University of Information Technology and Sciences

Department of CSE

Assignment

Course Code	CSE063212
Course Title	Data Structure & Algorithm I Lab
Assignment No	02
Assignment Title	Assignment on Searching, Sorting & Linked list
Submission Date	16-04-2025

Submitted by:	**Submitted to:**
Name: Md. Jawad-Ul-Karim **ID No:** 0432410005101029 **Batch:** 55 **Section:** 3A2	**Name:** Pabon Shaha Lecturer Department of CSE, University of Information Technology and Sciences.

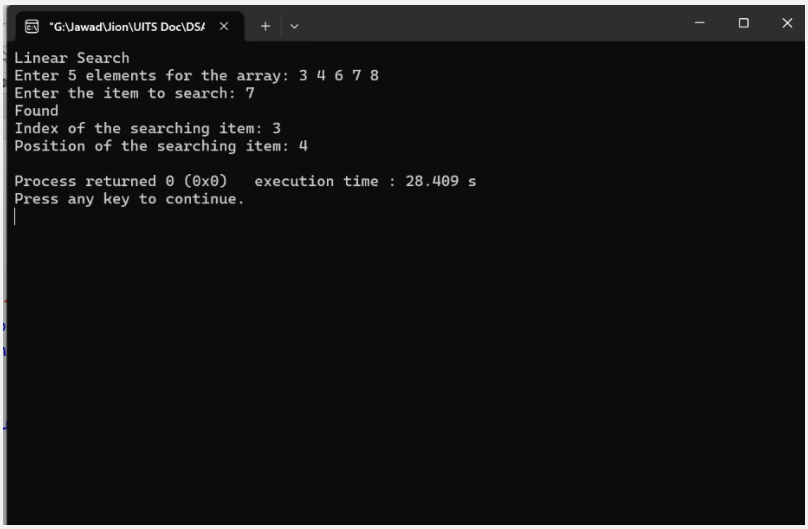
Searching

- **Linear Search-**

```
#include<iostream>
using namespace std;
int main ()
{
    cout << "Linear Search" << endl;

    int arr [5];
    int j=5;
    int item, loca = -1;

    cout << "Enter 5 elements for the array: ";

    for(int i = 0; i < j; i++) {
        cin >> arr[i];
    }
    cout << "Enter the item to search: ";
    cin >> item;

    for (int i = 0; i < j; i++) {
        if(arr[i] == item) {
```

```
            loca = i;
            break;
        }
    }
    if (loca != -1) {
        cout << "Found" << endl;
        cout << "Index of the searching item: " << loca << endl;
        cout << "Position of the searching item: " << loca + 1 << endl;
    }
    else {
        cout << "Not Found" << endl;
    }
    return 0;
}
```

Output:



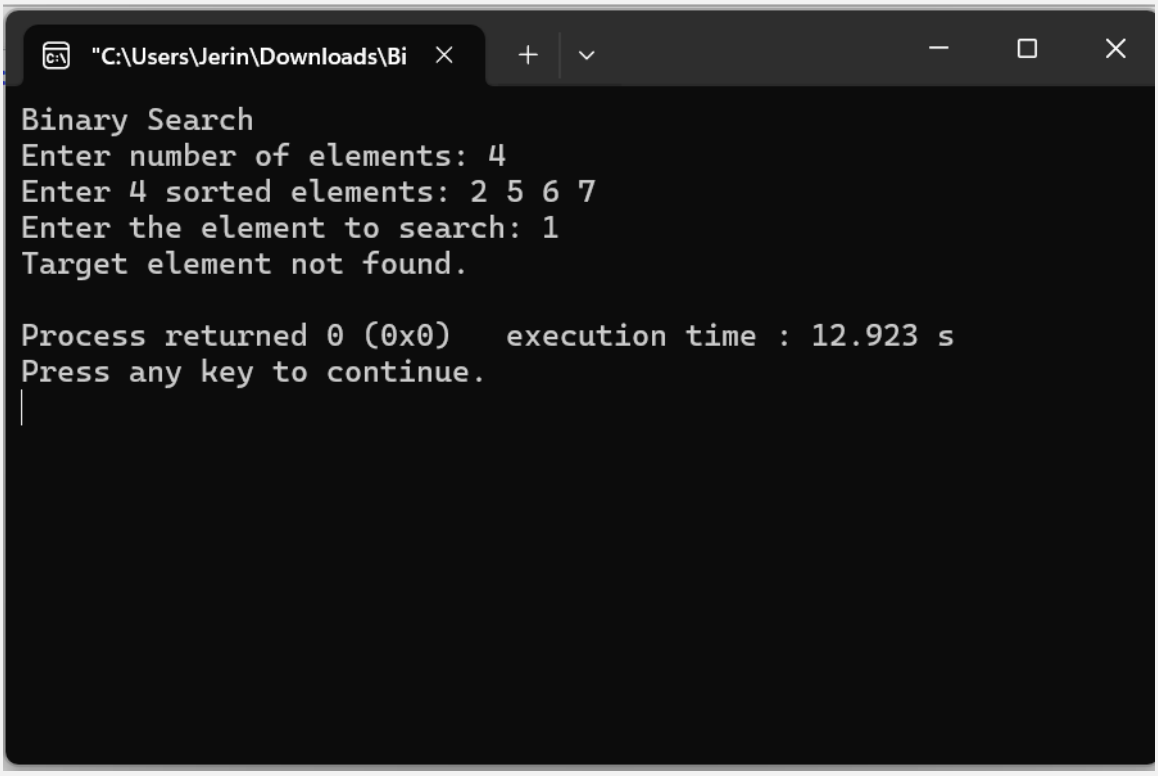
- **Binary Search-**

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

int main () {
    cout << "Binary Search" << endl;

    int j;
    cout << "Enter number of elements: ";
    cin >> j;

    int arr[j];
    cout << "Enter " << j << " sorted elements: ";
    for(int i = 0; i < j; i++) {
        cin >> arr[i];
    }
    int target, mid, flag = 0;
    cout << "Enter the element to search: ";
    cin >> target;
```

```
int low = 0;
int high = j - 1;

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

    if(arr[mid] == target) {
        flag = 1;
        break;
    }
    else if(arr[mid] < target) {
        low = mid + 1;
    }
    else {
        high = mid - 1;
    }
}

if (flag == 0) {
    cout << "Target element not found." << endl;
}
```

```
    else {
        cout << "Target element found." << endl;
}
    return 0;
}
```

Output:



Sorting

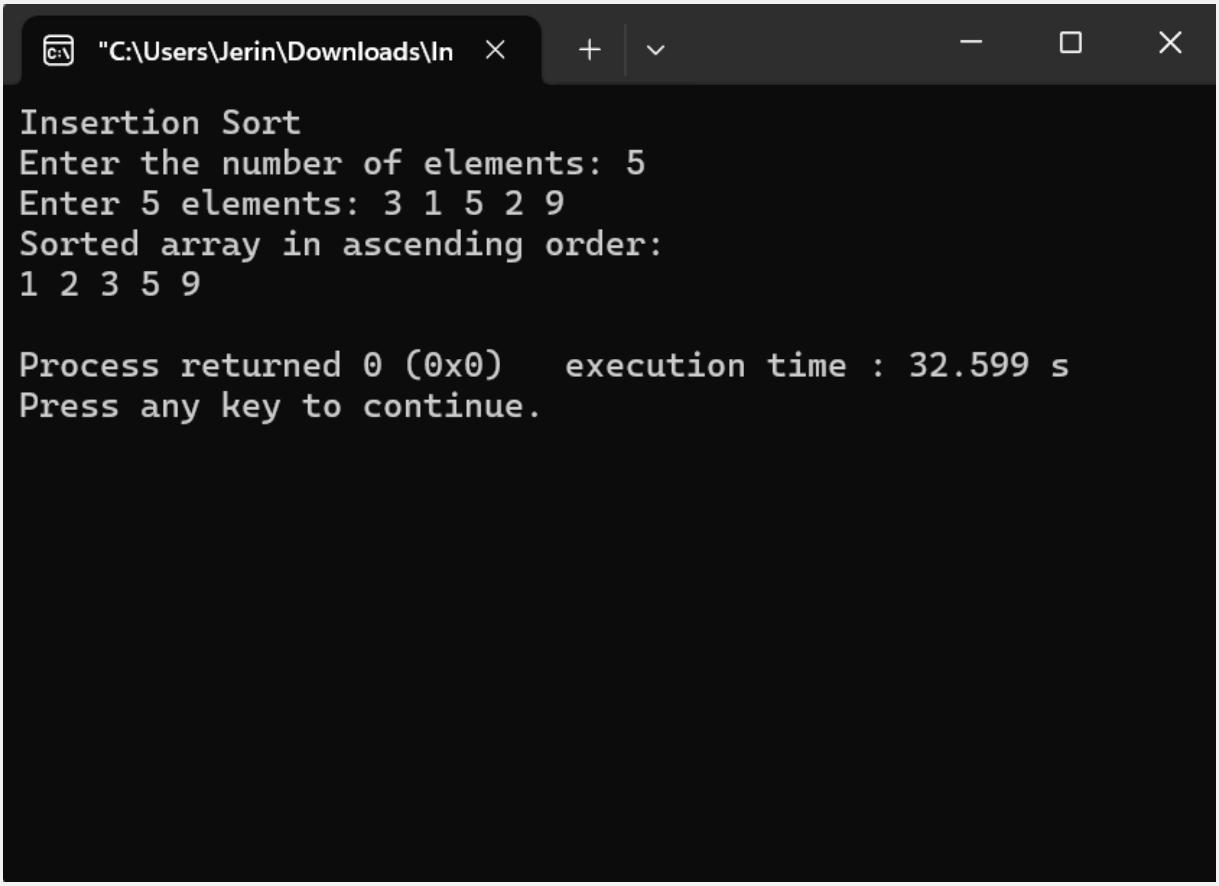
- **Insertion Sort-**

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

void printArray (int array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void insertionSort(int array[], int size) {
  for (int i = 1; i < size; i++) {
    int key = array[i];
    int j = i - 1;

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

```
    array [j + 1] = key;
  }
}

int main () {
  cout<<"Insertion Sort"<<endl;

  int size;
  cout << "Enter the number of elements: ";
  cin >> size;

  int* data = new int[size];

  cout << "Enter " << size << " elements: ";
  for (int k = 0; k < size; k++) {
    cin >> data[k];
  }
  insertionSort(data, size);

  cout << "Sorted array in ascending order:\n";
  printArray(data, size);
```

```
  return 0;
}
```

Output:

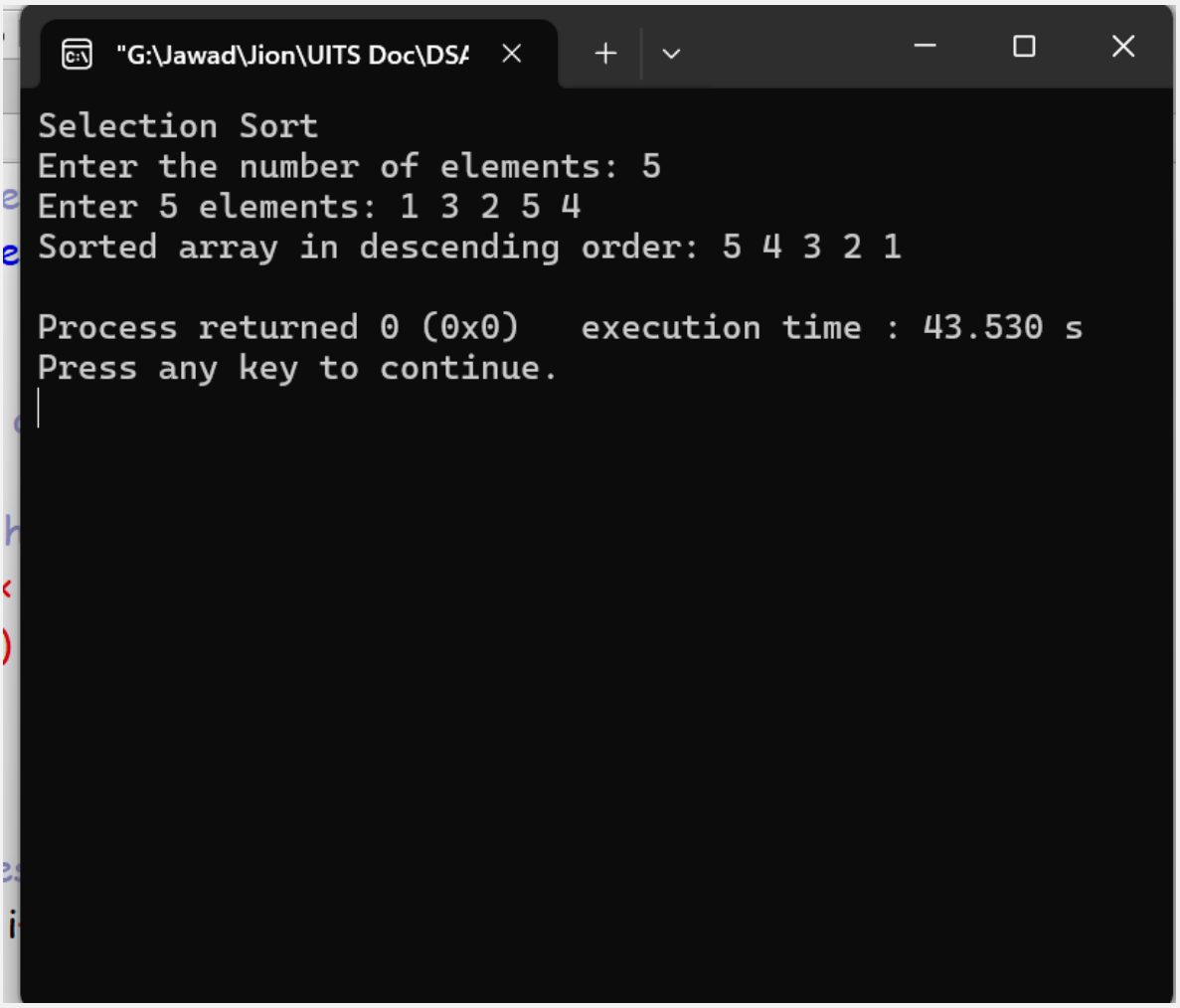


- **Selection Sort-**

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

int main () {
    cout<<"Selection Sort"<<endl;

    int num;

    cout << "Enter the number of elements: ";
    cin >> num;

    int arr[num];

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

    for (int i = 0; i < num - 1; i++) {
        int maxIndex = i;
```

```
        for (int j = i + 1; j < num; j++) {
            if (arr[j] > arr[maxIndex]) {
                maxIndex = j;
            }
        }

        if (maxIndex != i) {
            int temp = arr[i];
            arr[i] = arr[maxIndex];
            arr[maxIndex] = temp;
        }
    }

    cout << "Sorted array in descending order: ";
    for (int i = 0; i < num; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output:



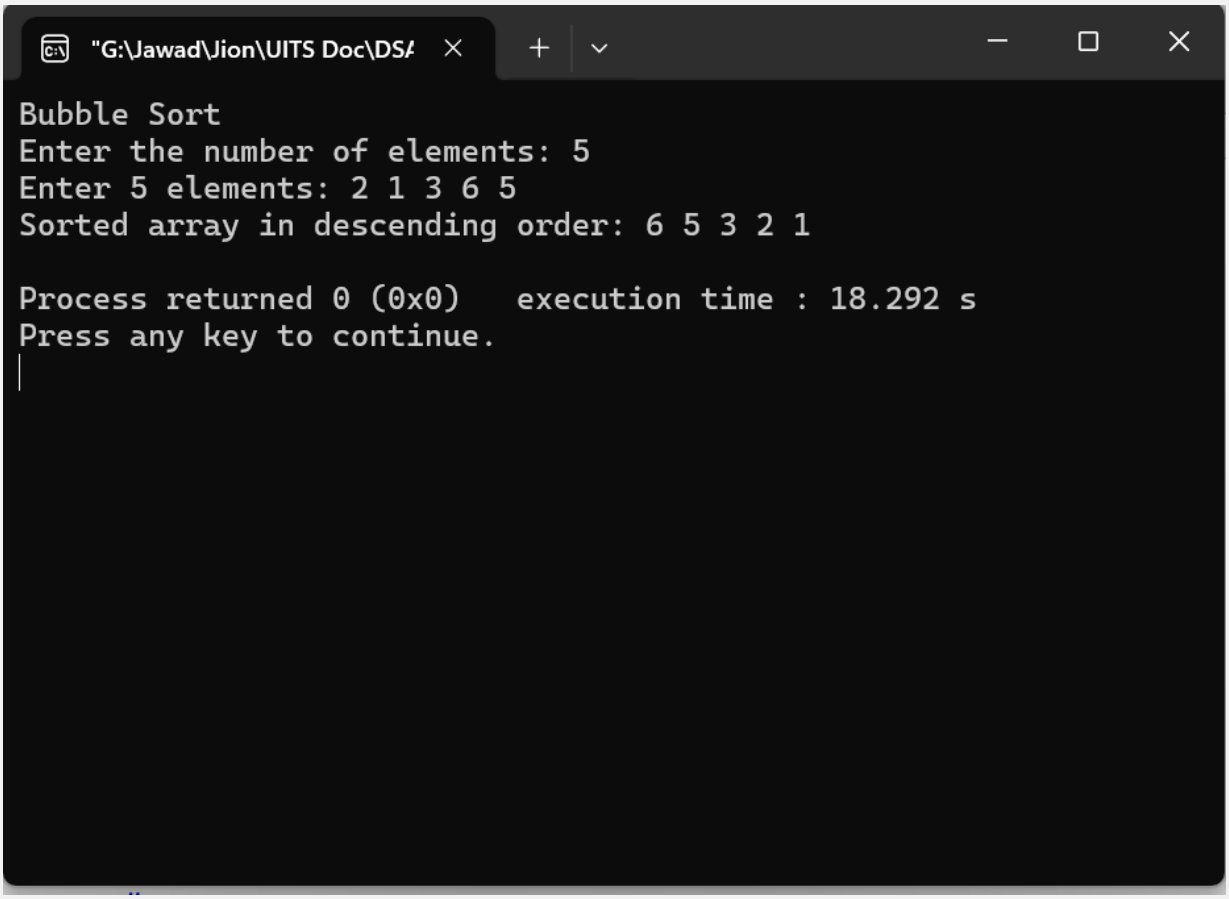
- **Bubble Sort-**

```
#include <iostream>

using namespace std;


int main () {

    cout<<"Bubble Sort"<<endl;
```

```
int num;
cout << "Enter the number of elements: ";
cin >> num;

int arr[num];
cout << "Enter " << num << " elements: ";
for (int j = 0; j < num; j++) {
    cin >> arr[j];
}
for (int j = 0; j < num - 1; j++) {
    for (int k = 0; k < num - j - 1; k++) {

        if (arr[k] < arr[k + 1]) {
            int temp = arr[k];
            arr[k] = arr[k + 1];
            arr[k + 1] = temp;
        }
    }
}
cout << "Sorted array in descending order: ";

for (int j = 0; j < num; j++) {
```

```
        cout << arr[j] << " ";
    }
cout << endl;


    return 0;
}
```

Output:



Linked List

- **Singly Linked List-**

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

struct Node {
    int data;
    Node* next;
};
class LinkedList {
    Node* head;

public:
    LinkedList() {
        head = NULL;
    }

    void insert(int value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = NULL;
```

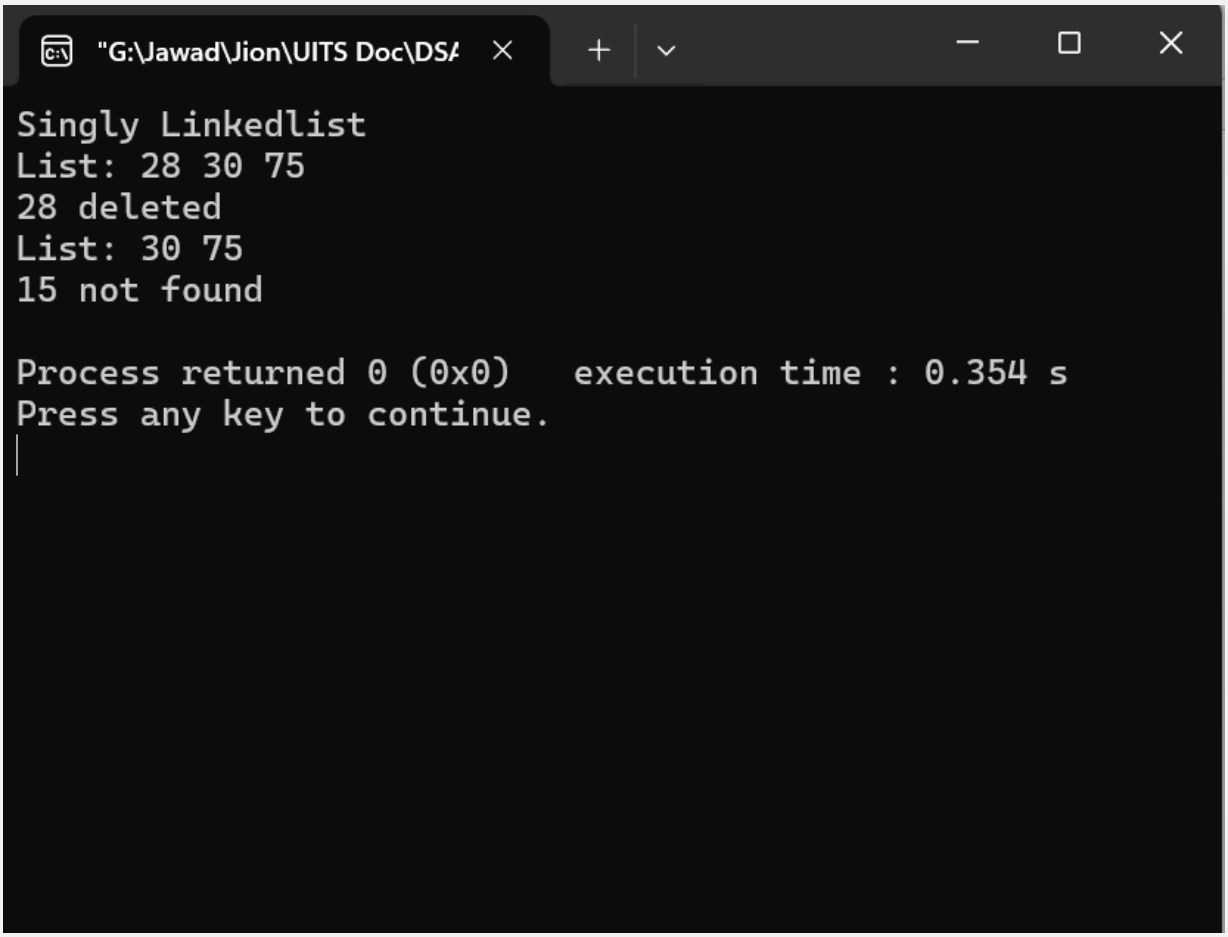
```
    if (head == NULL) {
        head = newNode;
        return;
    }

    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}
void display() {
    if (head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    cout << "List: ";
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
```

```
    }
    cout << endl;
}
void deleteNode(int value) {

    if (head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << value << " deleted" << endl;
        return;
    }

    Node* current = head;
    Node* previous = NULL;
    while (current != NULL && current->data != value) {
        previous = current;
        current = current->next;
```

```
        }

        if (current == NULL) {
            cout << value << " not found" << endl;
            return;
        }
        previous->next = current->next;
        delete current;
        cout << value << " deleted" << endl;
    }
};
int main () {
cout<<"Singly Linkedlist"<<endl;
    LinkedList list;

    list.insert(28);
    list.insert(30);
    list.insert(75);

    list.display();
    list.deleteNode(28);
```

```
    list.display();
    list.deleteNode(15);


    return 0;
}
```

Output:



- **Doubly Linked list-**

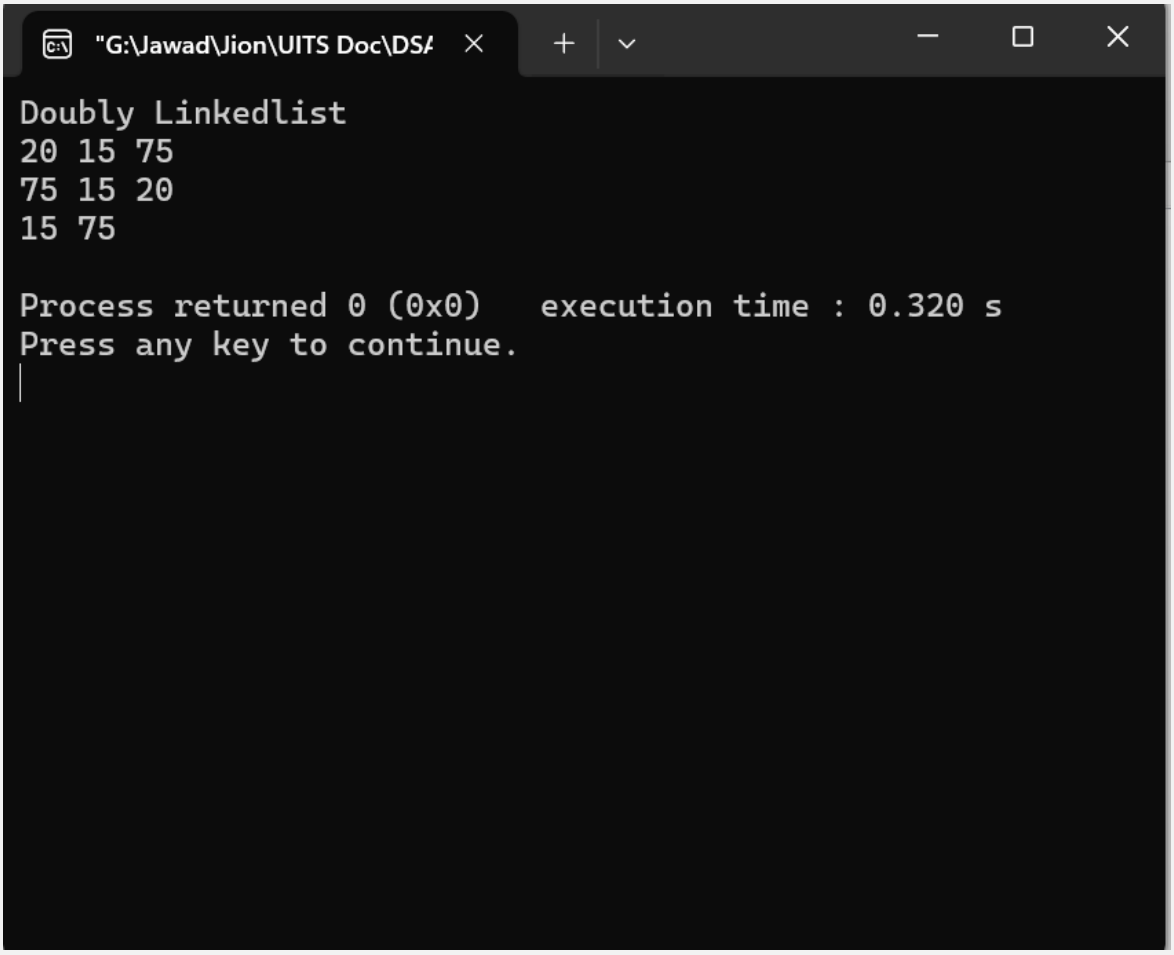
```
#include <iostream>
using namespace std;

struct Element {
    int value;
    Element* previous;
    Element* next;
};
class DoublyLinked {
private:
    Element* start;

public:
    DoublyLinked() : start(nullptr) {}

    void Beginning(int val) {
        Element* node = new Element{val, nullptr, start};
        if (start) start->previous = node;
        start = node;
    }
```

```
void End(int val) {
    Element* node = new Element{val, nullptr, nullptr};
    if (!start) {
        start = node;
        return;
    }
    Element* current = start;
    while (current->next) current = current->next;
    current->next = node;
    node->previous = current;
}
void removeNode(int val) {
    Element* current = start;
    while (current && current->value != val) current = current->next;
    if (!current) return;
    if (current->previous) current->previous->next = current->next;
    if (current->next) current->next->previous = current->previous;
    if (current == start) start = current->next;
    delete current;
}
void Forward() {
    Element* current = start;
```

```
        while (current) {
            cout << current->value << " ";
            current = current->next;
        }
        cout << endl;
    }
    void Backward() {
        Element* current = start;
        if (!current) return;
        while (current->next) current = current->next;
        while (current) {
            cout << current->value << " ";
            current = current->previous;
        }
        cout << endl;
    }
};
int main() {
  cout<<"Doubly Linkedlist"<<endl;

    DoublyLinked list;
    list.Beginning(15);
```

```
    list.Beginning(20);
    list.End(75);
    list.Forward();
    list.Backward();
    list.removeNode(20);
    list.Forward();
    return 0;
}
```

Output:



- **Circular Linked List-**

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

struct Node {
    int data;
    Node* next;
};

class CircularLinkedList {
private:
    Node* last;

public:
    CircularLinkedList() : last(nullptr) {}

    void insertEnd(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        if (last == nullptr) {
            last = newNode;
```

```
        last->next = last;
    } else {
        newNode->next = last->next;
        last->next = newNode;
        last = newNode;
    }
}

void insertBeginning(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    if (last == nullptr) {
        last = newNode;
        last->next = last;
    } else {
        newNode->next = last->next;
        last->next = newNode;
    }
}

void display() {
    if (last == nullptr) return;
```

```
    Node* temp = last->next;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != last->next);
    cout << endl;
}

void deleteNode(int value) {
    if (last == nullptr) return;
    Node* current = last->next;
    Node* previous = last;
    do {
        if (current->data == value) {
            if (current == last) {
                if (last->next == last) {
                    delete last;
                    last = nullptr;
                } else {
                    previous->next = current->next;
                    last = previous;
                    delete current;
```

```
                }
            } else {
                previous->next = current->next;
                delete current;
            }
            return;
        }
        previous = current;
        current = current->next;
    } while (current != last->next);
  }
};

int main() {
  cout<<"Circular Linkedlist"<<endl;
  CircularLinkedList cll;
cll.insertEnd(10);
cll.insertEnd(20);
cll.insertEnd(30);
cll.insertBeginning(5);
cll.display();
cll.deleteNode(20);
```

```
cll.display();

return 0;

}
```

Output: