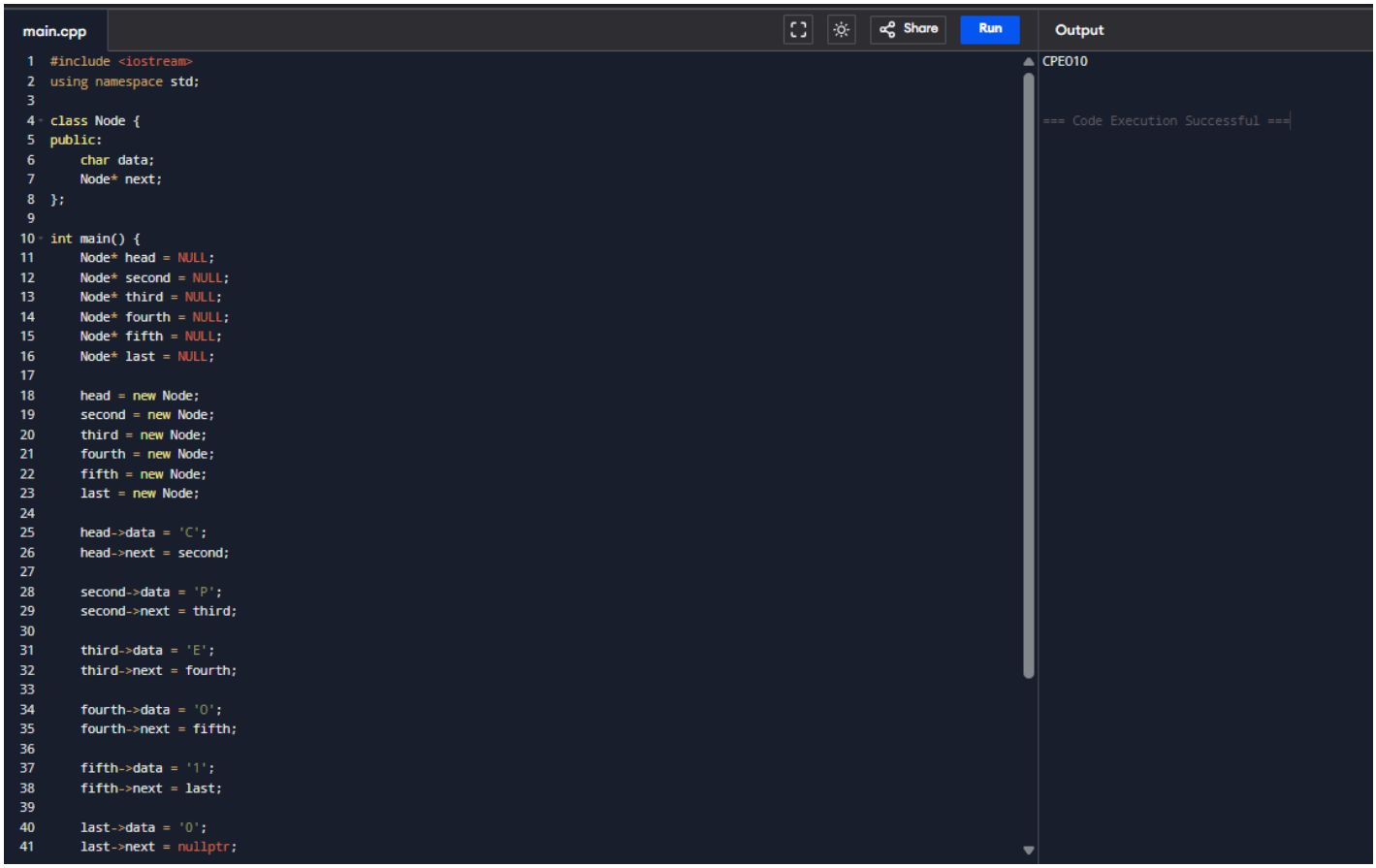
Activity No. 3.1	
Hands-on Activity Linked Lists	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 8/13/2025**
Section: CPE21S4	**Date Submitted: 8/14/2025**
Name(s): Avila, Vince Gabriel V.	**Instructor: Engr, Jimlord Quejado**

6. Output




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

class Node {
public:
    char data;
    Node* next;
};

int main() {
    Node* head = NULL;
    Node* second = NULL;
    Node* third = NULL;
    Node* fourth = NULL;
    Node* fifth = NULL;
    Node* last = NULL;

    head = new Node;
    second = new Node;
    third = new Node;
    fourth = new Node;
    fifth = new Node;
    last = new Node;

    head->data = 'C';
    head->next = second;

    second->data = 'P';
    second->next = third;

    third->data = 'E';
    third->next = fourth;

    fourth->data = '0';
    fourth->next = fifth;

    fifth->data = '1';
    fifth->next = last;

    last->data = '0';
    last->next = nullptr;
```

Analysis:

I created a linked list to store characters and display them in order by properly connecting the nodes.

Traversal:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* prev;
    Node* next;
};

void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    // Example List: C -> P -> E -> 0 -> 1 -> 0
    Node* head = new Node{'C', nullptr, nullptr};
    Node* second = new Node{'P', head, nullptr};
    head->next = second;
    Node* third = new Node{'E', second, nullptr};
    second->next = third;
    Node* fourth = new Node{'0', third, nullptr};
    third->next = fourth;
    Node* fifth = new Node{'1', fourth, nullptr};
    fourth->next = fifth;
    Node* sixth = new Node{'0', fifth, nullptr};
    fifth->next = sixth;

    traverse(head);
```

```
C P E 0 1 0

=== Code Execution Successful ===
```

Insertion at Head:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* prev;
    Node* next;
};

void insertAtHead(Node*& head, char value) {
    Node* newNode = new Node{value, nullptr, head};
    if (head != nullptr) {
        head->prev = newNode;
    }
    head = newNode;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    insertAtHead(head, 'E');
    insertAtHead(head, 'P');
    insertAtHead(head, 'C');

    traverse(head);

    return 0;
}
```

```
C P E 0 1 0

=== Code Execution Successful ===
```

Insertion at any part of the list:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* prev;
    Node* next;
};

void insertAtPosition(Node*& head, char value, int pos) {
    Node* newNode = new Node{value, nullptr, nullptr};
    if (pos == 1) { // Insert at head
        newNode->next = head;
        if (head) head->prev = newNode;
        head = newNode;
        return;
    }

    Node* temp = head;
    for (int i = 1; i < pos - 1 && temp != nullptr; ++i) {
        temp = temp->next;
    }

    if (temp == nullptr) return;

    newNode->next = temp->next;
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
    newNode->prev = temp;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{'C', nullptr, nullptr};
    head->next = new Node{'E', head, nullptr};
    insertAtPosition(head, 'P', 2);

    traverse(head);
```

```
C P E


=== Code Execution Successful ===
```

Insertion at the End:

main.cpp

Share Run

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

struct Node {
    char data;
    Node* prev;
    Node* next;
};

void insertAtEnd(Node*& head, char value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 'C');
    insertAtEnd(head, 'P');
    insertAtEnd(head, 'E');

    traverse(head);

    return 0;
}

```

Output

```
C P E


=== Code Execution Successful ===
```

Deletion of a Node:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* prev;
    Node* next;
};

void deleteNode(Node*& head, char value) {
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }

    if (temp == nullptr) return;

    if (temp->prev) temp->prev->next = temp->next;
    else head = temp->next;

    if (temp->next) temp->next->prev = temp->prev;

    delete temp;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{'C', nullptr, nullptr};
    head->next = new Node{'P', head, nullptr};
    head->next->next = new Node{'E', head->next, nullptr};

    traverse(head);
    deleteNode(head, 'P');
    traverse(head);

    return 0;
}
```

```
C P E
C E


=== Code Execution Successful ===
```

Source Code:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{'C', nullptr};
    head->next = new Node{'P', nullptr};
    head->next->next = new Node{'E', nullptr};

    traverse(head);

    return 0;
}
```

```
C P E


=== Code Execution Successful ===
```

Source Code:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* next;
};

void insertAtHead(Node*& head, char value) {
    Node* newNode = new Node{value, head};
    head = newNode;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{'P', nullptr};
    head->next = new Node{'E', nullptr};

    insertAtHead(head, 'C');

    traverse(head);

    return 0;
}
```

```
C P E


=== Code Execution Successful ===
```

Source Code:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* next;
};

void insertAfter(Node* prevNode, char value) {
    if (prevNode == nullptr) {
        cout << "Previous node cannot be null." << endl;
        return;
    }
    Node* newNode = new Node{value, prevNode->next};
    prevNode->next = newNode;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{'C', nullptr};
    head->next = new Node{'E', nullptr};

    insertAfter(head, 'P');

    traverse(head);
```

```
C P E


=== Code Execution Successful ===
```

Source Code:

main.cpp

Share Run

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

struct Node {
    char data;
    Node* next;
};

void insertAtEnd(Node*& head, char value) {
    Node* newNode = new Node{value, nullptr};
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{'C', nullptr};
    head->next = new Node{'P', nullptr};

    insertAtEnd(head, 'E');

    traverse(head);

    return 0;
}
```

Output

```
C P E


=== Code Execution Successful ===
```

Source Code:

main.cpp

Share Run Output

```cpp
struct Node {
    char data;
    Node* next;
};

void deleteNode(Node*& head, char value) {
    Node* temp = head;
    Node* prev = nullptr;

    while (temp != nullptr && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == nullptr) return;

    if (prev == nullptr) {
        head = temp->next;
    } else {
        prev->next = temp->next;
    }

    delete temp;
}

void traverse(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{'C', nullptr};
    head->next = new Node{'P', nullptr};
    head->next->next = new Node{'E', nullptr};

    deleteNode(head, 'P');
    traverse(head);

    return 0;
}
```

```
C E


=== Code Execution Successful ===
```

Screenshots:

main.cpp

Share Run

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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = nullptr;
    if (!head) {
        head = n;
        return;
    }
    Node* t = head;
    while (t->next) {
        t = t->next;
    }
    t->next = n;
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 'C');
    insertAtEnd(head, 'P');
    insertAtEnd(head, 'E');
    insertAtEnd(head, '1');
    insertAtEnd(head, '0');
    insertAtEnd(head, '1');
    cout << "Initial list: ";
    traverse(head);
    return 0;
}

```

Output

```
Initial list: CPE101


=== Code Execution Successful ===
```

Analysis:

To iterate through a linked list, a function moves a pointer, starting from the head, to each subsequent node and displays the character data held within

Screenshot:

main.cpp

Share Run

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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = nullptr;
    if (!head) {
        head = n;
        return;
    }
    Node* t = head;
    while (t->next) {
        t = t->next;
    }
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = head;
    head = n;
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 'C');
    insertAtEnd(head, 'P');
    insertAtEnd(head, 'E');
    insertAtEnd(head, '1');
    insertAtEnd(head, '0');
```

Output

```
After inserting 'G' at head: GCPE101


=== Code Execution Successful ===
```

Analysis:

To insert a node at the beginning of a linked list, a new node is created. The new node's next pointer is set to point to the current head of the list, and then the head is updated to point to the new node.

Screenshot:

main.cpp

Share Run

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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = nullptr;
    if (!head) {
        head = n;
        return;
    }
    Node* t = head;
    while (t->next) {
        t = t->next;
    }
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = head;
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node;
    n->data = v;
    n->next = prev->next;
    prev->next = n;
}
```

Output

```
After inserting 'E' after 'P': GCPEE101


=== Code Execution Successful ===
```

Analysis:

The process finds the node containing "P," creates a new node, and then updates the next pointers to insert the new node after "P."

Screenshot:

main.cpp

Share Run

Output

```
After deleting 'C': GCPEE101


--- Code Execution Successful ---
```

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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = nullptr;
    if (!head) {
        head = n;
        return;
    }
    Node* t = head;
    while (t->next) {
        t = t->next;
    }
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = head;
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node;
    n->data = v;
    n->next = prev->next;
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (head == nullptr) return;

    if (head->data == key) {
        Node* tmp = head;
        head = head->next;
        delete tmp;
        return;
    }

    Node* cur = head;
    while (cur->next && cur->next->data != key) {
        cur = cur->next;
    }

    if (cur->next) {
        Node* tmp = cur->next;
        cur->next = tmp->next;
        delete tmp;
    }
}

int main() {
```

Analysis:

locate the node preceding the one to be removed and update its next pointer to bypass the target node, effectively unlinking it from the list.

Screenshot:

main.cpp

Share Run

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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = nullptr;
    if (!head) {
        head = n;
        return;
    }
    Node* t = head;
    while (t->next) {
        t = t->next;
    }
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = head;
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node;
    n->data = v;
    n->next = prev->next;
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (head == nullptr) return;

    if (head->data == key) {
        Node* tmp = head;
        head = head->next;
        delete tmp;
        return;
    }

    Node* cur = head;
    while (cur->next && cur->next->data != key) {
        cur = cur->next;
    }

    if (cur->next) {
        Node* tmp = cur->next;
        cur->next = tmp->next;
        delete tmp;
    }
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 'C');
```

Output

```
After deleting 'C': GCPEE101

--- Code Execution Successful ---
```

Analysis:

Applying deletion again to remove “P”

Screenshot:

main.cpp

Share Run Output

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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    head = new Node;
    head->data = 'G';
    head->next = new Node;
    head->next->data = 'C';
    head->next->next = new Node;
    head->next->next->data = 'E';
    head->next->next->next = new Node;
    head->next->next->next->data = '1';
    head->next->next->next->next = new Node;
    head->next->next->next->next->data = '0';
    head->next->next->next->next->next = new Node;
    head->next->next->next->next->next->data = '1';
    head->next->next->next->next->next->next = nullptr;

    cout << "Final list: ";
    traverse(head);
    return 0;
}
```

```
Final list: GCE101


=== Code Execution Successful ===
```

Analysis:
Displaying the result after all operations

7. Supplementary Activity

main.cpp

Share Run

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

struct Song {
    string title;
    Song* next;
    Song* prev;
};

void addSong(Song*& head, const string& title) {
    Song* newSong = new Song{title, nullptr, nullptr};
    if (!head) {
        head = newSong;
        head->next = head;
        head->prev = head;
        return;
    }
    Song* tail = head->prev;
    tail->next = newSong;
    newSong->prev = tail;
    newSong->next = head;
    head->prev = newSong;
}

void removeSong(Song*& head, const string& title) {
    if (!head) return;
    Song* curr = head;
    do {
        if (curr->title == title) {
            if (curr->next == curr) {
                delete curr;
                head = nullptr;
                return;
            }
            curr->prev->next = curr->next;
            curr->next->prev = curr->prev;
            if (curr == head) {
                head = curr->next;
            }
            delete curr;
            return;
        }
        curr = curr->next;
    } while (curr != head);
}
```

Output

```
Initial Playlist:
Playing: Song A
Playing: Song B
Playing: Song C
Playing: Song D
Playing: Song E

Removing Song B...
Playing: Song A
Playing: Song C
Playing: Song D
Playing: Song E

Currently playing: Song A
Next song: Song C
Previous song: Song E


=== Code Execution Successful ===
```

```
            return;
        }
        curr = curr->next;
    } while (curr != head);
}

void playAll(Song* head) {
    if (!head) {
        cout << "Playlist is empty." << endl;
        return;
    }
    Song* curr = head;
    do {
        cout << "Playing: " << curr->title << endl;
        curr = curr->next;
    } while (curr != head);
}

Song* nextSong(Song* curr) {
    if (!curr) return nullptr;
    return curr->next;
}

Song* prevSong(Song* curr) {
    if (!curr) return nullptr;
    return curr->prev;
}

int main() {
    Song* playlist = nullptr;
    addSong(playlist, "Song A");
    addSong(playlist, "Song B");
    addSong(playlist, "Song C");
    addSong(playlist, "Song D");
    addSong(playlist, "Song E");

    cout << "\nInitial Playlist:\n";
    playAll(playlist);

    cout << "\nRemoving Song B...\n";
    removeSong(playlist, "Song B");
    playAll(playlist);

    Song* current = playlist;
    cout << "\nCurrently playing: " << current->title << endl;
    current = nextSong(current);
    cout << "Next song: " << current->title << endl;
    current = prevSong(current);
    current = prevSong(current);
    cout << "Previous song: " << current->title << endl;

    return 0;
}
```

8. Conclusion

Over this Activities 3.1 I have learnt to modify and connected with a doubly linked list, which also helped me in understanding the pointers as well. Then I learned the importance of next and previous things. As a whole, I still have much more to learn in organizing my code and cleaning it up before easier reading.

9. Assessment Rubric