Activity No. 3	
Hands-on Activity 3.1 Linked Lists	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 09-27-2024
Section: CpE21-S4	**Date Submitted:** 09-27-2024
Name(s): BONIFACIO, NYKO ADREIN L.	**Instructor:** Prof. Sayo

6. Output

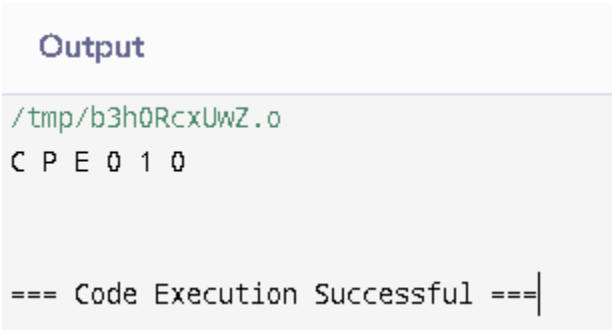
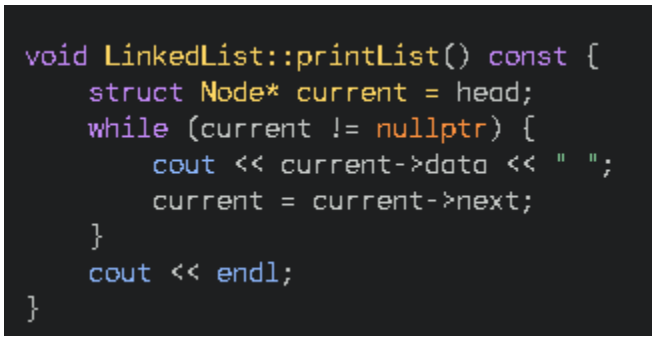
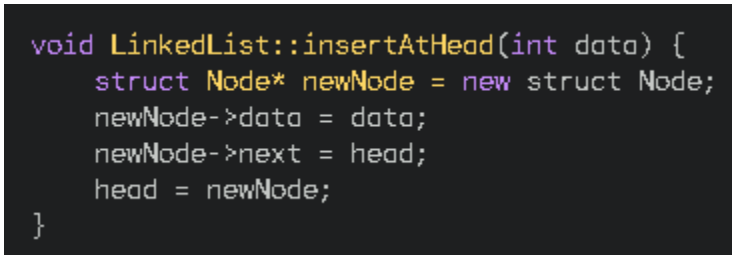
Screenshot	
Discussion	The code effectively creates a linked list, but it could be improved by using dynamic allocation, error handling, and adding functions for insertion, deletion, and modification. Iterators and templates could also be considered for flexibility. There was no output in the given code, but after a few modifications the output is shown

Table 3.1 Output of Initial/Sample Implementation

Operation	Screenshot
Traversal	
Insert at head	

```
void LinkedList::printList() const {
    struct Node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}
```

```
void LinkedList::insertAtHead(int data) {
    struct Node* newNode = new struct Node;
    newNode->data = data;
    newNode->next = head;
    head = newNode;
}
```

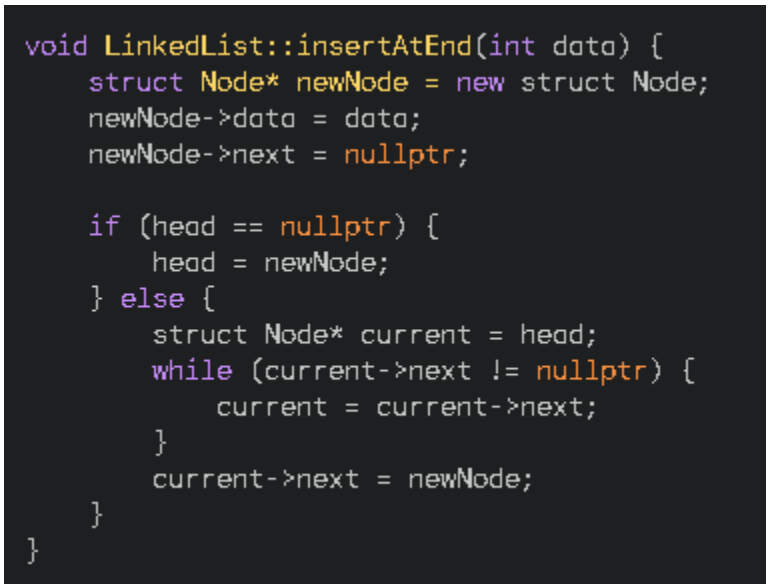
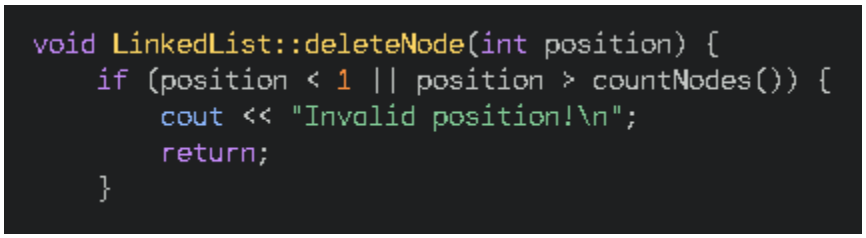
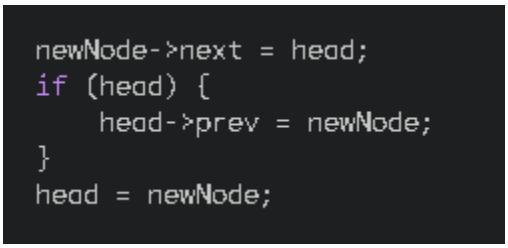
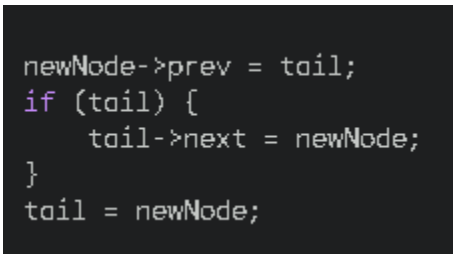
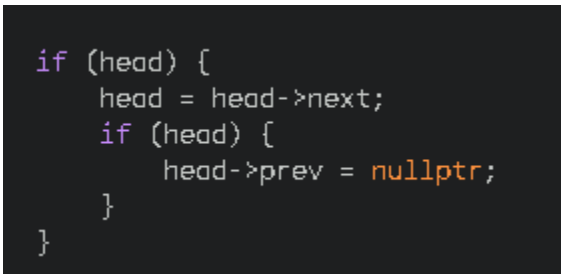
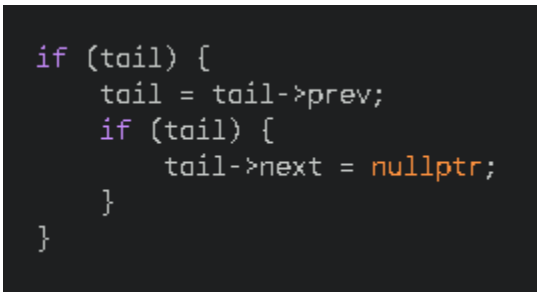
Insert at any part of the list	
Insert at the end	
Deletion of a node	

Table 3.2 Code for the List Operation

a.	**Source Code**	The `printList` function is used to traverse the list and print its elements
	Console	Original list: CPE101
b.	**Source Code**	The `insertAtStart` function is used to insert 'G' at the beginning of the list.
	Console	After inserting 'G': GCPE101
c.	**Source Code**	The `insertAfter` function is used to insert 'E' after the node containing 'P'.
	Console	After inserting 'E': GCPEE101
d.	**Source Code**	The `deleteNode` function is used to delete the node containing 'C'.
	Console	After deleting 'C': GPEE101
e.	**Source Code**	The `deleteNode` function is used to delete the node containing 'P'.

	Console	After deleting 'P': GEE101
f.	**Source Code**	The `printList` function is used to print the final list.
	Console	Final list: GEE101

Table 3.3 Code and Analysis for Singly Linked

Screenshot(s)	Analysis
	**Insertion** At the beginning: The new node becomes the new head, connecting to the previous head (if any) and the current head.
	**Insertion** At the end: The new node becomes the new tail, connecting to the previous tail and the current tail.
	**Deletion** At the beginning: The head is simply removed, and the next node becomes the new head.
	**Deletion** At the end: The tail is simply removed, and the previous node becomes the new tail.

7. Supplementary Activity

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

class Node {
public:
    string song;
    Node* next;

    Node(const string& song) : song(song), next(nullptr) {}
};

class CircularLinkedList {
private:
    Node* head;

public:
    CircularLinkedList() : head(nullptr) {}

    void addSong(const string& song) {
        Node* newNode = new Node(song);

        if (!head) {
            head = newNode;
            head->next = head; // Make it circular
        } else {
            Node* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            temp->next = newNode;
            newNode->next = head;
        }
    }

    void removeSong(const string& song) {
        if (head) {
            if (head->song == song) {
                if (head->next == head) {
                    delete head;
                    head = nullptr;
                } else {
                    Node* temp = head;
                    while (temp->next != head) {
                        temp = temp->next;
                    }
                    Node* toDelete = head;
                    temp->next = head->next;
                    head = head->next;
```

```
            delete toDelete;
        }
    } else {
        Node* prev = nullptr;
        Node* temp = head;
        while (temp->next != head && temp->song != song) {
            prev = temp;
            temp = temp->next;
        }
        if (temp->song == song) {
            prev->next = temp->next;
            delete temp;
        }
    }
}

void playAllSongs() const {
    if (head) {
        Node* temp = head;
        do {
            cout << temp->song << endl;
            temp = temp->next;
        } while (temp != head);
    }
}
};

int main() {
    CircularLinkedList playlist;
    int choice;
    string song;

    while (true) {
        cout << "\nOptions:\n";
        cout << "1. Add song\n";
        cout << "2. Remove song\n";
        cout << "3. Play all songs\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter the song name: ";
                cin.ignore();
                getline(cin, song);
                playlist.addSong(song);
                break;
            case 2:
                cout << "Enter the song name to remove: ";
                cin.ignore();
                getline(cin, song);
                playlist.removeSong(song);
                break;
            case 3:
                cout << "Playing all songs in the playlist:\n";
                playlist.playAllSongs();
                break;
            case 4:
                return 0;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }
}
```

8. Conclusion
I successfully completed the linked list activity, gaining a solid understanding of their structure, operations, and advantages over arrays. I was able to implement both singly and doubly linked lists, demonstrating my ability to apply the concepts to practical coding problems. While I feel confident in my understanding, I am eager to explore more advanced linked list topics and practice more complex coding challenges to further enhance my skills.
9. Assessment Rubric