|  |  |
| --- | --- |
| **Practical: 1** | **Write a program that implement array operations a) Insertion b) Deletion** |
| **25-07-2024** |

**1a) Inserting the element into the array at any specific position.**

**Program**:

*#include* <iostream>

using namespace std;

int main()

{

    int arreaySize\_input;

    cout << "Enter the size of array: " << endl;

    cin >> arreaySize\_input;

    int arraySize = arreaySize\_input;

    int array[arraySize + 1];

    int i;

    cout << "Enter the array elements: " << endl;

*for* (i = 0; i < arraySize; i++)

    {

        cin >> array[i];

    }

    int position, element;

    cout << "Enter the position to insert the new element (0 to " << arraySize << "): ";

    cin >> position;

    cout << "Enter the element to insert: ";

    cin >> element;

*for* (i = arraySize; i > position; i--)

    {

        array[i] = array[i - 1];

    }

    array[position] = element;

    arraySize++;

    cout << "The array elements after insertion are: ";

*for* (i = 0; i < arraySize; i++)

    {

        cout << array[i] << " ";

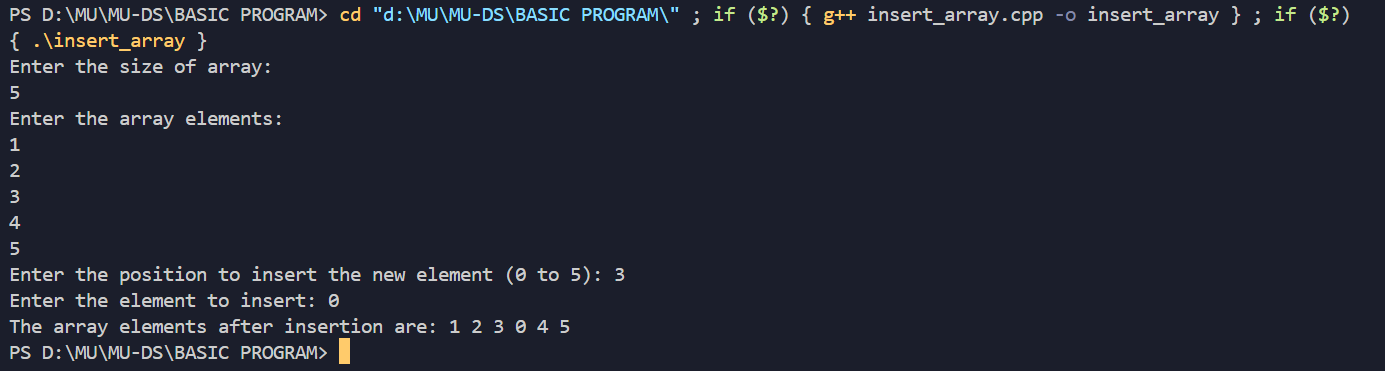
    }

    cout << endl;

*return* 0;

}

**Output:**



**1b) Deleting the element from the array.**

**Program:**

*#include* <iostream>

using namespace std;

int main()

{

    int arreaySize\_input;

    cout << "Enter the size of array: " << endl;

    cin >> arreaySize\_input;

    int arraySize = arreaySize\_input;

    int array[arraySize];

    int i;

    cout << "Enter the array elements: " << endl;

*for* (i = 0; i < arraySize; i++)

    {

*// Read elements into the array*

        cin >> array[i];

    }

    int position;

    cout << "Enter the position of the element to delete (0 to " << arraySize - 1 << "): ";

    cin >> position;

*if* (position < 0 || position >= arraySize)

    {

        cout << "Invalid position!" << endl;

    }

*else*

    {

*// Shift left*

*for* (i = position; i < arraySize - 1; i++)

        {

            array[i] = array[i + 1];

        }

        arraySize--;

*// Update array*

        cout << "The array elements after deletion are: ";

*for* (i = 0; i < arraySize; i++)

        {

            cout << array[i] << " ";

        }

        cout << endl;

    }

*return* 0;

}

**Output:**

****

**Conclusion:**

Deleting an element from an array involves shifting subsequent elements to fill the gap, maintaining contiguity, and has a time complexity of O(n). Inserting an element at a specific position requires shifting elements to the right to make room, also with a time complexity of O(n), and may require resizing the array if it is full.

|  |  |
| --- | --- |
| **Practical: 2** | **Write a program that implements the following sorting**  **a) Bubble sort b) Insertion sort c) Selection sort** |
| **01-08-2024** |

**2 a) Program to implement Bubble sort.**

**Program:**

*#include* <iostream>

using namespace std;

int main()

{

    int i, arr[50], n, x, y;

    cout << "Enter the size of array:";

    cin >> n;

    cout << "Enter the elements in an array:";

*for* (i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

*for* (i = 0; i < n; i++)

    {

*for* (int j = 0; j < n; j++)

        {

*if* (arr[j] > arr[j + 1])

            {

                int temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

*for* (i = 0; i < n; i++)

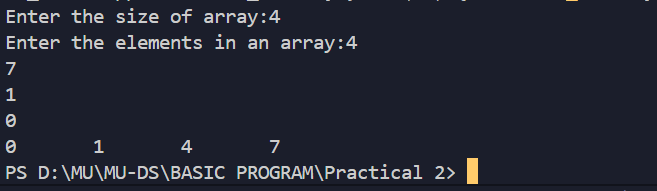
    {

        cout << arr[i] << "\t";

    }

}

**Output:**



**2 b) Program to implement Insertion sort.**

**Program:**

*#include* <iostream>

using namespace std;

int main()

{

    int i, arr[50], n, j, current;

    cout << "Enter the size of array:";

    cin >> n;

    cout << "Enter the elements in an array:";

*for* (i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

*for* (i = 1; i < n; i++)

    {

        current = arr[i];

        j = i - 1;

*while* (arr[j] > current && j >= 0)

        {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = current;

    }

*for* (i = 0; i < n; i++)

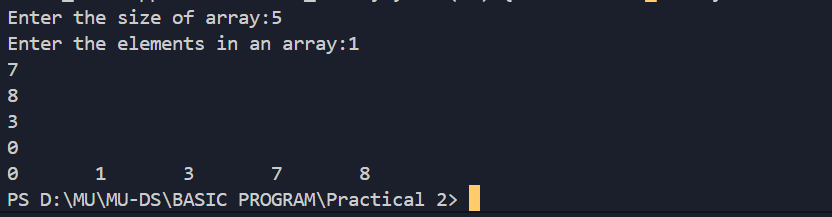
    {

        cout << arr[i] << "\t";

    }

}

**Output:**

****

**2 c) Program to implement Selection sort.**

**Program:**

*#include* <iostream>

using namespace std;

int main()

{

    int i, arr[50], n, x, y;

    cout << "Enter the size of array:";

    cin >> n;

    cout << "Enter the elements in an array:";

*for* (i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

*for* (i = 0; i < n - 1; i++)

    {

*for* (int j = i + 1; j < n; j++)

        {

*if* (arr[i] > arr[j])

            {

                int temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

    }

*for* (i = 0; i < n; i++)

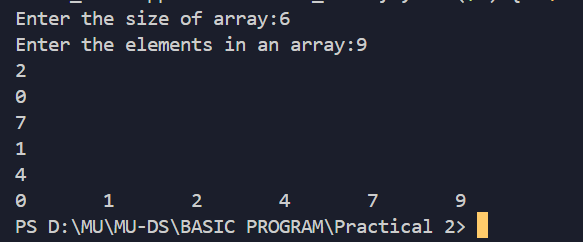
    {

        cout << arr[i] << "\t";

    }

}

**Output:**

****

**Conclusion:**

In this experiment we have learned how to Sort elements in an Array using Bubble Sorting,Insertion Sorting and Selection sorting.

|  |  |
| --- | --- |
| **Practical: 3** | **Write a program that implements the following**  **a) Quick Sort b) Merge sort** |
|  |

**3 a) Program to implement Quick sort.**

**Program:**

*#include* <iostream>

using namespace std;

*// Function to print an array*

void printArr(int a[], int n)

{

*for* (int i = 0; i < n; i++)

        cout << a[i] << " ";

    cout << endl;

}

int partition(int a[], int start, int end)

{

    int pivot = a[end]; *// pivot element*

    int i = start - 1;

*for* (int j = start; j <= end - 1; j++)

    {

*if* (a[j] < pivot)

        {

            i++;

            swap(a[i], a[j]);

        }

    }

    swap(a[i + 1], a[end]);

*return* i + 1;

}

void quick(int a[], int start, int end)

{

*if* (start < end)

    {

        int p = partition(a, start, end);

        quick(a, start, p - 1);

        quick(a, p + 1, end);

    }

}

int main()

{

int arraySize;

    cout << "Enter the size of the array: ";

    cin >> arraySize;

    int a[arraySize];

    cout << "Enter the array elements: " << endl;

*for* (int i = 0; i < arraySize; i++)

    {

        cin >> a[i];

    }

    cout << "The array elements are: ";

*for* (int i = 0; i < arraySize; i++)

    {

        cout << a[i] << " ";

    }

    cout << endl;

    int n = sizeof(a) / sizeof(a[0]);

    cout << "Before sorting, array elements are:" << endl;

    printArr(a, n);

    quick(a, 0, n - 1);

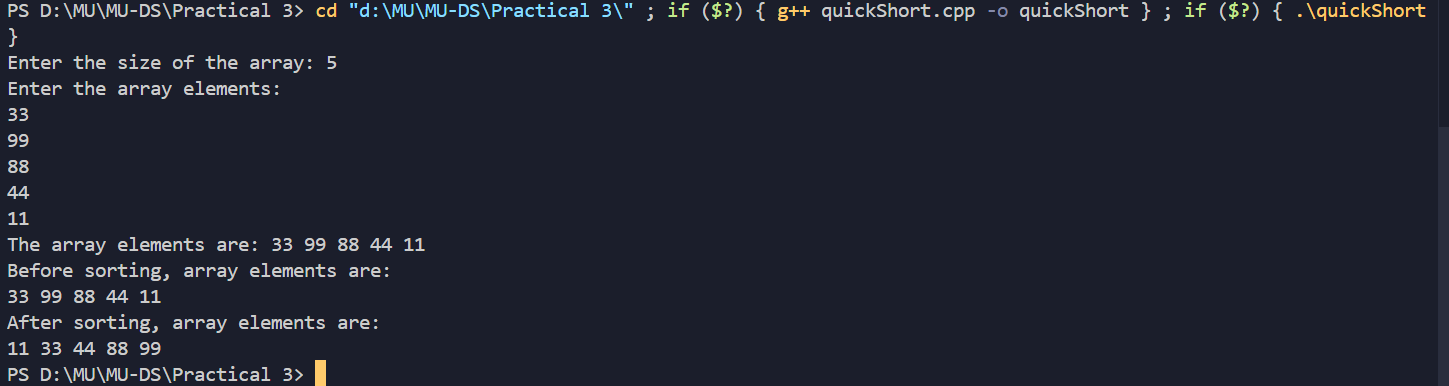
    cout << "After sorting, array elements are:" << endl;

    printArr(a, n);

*return* 0;

}

**Output:**



**3 b) Program to implement Merge sort.**

**Program:**

*#include* <iostream>

using namespace std;

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

{

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;

    int \*L = new int[n1];

    int \*R = new int[n2];

*for* (i = 0; i < n1; i++)

        L[i] = arr[l + i];

*for* (j = 0; j < n2; j++)

        R[j] = arr[m + 1 + j];

    i = 0;

    j = 0;

    k = l;

*while* (i < n1 && j < n2)

    {

*if* (L[i] <= R[j])

        {

            arr[k] = L[i];

            i++;

        }

*else*

        {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

*while* (i < n1)

    {

        arr[k] = L[i];

        i++;

        k++;

    }

*while* (j < n2)

    {

        arr[k] = R[j];

        j++;

        k++;

    }

    delete[] L;

    delete[] R;

}

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

{

*if* (l < r)

    {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }

}

void printArray(int A[], int size)

{

*for* (int i = 0; i < size; i++)

        cout << A[i] << " ";

    cout << endl;

}

*/\* Driver code \*/*

int main()

{

    int arraySize;

    cout << "Enter the size of the array: ";

    cin >> arraySize;

    int arr[arraySize];

    cout << "Enter the array elements: " << endl;

*for* (int i = 0; i < arraySize; i++)

    {

        cin >> arr[i];

    }

    cout << "The array elements are: ";

*for* (int i = 0; i < arraySize; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

    int arr\_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";

    printArray(arr, arr\_size);

    mergeSort(arr, 0, arr\_size - 1);

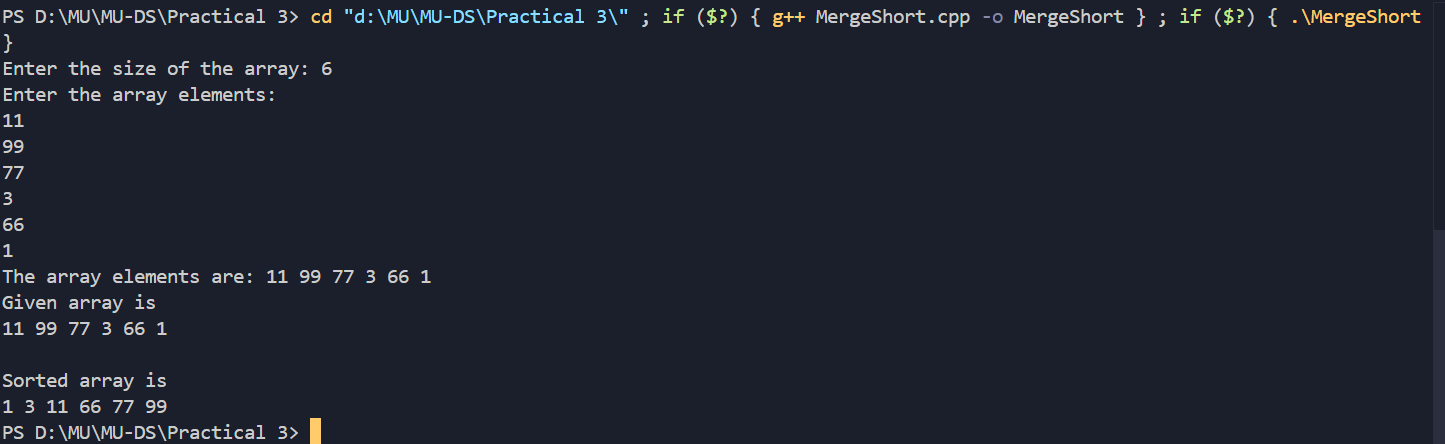
    cout << "\nSorted array is \n";

    printArray(arr, arr\_size);

*return* 0;

}

**Output:**



**Conclusion:**

Quick Sort and Merge Sort are both efficient sorting algorithms with distinct characteristics. Quick Sort uses a pivot to partition the array and sorts in place, offering an average time complexity of O(nlogn)O(n \log n)O(nlogn) but with a worst-case of O(n2)O(n^2)O(n2). It’s generally faster and more memory-efficient. Merge Sort, on the other hand, consistently performs at O(nlogn)O(n \log n)O(nlogn), as it splits the array, sorts the halves, and merges them. Though stable and predictable, Merge Sort requires additional memory. Quick Sort is preferred for its speed, while Merge Sort is favored when stability and worst-case guarantees are needed.

|  |  |
| --- | --- |
| **Practical: 4** | **Write a program for searching an element from the given list**  **a) Linear search b) Binary search.** |
|  |

**4 a) Program to implement Linear sort.**

**Program:**

*#include* <iostream>

using namespace std;

int search(int arr[], int N, int x)

{

*for* (int i = 0; i < N; i++)

*if* (arr[i] == x)

*return* i;

*return* -1;

}

*// Driver's code*

int main()

{

    int arraySize;

    cout << "Enter the size of the array: ";

    cin >> arraySize;

    int array[arraySize];

    cout << "Enter the array elements: " << endl;

*for* (int i = 0; i < arraySize; i++)

    {

        cin >> array[i];

    }

    cout << "The array elements are: ";

*for* (int i = 0; i < arraySize; i++)

    {

        cout << array[i] << " ";

    }

    cout << endl;

    int x;

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

    cin >> x;

*//TODO: Function call*

    int result = search(array, arraySize, x);

*if* (result == -1)

        cout << "Element is not present in the array" << endl;

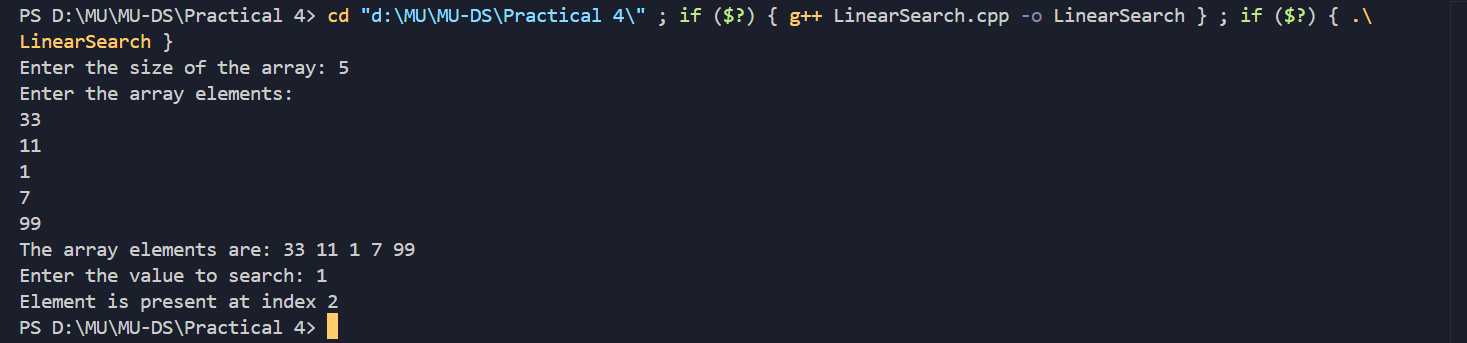
*else*

        cout << "Element is present at index " << result << endl;

*return* 0;

}

**Output:**



**4 b) Program to implement Binary sort.**

**Program:**

*#include* <iostream>

using namespace std;

int sorted(int a[], int n)

{

    int i, j;

*for* (i = 0; i < n; i++)

    {

*for* (int j = 0; j < n - 1; j++)

        {

*if* (a[j] > a[j + 1])

            {

                int temp = a[j];

                a[j] = a[j + 1];

                a[j + 1] = temp;

            }

        }

    }

*return* a[50];

}

int main()

{

    int i, arr[50], n, x, y;

    cout << "Enter the size of array:";

    cin >> n;

    cout << "Enter the elements in an array:";

*for* (i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    sorted(arr, n);

*for* (i = 0; i < n; i++)

    {

        cout << arr[i] << "\t";

    }

    int end = n - 1, beg = 0;

    cout << "\nEnter the elements to be found:";

    cin >> x;

*for* (i = 0; i < n; i++)

    {

        int mid = (beg + end) / 2;

*if* (arr[mid] == x)

        {

            cout << "The element is found at index:" << mid;

*break*;

        }

*else* *if* (arr[mid] > x)

        {

            end = mid - 1;

        }

*else*

        {

            beg = mid + 1;

        }

    }

}

**Output:**

****

**Conclusion:**

Linear search is a simple algorithm that checks each element of the list sequentially, making it effective for small or unsorted lists. Its time complexity is O(n)O(n)O(n). Binary search, on the other hand, is more efficient with a time complexity of O(log n)O(\log n)O(logn), but it requires the list to be sorted. Binary search is preferred for large, sorted lists due to its faster performance.

|  |  |
| --- | --- |
| **Practical: 5** | **Write a program to implement STACK using array that performs following operations: (a) PUSH (b) POP (c) Display (d) isEmpty (e) isFull** |
|  |

**Program:**

*#include* <iostream>

*#define* n 10

using namespace std;

int top = 0;

int st[n];

int st\_push(int);

int st\_pop();

void st\_display();

int main()

{

    int ch, x, ind, rpindex, rpvalue;

    char con;

*do*

    {

        cout << "Press 1 for Push\n";

        cout << "Press 2 for Pop\n";\

        cout << "Enter your choice::\n";

        cin >> ch;

*switch* (ch)

        {

*case* 1:

        {

            cout << "Enter Element you want to insert\n";

            cin >> x;

            st\_push(x);

*break*;

        }

*case* 2:

        {

            st\_pop();

*break*;

        }

*default*:

        {

            cout << "Enter valid choice\n";

        }

        }

        cout << "Do you want to continue (y/n)?\n";

        cin >> con;

    } *while* (con == 'y');

*return* 0;

}

int st\_push(int x)

{

*if* (top >= n)

    {

        cout << "Overflow on stack \n";

*return* 0;

    }

    top++;

    st[top] = x;

    cout << "The new top value is " << top << "\n";

    st\_display();

*return* 0;

}

int st\_pop()

{

*if* (top <= 0)

    {

        cout << "Underflow on stack \n";

*return* 0;

    }

    cout << "Element deleted from stack is " << st[top] << "\n";

    top = top - 1;

    st\_display();

*return* 0;

}

void st\_display()

{

*for* (int i = 1; i <= top; i++)

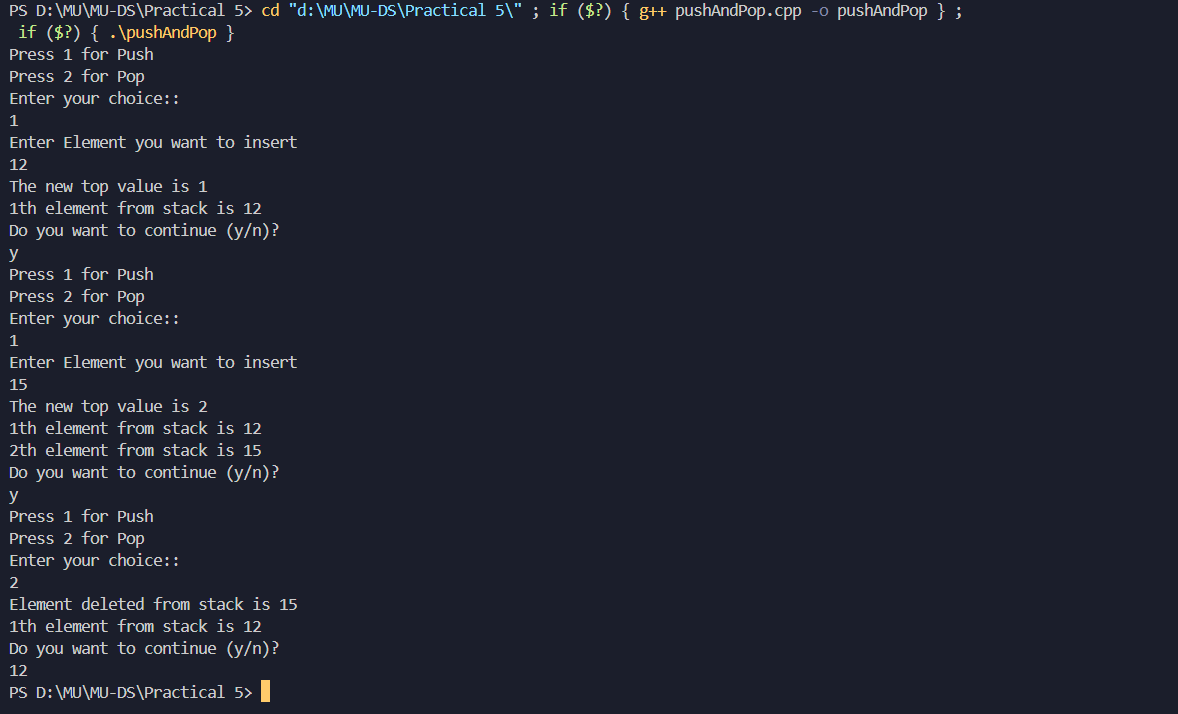
    {

        cout << i << "th element from stack is " << st[i] << "\n";

    }

}

**Output:**



|  |  |
| --- | --- |
| **Practical: 6** | **Write a program to implement Queue using arrays that perform the following operations. (a) Insert (b) Delete (c) Display (d) isEmpty (e) isFull** |
|  |

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

int queue[MAX];

int front = -1, rear = -1;

int isEmpty() {

if (front == -1)

return 1;

else

return 0;

}

int isFull() {

if (rear == MAX - 1)

return 1;

else

return 0;

}

void enqueue(int value) {

if (isFull()) {

printf("Queue is full. Cannot insert.\n");

return;

}

if (front == -1)

front = 0;

rear++;

queue[rear] = value;

printf("%d inserted into the queue.\n", value);

}

void dequeue() {

if (isEmpty()) {

printf("Queue is empty. Cannot delete.\n");

return;

}

printf("%d deleted from the queue.\n", queue[front]);

if (front == rear) {

// Reset queue after the last element is deleted

front = rear = -1;

} else {

front++;

}

}

void display() {

if (isEmpty()) {

printf("Queue is empty.\n");

return;

}

printf("Queue elements are: ");

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

printf("\n");

}

int main() {

int choice, value;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\n");

printf("4. Check if Empty\n");

printf("5. Check if Full\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

if (isEmpty())

printf("Queue is empty.\n");

else

printf("Queue is not empty.\n");

break;

case 5:

if (isFull())

printf("Queue is full.\n");

else

printf("Queue is not full.\n");

break;

case 6:

exit(0);

default:

printf("Invalid choice! Please try again.\n");

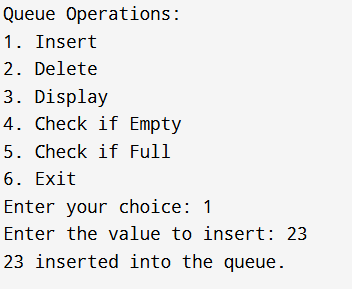
}

}

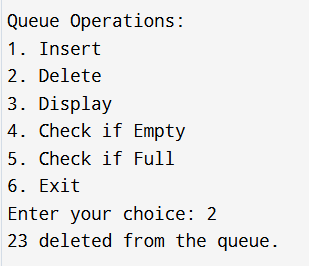
return 0;

}

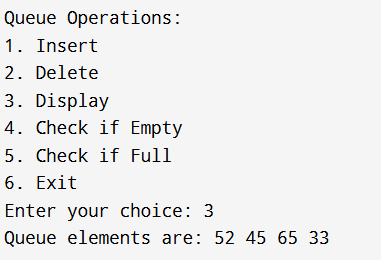
**Output:**

**a)Inserting an element:**

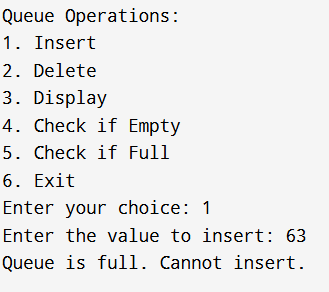
**b)Deleting an element:**

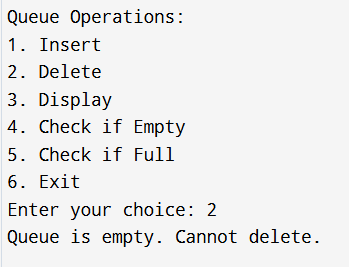


**c)Display the Queue:**



**d)Queue is full:**



**e)Queue is Empty:**

|  |  |
| --- | --- |
| **Practical: 7** | **Write a program to implement Circular Queue using arrays that perform the following operations. (a) Insert (b) Delete (c) Display (d) isEmpty (e) isFull** |
|  |

**Program:**

*#include* <iostream>

using namespace std;

*#define* SIZE 5 *// Define the size of the circular queue*

class CircularQueue

{

private:

    int queue[SIZE]; *// Array to store queue elements*

    int front, rear; *// Front and rear pointers*

public:

*// Constructor to initialize the front and rear pointers*

    CircularQueue()

    {

        front = -1;

        rear = -1;

    }

*// Check if the queue is empty*

    bool isEmpty()

    {

*return* (front == -1);

    }

*// Check if the queue is full*

    bool isFull()

    {

*return* ((rear + 1) % SIZE == front);

    }

*// Insert an element into the queue*

    void insert(int value)

    {

*if* (isFull())

        {

            cout << "Queue is full. Cannot insert " << value << endl;

*return*;

        }

*if* (isEmpty())

        {

            front = 0;

        }

        rear = (rear + 1) % SIZE; *// Circular increment*

        queue[rear] = value;

        cout << value << " inserted into the queue." << endl;

    }

*// Delete an element from the queue*

    void del()

    {

*if* (isEmpty())

        {

            cout << "Queue is empty. Cannot delete." << endl;

*return*;

        }

        cout << queue[front] << " deleted from the queue." << endl;

*if* (front == rear)

        {

*// If only one element is present in the queue*

            front = rear = -1;

        }

*else*

        {

            front = (front + 1) % SIZE; *// Circular increment*

        }

    }

*// Display the elements of the queue*

    void display()

    {

*if* (isEmpty())

        {

            cout << "Queue is empty." << endl;

*return*;

        }

        cout << "Queue elements: ";

        int i = front;

*while* (true)

        {

            cout << queue[i] << " ";

*if* (i == rear)

*break*;

            i = (i + 1) % SIZE; *// Circular increment*

        }

        cout << endl;

    }

};

int main()

{

    CircularQueue cq;

    int choice, value;

*do*

    {

        cout << "\nCircular Queue Operations:";

        cout << "\n1. Insert";

        cout << "\n2. Delete";

        cout << "\n3. Display";

        cout << "\n4. Check if Empty";

        cout << "\n5. Check if Full";

        cout << "\n6. Exit";

        cout << "\nEnter your choice: ";

        cin >> choice;

*switch* (choice)

        {

*case* 1:

            cout << "Enter the value to insert: ";

            cin >> value;

            cq.insert(value);

*break*;

*case* 2:

            cq.del();

*break*;

*case* 3:

            cq.display();

*break*;

*case* 4:

*if* (cq.isEmpty())

                cout << "Queue is empty." << endl;

*else*

                cout << "Queue is not empty." << endl;

*break*;

*case* 5:

*if* (cq.isFull())

                cout << "Queue is full." << endl;

*else*

                cout << "Queue is not full." << endl;

*break*;

*case* 6:

            cout << "Exiting..." << endl;

*break*;

*default*:

            cout << "Invalid choice! Please try again." << endl;

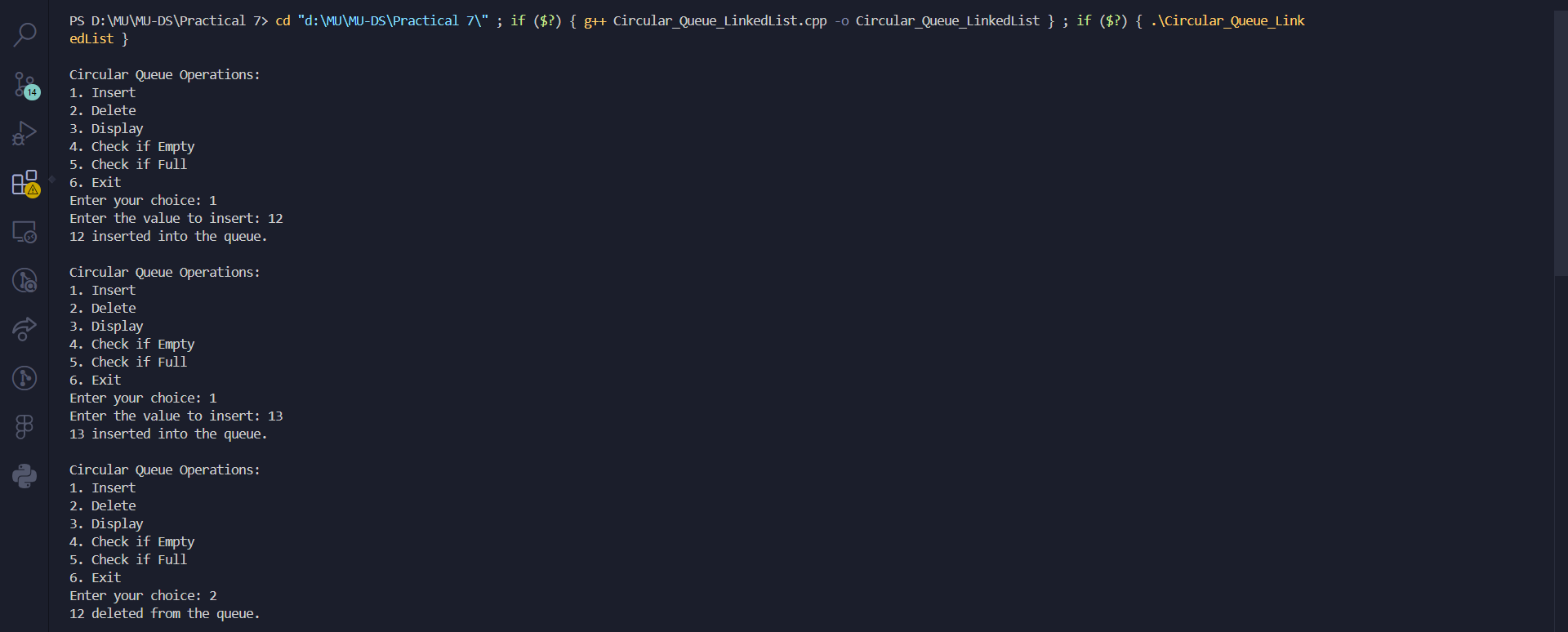
        }

    } *while* (choice != 6);

*return* 0;

}

**Output:**

****

****

****