|  |  |
| --- | --- |
| **Cycle in C++** | |
| #include <iostream>  using namespace std;  // Definition of a Node in the linked list  struct Node {  int val;  Node\* next;  Node(int x) {  val = x; // Assigns the parameter x to the member variable val  next = nullptr; // Initializes the next pointer to nullptr  }  };  // Function to detect if there is a cycle in the linked list  bool hasCycle(Node\* head) {  if (head == nullptr || head->next == nullptr) {  return false;  }  Node\* slow = head;  Node\* fast = head;  while (fast != nullptr && fast->next != nullptr) {  slow = slow->next;  fast = fast->next->next;  if (slow == fast) {  return true; // Cycle detected  }  }  return false; // No cycle found  }  int main() {  // Creating a linked list: 1 -> 2 -> 3 -> 4 -> 5  Node\* head = new Node(1);  head->next = new Node(2);  head->next->next = new Node(3);  head->next->next->next = new Node(4);  head->next->next->next->next = new Node(5);  // Creating a cycle by pointing the next of last node to the node with value 3 (index 2)  Node\* tail = head;  while (tail->next != nullptr) {  tail = tail->next;  }  Node\* cycleNode = head->next->next; // Node with value 3  tail->next = cycleNode;  // Check if the cycle is present  cout << (hasCycle(head) ? "Cycle is present" : "No cycle") << endl;  return 0;  } | **Core Logic Recap**  Floyd’s algorithm uses:   * slow: moves 1 step at a time. * fast: moves 2 steps at a time.   If there's a cycle, slow and fast will eventually meet inside the loop.  **🧪 Dry Run**  **Linked List:**  1 -> 2 -> 3 -> 4 -> 5  ^ |  |\_\_\_\_\_\_\_\_|  Cycle: 5 -> 3 creates a loop back to node with value 3.  **🧾 Dry Run Table**   | **Iteration** | **slow value** | **fast value** | **Notes** | | --- | --- | --- | --- | | 1 | 2 | 3 | both moved: slow+1, fast+2 | | 2 | 3 | 5 | fast jumps into cycle | | 3 | 4 | 4 | slow == fast → cycle found |   **🧠 Output:**  Cycle is present |
| Output:- Cycle is present | |