs. Public IPs:

- **Private IPs:** Used inside a local network (e.g., 192.168.1.1).
- **Public IPs:** Used on the Internet and assigned by **ISPs**.

Experiment 4:

Connect the Computers in Local Area Network

Objective:

- To **set up and configure** a **Local Area Network (LAN)**.
- To understand the **basic network setup** and **peer-to-peer communication**.

Requirements:

- Minimum **two computers/laptops**.
- **Ethernet cables (RJ-45)** or **Wi-Fi connection**.
- **Network switch** (if multiple devices).
- **Operating System with network configuration support**.

Steps to Connect Computers in LAN:

Step 1: Physically Connect the Computers

1. Use a **straight-through Ethernet cable** to connect each computer to a **network switch**.
2. If only two computers, use a **cross-over cable** to connect them directly.
3. Alternatively, connect computers to the same **Wi-Fi network**.

Step 2: Configure IP Addresses (Manual Method)

1. Go to **Control Panel → Network and Sharing Center**.
2. Click **Change adapter settings**.
3. Right-click on **Ethernet/Wi-Fi adapter → Properties**.
4. Select **Internet Protocol Version 4 (TCP/IPv4)** → Click **Properties**.
5. Choose **"Use the following IP address"** and assign:
    - **Computer 1:**
        - IP Address: `192.168.1.1`
        - Subnet Mask: `255.255.255.0`
    - **Computer 2:**
        - IP Address: `192.168.1.2`
        - Subnet Mask: `255.255.255.0`
6. Click **OK** and close all windows.

Step 3: Test the Connection (Ping Command)

1. Open **Command Prompt (cmd)** on **Computer 1**.
2. Type: `ping 192.168.1.2`
3. If the connection is successful, you will see:

```
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
```

4. Repeat on **Computer 2** using the IP **192.168.1.1**.

Step 4: Enable File and Printer Sharing (Optional)

1. Go to **Control Panel → Network and Sharing Center**.
2. Click **Advanced Sharing Settings**.
3. Turn on **Network Discovery** and **File & Printer Sharing**.

Experiment 5:

Study the Basic Network Commands and Network Configuration Commands

Objective:

- To understand and use **basic network commands** for troubleshooting and network configuration.
- To learn how to check **network connectivity**, **IP configuration**, and **network routing** using commands.

Common Network Commands and Their Usage:

1. `ipconfig` (Windows) / `ifconfig` (Linux & macOS)

- **Displays network configuration details** (IP address, subnet mask, default gateway).
- **Command:**

```
ipconfig   # Windows
ifconfig   # Linux/macOS (deprecated, use `ip a`)
```

2. `ping` (Check Connectivity to a Host)

- **Tests network connectivity** between two devices.
- **Example:**

```
ping 8.8.8.8  # Pinging Google's public DNS server
```

3. `tracert` (Windows) / `traceroute` (Linux & macOS)

- **Shows the path** packets take to reach a destination.
- **Example:**

```
tracert google.com  # Windows
traceroute google.com  # Linux/macOS
```

4. `nslookup` (Query DNS Records)

- **Checks domain name resolution (DNS lookup).**
- **Example:**

```
nslookup google.com
```

5. `netstat` (Display Active Connections and Network Statistics)

- **Shows active TCP/UDP connections, listening ports, and routing tables.**
- **Example:**

```
netstat -an
```

6. `arp` (Address Resolution Protocol Table)

- **Displays MAC addresses of devices in the network.**
- **Example:**

```
arp -a
```

7. `route` (View and Modify Routing Table)

- **Displays or modifies the routing table of the system.**
- **Example:**

```
route print  # Windows
route -n  # Linux/macOS
```

8. `ftp` (File Transfer Protocol)

- **Used to transfer files between systems over a network.**
- **Example:**

```
ftp 192.168.1.1
```

Experiment 6:

Performing an Initial Switch Configuration

Objective:

To configure a switch with basic settings such as hostname, password, and management IP address.

Steps for Initial Switch Configuration:

1. **Connect to the Switch using Console Cable**
    - Use a terminal emulator (e.g., PuTTY, Tera Term) to access the switch via the **console port**.
2. **Enter Privileged EXEC Mode**
    - Default switch prompt:

      ```
      Switch>
      ```

    - Enter privileged EXEC mode:

      ```
      enable
      ```

        - The prompt changes to `Switch#`.
3. **Enter Global Configuration Mode**

   ```
   configure terminal
   ```

    - Prompt changes to `Switch(config)#`.
4. **Set a Hostname for the Switch**

   ```
   hostname MySwitch
   ```

    - Changes the name from `Switch` to `MySwitch`.
5. **Set a Console Password (for Local Access)**

   ```
   line console 0
   password cisco123
   login
   exit
   ```

6. **Set an Enable (Privileged Mode) Password**

   ```
   enable secret admin123
   ```

7. **Configure a Management IP Address (VLAN 1)**

   ```
   interface vlan 1
   ip address 192.168.1.100 255.255.255.0
   no shutdown
   exit
   ```

    - Assigns IP **192.168.1.100** to VLAN 1 for remote management.
8. **Configure Default Gateway (If Needed)**

   ```
   ip default-gateway 192.168.1.1
   ```

9. **Save the Configuration**

   ```
   write memory
   ```

   - Saves the configuration to **NVRAM**.

10. **Verify Configuration**

    - **Check VLAN and IP Configuration:**

      ```
      show ip interface brief
      ```

    - **Check Running Configuration:**

      ```
      show running-config
      ```

    - **Check VLANs:**

      ```
      show vlan brief
      ```

Experiment 7:

Performing an Initial Routing Configuration

Objective:

To configure a router with basic settings such as hostname, interface IP addresses, and enabling routing protocols.

Steps for Initial Router Configuration:

1. **Access the Router via Console**
    - Use a terminal emulator (e.g., **PuTTY**, **Tera Term**) to connect via the **console port**.
2. **Enter Privileged EXEC Mode**

```
Router>
enable
```

    - The prompt changes to `Router#`.
3. **Enter Global Configuration Mode**

```
configure terminal
```

    - Prompt changes to `Router(config)#`.
4. **Set a Hostname**

```
hostname MyRouter
```

    - Changes the router name to `MyRouter`.
5. **Set a Console Password**

```
line console 0
password cisco123
login
exit
```

6. **Set an Enable (Privileged Mode) Password**

```
enable secret admin123
```

7. **Configure an Interface with IP Address**
    - Assume configuring **FastEthernet 0/0**:

```
interface FastEthernet0/0
ip address 192.168.1.1 255.255.255.0
no shutdown
exit
```

    - Configure **another interface** (e.g., for WAN connectivity):

```
interface Serial0/0/0
ip address 10.0.0.1 255.255.255.252
clock rate 64000
no shutdown
exit
```

8. **Enable Routing (Static Routing Example)**

```
ip route 192.168.2.0 255.255.255.0 10.0.0.2
```

   - Routes **192.168.2.0/24** network through **10.0.0.2**.

9. **Enable a Dynamic Routing Protocol (RIP Example)**

```
router rip
version 2
network 192.168.1.0
network 10.0.0.0
exit
```

10. **Save the Configuration**

```
write memory
```

11. **Verify the Configuration**

   - **Check Interface Status:**

     ```
     show ip interface brief
     ```

   - **Check Routing Table:**

     ```
     show ip route
     ```

   - **Check Running Configuration:**

     ```
     show running-config
     ```

Experiment 8:

Configuring and Troubleshooting a Switched Network

Objective:

To configure a switch with basic settings, VLANs, and security features, and troubleshoot network issues.

Steps for Configuring a Switch:

1. **Access the Switch via Console**
    - Use **PuTTY** or another terminal emulator to connect via the **console port**.
2. **Enter Privileged EXEC Mode**

```
Switch>
enable
```

    - The prompt changes to `Switch#`.
3. **Enter Global Configuration Mode**

```
configure terminal
```

4. **Set a Hostname**

```
hostname MySwitch
```

5. **Configure Management Interface (VLAN 1)**
    - Assign an IP address to VLAN 1 for remote access:

```
interface vlan 1
ip address 192.168.1.2 255.255.255.0
no shutdown
exit
```

    - Set the **default gateway** (for communication with routers):

```
ip default-gateway 192.168.1.1
```

6. **Configure Port Security on a Specific Port**
    - Restrict the number of devices allowed on **FastEthernet 0/1**:

```
interface FastEthernet0/1
switchport mode access
switchport port-security
switchport port-security maximum 2
switchport port-security violation restrict
switchport port-security mac-address sticky
exit
```

7. **Create and Assign VLANs**
    - Create VLANs:

```
vlan 10
name SALES
vlan 20
```

```
name HR
exit
```

- Assign VLAN to a port:

```
interface FastEthernet0/2
switchport mode access
switchport access vlan 10
exit
```

8. **Enable Trunking on an Uplink Port**
    - Allow multiple VLANs on **FastEthernet 0/24**:

```
interface FastEthernet0/24
switchport mode trunk
switchport trunk allowed vlan 10,20
exit
```

9. **Save the Configuration**

```
write memory
```

Troubleshooting a Switched Network:

1. **Check Interface Status**

```
show interfaces status
```

2. **Verify VLAN Configuration**

```
show vlan brief
```

3. **Check MAC Address Table**

```
show mac address-table
```

4. **Check Port Security Violations**

```
show port-security interface FastEthernet0/1
```

5. **Verify Trunking Configuration**

```
show interfaces trunk
```

6. **Test Connectivity**

```
ping 192.168.1.1
```

Subject Code: LESIT-101

Subject Name: Digital Circuits and Logic Design Laboratory

Programme: B.Tech. (IT)	**L: 0 T: 0 P: 2**
Semester: 3	**Teaching Hours: 26**
Theory/Practical: Practical	**Credits: 1**
Internal Marks: 30	**Percentage of Numerical/Design Problems: 100%**
External Marks: 20	**Duration of End Semester Exam(ESE): 1.5 hr**
Total Marks: 50	**Elective Status: Compulsory**

Name of Practicals:

1. Truth-table verification of OR, AND, NOT, XOR, NAND and NOR gates (using various IC's and programming languages)
2. Realization of OR, AND, NOT and XOR functions using universal gates IC's 7400 and 7402
3. Half Adder / Full Adder: Realization using basic and XOR gates IC's(also using Programming Languages)
4. Half Subtractor / Full Subtractor: Realization using IC's 7400 and 7402
5. Realisation of IC7483 as Parallel Adder/Subtractor
6. 4-Bit Binary-to-Gray & Gray-to-Binary Code Converter: Realization using Basic, XOR gates and Universal gates
7. 4-Bit and 8-Bit Comparator: Implementation using IC7485 magnitude comparator chips
8. Multiplexer: Truth-table verification and realization of Half adder and Full adder using IC74153 chip
9. Demultiplexer: Truth-table verification and realization of Half subtractor and Full subtractor using IC74139 chip
10. Flip Flops: Truth-table verification of JK Master Slave FF, T-type and D-type FF using IC7476 chip.
11. Asynchronous Counter: Realization of 4-bit up counter and Mod-N counter using IC7490 & IC7493 chip
12. Synchronous Counter: Realization of 4-bit up/down counter and Mod-N counter using IC74192 & IC74193 chip.

Experiment 1:

Truth-table verification of OR, AND, NOT, XOR, NAND and NOR gates (using various ICs and programming languages)

Verification Using ICs:
Logic gates can be verified using different **ICs (Integrated Circuits)** such as:

- **AND Gate** → IC 7408
- **OR Gate** → IC 7432
- **NOT Gate** → IC 7404
- **XOR Gate** → IC 7486
- **NAND Gate** → IC 7400
- **NOR Gate** → IC 7402

Each IC has a **Vcc (Power)** and **GND (Ground)** pin. Inputs are given through the respective pins, and output is observed using an LED or multimeter.

Truth Tables for Each Gate:

1. **AND Gate (A · B)**

A	B	Output (A · B)
0	0	0
0	1	0
1	0	0
1	1	1

2. **OR Gate (A + B)**

A	B	Output (A + B)
0	0	0
0	1	1
1	0	1
1	1	1

3. **NOT Gate (~A)**

A	Output (~A)
0	1
1	0

4. **XOR Gate (A ⊕ B)**

A	B	Output (A ⊕ B)
0	0	0
0	1	1
1	0	1
1	1	0

5. **NAND Gate (~(A · B))**

A	B	Output (~(A · B))
0	0	1
0	1	1
1	0	1
1	1	0

6. **NOR Gate (~(A + B))**

A	B	Output (~(A + B))
0	0	1
0	1	0
1	0	0
1	1	0

Experiment 2:

Realization of OR, AND, NOT, and XOR functions using Universal Gates (IC 7400 and IC 7402)

Introduction

Universal gates (**NAND and NOR**) can be used to implement any Boolean function. Here, we use **IC 7400 (NAND Gate)** and **IC 7402 (NOR Gate)** to realize OR, AND, NOT, and XOR functions.

Components Required

- IC 7400 (Quad 2-input NAND gates)
- IC 7402 (Quad 2-input NOR gates)
- Breadboard
- Connecting wires
- LED indicators
- Power supply (5V)

Pin Configurations

1. **IC 7400 (NAND Gate)**
    - It contains **four** NAND gates.
    - **Pin 7:** GND, **Pin 14:** Vcc (5V).
2. **IC 7402 (NOR Gate)**
    - It contains **four** NOR gates.
    - **Pin 7:** GND, **Pin 14:** Vcc (5V).

Realization Using NAND Gates (IC 7400)

- **NOT (A')**: Connect both inputs of a NAND gate to A → Output = **NOT A**
- **AND (A · B)**: Use double NAND operation → Output = **A · B**
- **OR (A + B)**: Apply De Morgan's Theorem → Output = **A + B**
- **XOR (A ⊕ B)**: Implement using a combination of NAND gates

Realization Using NOR Gates (IC 7402)

- **NOT (A')**: Connect both inputs of a NOR gate to A → Output = **NOT A**
- **AND (A · B)**: Use De Morgan's Theorem → Output = **A · B**
- **OR (A + B)**: Use double NOR operation → Output = **A + B**
- **XOR (A ⊕ B)**: Implement using a combination of NOR gates

Procedure

1. **Power the ICs**: Connect **Pin 14** to **+5V** and **Pin 7** to **GND**.

2. **Connect Inputs and Outputs**: Use wires to set logic inputs (0 or 1) and connect LEDs to observe output.
3. **Test Each Logic Function**: Apply different input combinations and verify the output using a truth table.
4. **Record Observations**: Compare the results with theoretical values to confirm the correctness.

Observations

- NAND and NOR gates can be used to implement any Boolean function.
- The output matches the expected truth tables for OR, AND, NOT, and XOR.
- Universal gates simplify circuit design by reducing the number of different ICs required.

Experiment 3:

Half Adder / Full Adder: Realization using basic and XOR gates ICs (also using Programming Languages)

Half Adder

A half adder is a combinational logic circuit that performs the addition of two binary bits. It produces two outputs:

- **Sum (S)** = A ⊕ B (XOR operation)
- **Carry (C)** = A * B (AND operation)

Implementation using ICs:

- **XOR Gate (IC 7486):** Used for Sum output.
- **AND Gate (IC 7408):** Used for Carry output.

Truth Table for Half Adder:

A	B	Sum (A ⊕ B)	Carry (A * B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder

A full adder performs binary addition on three bits: two input bits and a carry-in bit. It produces two outputs:

- **Sum (S) = (A ⊕ B) ⊕ Cin**
- **Carry (Cout) = (A * B) + (Cin * (A ⊕ B))**

Implementation using ICs:

- **XOR Gate (IC 7486):** Used for Sum operation.
- **AND Gate (IC 7408) & OR Gate (IC 7432):** Used for Carry output.

Truth Table for Full Adder:

A	B	Cin	Sum (A ⊕ B ⊕ Cin)	Carry (Cout)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Experiment 4:

Half Subtractor / Full Subtractor: Realization using ICs 7400 and 7402 (also using Programming Languages)

Half Subtractor

A half subtractor is a combinational circuit used to subtract two single-bit binary numbers. It has two inputs (**A and B**) and two outputs:

- **Difference (D) = A ⊕ B (XOR operation)**
- **Borrow (B_out) = A' * B (A NOT AND B)**

Implementation using ICs:

- **XOR Gate (IC 7486):** Used for Difference output.
- **NAND Gate (IC 7400) & NOR Gate (IC 7402):** Used for Borrow output.

Truth Table for Half Subtractor:

A	B	Difference (A ⊕ B)	Borrow (A' * B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Full Subtractor

A full subtractor performs binary subtraction on three bits: two input bits (**A, B**) and a **Borrow-in (Bin)**. It has two outputs:

- **Difference (D) = (A ⊕ B) ⊕ Bin**
- **Borrow-out (Bout) = (A' * B) + (Bin * (A ⊕ B))**

Implementation using ICs:

- **XOR Gate (IC 7486):** Used for Difference operation.
- **NAND Gate (IC 7400) & NOR Gate (IC 7402):** Used for Borrow output

Truth Table for Full Subtractor:

A	B	Bin	Difference (A ⊕ B ⊕ Bin)	Borrow (Bout)
0	0	0	0	0

A	B	Bin	Difference (A ⊕ B ⊕ Bin)	Borrow (Bout)
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Experiment 5:

Realization of IC 7483 as Parallel Adder/Subtractor

IC 7483 – 4-bit Binary Adder

IC 7483 is a **4-bit full adder** used for performing binary addition of two 4-bit numbers. It has:

- Two 4-bit inputs: **A (A3 A2 A1 A0) and B (B3 B2 B1 B0)**
- Carry input (**Cin**)
- A 4-bit sum output (**S3 S2 S1 S0**)
- Carry output (**Cout**)

Truth Table for IC 7483 (Basic 4-bit Addition Example)

A3 A2 A1 A0	B3 B2 B1 B0	Cin	Sum (S3 S2 S1 S0)	Cout
0001 (1)	0010 (2)	0	0011 (3)	0
0101 (5)	0110 (6)	0	1011 (11)	0
1111 (15)	0001 (1)	0	0000 (0)	1

Using IC 7483 as a Parallel Adder/Subtractor

To use **IC 7483** as a **parallel adder-subtractor**, we apply the **2's complement method** for subtraction.

Circuit Modification for Subtraction:

- Subtraction is performed as **A - B = A + (-B)**
- To obtain **-B**, take the **2's complement** of B:
    - **1's complement:** Invert all bits of B
    - **Add 1** using Carry-in (**Cin = 1**)

Adder/Subtractor Mode Selection:

- **For Addition:** Connect **M = 0**, so B is passed as it is.
- **For Subtraction:** Connect **M = 1**, so B is inverted and 1 is added using Carry-in.

M (Mode)	Operation
0	A + B
1	A - B

Implementation Steps:

1. **Connect A (A3 A2 A1 A0) and B (B3 B2 B1 B0) to IC 7483**
2. **Use XOR gates (IC 7486) to invert B when M = 1**
3. **Connect M to Carry-in (Cin) for 2’s complement**
4. **Read the Sum and Carry-out values for the result**

Example of Parallel Addition and Subtraction using IC 7483

Addition (A = 0110, B = 0011, M = 0, Cin = 0)

A	B	M	Cin	Result (Sum)	Cout
0110	0011	0	0	1001 (9)	0

Subtraction (A = 0110, B = 0011, M = 1, Cin = 1)

- 1’s complement of B = **1100**
- Adding 1 (Cin = 1) → **1101**
- **A (0110) + 2's complement of B (1101) = 0011 (3) with Carry-out = 1**

A	B	M	Cin	Result (A - B)	Cout
0110	0011	1	1	0011 (3)	1

Experiment 6:

4-Bit Binary-to-Gray & Gray-to-Binary Code Converter: Realization using Basic, XOR Gates, and Universal Gates

Binary-to-Gray Code Converter

A **Binary-to-Gray** converter converts a **4-bit binary number** into its equivalent **Gray code**.

- **Gray Code**: A type of binary code where only **one bit changes at a time** between consecutive numbers.

Logic for Binary-to-Gray Conversion

The Gray code is derived using the following formula:

- **G3 = B3**
- **G2 = B3 ⊕ B2**
- **G1 = B2 ⊕ B1**
- **G0 = B1 ⊕ B0**

Where:

- **Bi** = Binary bits
- **Gi** = Corresponding Gray code bits
- **⊕** = XOR operation

Truth Table for Binary-to-Gray Conversion

Binary (B3 B2 B1 B0)	Gray (G3 G2 G1 G0)
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Implementation using XOR Gates

1. Connect **B3** directly to **G3**.
2. Use **XOR gates (IC 7486)** to compute **G2, G1, and G0**.
3. The final outputs represent the **Gray code equivalent**.

Gray-to-Binary Code Converter

A **Gray-to-Binary** converter reverses the process, converting **Gray code** back into **binary**.

Logic for Gray-to-Binary Conversion

The Binary code is obtained using the following formula:

- **B3 = G3**
- **B2 = B3 ⊕ G2**
- **B1 = B2 ⊕ G1**
- **B0 = B1 ⊕ G0**

Truth Table for Gray-to-Binary Conversion

Gray (G3 G2 G1 G0)	Binary (B3 B2 B1 B0)
0000	0000
0001	0001
0011	0010
0010	0011
0110	0100
0111	0101
0101	0110
0100	0111
1100	1000
1101	1001
1111	1010
1110	1011
1010	1100
1011	1101
1001	1110
1000	1111

Implementation using XOR Gates

1. Connect **G3** directly to **B3**.
2. Use **XOR gates (IC 7486)** to compute **B2, B1, and B0**.
3. The final outputs represent the **Binary equivalent**.

Realization using Universal Gates (NAND/NOR Gates)

Using **Universal Gates (NAND - IC 7400 or NOR - IC 7402)**, the same logic can be implemented:

- **XOR operation** can be built using NAND gates:
    - **A ⊕ B = ((A NAND (A NAND B)) NAND (B NAND (A NAND B)))**
- **NAND-based logic circuits** replace XOR gates in the previous design.

Experiment 7:

4-Bit and 8-Bit Comparator: Implementation using IC 7485 Magnitude Comparator Chips

Introduction

A **comparator** is a combinational circuit that compares two binary numbers and determines their relationship (**equal, greater than, or less than**). The **IC 7485** is a **4-bit magnitude comparator** used to compare two **4-bit numbers (A and B)**.

For **8-bit comparison**, two **IC 7485** chips are cascaded to compare **higher and lower 4-bit sections separately**.

IC 7485: 4-Bit Magnitude Comparator

Pin Configuration of IC 7485

Pin No.	Pin Name	Description
1	A3	Most significant bit (MSB) of A
2	B3	Most significant bit (MSB) of B
3	A2	Second bit of A
4	B2	Second bit of B
5	A1	Third bit of A
6	B1	Third bit of B
7	A0	Least significant bit (LSB) of A
8	B0	Least significant bit (LSB) of B
9	A < B	Output: A is less than B
10	A = B	Output: A is equal to B
11	A > B	Output: A is greater than B
12	Cascading Input (A < B)	Used for 8-bit comparison
13	Cascading Input (A = B)	Used for 8-bit comparison
14	Cascading Input (A > B)	Used for 8-bit comparison
15	VCC	Power supply (+5V)
16	GND	Ground

Truth Table for IC 7485 (4-bit Comparator)

A (4-bit)	B (4-bit)	A > B	A = B	A < B
0000	0000	0	1	0
0001	0000	1	0	0
0000	0001	0	0	1
0101	0011	1	0	0
1100	1100	0	1	0
0111	1000	0	0	1

Working of IC 7485

- The IC **compares two 4-bit binary numbers (A3 A2 A1 A0) and (B3 B2 B1 B0)**.
- It produces **three outputs**:
    - **A > B → HIGH** when A is greater.
    - **A = B → HIGH** when A is equal to B.
    - **A < B → HIGH** when A is smaller.
- The IC supports **cascading** to extend comparison for more than 4 bits.

Implementation of 8-Bit Comparator using Two IC 7485 Chips

To compare **8-bit numbers**, two ICs are cascaded:

1. **Lower 4-bits** of A and B are compared using the **first IC 7485**.
2. **Higher 4-bits** of A and B are compared using the **second IC 7485**.
3. The outputs from the first IC are fed into the **cascading inputs** of the second IC to decide the final result.

Connections for 8-bit Comparison

- **First IC 7485 (Lower 4 bits)**:
    - Inputs: **A3 A2 A1 A0** and **B3 B2 B1 B0**
    - Outputs: **(A > B), (A = B), (A < B)**
    - **Cascading outputs are connected to the second IC**.
- **Second IC 7485 (Higher 4 bits)**:
    - Inputs: **A7 A6 A5 A4** and **B7 B6 B5 B4**
    - **Cascading inputs** from the first IC are used to determine the final output.

Experiment 8:

Multiplexer: Truth-Table Verification and Realization of Half Adder and Full Adder Using IC 74153

Introduction

A **Multiplexer (MUX)** is a combinational circuit that selects one input from multiple inputs based on select lines and passes it to the output. The **IC 74153** is a **dual 4-to-1 multiplexer** that contains two independent 4-input MUXs.

In this experiment, we will:

1. **Verify the truth table of a multiplexer (IC 74153).**
2. **Use IC 74153 to implement Half Adder and Full Adder logic.**

IC 74153: 4-to-1 Multiplexer

Pin Configuration of IC 74153

Pin No.	Pin Name	Description
1	Strobe (G)	Active LOW Enable
2	S1	Select line 1
3	I3 (Data Input)	Input 3
4	I2 (Data Input)	Input 2
5	I1 (Data Input)	Input 1
6	I0 (Data Input)	Input 0
7	Y (Output)	Output of MUX
8	GND	Ground
9	Y' (Output)	Output of second MUX
10	I0'	Input 0 (MUX 2)
11	I1'	Input 1 (MUX 2)
12	I2'	Input 2 (MUX 2)
13	I3'	Input 3 (MUX 2)
14	S0	Select line 0
15	Strobe' (G')	Active LOW Enable (MUX 2)
16	VCC	Power Supply (+5V)

Truth Table of 4-to-1 Multiplexer (IC 74153)

Select Lines (S1 S0)	Output (Y)
00	I0
01	I1
10	I2
11	I3

- **Strobe (G) must be LOW (0) for MUX to function**.
- **The selected input is routed to the output based on S1 and S0 values**.

Implementation of Half Adder Using IC 74153

Half Adder Logic

A Half Adder performs addition of two binary bits and produces a **Sum (S)** and **Carry (C)**.

A	B	Sum (A ⊕ B)	Carry (A · B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

MUX Implementation of Half Adder

- **For Sum (A ⊕ B), use one 4:1 MUX with inputs as follows:**
  - **I0 = 0**, **I1 = 1**, **I2 = 1**, **I3 = 0**
  - Select lines: **A and B**
  - **Sum = I0 (00), I1 (01), I2 (10), I3 (11)**
- **For Carry (A · B), use another 4:1 MUX with inputs as follows:**
  - **I0 = 0**, **I1 = 0**, **I2 = 0**, **I3 = 1**
  - Select lines: **A and B**
  - **Carry = I0 (00), I1 (01), I2 (10), I3 (11)**

Implementation of Full Adder Using IC 74153

Full Adder Logic

A Full Adder adds three binary inputs: **A, B, and Carry-in (Cin)** and produces **Sum (S) and Carry-out (Cout)**.

A	B	Cin	Sum (A ⊕ B ⊕ Cin)	Carry (A·B + B·Cin + Cin·A)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

MUX Implementation of Full Adder

- **For Sum (A ⊕ B ⊕ Cin), use one 4:1 MUX with inputs as follows:**
  - **I0 = 0**, **I1 = 1**, **I2 = 1**, **I3 = 0**

    - Select lines: **B and Cin**
    - **Sum = I0 (00), I1 (01), I2 (10), I3 (11)**
- **For Carry (A·B + B·Cin + Cin·A), use another 4:1 MUX with inputs as follows:**
    - **I0 = 0**, **I1 = 0**, **I2 = 0**, **I3 = 1**
    - Select lines: **B and Cin**
    - **Carry = I0 (00), I1 (01), I2 (10), I3 (11)**
- The **second multiplexer** is cascaded with an **OR gate** to implement the complete carry logic.

Experiment 9:

Demultiplexer: Truth-Table Verification and Realization of Half Subtractor and Full Subtractor Using IC 74139

Introduction

A **Demultiplexer (DEMUX)** is a combinational circuit that takes a **single input** and **routes it to one of many outputs** based on the select lines. The **IC 74139** is a **dual 2-to-4 demultiplexer** that contains **two independent 1-to-4 DEMUX circuits**.

In this experiment, we will:

1. **Verify the truth table of a demultiplexer (IC 74139).**
2. **Use IC 74139 to implement Half Subtractor and Full Subtractor logic.**

IC 74139: 2-to-4 Demultiplexer

Pin Configuration of IC 74139

Pin No.	Pin Name	Description
1	Enable (G1)	Active LOW Enable
2	S0 (Select Line)	Select input for DEMUX 1
3	S1 (Select Line)	Select input for DEMUX 1
4	Y0 (Output)	Output 0
5	Y1 (Output)	Output 1
6	Y2 (Output)	Output 2
7	Y3 (Output)	Output 3
8	GND	Ground
9	Y3' (Output)	Output 3 of DEMUX 2
10	Y2' (Output)	Output 2 of DEMUX 2
11	Y1' (Output)	Output 1 of DEMUX 2
12	Y0' (Output)	Output 0 of DEMUX 2
13	S1' (Select Line)	Select input for DEMUX 2
14	S0' (Select Line)	Select input for DEMUX 2
15	Enable (G2)	Active LOW Enable (DEMUX 2)
16	VCC	Power Supply (+5V)

Truth Table of 2-to-4 Demultiplexer (IC 74139)

Enable (G)	Select Lines (S1 S0)	Output Y0	Output Y1	Output Y2	Output Y3
0 (Active)	00	0	1	1	1
0 (Active)	01	1	0	1	1
0 (Active)	10	1	1	0	1
0 (Active)	11	1	1	1	0
1 (Inactive)	XX	1	1	1	1

- **When Enable (G) is 0, one of the four outputs is LOW based on S1 and S0.**

- **When Enable (G) is 1, all outputs remain HIGH (inactive state).**

Implementation of Half Subtractor Using IC 74139

Half Subtractor Logic

A **Half Subtractor** takes two binary inputs **A and B** and produces:

1. **Difference (D) = A ⊕ B**
2. **Borrow (B_out) = A'B**

A	B	Difference (A ⊕ B)	Borrow (A'B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

DEMUX Implementation of Half Subtractor

- **For Difference (A ⊕ B):**
  - Use one **2-to-4 DEMUX**.
  - **S1 = A, S0 = B** (Select lines).
  - **Set inputs such that the output represents XOR logic.**
- **For Borrow (A'B):**
  - Use another **2-to-4 DEMUX**.
  - **S1 = A, S0 = B**.
  - **Set inputs such that the output represents A'B.**

Implementation of Full Subtractor Using IC 74139

Full Subtractor Logic

A **Full Subtractor** takes three binary inputs **A, B, and Borrow-in (Bin)** and produces:

1. **Difference (D) = A ⊕ B ⊕ Bin**
2. **Borrow-out (Bout) = A'B + A'Bin + B·Bin**

A	B	Bin	Difference (A ⊕ B ⊕ Bin)	Borrow (A'B + A'Bin + B·Bin)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

DEMUX Implementation of Full Subtractor

- **For Difference (A ⊕ B ⊕ Bin):**
    - Use one **2-to-4 DEMUX**.
    - **S1 = B, S0 = Bin**.
    - **Set inputs such that the output represents XOR logic.**
- **For Borrow (A'B + A'Bin + B·Bin):**
    - Use another **2-to-4 DEMUX**.
    - **S1 = B, S0 = Bin**.
    - **Set inputs such that the output represents borrow logic.**

Experiment 10:

Flip-Flops: Truth-Table Verification of JK Master-Slave, T-type, and D-type Flip-Flops Using IC 7476

Introduction

A **flip-flop** is a sequential circuit that can store **one bit of data** and is used as a fundamental building block in memory and registers. The **IC 7476** is a **dual JK Master-Slave Flip-Flop** with **preset and clear** functionality. By configuring the JK flip-flop, we can also create **T-type and D-type flip-flops**.

In this experiment, we will:

1. **Verify the truth tables of JK Master-Slave, T-type, and D-type Flip-Flops using IC 7476.**
2. **Understand their behavior in digital circuits.**

IC 7476: JK Master-Slave Flip-Flop

Pin Configuration of IC 7476

Pin No.	Pin Name	Description
1	CLK1	Clock for Flip-Flop 1
2	CLR1	Clear (Active LOW) for Flip-Flop 1
3	J1	J input for Flip-Flop 1
4	K1	K input for Flip-Flop 1
5	PRE1	Preset (Active LOW) for Flip-Flop 1
6	Q1	Output Q for Flip-Flop 1
7	$\bar{Q}$1	Complement Output for Flip-Flop 1
8	GND	Ground
9	$\bar{Q}$2	Complement Output for Flip-Flop 2
10	Q2	Output Q for Flip-Flop 2
11	PRE2	Preset (Active LOW) for Flip-Flop 2
12	K2	K input for Flip-Flop 2
13	J2	J input for Flip-Flop 2
14	CLR2	Clear (Active LOW) for Flip-Flop 2
15	CLK2	Clock for Flip-Flop 2
16	VCC	Power Supply (+5V)

1. Truth Table of JK Master-Slave Flip-Flop (IC 7476)

A **JK flip-flop** is a **universal flip-flop** that can function as an SR, T, or D flip-flop. It has inputs **J, K, and a clock (CLK)**.

J	K	Clock (↑)	Q (Next State)	$\bar{Q}$ (Complement)
0	0	↑	No Change	No Change

J	K	Clock (↑)	Q (Next State)	$\bar{Q}$ (Complement)
0	1	↑	0	1
1	0	↑	1	0
1	1	↑	Toggle	Toggle

- **J = 0, K = 0** → No change (Memory State)
- **J = 0, K = 1** → Reset (Q = 0, $\bar{Q}$ = 1)
- **J = 1, K = 0** → Set (Q = 1, $\bar{Q}$ = 0)
- **J = 1, K = 1** → Toggle (Q changes to opposite state on clock pulse)

2. Truth Table of T-type Flip-Flop

A **T flip-flop** is obtained from a **JK flip-flop** by connecting **J = K = T**. It toggles on each clock pulse when **T = 1**.

T	Clock (↑)	Q (Next State)	$\bar{Q}$ (Complement)
0	↑	No Change	No Change
1	↑	Toggle	Toggle

- **T = 0** → No change
- **T = 1** → Toggle (Flip state of Q)

3. Truth Table of D-type Flip-Flop

A **D flip-flop** (Data flip-flop) is obtained from a JK flip-flop by setting **J = D** and **K = $\bar{D}$**. It transfers **input D** to **output Q** on the rising edge of the clock.

D	Clock (↑)	Q (Next State)	$\bar{Q}$ (Complement)
0	↑	0	1
1	↑	1	0

- **D = 0** → Q = 0
- **D = 1** → Q = 1

Procedure for Verification

1. JK Flip-Flop Verification

- Connect the **VCC (Pin 16) and GND (Pin 8)**.
- Apply **J and K inputs**.
- Give a **clock pulse**.
- Observe the **Q and $\bar{Q}$ outputs**.

2. T Flip-Flop Verification

- Connect **J and K together as T**.
- Apply **T input and clock pulses**.
- Observe the **toggle behavior of Q**.

3. D Flip-Flop Verification

- Connect **J = D and K = $\bar{D}$**.
- Apply **D input and clock pulses**.
- Observe that **Q follows D**.

Experiment 11:

Asynchronous Counter: Realization of 4-bit Up Counter and Mod-N Counter using IC 7490 & IC 7493

Introduction

An **asynchronous counter**, also called a **ripple counter**, is a sequential circuit in which the flip-flops are **not clocked simultaneously**. The **clock input of each flip-flop is driven by the output of the preceding flip-flop**, causing a **delay (ripple effect)**.

IC **7490** and **IC 7493** are commonly used **binary and decade counters**.

In this experiment, we will:

1. **Implement a 4-bit up counter using IC 7493.**
2. **Implement a Mod-N counter using IC 7490.**

1. IC 7490: Decade Counter (MOD-10 Counter)

The **IC 7490** is a **4-bit decade counter** (MOD-10), meaning it counts from **0 to 9 (0000 to 1001 in binary)** and then resets to **0**.

Pin Configuration of IC 7490

Pin No.	Pin Name	Description
1	B	Clock for divide-by-2 section
2	R1	Reset 1 (Active HIGH)
3	R2	Reset 2 (Active HIGH)
4	NC	Not Connected
5	QA	Output Bit 0
6	QB	Output Bit 1
7	GND	Ground (0V)
8	QC	Output Bit 2
9	QD	Output Bit 3
10	NC	Not Connected
11	CLK	Clock Input
12	NC	Not Connected
13	VCC	Power Supply (+5V)
14	A	Clock for divide-by-5 section

Truth Table for MOD-10 Counter using IC 7490

Clock Pulses	QD	QC	QB	QA	Decimal Equivalent
0	0	0	0	0	0

Clock Pulses	QD	QC	QB	QA	Decimal Equivalent
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	0	0	0	0	Reset to 0

- The counter **repeats after 10 pulses**, making it a **MOD-10 counter**.

2. IC 7493: 4-bit Binary Counter (MOD-16 Counter)

The **IC 7493** is a **4-bit binary counter** (MOD-16), meaning it counts from **0 to 15 (0000 to 1111 in binary)** before resetting.

Pin Configuration of IC 7493

Pin No.	Pin Name	Description
1	QD	Output Bit 3
2	NC	Not Connected
3	CLK B	Clock Input (MSB Counter)
4	NC	Not Connected
5	R1	Reset 1 (Active HIGH)
6	R2	Reset 2 (Active HIGH)
7	GND	Ground (0V)
8	QA	Output Bit 0
9	CLK A	Clock Input (LSB Counter)
10	QB	Output Bit 1
11	QC	Output Bit 2
12	VCC	Power Supply (+5V)

Truth Table for 4-bit Counter using IC 7493

Clock Pulses	QD	QC	QB	QA	Decimal Equivalent
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2

Clock Pulses	QD	QC	QB	QA	Decimal Equivalent
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	1	0	1	0	10
11	1	0	1	1	11
12	1	1	0	0	12
13	1	1	0	1	13
14	1	1	1	0	14
15	1	1	1	1	15
16	0	0	0	0	Reset to 0

- The counter **repeats after 16 pulses**, making it a **MOD-16 counter**.

Procedure for Implementation

1. Implementation of 4-bit Up Counter using IC 7493

1. Connect **VCC (Pin 12) and GND (Pin 7)**.
2. Connect **QA (Pin 8), QB (Pin 10), QC (Pin 11), and QD (Pin 1)** as outputs.
3. Connect **Clock input (Pin 9)** to an external pulse generator or push button.
4. Observe the **binary output sequence** on LEDs or a logic analyzer.

2. Implementation of MOD-N Counter using IC 7490

1. Connect **VCC (Pin 13) and GND (Pin 7)**.
2. Connect **QA (Pin 5), QB (Pin 6), QC (Pin 8), and QD (Pin 9)** as outputs.
3. Apply a **clock signal to Pin 11**.
4. Use **reset pins (Pin 2 and Pin 3)** to reset at the required MOD-N count.
5. Observe the **binary count sequence**.

Experiment 12:

Synchronous Counter: Realization of 4-bit Up/Down Counter and Mod-N Counter using IC 74192 & IC 74193

Introduction

A **synchronous counter** is a type of counter in which **all flip-flops are clocked simultaneously**, avoiding the ripple effect seen in asynchronous counters.

IC **74192** and **IC 74193** are **4-bit synchronous up/down counters** that can be used to implement **up counters, down counters, and Mod-N counters**.

1. IC 74192: BCD Up/Down Counter (MOD-10 Counter)

The **IC 74192** is a **synchronous BCD (Binary-Coded Decimal) counter**, meaning it counts from **0 to 9 (0000 to 1001 in binary)** before resetting to **0**.

Pin Configuration of IC 74192

Pin No.	Pin Name	Description
1	D	Data Input Bit 3 (MSB)
2	C	Data Input Bit 2
3	B	Data Input Bit 1
4	A	Data Input Bit 0 (LSB)
5	Load	Parallel Load Input (Active LOW)
6	Down	Down Clock Input
7	GND	Ground (0V)
8	Borrow	Borrow Output
9	QA	Output Bit 0
10	QB	Output Bit 1
11	QC	Output Bit 2
12	QD	Output Bit 3
13	Carry	Carry Output
14	Up	Up Clock Input
15	Reset	Asynchronous Reset (Active HIGH)
16	VCC	Power Supply (+5V)

Truth Table for 4-bit Up/Down Counter using IC 74192

Clock Pulses	Up/Down	QD	QC	QB	QA	Decimal Equivalent
0	Up	0	0	0	0	0
1	Up	0	0	0	1	1
2	Up	0	0	1	0	2
3	Up	0	0	1	1	3
4	Up	0	1	0	0	4
5	Up	0	1	0	1	5
6	Up	0	1	1	0	6
7	Up	0	1	1	1	7
8	Up	1	0	0	0	8
9	Up	1	0	0	1	9
10	Up	0	0	0	0	Reset to 0
9	Down	1	0	0	1	9
8	Down	1	0	0	0	8
7	Down	0	1	1	1	7
6	Down	0	1	1	0	6
5	Down	0	1	0	1	5
4	Down	0	1	0	0	4
3	Down	0	0	1	1	3
2	Down	0	0	1	0	2
1	Down	0	0	0	1	1
0	Down	0	0	0	0	Reset to 9

- The **counter increments when UP is active** and **decrements when DOWN is active**.
- The **counter resets at 10 (0000) when counting up** and **resets at 9 (1001) when counting down**.

2. IC 74193: 4-bit Binary Up/Down Counter (MOD-16 Counter)

The **IC 74193** is a **synchronous 4-bit binary counter** (MOD-16), meaning it counts from **0 to 15 (0000 to 1111 in binary)**.

Pin Configuration of IC 74193

Pin No.	Pin Name	Description
1	D	Data Input Bit 3 (MSB)
2	C	Data Input Bit 2
3	B	Data Input Bit 1
4	A	Data Input Bit 0 (LSB)
5	Load	Parallel Load Input (Active LOW)
6	Down	Down Clock Input
7	GND	Ground (0V)
8	Borrow	Borrow Output
9	QA	Output Bit 0

Pin No.	Pin Name	Description
10	QB	Output Bit 1
11	QC	Output Bit 2
12	QD	Output Bit 3
13	Carry	Carry Output
14	Up	Up Clock Input
15	Reset	Asynchronous Reset (Active HIGH)
16	VCC	Power Supply (+5V)

Truth Table for 4-bit Up/Down Counter using IC 74193

Clock Pulses	Up/Down	QD	QC	QB	QA	Decimal Equivalent
0	Up	0	0	0	0	0
1	Up	0	0	0	1	1
2	Up	0	0	1	0	2
3	Up	0	0	1	1	3
4	Up	0	1	0	0	4
...	...	...	...	...	...	...
15	Up	1	1	1	1	15
16	Up	0	0	0	0	Reset to 0

- **IC 74193 is a MOD-16 counter** that resets after 16 pulses.
- It can be used to create **MOD-N counters** by using external reset logic.

Procedure for Implementation

1. Implementation of 4-bit Up/Down Counter using IC 74192

1. Connect **VCC (Pin 16) and GND (Pin 7)**.
2. Connect **QA (Pin 9), QB (Pin 10), QC (Pin 11), and QD (Pin 12)** as outputs.
3. Apply **clock pulses to Pin 14 (UP) for counting up** and **Pin 6 (DOWN) for counting down**.
4. Observe the **binary output sequence on LEDs or a logic analyzer**.

2. Implementation of MOD-N Counter using IC 74193

1. Connect **VCC (Pin 16) and GND (Pin 7)**.
2. Connect **QA (Pin 9), QB (Pin 10), QC (Pin 11), and QD (Pin 12)** as outputs.
3. Apply a **clock signal to Pin 14 (UP)**.
4. Use **reset logic to create a MOD-N counter**.
5. Observe the **binary count sequence**.