

***Name:*** *Taha Nawaz*

***Section:*** *Bssem-3A*

***ROLL NO:*** *Bssem-015*

***Assignment DSA Lab***

***SUBMITTED TO:***

***Sir Rasikh ALi***

***LAB TASK***

***TASK #1***

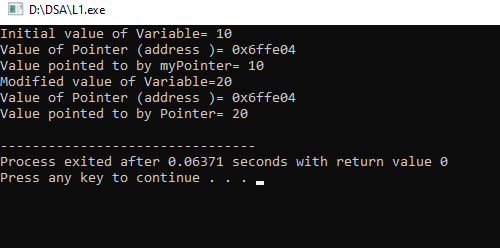
***Variable Declaration:*** *An integer variable is declared and initialized to 10.*

***Pointer Declaration:*** *A pointer is created to hold the address of the integer variable.*

***Display Initial Values:*** *The program prints the initial value of the variable, the address stored in the pointer, and the value pointed to by the pointer.*

***Modify Value:*** *The value of the variable is changed to 20 using the pointer (\*pointer = 20;).*

***Display Modified Values:*** *The program prints the updated value of the variable, the same address in the pointer, and the new value pointed to by the pointer.*

**

***TASK #2***

***Function findMax:***

*It takes an array arr and its size n as parameters.*

*It initializes max to the first element of the array.*

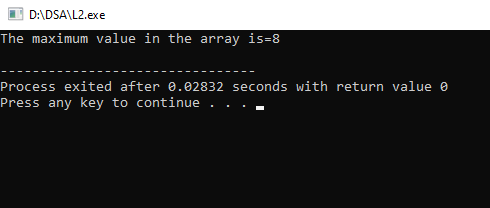
*It loops through the array starting from the second element (index 1) and updates max whenever it finds a larger value.*

***main Function:***

*An example array is defined.*

*The size of the array is calculated.*

*The findMax function is called, and the result is printed.*

**

***TASK #3***

***Node Class:***

*Each node contains an integer data and a pointer next to the next node.*

*The constructor initializes data and sets next to NULL.*

***SinglyLinkedList Class:***

*Contains a pointer head to the first node of the list.*

***Constructor:*** *Initializes head to NULL.*

***insertAtStart(int value):*** *Creates a new node, points it to the current head, and updates the head to the new node. It then calls display() to show the list.*

***insertAtEnd(int value):*** *Creates a new node and traverses to the end of the list to link the new node. It also calls display() to show the updated list.*

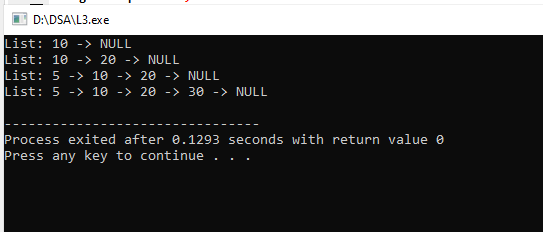
***display():*** *Traverses the list and prints the data of each node****.***

***Destructor:*** *Frees the memory allocated for the nodes when the list is destroyed.*

***Main Function:***

*Creates an instance of SinglyLinkedList.*

*Inserts nodes at the start and end, displaying the list after each insertion*

*.*

***TASK #4***

***Node Class****: Represents a single node in the linked list, containing an integer****data****and a pointer****next****to the next node.*

***SinglyLinkedList Class****:*

*Contains a pointer****head****to the first node of the list.*

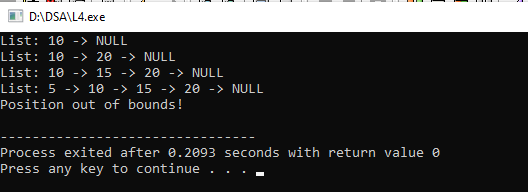
***insertAtPosition(int value, int position)****:*

*Checks if the position is valid (non-negative).*

* + - *If the position is****0****, it inserts the new node at the start of the list.*
    - *Inserts the new node at the specified position.*
  + ***display()****: Traverses the list and prints the values of the nodes.*
  + ***Destructor****: Frees the memory allocated for the nodes when the list is destroyed.*

***Main Function****:*

* + *Creates an instance of****SinglyLinkedList*** *AND Inserts nodes at various positions, demonstrating the functionality of the insertion method.*

**

***TASK #5***

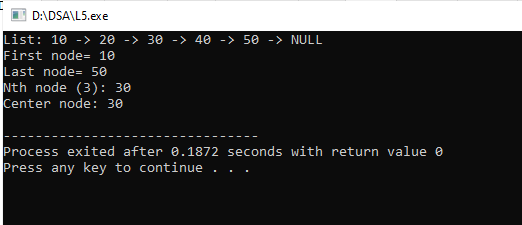
***Node Class****: Represents a single node in the linked list, containing an integer****data****and a pointer****next****.*

***SinglyLinkedList Class****:*

* + *Contains a pointer****head****to the first node of the list.*
  + ***insertAtEnd(int value)****: Inserts a new node at the end of the list.*
  + ***displayFirstNode()****: Displays the data of the first node.*
  + ***displayLastNode()****: Traverses the list to find and display the data of the last node.*
  + ***displayNthNode(int n)****: Displays the data of the Nth node, checking for valid positions.*
  + ***displayCenterNode()****: Uses the slow and fast pointer technique to find and display the center node of the list.*
  + ***display()****: Displays the entire list.*
  + ***Destructor****: Frees the memory allocated for the nodes when the list is destroyed.*

***Main Function****:*

*Creates an instance of****SinglyLinkedList****. AND Inserts nodes into the list.Calls the functions to display the first node, last node*

*,*

***TASK #6***

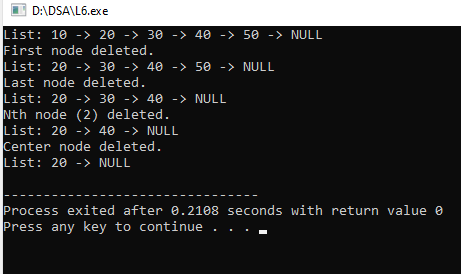
***Node Class:*** *Represents a single node with an integer data and a pointer next****.***

***SinglyLinkedList Class:***

* + ***insertAtEnd(int value):*** *Adds a new node with the given value at the end of the list.*
  + ***deleteFirstNode():*** *Removes the first node from the list****.***
  + ***deleteLastNode():*** *Removes the last node from the list.*
  + ***deleteNthNode(int n):*** *Removes the node at the specified position n****.***
  + ***deleteCenterNode():*** *Removes the center node using the slow and fast pointer technique.*
  + ***display():*** *Prints the entire list****.***
  + ***Destructor:*** *Frees memory for all nodes when the list is destroyed.*

***Main Function:***

*Creates a SinglyLinkedList instance.Inserts nodes and demonstrates deletion functions, displaying the list after each operation.*

**

***TASK #7***

***Node Class:*** *Represents a single node in the doubly linked list, containing:*

* + *int data: The value stored in the node.*
  + *Node\* next: A pointer to the next node.*
  + *Node\* prev: A pointer to the previous node.*

1. ***DoublyLinkedList Class:***

*Contains a pointer head to the first node of the list.*

*Inserts a new node at the beginning of the list.*

*Inserts a new node at the end of the list.*

*Inserts a new node at the specified position n.*

*Inserts a new node in the center of the list using the slow and fast pointer technique.*

*Displays the list from head to tail.*

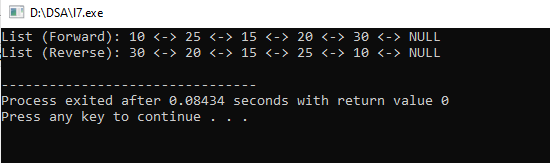
*Displays the list from tail to head.*

*Frees the memory allocated for the nodes when the list is destroyed.*

1. ***Main Function:***

*Creates an instance of DoublyLinkedList.*

*Inserts nodes at various positions and displays the list in both forward and reverse order.*

******

***TASK #8.1***

1. ***Node Class:*** *Represents a single node in the singly linked list, containing:*

*int data: The value stored in the node.*

*Node\* next: A pointer to the next node.*

1. ***SinglyLinkedList Class:***

*Contains a pointer head to the first node****.***

***insertAtEnd(int value):*** *Adds a new node with the specified value at the end of the list.*

***getHead():*** *Returns the head of the list.*

***display():*** *Prints the entire list in order****.***

***Destructor:*** *Cleans up memory by deleting all nodes when the list is destroyed.*

1. ***mergeSinglyLinkedLists():*** *Merges two singly linked lists by:*

*Creating a new SinglyLinkedList.*

*Traversing both input lists and inserting their elements into the merged list.*

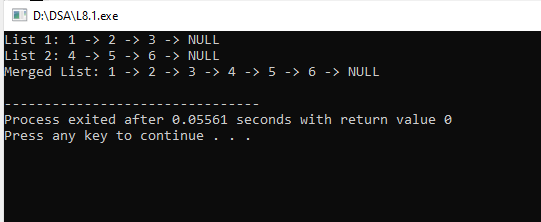
1. ***Main Function:***

*Creates two instances of SinglyLinkedList.*

*Inserts values into both lists.*

*Displays the original lists.*

*Merges the lists and displays the merged list.*

******

***TASK #8.2***

***DoublyNode Class:*** *Represents a single node in the doubly linked list, containing:*

***int data:*** *The value stored in the node.*

***DoublyNode\* next:*** *A pointer to the next node.*

***DoublyNode\* prev:*** *A pointer to the previous node.*

***DoublyLinkedList Class:***

* + *Contains a pointer head to the first node****.***
  + ***insertAtEnd(int value):*** *Adds a new node with the specified value at the end of the list.*
  + ***getHead():*** *Returns the head of the list****.***
  + ***display():*** *Prints the entire list in order****.***
  + ***Destructor:*** *Cleans up memory by deleting all nodes when the list is destroyed.*

1. ***mergeDoublyLinkedLists():*** *Merges two doubly linked lists by:*

*Creating a new DoublyLinkedList.*

*Traversing both input lists and inserting their elements into the merged list.*

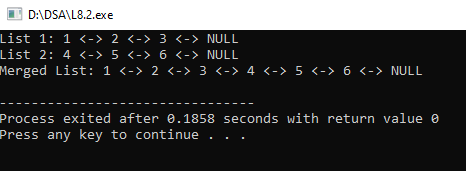
1. ***Main Function:***

*Creates two instances of DoublyLinkedList.*

*Inserts values into both lists.*

*Displays the original lists.*

*Merges the lists and displays the merged list.*

******

***TASK #9***

1. ***Node Class:***
   * *Represents a single node in the circular linked list.*
   * *Contains:*
     + *int data: The value stored in the node.*
     + *Node\* next: Pointer to the next node.*
     + *Node\* prev: Pointer to the previous node (for reverse traversal).*
2. ***CircularLinkedList Class:***
   * *Contains a pointer head to the first node.*
   * *insertAtFirst(int value): Inserts a new node at the beginning of the list.*
   * *insertAtLast(int value): Inserts a new node at the end of the list.*

*Inserts a new node at the specified position n.*

*Inserts a new node in the center of the list.*

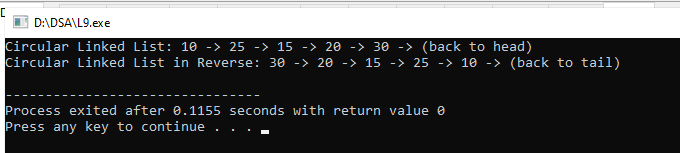
*Prints the list in order, starting from the head.*

*Prints the list in reverse order, starting from the tail.*

*Cleans up memory by deleting all nodes when the list is destroyed.*

1. ***Main Function:***

*Creates an instance of CircularLinkedList. And then Inserts several values into the list.After that Displays the list in both normal and reverse order.*

**