**Experiment 5**

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C013 | Name: Ashmit Jain |
| Class : B | Batch : B1 |
| Date of Experiment: | Date of Submission: 8/9/24 |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your Matlab code completed during the 2 hours of practical in the lab here)***

**Task1:**

**1.1**



**1.2**





**1.3**

****

**Task1:**

**Observations**

1. **Task 1.1: Creating and Displaying a Linked List**
   * The linked list structure was successfully created with nodes containing integer data.
   * The displayList function correctly traversed the list and displayed the elements in the order they were linked.
   * The linked list terminates with NULL, indicating the end of the list, which was displayed as NULL at the end of the output.
2. **Task 1.2: Inserting Nodes at Different Positions**
   * **Insertion at the Beginning**:
     + Nodes were successfully added to the beginning of the list, and the head pointer was correctly updated.
     + The list displayed after insertion at the beginning showed that the new node was at the start, pushing previous elements to the right.
   * **Insertion at the End**:
     + Nodes were successfully appended to the end of the list.
     + The list displayed after insertion at the end showed that the new node appeared at the end of the list.
   * **Insertion at a Specific Position**:
     + Nodes were inserted at specific positions within the list.
     + The list displayed after insertion at a specific position showed that the new node was correctly placed between the appropriate existing nodes.
     + When attempting to insert at an out-of-bounds position, the program handled the error by displaying an appropriate message.
3. **Task 1.3: Deleting Nodes at Different Positions**
   * **Deletion at the Beginning**:
     + The head node was successfully deleted, and the head pointer was updated to the next node.
     + The list displayed after deletion at the beginning showed that the previous head was removed, and the second node became the new head.
   * **Deletion at the End**:
     + The last node was successfully removed.
     + The list displayed after deletion at the end confirmed that the final node was deleted, and the second-to-last node became the new end, pointing to NULL.
   * **Deletion at a Specific Position**:
     + Nodes were correctly removed from specified positions within the list.
     + The list displayed after deletion at a specific position showed that the targeted node was removed and the links between the remaining nodes were correctly maintained.
     + The program correctly handled cases where the position was out of bounds by displaying an appropriate message.

**Conclusion**

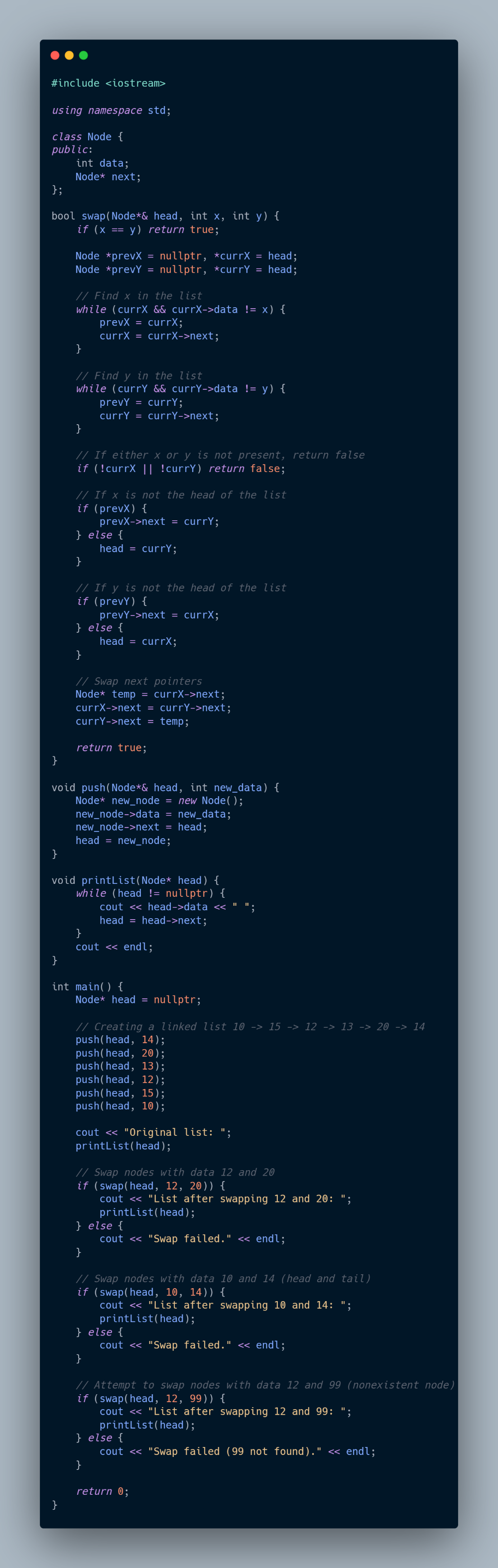
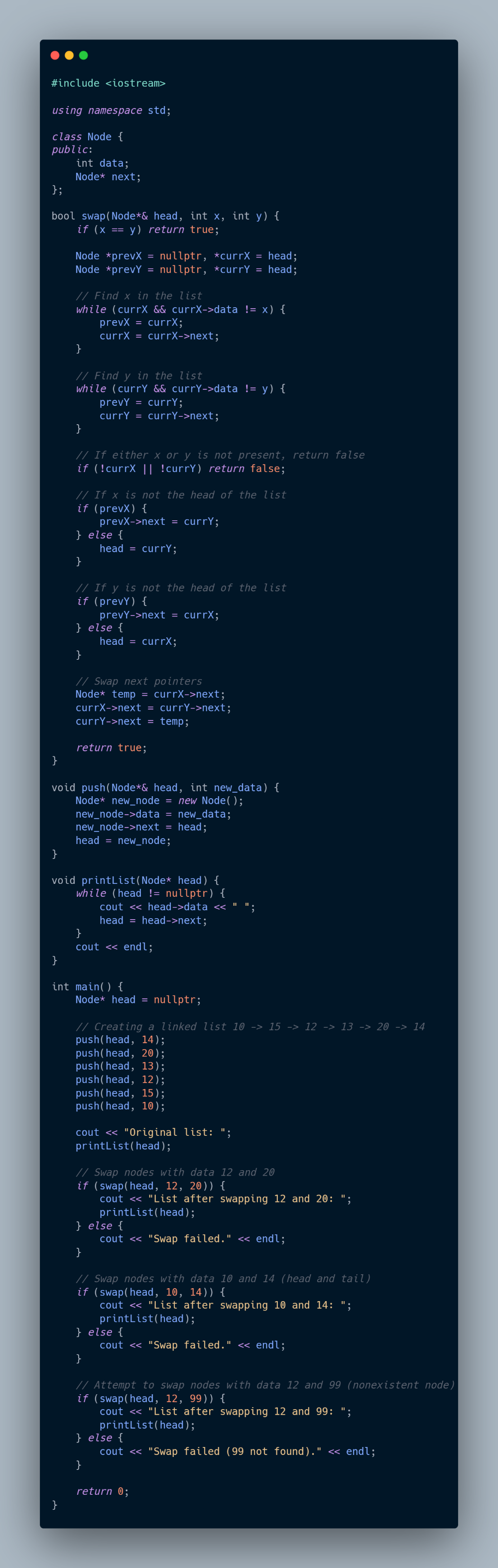
* **Task 1.1** confirmed that a basic linked list can be effectively constructed and traversed. This demonstrated an understanding of the fundamental operations required to manage a singly linked list in C++.
* **Task 1.2** illustrated that insertion operations at the beginning, end, and middle positions of the list can be managed seamlessly. This involved updating the head pointer, handling cases where the list was initially empty, and ensuring that all links between nodes were maintained correctly.
* **Task 1.3** demonstrated the ability to delete nodes from various positions within the list, with careful management of pointers to ensure no memory leaks or dangling pointers occurred. The program also handled edge cases, such as attempting to delete nodes from an empty list or from out-of-bounds positions, by providing clear feedback to the user.

Overall, these tasks collectively validated that the linked list implementation was both robust and flexible, capable of handling common operations like insertion and deletion in a variety of scenarios. The code effectively manages memory and maintains the integrity of the list structure during all operations, which is essential in the context of dynamic data structures like linked lists.

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

1. ***Write an algorithm and C\C++ program that swaps (exchanges) two nodes in a list. The nodes are identified by number and are passed as parameters. For example, to exchange nodes 5 and 8, you would call swap(5,8). If the exchange is successful, the algorithm is to return true. If it encounters an error, such as an invalid node number, it returns false. Use linked list implementation.***



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*