

Superior University, Lahore



Name: Wardah Shoaib

Roll No: BSSEM-S24-095

DSA LAB TASKS

Submitted to: Sir Rasikh Ali

Lab Task 01: Pointers

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

int main() {
    int num = 10;
    int* ptr = &num;

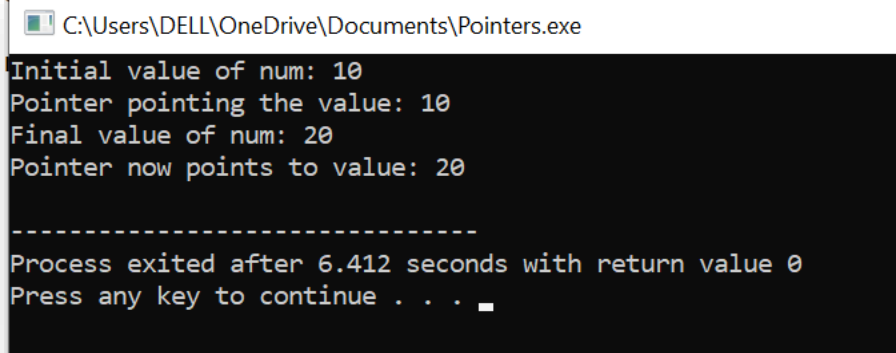
    cout << "Initial value of num: " << num << endl;
    cout << "Pointer pointing the value: " << *ptr << endl;

    *ptr = 20;

    cout << "Final value of num: " << num << endl;
    cout << "Pointer now points to value: " << *ptr << endl;

    return 0;
}
```

- It declares an integer num and sets it to 10.
- A pointer ptr is then created to store the memory address of num.
- Then, the value of num is changed through the pointer by setting *ptr = 20; which updates num directly. The program prints the new values to show that num has changed.



```
C:\Users\DELL\OneDrive\Documents\Pointers.exe
Initial value of num: 10
Pointer pointing the value: 10
Final value of num: 20
Pointer now points to value: 20

-----
Process exited after 6.412 seconds with return value 0
Press any key to continue . . .
```

Lab Task 02: Big O Notation (Loops and Arrays)

```

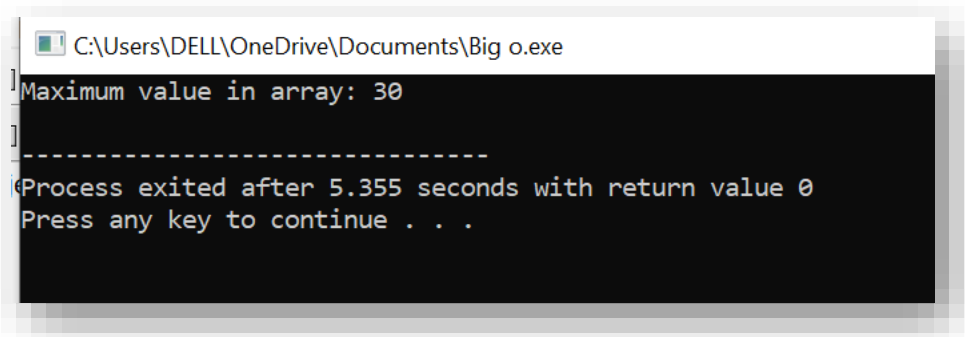
1  #include <iostream>
2  using namespace std;
3
4  int findMax(int arr[], int n) {
5      int maxVal = arr[0];
6      for (int i = 1; i < n; i++) {
7          if (arr[i] > maxVal) {
8              maxVal = arr[i];
9          }
10     }
11     return maxVal;
12 }
13
14 int main() {
15     int arr[] = {10, 25, 8, 17, 30, 5};
16     int n = sizeof(arr) / sizeof(arr[0]);
17
18     cout << "Maximum value in array: " << findMax(arr, n) << endl;
19
20     return 0;
21 }
22

```

The time complexity is **O(n)**, where n is the size of the array. This is because the number of operations increases linearly with the number of elements, making it an efficient approach for finding the maximum value.

- The findMax function starts by assuming the first number is the largest.
- If it finds a bigger number, it updates the maximum value.
- After checking all the numbers, it returns the biggest one.
- The program then calls findMax and prints the result.

The time complexity is **O(n)**, meaning it works efficiently even for larger arrays.



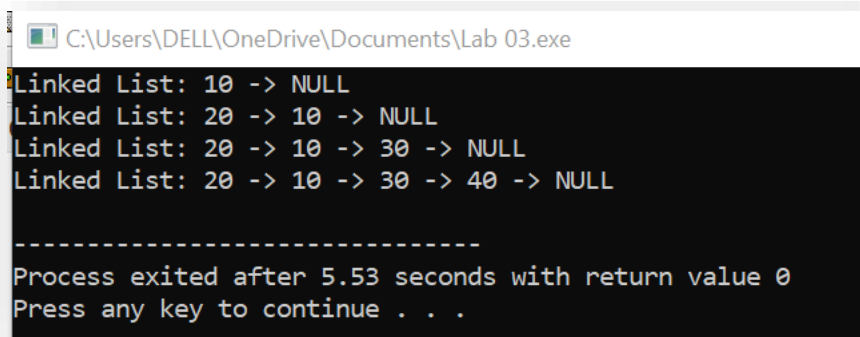
```

C:\Users\DELL\OneDrive\Documents\Big o.exe
Maximum value in array: 30
-----
Process exited after 5.355 seconds with return value 0
Press any key to continue . . .

```

Lab Task 03: Singly Linked List (Insert at End, Insert at Start)

- The Node class stores a number and a pointer to the next node.
- The SinglyLinkedList class manages the list, starting with an empty head.
- The insertAtStart function adds a new node at the beginning by making it the new head,
- while insertAtEnd adds a new node at the end by going through the list until it finds the last node.

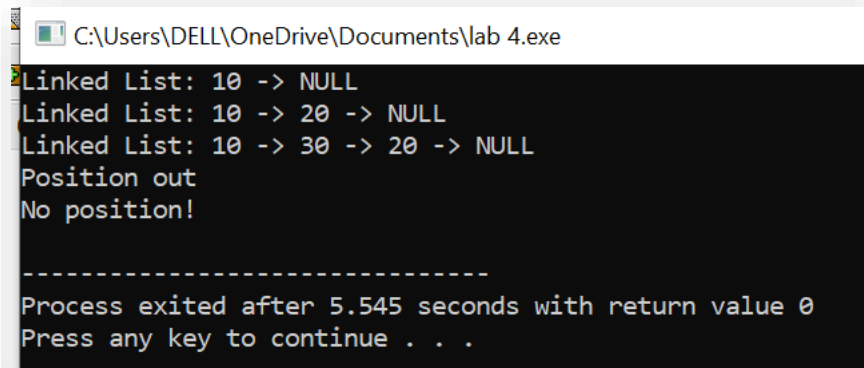


```
C:\Users\DELL\OneDrive\Documents\Lab 03.exe
Linked List: 10 -> NULL
Linked List: 20 -> 10 -> NULL
Linked List: 20 -> 10 -> 30 -> NULL
Linked List: 20 -> 10 -> 30 -> 40 -> NULL

-----
Process exited after 5.53 seconds with return value 0
Press any key to continue . . .
```

Lab Task 04: Singly Linked List (Insert at Specific Location)

- The SinglyLinkedList class manages the list, starting with an empty head.
- The insertAtPosition function adds a new node at the given position.
- If the position is 1, the new node becomes the head.
- If the position is too far, it shows an error.



```
C:\Users\DELL\OneDrive\Documents\lab 4.exe
Linked List: 10 -> NULL
Linked List: 10 -> 20 -> NULL
Linked List: 10 -> 30 -> 20 -> NULL
Position out
No position!

-----
Process exited after 5.545 seconds with return value 0
Press any key to continue . . .
```

Lab Task 05: Singly Linked List (Display Nodes)

(int val, int pos) function inserts a node at a specific position. If the list is empty or pos is 1, the new node becomes the head. Otherwise, it traverses to the correct position and inserts the node.

The display() function prints all nodes in order. **Other display functions retrieve specific nodes:**

- displayFirstNode() prints the first node.
- displayLastNode() finds and prints the last node.
- displayNthNode(n) prints the **Nth node**, checking for out-of-range errors.
- displayCenterNode() uses **slow and fast pointers** to find and display the middle node efficiently.

```
C:\Users\DELL\OneDrive\Documents\lab 05.exe
Linked List: 10 -> NULL
Linked List: 10 -> 20 -> NULL
Linked List: 10 -> 30 -> 20 -> NULL
Linked List: 10 -> 30 -> 20 -> 40 -> NULL
Linked List: 10 -> 30 -> 50 -> 20 -> 40 -> NULL
First Node: 10
Last Node: 40
Node at position 3: 50
Center Node: 50

-----
Process exited after 5.521 seconds with return value 0
Press any key to continue . . .
```

Lab Task 06: Singly Linked List (Delete Nodes)

The **SinglyLinkedList** class manages the list using a head pointer and provides functions to insert, delete, and display nodes.

Insertion & Deletion:

- **insertAtPosition(val, pos):** Inserts a new node at a given position. If at pos = 1 or empty, it becomes the new head; otherwise, it finds the correct position and inserts the node.
- **deleteFirstNode():** Removes the first node by updating head to the next node.
- **deleteLastNode():** If one node exists, deletes it. Otherwise, finds the second last node and removes the last one.
- **deleteNthNode(n):** Removes the node at position n; calls deleteFirstNode() if n == 1.
- **deleteCenterNode():** Uses **slow and fast pointers** to find and delete the middle node efficiently.

Display:

- **display()** prints the list in value -> value -> NULL format after every operation.

```
C:\Users\DELL\OneDrive\Documents\lab 06.exe
10 -> NULL
10 -> 20 -> NULL
10 -> 20 -> 30 -> NULL
10 -> 20 -> 30 -> 40 -> NULL
10 -> 20 -> 30 -> 40 -> 50 -> NULL
20 -> 30 -> 40 -> 50 -> NULL
20 -> 30 -> 40 -> NULL
20 -> 40 -> NULL
20 -> NULL

-----
Process exited after 5.609 seconds with return value 0
Press any key to continue . . .
```

Lab Task 07: Doubly Linked List (Insert & Display Nodes)

The **DoublyLinkedList** class manages the list using two pointers: head (first node) and tail (last node). It provides functions to **insert nodes**:

- **insertAtFirst(val)**: Adds a node at the beginning. If empty, updates both head and tail.
- **insertAtLast(val)**: Adds a node at the end. If empty, behaves like insertAtFirst().
- **insertAtNth(val, pos)**: Inserts at a specific position; calls insertAtFirst() if pos == 1. Updates tail if added at the end.
- **insertAtCenter(val)**: Uses **slow & fast pointers** to find the middle and calls insertAtNth().

```

C:\Users\DELL\OneDrive\Documents\lab 07.exe
10 <-> NULL
10 <-> 20 <-> NULL
10 <-> 15 <-> 20 <-> NULL
10 <-> 25 <-> 15 <-> 20 <-> NULL
List in order: 10 <-> 25 <-> 15 <-> 20 <-> NULL
List in reverse order: 20 <-> 15 <-> 25 <-> 10 <-> NULL

-----
Process exited after 11.29 seconds with return value 0
Press any key to continue . . .

```

Lab Task 08: Merge two LinkedLists

The program **creates and merges** both **Singly and Doubly Linked Lists**.

1. **Singly Linked List (SinglyLinkedList)**
 - Each **node** stores an integer and a next pointer.
 - **insert(val)** adds a node at the end.
 - **merge(other)** links the last node of the first list to the second list's head.
2. **Doubly Linked List (DoublyLinkedList)**
 - Each **node** has a next (next node) and prev (previous node) pointer.
 - **insert(val)** adds a node at the end and updates tail for easy traversal.
 - **merge(other)** links both lists and updates prev pointers to maintain bidirectional links.

```

Singly Linked List 1: 1 -> 3 -> 5 -> NULL
Singly Linked List 2: 2 -> 4 -> 6 -> NULL
Merged Singly Linked List: 1 -> 3 -> 5 -> 2 -> 4 -> 6 -> NULL
1st Doubly Linked List: 10 <-> 30 <-> 50 <-> NULL
2nd Doubly Linked List: 20 <-> 40 <-> 60 <-> NULL
Merged Doubly Linked List: 10 <-> 30 <-> 50 <-> 20 <-> 40 <-> 60 <-> NULL

-----
Process exited after 7.874 seconds with return value 0
Press any key to continue . . .

```


Lab Task 09: Circular Linked List (Insert & Display Nodes)

Node Structure → Stores an integer (data) and a next pointer (points to the next node).

CircularLinkedList Class → Uses head to track the first node.

Insertion Methods:

- **insertAtFirst(val)** → If empty, creates a node pointing to itself; otherwise, updates the last node's next and moves head.
- **insertAtLast(val)** → Traverses to the last node and inserts a new one, maintaining the circular link.
- **insertAtNth(val, pos)** → Finds the correct position and inserts without breaking the circular structure.
- **insertAtCenter(val)** → Uses **slow & fast pointers** to locate the middle and calls insertAtNth().

Display Methods:

- **display()** → Prints the list in order, stopping when it loops back to head.
- **displayReverse()** → Stores values in an array, then prints them in reverse

```
C:\Users\DELL\OneDrive\Documents\lab 09.exe
10 -> (back to head)
10 -> 30 -> (back to head)
10 -> 20 -> 30 -> (back to head)
10 -> 20 -> 25 -> 30 -> (back to head)
Circular Linked List: 10 -> 20 -> 25 -> 30 -> (back to head)
Reverse: 30 -> 25 -> 20 -> 10 -> (back to head)

-----
Process exited after 5.966 seconds with return value 0
Press any key to continue . . .
```