Experiment No. 1

A) Linear Search

Code:

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

int main() {
   int arr[11] = {22, 30, 33, 40, 44, 55, 60, 66, 77, 88, 11};
   int n = 11;
   int pos = -1;
   int k;

   for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }

   cout << endl;
   cout << "Enter the value you want to search for: ";
   cin >> k;

   for (int i = 0; i < n; i++) {
      if (arr[i] == k) {
         pos = i;
         break;
      }
   }

   if (pos == -1) {
      cout << "Element is not present";
   } else {
      cout << k << " Found at location: " << pos;
   }

   return 0;
}
```

Output:

```
/tmp/87ZGz1vqiE.o
11 22 30 33 40 44 55 60 66 77 88 99
Enter the value you want to search for: 33
33 is present at 3

=== Code Execution Successful ===
```

```
/tmp/0P7HxAqOqQ.o
11 22 30 33 40 44 55 60 66 77 88 99
Enter the value you want to search for: 101
101 is not present

=== Code Execution Successful ===
```

Experiment No. 1

B) Binary Search

Code:

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

int main() {
   int arr[12] = {11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 88, 99};

   for (int i = 0; i < 12; i++) {
      cout << arr[i] << " ";
   }

   cout << endl;

   int lb = 0;
   int ub = (sizeof(arr) / sizeof(int)) - 1;
   int mid;
   int key, loc = -1;

   cout << "Enter the value you want to search for: ";
   cin >> key;

   while (lb <= ub) {
      mid = (lb + ub) / 2;
      if (arr[mid] == key) {
         loc = mid;
         break;
      } else if (key < arr[mid]) {
         ub = mid - 1;
      } else {
         lb = mid + 1;
      }
   }

   if (loc != -1) {
      cout << key << " is present at " << loc;
```

```
    } else {
        cout << key << " is not present ";
    }

    return 0;
}
```

Output:

```
/tmp/pZFId2HuO2.o
11 22 30 33 40 44 55 60 66 77 88 99
Enter the value you want to search for: 60
60 is present at 7

=== Code Execution Successful ===
```

```
/tmp/0P7HxAqOqQ.o
11 22 30 33 40 44 55 60 66 77 88 99
Enter the value you want to search for: 101
101 is not present

=== Code Execution Successful ===
```

Experiment No. 2

A) Bubble Sort

Code:

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

int main() {
   int data[8] = {32, 51, 27, 85, 66, 23, 13, 57};
   int n = sizeof(data) / sizeof(int);
   int ptr = 0;
   int temp;

   cout << "Unsorted array:" << endl;
   for (int k = 0; k < n; k++) {
      cout << data[k] << " ";
   }
   cout << endl;

   for (int k = 0; k < n; k++) {
      while (ptr < n - k - 1) {
         if (data[ptr] > data[ptr + 1]) {
            temp = data[ptr];
            data[ptr] = data[ptr + 1];
            data[ptr + 1] = temp;
         }
         ptr += 1;
      }
      ptr = 0;
   }
   cout << "Sorted array:" << endl;
   for (int k = 0; k < n; k++) {
      cout << data[k] << " ";
   }
   cout << endl;
   ptr = 0;
}
```

Output:

```
/tmp/Gljk3UchEb.o
Unsorted array:
32 51 27 85 66 23 13 57
Sorted array:
13 23 27 32 51 57 66 85


=== Code Execution Successful ===
```

Experiment No. 2

B) Insertion Sort

Code:

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

int main() {
   int a[8] = {77, 33, 44, 11, 88, 22, 66, 55};
   int i = 0;
   int ptr;
   int temp;
   int n = sizeof(a) / sizeof(int);

   cout << "Unsorted Array" << endl;
   for (int i = 0; i < n; i++) {
      cout << a[i] << " ";
   }

   for (int i = 1; i < n; i++) {
      temp = a[i];
      ptr = i - 1;
      while (temp < a[ptr] && ptr >= 0) {
         a[ptr + 1] = a[ptr];
         ptr = ptr - 1;
      }
      a[ptr + 1] = temp;
   }

   cout << endl;
   cout << "Sorted Array" << endl;
   for (int i = 0; i < n; i++) {
      cout << a[i] << " ";
   }

   return 0;
}
```

Output:

```
/tmp/Ti1jmLF7XE.o
Unsorted Array
77 33 44 11 88 22 66 55
Sorted Array
11 22 33 44 55 66 77 88

=== Code Execution Successful ===
```

Experiment No. 2

C) Quick Sort

Code:

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

// Partition function
int partition(int arr[], int low, int high) {
   int pivot = arr[high]; // Pivot element
   int i = low - 1; // Index of smaller element

   for (int j = low; j < high; j++) {
      // If the current element is smaller than or equal to the pivot
      if (arr[j] <= pivot) {
         i++; // Increment the index of the smaller element
         swap(arr[i], arr[j]);
      }
   }

   swap(arr[i + 1], arr[high]); // Move the pivot element to the correct position
   return i + 1;
}

// QuickSort function
void quickSort(int arr[], int low, int high) {
   if (low < high) {
      int pi = partition(arr, low, high); // Partitioning index
      // Recursively sort elements before and after partition
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
   }
}

int main() {
   int arr[] = {77, 33, 44, 11, 88, 22, 66, 55};
   int n = sizeof(arr) / sizeof(int);
```

```
    cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    cout << endl << endl;
    quickSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

Output:

```
/tmp/3QaBJixhRl.o
Original array: 77 33 44 11 88 22 66 55

Sorted array: 11 22 33 44 55 66 77 88

=== Code Execution Successful ===
```

Experiment No. 3

Code:

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

int main() {
   int row, col;
   cout << "Enter the number of rows: ";
   cin >> row;
   cout << "Enter the number of columns: ";
   cin >> col;

   int matrix[row][col];
   cout << "Enter the matrix:" << endl;

   for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
         cout << "(" << i << "," << j << ") = ";
         cin >> matrix[i][j];
      }
   }

   cout << "Original Matrix:" << endl;
   for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
         cout << matrix[i][j] << " ";
      }
      cout << endl;
   }

   int size = 0;
   for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
         if (matrix[i][j] != 0) {
            size++;
         }
      }
   }

   // Making new matrix
```

```
int sparseMatrix[3][size];
int k = 0;
for (int i = 0; i < row; i++) {
   for (int j = 0; j < col; j++) {
      if (matrix[i][j] != 0) {
         sparseMatrix[0][k] = i;      // Row index
         sparseMatrix[1][k] = j;      // Column index
         sparseMatrix[2][k] = matrix[i][j]; // Value
         k++;
      }
   }
}

cout << "Sparse Matrix:" << endl;
for (int i = 0; i < 3; i++) {
   if (i == 0) {
      cout << "R : ";
   } else if (i == 1) {
      cout << "C : ";
   } else {
      cout << "V : ";
   }
   for (int j = 0; j < size; j++) {
      cout << sparseMatrix[i][j] << " ";
   }
   cout << endl;
}

// Transpose matrix
int transposeMatrix[3][size];
for (int i = 0; i < 3; i++) {
   if (i == 2) {
      for (int j = 0; j < size; j++) {
         transposeMatrix[i][j] = sparseMatrix[i][j];
      }
   } else {
      for (int j = 0; j < size; j++) {
         transposeMatrix[0][j] = sparseMatrix[1][j];
         transposeMatrix[1][j] = sparseMatrix[0][j];
      }
```

```
        }
    }

    cout << "Transpose Matrix:" << endl;
    for (int i = 0; i < 3; i++) {
        if (i == 0) {
            cout << "R : ";
        } else if (i == 1) {
            cout << "C : ";
        } else {
            cout << "V : ";
        }
        for (int j = 0; j < size; j++) {
            cout << transposeMatrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output:

```
/tmp/MIu7YlZzJ1.o
Enter the number of rows:3
Enter the number of columns:3
Enter the matrix:
(0,0) = 0
(0,1) = 1
(0,2) = 0
(1,0) = 0
(1,1) = 0
(1,2) = 2
(2,0) = 4
(2,1) = 3
(2,2) = 0
Original Matrix:
0 1 0
0 0 2
4 3 0
Sparse Matrix:
R : 0 1 2 2
C : 1 2 0 1
V : 1 2 4 3
Transpose Matrix:
R : 1 2 0 1
C : 0 1 2 2
V : 1 2 4 3
```

Experiment No. 4

Code:

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

struct Node {
   int data;
   Node* next;
   Node(int val) : data(val), next(nullptr) {}
};

// Function to add a node at the end
void addNode(Node*& head, int data) {
   Node* newNode = new Node(data);
   if (!head) {
      head = newNode;
   } else {
      Node* temp = head;
      while (temp->next) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
}

// Function to delete a node with a specific value
void deleteNode(Node*& head, int data) {
   if (!head) return;
   if (head->data == data) {
      Node* temp = head;
      head = head->next;
      delete temp;
      return;
   }
   Node* temp = head;
   while (temp->next && temp->next->data != data) {
      temp = temp->next;
```

```
    }
    if (temp->next) {
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        delete nodeToDelete;
    }
}

// Function to count the number of nodes
int countNodes(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp) {
        count++;
        temp = temp->next;
    }
    return count;
}

// Function to display the list in reverse order
void displayReverse(Node* node) {
    if (!node) return;
    displayReverse(node->next);
    cout << node->data << " ";
}

// Function to display the list
void displayList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
```

```
    Node* head = nullptr;

    // Adding nodes
    addNode(head, 10);
    addNode(head, 20);
    addNode(head, 30);
    addNode(head, 40);

    // Display list
    cout << "Original list: ";
    displayList(head);

    // Count nodes
    cout << "Total nodes: " << countNodes(head) << endl;

    // Display reverse list
    cout << "List in reverse order: ";
    displayReverse(head);
    cout << endl;

    // Delete a node
    deleteNode(head, 20);
    cout << "After deleting 20: ";
    displayList(head);

    return 0;
}
```

Output:

```
/tmp/T0tSY5eLt3.o
Original list: 10 20 30 40
Total nodes: 4
List in reverse order: 40 30 20 10
After deleting 20: 10 30 40
```

Experiment No. 5

Code:

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

// Node class for representing each term in the polynomial
class Node {
public:
   int coefficient;
   int exponent;
   Node* next;

   Node(int coeff, int exp) {
      coefficient = coeff;
      exponent = exp;
      next = nullptr;
   }
};

// Function to add two polynomials
Node* add_polynomials(Node* poly1, Node* poly2) {
   Node* result_head = new Node(0, 0);
   Node* current_result = result_head;

   while (poly1 != nullptr || poly2 != nullptr) {
      if (poly1 == nullptr) {
         current_result->next = poly2;
         break;
      } else if (poly2 == nullptr) {
         current_result->next = poly1;
         break;
      }

      if (poly1->exponent > poly2->exponent) {
         current_result->next = new Node(poly1->coefficient, poly1->exponent);
         poly1 = poly1->next;
      } else if (poly1->exponent < poly2->exponent) {
```

```
            current_result->next = new Node(poly2->coefficient, poly2->exponent);
            poly2 = poly2->next;
        } else {
            int new_coefficient = poly1->coefficient + poly2->coefficient;
            if (new_coefficient != 0) {
                current_result->next = new Node(new_coefficient, poly1->exponent);
            }
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
        current_result = current_result->next;
    }

    return result_head->next;
}

// Function to display the polynomial
void display_polynomial(Node* poly) {
    while (poly != nullptr) {
        cout << poly->coefficient << "x^" << poly->exponent;
        if (poly->next != nullptr && poly->next->coefficient >= 0) {
            cout << " + ";
        }
        poly = poly->next;
    }
    cout << endl;
}

int main() {
    // Polynomial 1: 3x^2 + 2x^1 + 5
    Node* poly1 = new Node(3, 2);
    poly1->next = new Node(2, 1);
    poly1->next->next = new Node(5, 0);

    // Polynomial 2: -1x^2 + 4x^1 - 1
    Node* poly2 = new Node(-1, 2);
    poly2->next = new Node(4, 1);
```

```
    poly2->next->next = new Node(-1, 0);

    // Add the two polynomials
    Node* result = add_polynomials(poly1, poly2);

    // Display the polynomials
    cout << "Polynomial 1: ";
    display_polynomial(poly1);

    cout << "Polynomial 2: ";
    display_polynomial(poly2);

    cout << "Sum: ";
    display_polynomial(result);

    return 0;
}
```

Output:

```
/tmp/iOrjoLnp7K.o
Polynomial 1: 3x^2 + 2x^1 + 5x^0
Polynomial 2: -1x^2 + 4x^1-1x^0
Sum: 2x^2 + 6x^1 + 4x^0
```

Experiment No. 6

Code:

```
#include <bits/stdc++.h>
using namespace std;

// Creating a linked list
class Node {
public:
   int data;
   Node* link;

   // Constructor
   Node(int n) {
      this->data = n;
      this->link = NULL;
   }
};

class Stack {
   Node* top;

public:
   Stack() { top = NULL; }

   void push(int data) {
      // Create new node temp and allocate memory in heap
      Node* temp = new Node(data);

      // Check if stack (heap) is full. Inserting an element would
      // lead to stack overflow
      if (!temp) {
         cout << "\nStack Overflow";
         exit(1);
      }

      // Initialize data into temp data field
      temp->data = data;
```

```
    // Put top pointer reference into temp link
    temp->link = top;

    // Make temp as top of Stack
    top = temp;
}

// Utility function to check if the stack is empty or not
bool isEmpty() {
    // If top is NULL it means that there are no elements in the stack
    return top == NULL;
}

// Utility function to return the top element in a stack
int peek() {
    // If stack is not empty, return the top element
    if (!isEmpty())
        return top->data;
    else
        exit(1);
}

// Function to remove a key from the given stack
void pop() {
    Node* temp;

    // Check for stack underflow
    if (top == NULL) {
        cout << "\nStack Underflow" << endl;
        exit(1);
    } else {
        // Assign top to temp
        temp = top;

        // Assign second node to top
        top = top->link;
```

```
        // Release memory of the top node
        free(temp);
      }
   }

   // Function to print all the elements of the stack
   void display() {
      Node* temp;

      // Check for stack underflow
      if (top == NULL) {
         cout << "\nStack Underflow";
         exit(1);
      } else {
         temp = top;
         while (temp != NULL) {
            // Print node data
            cout << temp->data;

            // Assign temp link to temp
            temp = temp->link;

            if (temp != NULL)
               cout << " -> ";
         }
      }
   }
};

// Driven Program
int main() {
   // Creating a stack
   Stack s;

   // Push the elements of stack
   s.push(11);
```

```
    s.push(22);
    s.push(33);
    s.push(44);
    s.push(55);
    s.push(66);
    s.push(77);

    // Display stack elements
    s.display();

    // Print top element of stack
    cout << "\nTop element is " << s.peek() << endl;

    // Delete top elements of stack
    s.pop();
    s.pop();

    // Display stack elements
    s.display();

    // Print top element of stack
    cout << "\nTop element is " << s.peek() << endl;

    return 0;
}
```

Output:

```
/tmp/PhzBqNuMFY.o
77 -> 66 -> 55 -> 44 -> 33 -> 22 -> 11
Top element is 77
55 -> 44 -> 33 -> 22 -> 11
Top element is 55
```

Experiment No. 7

Code:

```
// expression where tokens are
// separated by space.
#include <bits/stdc++.h>
using namespace std;

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

// Function to perform arithmetic operations.
int applyOp(int a, int b, char op) {
   switch (op) {
      case '+': return a + b;
      case '-': return a - b;
      case '*': return a * b;
      case '/': return a / b;
   }
}

// Function that returns value of expression after evaluation.
int evaluate(string tokens) {
   int i;
   // stack to store integer values.
   stack<int> values;
   // stack to store operators.
   stack<char> ops;

   for (i = 0; i < tokens.length(); i++) {
      // Current token is a whitespace, skip it.
      if (tokens[i] == ' ')
```

```
    continue;
// Current token is an opening brace, push it to 'ops'
else if (tokens[i] == '(') {
    ops.push(tokens[i]);
}
// Current token is a number, push it to stack for numbers.
else if (isdigit(tokens[i])) {
    int val = 0;
    // There may be more than one digits in number.
    while (i < tokens.length() && isdigit(tokens[i])) {
        val = (val * 10) + (tokens[i] - '0');
        i++;
    }

    values.push(val);

    // right now the i points to the character next to the digit,
    // since the for loop also increases the i, we would skip one
    // token position; we need to decrease the value of i by 1
    // to correct the offset.
    i--;
}
// Closing brace encountered, solve entire brace.
else if (tokens[i] == ')') {
    while (!ops.empty() && ops.top() != '(') {
        int val2 = values.top();
        values.pop();

        int val1 = values.top();
        values.pop();

        char op = ops.top();
        ops.pop();

        values.push(applyOp(val1, val2, op));
    }
```

```
            // pop opening brace.
            if (!ops.empty())
                ops.pop();
        }
        // Current token is an operator.
        else {
            // While top of 'ops' has same or greater precedence
            // to current token, which is an operator. Apply operator
            // on top of 'ops' to top two elements in values stack.
            while (!ops.empty() && precedence(ops.top()) >= precedence(tokens[i]))
{
                int val2 = values.top();
                values.pop();

                int val1 = values.top();
                values.pop();

                char op = ops.top();
                ops.pop();

                values.push(applyOp(val1, val2, op));
            }

            // Push current token to 'ops'.
            ops.push(tokens[i]);
        }
    }

    // Entire expression has been parsed at this point, apply
    // remaining ops to remaining values.
    while (!ops.empty()) {
        int val2 = values.top();
        values.pop();

        int val1 = values.top();
        values.pop();
```

```
        char op = ops.top();
        ops.pop();

        values.push(applyOp(val1, val2, op));
    }

    // Top of 'values' contains result, return it.
    return values.top();
}

int main() {
    cout << "10 + 2 * 6 = " << evaluate("10 + 2 * 6") << "\n";
    cout << "100 * 2 + 12 = " << evaluate("100 * 2 + 12") << "\n";
    cout << "100 * ( 2 + 12 ) = " << evaluate("100 * ( 2 + 12 )") << "\n";
    cout << "100 * ( 2 + 12 ) / 14 = " << evaluate("100 * ( 2 + 12 ) / 14") << "\n";
    return 0;
}
```

Output:

```
/tmp/dkXWDxllde.o
10 + 2 * 6 = 22
100 * 2 + 12 = 212
100 * ( 2 + 12 ) = 1400
100 * ( 2 + 12 ) / 14 = 100
```

Experiment No. 8

Code:

```
#include <iostream>
using namespace std;
void towers_of_hanoi(int n, const string& a, const string& b, const string& c)  {
   if (n == 1) {
      ++cnt;
      cout << "\n" << cnt << ": Move disk 1 from " << a << " to " << c;
      return;
   } else {
      towers_of_hanoi(n - 1, a, c, b);
      ++cnt;
      cout << "\n" << cnt << ": Move disk " << n << " from " << a << " to " << c;
      towers_of_hanoi(n - 1, b, a, c);
      return;
   }

int cnt = 0;

int main() {
   int n;
   cout << "Enter number of discs: ";
   cin >> n;
   towers_of_hanoi(n, "Tower 1", "Tower 2", "Tower 3");
   return 0;
}
```

Output:

```
/tmp/nNDXuAxL39.o
Enter number of discs: 3

1: Move disk 1 from Tower 1 to Tower 3
2: Move disk 2 from Tower 1 to Tower 2
3: Move disk 1 from Tower 3 to Tower 2
4: Move disk 3 from Tower 1 to Tower 3
5: Move disk 1 from Tower 2 to Tower 1
6: Move disk 2 from Tower 2 to Tower 3
7: Move disk 1 from Tower 1 to Tower 3
```

Experiment No. 9

Code:

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

#define SIZE 5

class Dequeue {
   int a[SIZE];
   int front, rear, count;

public:
   Dequeue();
   void add_at_beg(int);
   void add_at_end(int);
   void delete_fr_front();
   void delete_fr_rear();
   void display();
};

Dequeue::Dequeue() {
   front = -1;
   rear = -1;
   count = 0;
}

void Dequeue::add_at_beg(int item) {
   if (count >= SIZE) {
      cout << "\nInsertion is not possible: overflow!!!";
      return;
   }

   if (front == -1) {
      front = 0;
      rear = 0;
   } else if (front == 0) {
      cout << "\nInsertion is not possible: front is at the beginning!!!";
```

```
        return;
    } else {
        front--;
    }

    a[front] = item;
    count++;
}

void Dequeue::add_at_end(int item) {
    if (count >= SIZE) {
        cout << "\nInsertion is not possible: overflow!!!";
        return;
    }

    if (front == -1) {
        front = 0;
        rear = 0;
    } else {
        rear++;
    }

    a[rear] = item;
    count++;
}

void Dequeue::display() {
    if (front == -1) {
        cout << "\nDequeue is empty!";
        return;
    }

    cout << "\nElements in Dequeue: ";
    for (int i = front; i <= rear; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
```

```
}

void Dequeue::delete_fr_front() {
   if (front == -1) {
      cout << "Deletion is not possible: Dequeue is empty";
      return;
   }

   cout << "The deleted element is " << a[front] << endl;
   front++;

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

   count--;
}

void Dequeue::delete_fr_rear() {
   if (front == -1) {
      cout << "Deletion is not possible: Dequeue is empty";
      return;
   }

   cout << "The deleted element is " << a[rear] << endl;
   rear--;

   if (rear < front) {
      front = rear = -1; // Reset if empty
   }

   count--;
}

int main() {
   int c, item;
   Dequeue d1;
```

```
do {
   cout << "\n\n****DEQUEUE OPERATION****\n";
   cout << "\n1- Insert at beginning";
   cout << "\n2- Insert at end";
   cout << "\n3- Display";
   cout << "\n4- Deletion from front";
   cout << "\n5- Deletion from rear";
   cout << "\n6- Exit";
   cout << "\nEnter your choice (1-6): ";
   cin >> c;
   switch (c) {
      case 1:
         cout << "Enter the element to be inserted: ";
         cin >> item;
         d1.add_at_beg(item);
         break;
      case 2:
         cout << "Enter the element to be inserted: ";
         cin >> item;
         d1.add_at_end(item);
         break;
      case 3:
         d1.display();
         break;
      case 4:
         d1.delete_fr_front();
         break;
      case 5:
         d1.delete_fr_rear();
         break;
      case 6:
         exit(0);
         break;
      default:
         cout << "Invalid choice";
         break;
   }
```

```
    } while (true);

    return 0;
}
```

Output:

```
****DEQUEUE OPERATION****

1- Insert at beginning
2- Insert at end
3- Display
4- Deletion from front
5- Deletion from rear
6- Exit
Enter your choice (1-6): 1
Enter the element to be inserted: 45


****DEQUEUE OPERATION****

1- Insert at beginning
2- Insert at end
3- Display
4- Deletion from front
5- Deletion from rear
6- Exit
Enter your choice (1-6): 2
Enter the element to be inserted: 46
```

```
****DEQUEUE OPERATION****

1- Insert at beginning
2- Insert at end
3- Display
4- Deletion from front
5- Deletion from rear
6- Exit
Enter your choice (1-6): 3

Elements in Dequeue: 45 46


****DEQUEUE OPERATION****

1- Insert at beginning
2- Insert at end
3- Display
4- Deletion from front
5- Deletion from rear
6- Exit
Enter your choice (1-6): 4
The deleted element is 45
```

```
****DEQUEUE OPERATION****

1- Insert at beginning
2- Insert at end
3- Display
4- Deletion from front
5- Deletion from rear
6- Exit
Enter your choice (1-6): 3

Elements in Dequeue: 46


****DEQUEUE OPERATION****

1- Insert at beginning
2- Insert at end
3- Display
4- Deletion from front
5- Deletion from rear
6- Exit
Enter your choice (1-6): 5
The deleted element is 46
```

```
****DEQUEUE OPERATION****

1- Insert at beginning
2- Insert at end
3- Display
4- Deletion from front
5- Deletion from rear
6- Exit
Enter your choice (1-6): 6
Exited
```

Experiment No. 10

Code:

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

struct node {
   int data;
   node *L;
   node *R;
};

class bst {
public:
   node *root;
   int count;

   bst() {
      root = NULL;
      count = 0;
   }

   void create();
   void insert(node*, node*);
   void disin(node*);
   void dispre(node*);
   void dispost(node*);
   void search(node*, int);
   int height(node*);
   void mirror(node*);
   void min(node*);
};

void bst::create() {
   char ans;
   do {
      node *temp = new node;
      cout << "Enter the data: ";
```

```
        cin >> temp->data;
        temp->L = NULL;
        temp->R = NULL;

        if (root == NULL) {
            root = temp;
        } else {
            insert(root, temp);
        }

        count++;
        cout << "Do you want to insert more value (y/n)? ";
        cin >> ans;
        cout << endl;
    } while (ans == 'y');

    cout << "The Total number of nodes is: " << count << endl;
}

void bst::insert(node *root, node* temp) {
    if (temp->data > root->data) {
        if (root->R == NULL) {
            root->R = temp;
        } else {
            insert(root->R, temp);
        }
    } else {
        if (root->L == NULL) {
            root->L = temp;
        } else {
            insert(root->L, temp);
        }
    }
}

void bst::disin(node *root) {
    if (root != NULL) {
```

```
        disin(root->L);
        cout << root->data << "\t";
        disin(root->R);
    }
}

void bst::dispre(node *root) {
    if (root != NULL) {
        cout << root->data << "\t";
        dispre(root->L);
        dispre(root->R);
    }
}

void bst::dispost(node *root) {
    if (root != NULL) {
        dispost(root->L);
        dispost(root->R);
        cout << root->data << "\t";
    }
}

void bst::search(node *root, int key) {
    cout << "\nEnter your key: ";
    cin >> key;
    node *temp = root;
    while (temp != NULL) {
        if (key == temp->data) {
            cout << "KEY FOUND\n";
            return;
        }
        if (key > temp->data) {
            temp = temp->R;
        } else {
            temp = temp->L;
        }
    }
```

```
    cout << "KEY NOT FOUND\n";
}

int bst::height(node *root) {
    if (root == NULL) {
        return 0;
    }
    int hl = height(root->L);
    int hr = height(root->R);
    return 1 + max(hl, hr); // Return height from both sides
}

void bst::min(node *root) {
    node *temp = root;
    while (temp && temp->L != NULL) {
        temp = temp->L;
    }
    if (temp) {
        cout << "The minimum element is: " << temp->data << endl;
    } else {
        cout << "The tree is empty." << endl;
    }
}

void bst::mirror(node *root) {
    if (root != NULL) {
        mirror(root->L);
        mirror(root->R);
        swap(root->L, root->R); // Swap the left and right children
    }
}

int main() {
    bst t;
    int ch;
    char ans;
    do {
```

```
        cout << "\n1) Insert new node\n"
             << "2) Number of nodes in longest path\n"
             << "3) Minimum\n"
             << "4) Mirror\n"
             << "5) Search\n"
             << "6) Inorder\n"
             << "7) Preorder\n"
             << "8) Postorder\n";
        cout << "Enter your choice (1-8): ";
        cin >> ch;

        switch (ch) {
            case 1:
                t.create();
                break;
            case 2:
                cout << "Number of nodes in longest path: " << t.height(t.root) <<
endl;
                break;
            case 3:
                t.min(t.root);
                break;
            case 4:
                t.mirror(t.root);
                cout << "The mirror of tree is: ";
                t.disin(t.root);
                break;
            case 5:
                t.search(t.root, 0); // Passing 0 as the initial value
                break;
            case 6:
                cout << "\n***************INORDER**************" << endl;
                t.disin(t.root);
                break;
            case 7:
                cout << "\n***************PREORDER**************" << endl;
                t.dispre(t.root);
```

```
            break;
        case 8:
            cout << "\n********************POSTORDER**************" <<
endl;
            t.dispost(t.root);
            break;
        default:
            cout << "Invalid choice" << endl;
      }
      cout << "\nDo you want to continue (y/n)? ";
      cin >> ans;
   } while (ans == 'y');

   return 0;
}
```

Output:

```
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 1
Enter the data: 3
Do you want to insert more value (y/n)? y
Enter the data: 4
Do you want to insert more value (y/n)? y
Enter the data: 1
Do you want to insert more value (y/n)? y
Enter the data: 6
Do you want to insert more value (y/n)? y
Enter the data: 9
Do you want to insert more value (y/n)? y
Enter the data: 7
Do you want to insert more value (y/n)? n
The Total number of nodes is: 6
```

```
Do you want to continue (y/n)? y
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 2
Number of nodes in longest path: 5

Do you want to continue (y/n)? y
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 3
The minimum element is: 3
```

```
Do you want to continue (y/n)? y
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 4
The mirror of the tree is: 9 7 6 4 3 1

Do you want to continue (y/n)? y
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 5
Enter your key: 1
KEY FOUND
```

```
Do you want to continue (y/n)? y
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 6

***************INORDER**************
1 3 4 6 7 9
```

```
Do you want to continue (y/n)? y
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 7

***************PREORDER**************
3 1 4 6 9 7
```

```
Do you want to continue (y/n)? y
1) Insert new node
2) Number of nodes in longest path
3) Minimum
4) Mirror
5) Search
6) Inorder
7) Preorder
8) Postorder
Enter your choice (1-8): 8

*******************POSTORDER**************
1 7 9 6 4 3

Do you want to continue (y/n)? n
```