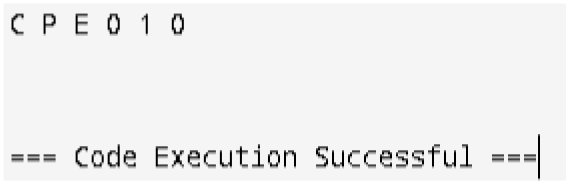
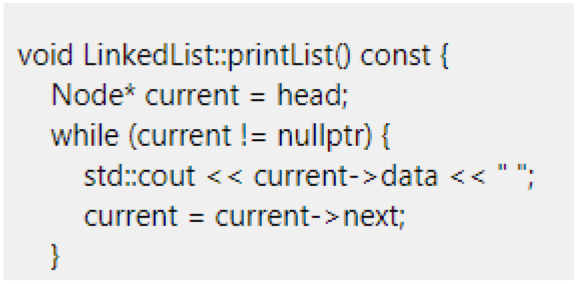
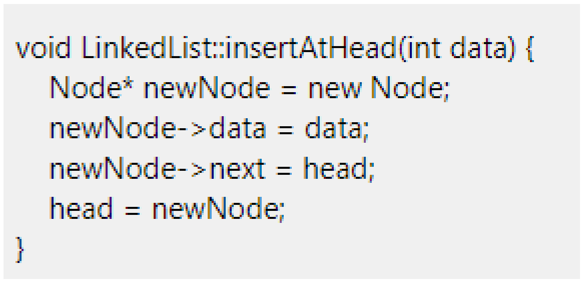
Activity No. <n>	
<Replace with Title>	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 9/27/24**
Section: CPE 21S4	**Date Submitted:9/27/24**
Name(s): Solis, Paul Vincent M.	**Instructor: Prof. Sayo**
6. Output	

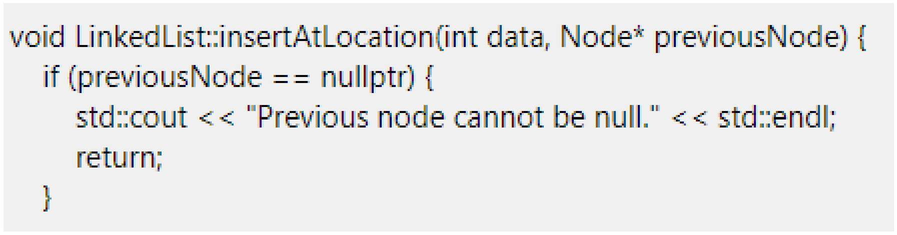
Screenshot	 C P E 0 1 0 === Code Execution Successful ===
Discussion	Although the code generates a linked list successfully, it might be made better by adding functions for insertion, deletion, and modification, handling errors, and dynamic allocation. For flexibility, iterators and templates might also be taken into account. The code provided produced no output, however after some adjustments, the output is displayed.

Tabe 3-1. Output of Initial / Simple Implementation

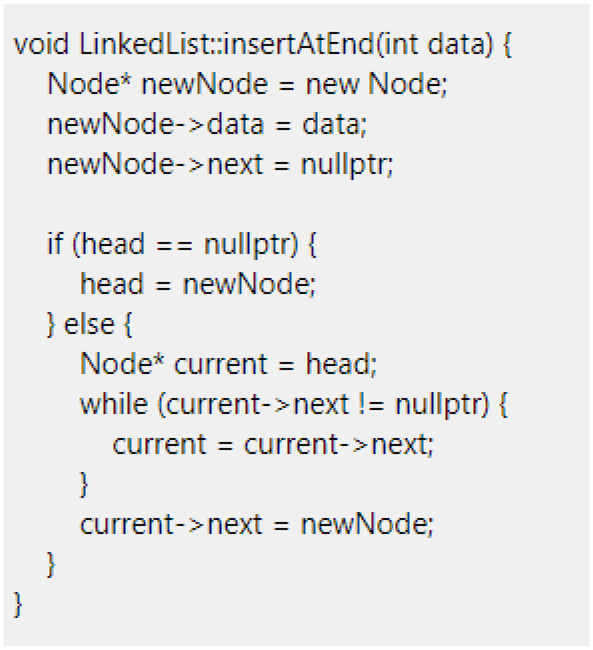
Operation	Screenshot

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

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

Insertion at any part of the list	 void LinkedList::insertAtLocation(int data, Node* previousNode) {
	    if (previousNode == nullptr) {
	        std::cout << "Previous node cannot be null." << std::endl;
	        return;
	    }

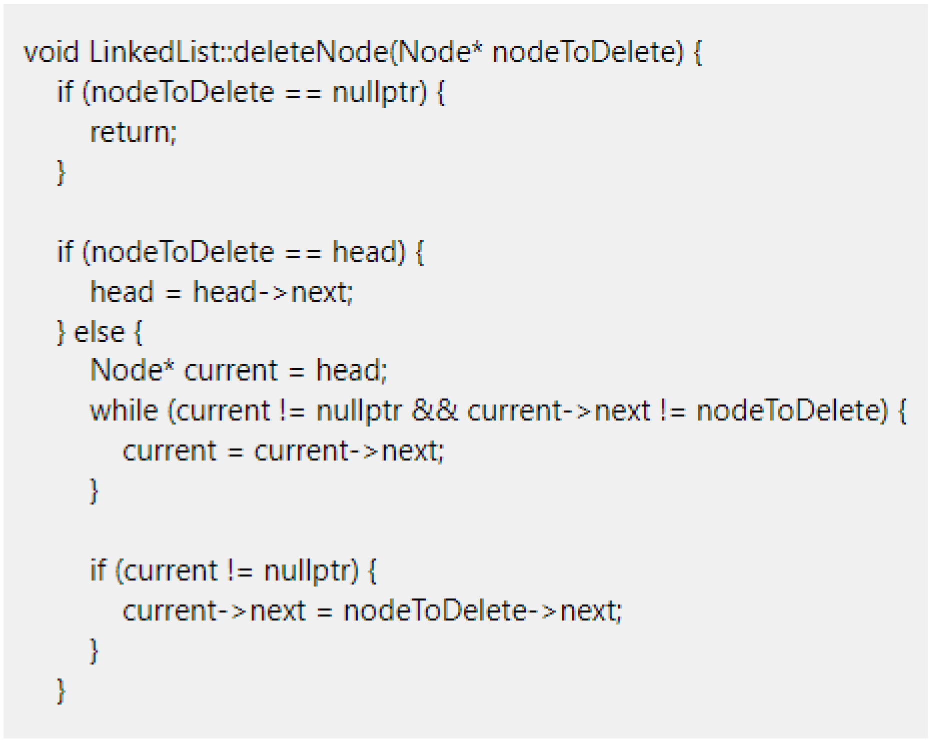
Insertion at the end



```
void LinkedList::insertAtEnd(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
    } else {
        Node* current = head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}
```

Deletion of a node



```
void LinkedList::deleteNode(Node* nodeToDelete) {
    if (nodeToDelete == nullptr) {
        return;
    }

    if (nodeToDelete == head) {
        head = head->next;
    } else {
        Node* current = head;
        while (current != nullptr && current->next != nodeToDelete) {
            current = current->next;
        }

        if (current != nullptr) {
            current->next = nodeToDelete->next;
        }
    }
```

Table3-2. Coder of the list operation

a.	Source code Console	To go over the list and print each entry, utilize the printList method. CPE101 was the original list.
b.	Source code Console	'G' is inserted at the start of the list using the insertAtStart function. Following 'G' insertion: GCPE101
c.	Source code Console	'E' is inserted after the node holding 'P' using the insertAfter method. Upon adding 'E': GCPEE101

d.	Source code Console	The node containing 'C' is deleted using the deleteNode method. Upon eliminating 'C': GPEE101
e.	Source code Console	The function deleteNode is utilized for removing the node that holds the character 'P'. After removing the letter 'P': GEE101
f.	Source code Console	The final list is printed using the printList function. End roster: GEE101

Table 3-3. Code and analysis for singly linked lists

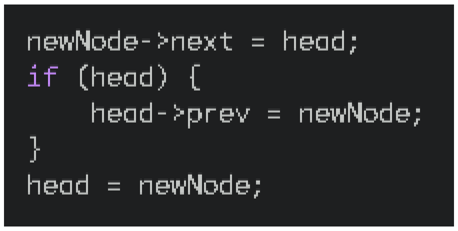
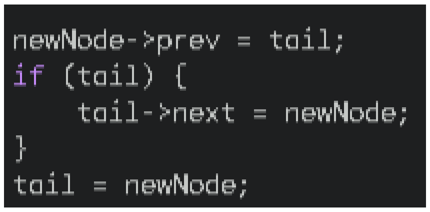
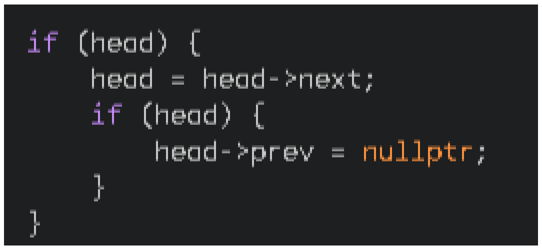
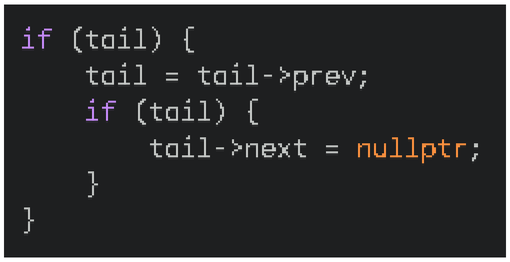
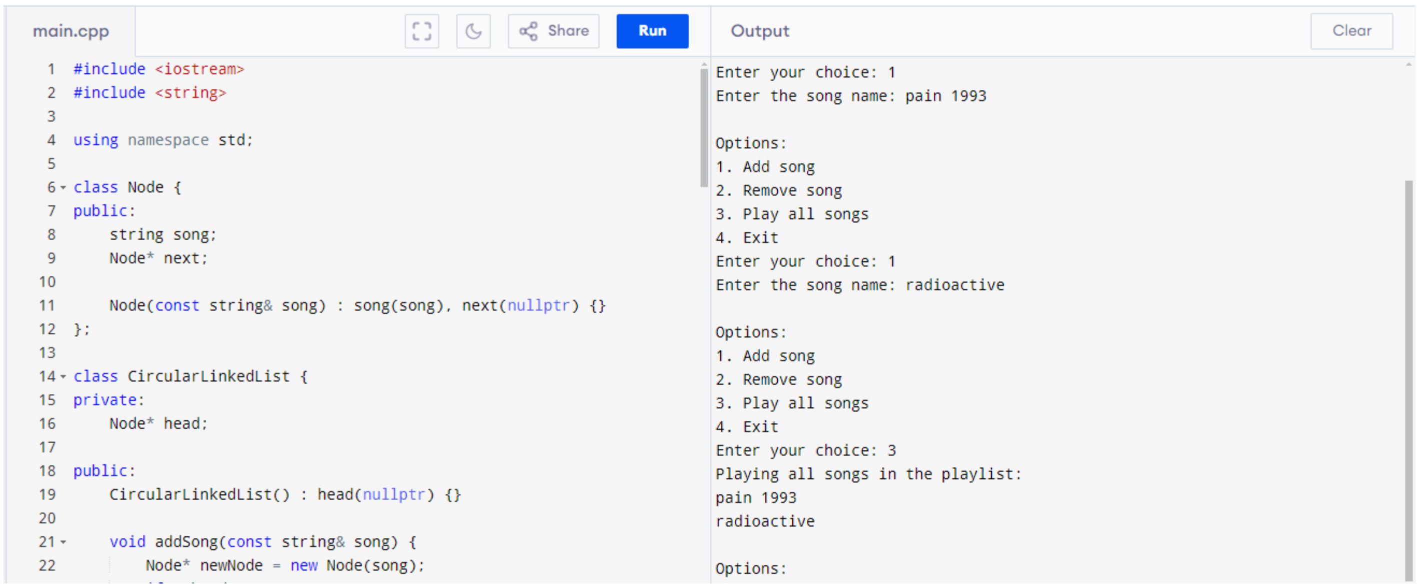
Screenshot (s)	Analysis
 `newNode->next = head;` `if (head) {` `head->prev = newNode;` `}` `head = newNode;`	Insertion The new head is connected to the previous head (if there is one) and the current head, making it the new node.
 `newNode->prev = tail;` `if (tail) {` `tail->next = newNode;` `}` `tail = newNode;`	Insertion The fresh node turns into the current tail by linking to both the previous tail and the current tail.
 `if (head) {` `head = head->next;` `if (head) {` `head->prev = nullptr;` `}` `}`	Deletion In the initial stages: The head is just taken off, and the following node takes its place as the new head.
 `if (tail) {` `tail = tail->prev;` `if (tail) {` `tail->next = nullptr;` `}` `}`	Deletion In conclusion: The tail is taken off, and the previous node turns into the new tail.

Table 3-4. Modified operations for doubly linked lists

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);
```

```
Enter your choice: 1
Enter the song name: pain 1993

Options:
1. Add song
2. Remove song
3. Play all songs
4. Exit
Enter your choice: 1
Enter the song name: radioactive

Options:
1. Add song
2. Remove song
3. Play all songs
4. Exit
Enter your choice: 3
Playing all songs in the playlist:
pain 1993
radioactive

Options:
```

CODE:

```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;
        } 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 accomplished the linked list task with a strong grasp of their layout, functions, and benefits compared to arrays. I successfully utilized both singly and doubly linked lists, showcasing my practical coding skills in implementing these concepts. Though I am confident in my grasp of the concept, I am keen to delve into more advanced linked list topics and tackle more challenging coding exercises in order to improve my skills even further.

9. Assessment Rubric