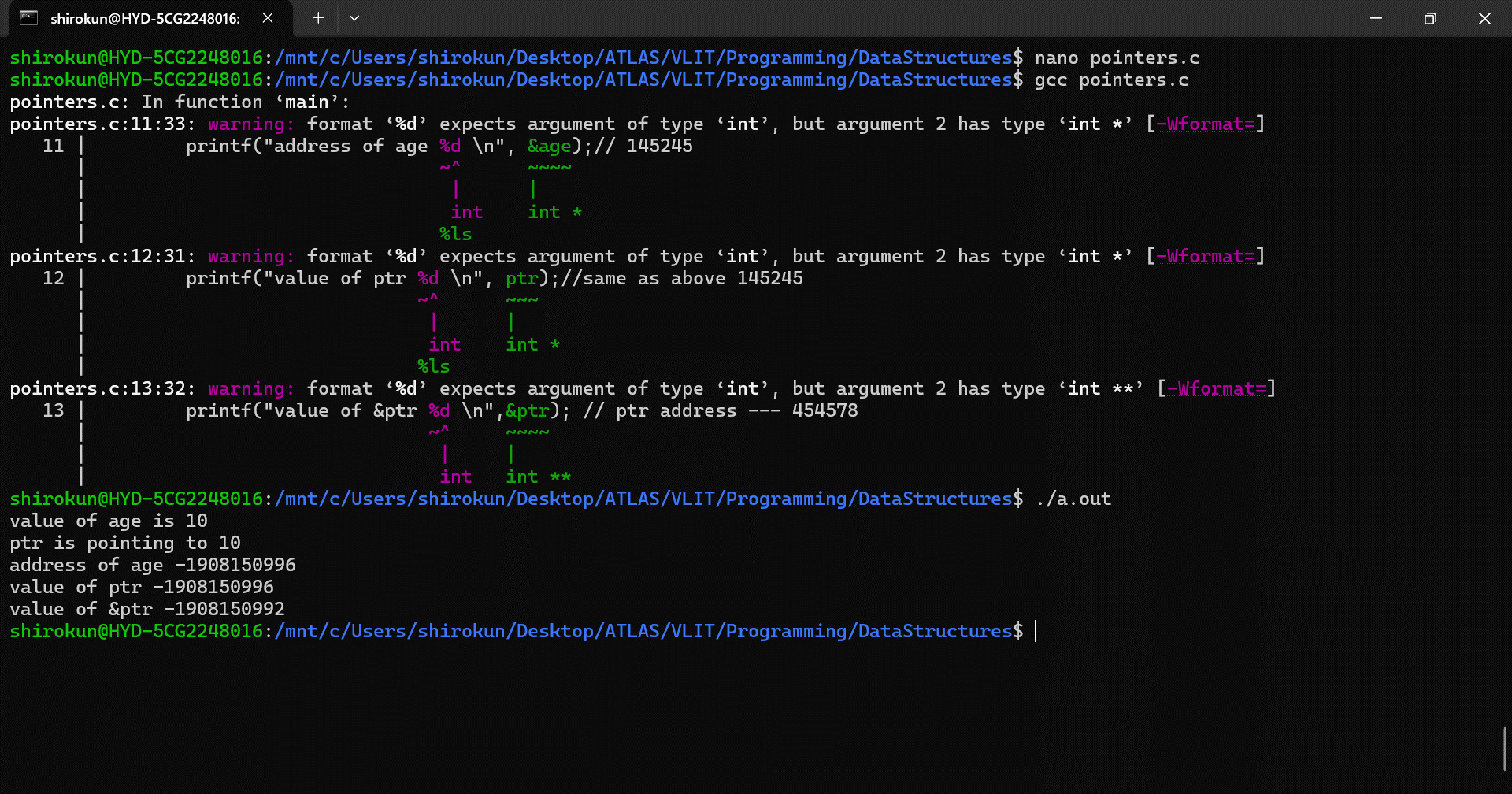
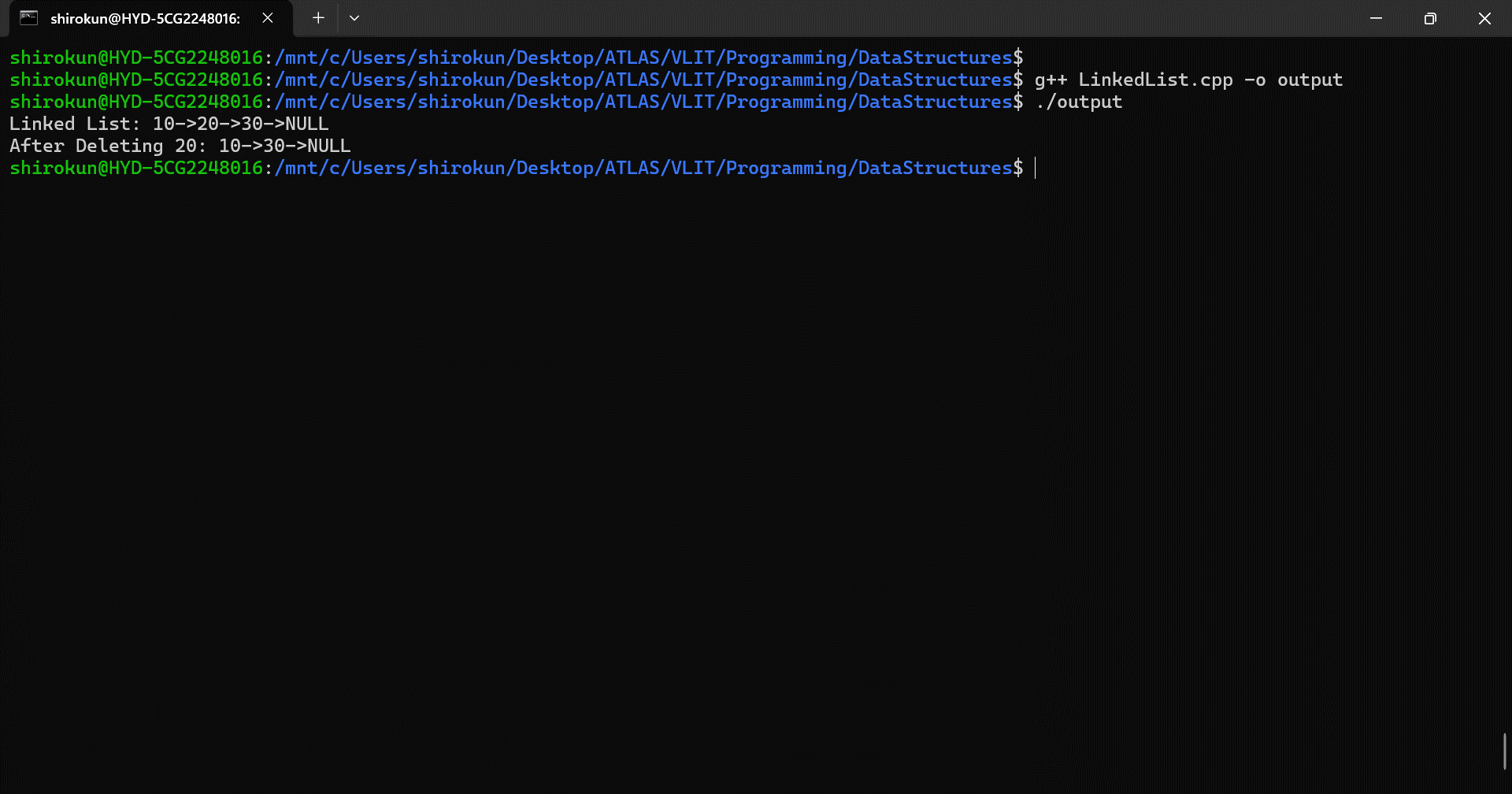
Day 13 – 28/06/2025

// Task001: Pointers.  
  
#include<stdio.h>  
void main(){  
 int age;  
 int \*ptr;  
 age = 10;  
 ptr = &age;  
 printf("value of age is %d\n", age);   
 printf("ptr is pointing to %d\n", \*ptr);  
 printf("address of age %d \n", &age);   
 printf("value of ptr %d \n", ptr);  
 printf("value of &ptr %d \n",&ptr);  
  
}



// Task002: LinkedList  
  
#include <bits/stdc++.h>  
using namespace std;  
  
// Define a Node class  
class Node{  
 public:  
 int data; // Data part of the node  
 Node\* next; // Pointer to the next node  
  
 // Constructor for convenience  
 Node(int value) : data(value), next(nullptr) {}  
};  
  
// Class for singly linked list  
class Linkedlist{  
private:  
 Node\* head; // Pointer to the head of the list  
  
public:  
 // Constructor to initialize an empty list  
 Linkedlist(){  
 head = nullptr;  
 }  
 // Function to insert a node at the end  
 void insertAtEnd(int value){  
 Node\* newNode = new Node(value);  
 if(head == nullptr){  
 head = newNode; // If list is empty, make newNode the head  
 }  
 else{  
 Node\* temp = head;  
 while (temp->next != nullptr){  
 temp = temp->next; // Traverse to the last node  
 }  
 temp->next = newNode; // Link the last node to newNode  
 }  
 }  
  
 // Function to delete a Node by Value  
 void deleteByValue(int value){  
 if(head == nullptr){  
 return;  
 }  
 if(head->data == value){  
 Node\* temp = head;  
 head = head->next; // Move head to the next node  
 delete temp; // Free memory of the deleted node  
 return;  
 }  
 Node\* temp = head;  
 while(temp->next && temp->next->data != value){  
 temp = temp->next; // Traverse to find the node to delete  
 }  
 if(temp->next){  
 Node\* nodeToDelete = temp->next;  
 temp->next = temp->next->next; // Unlink the node  
 delete nodeToDelete; //Free Memory  
 }  
 }  
 // Function to display the list  
 void display(){  
 Node\* temp = head;  
 while(temp != nullptr){  
 cout << temp->data << "->";  
 temp = temp->next;  
 }  
 cout << "NULL" <<endl;  
 }

// Destructor to free all allocated memory  
 ~Linkedlist() {  
 Node\* temp;  
 while (head) {  
 temp = head;  
 head = head->next;  
 delete temp;  
 }  
 }  
};  
int main() {  
 Linkedlist list;  
  
 list.insertAtEnd(10);  
 list.insertAtEnd(20);  
 list.insertAtEnd(30);  
  
 cout << "Linked List: ";  
 list.display();  
  
 list.deleteByValue(20);  
  
 cout << "After Deleting 20: ";  
 list.display();  
  
 return 0;  
}



// Task003: Use the above code to create a  
// Java code which creates a linked list.  
  
package LinkedLists;  
  
class Node {  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
  
class LinkedList {  
 Node head;  
 public LinkedList() {  
 head = null;  
 }  
  
 public void add(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 } else {  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
 }  
  
 public void deleteAtPosition(int position) {  
 if (head == null) {  
 System.*out*.println("List is empty");  
 return;  
 }  
  
 if (position < 0) {  
 System.*out*.println("Invalid position");  
 return;  
 }  
  
 if (position == 0) {  
 head = head.next;  
 return;  
 }  
  
 Node current = head;  
 int currentIndex = 0;  
  
 while (current != null && currentIndex < position - 1) {  
 current = current.next;  
 currentIndex++;  
 }  
  
 if (current == null || current.next == null) {  
 System.*out*.println("Position out of bounds");  
 return;  
 }  
  
 current.next = current.next.next;  
 }  
  
  
 public void printList() {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + "->");  
 current = current.next;  
 }  
 System.*out*.println("NULL");  
 }  
}  
  
public class Task003 {  
 public static void main(String[] args) {  
 LinkedList linkedList = new LinkedList();  
 linkedList.add(10);  
 linkedList.add(20);  
 linkedList.add(30);  
 linkedList.add(40);  
 linkedList.add(50);  
 linkedList.printList();  
 }  
}

