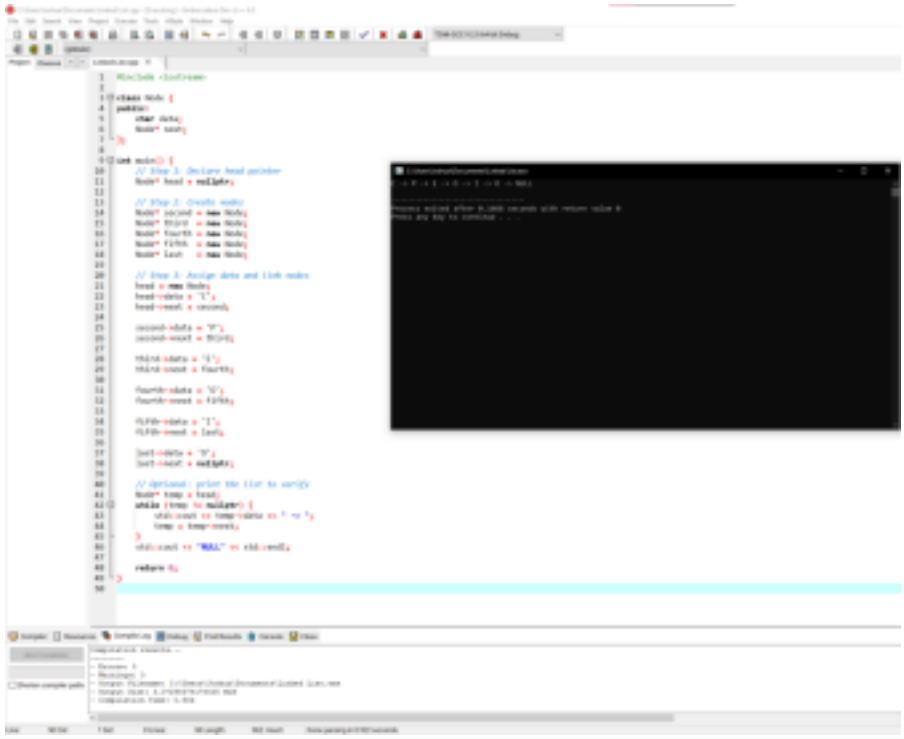
Activity No. <3>	
<Hands-on Activity 3.1 Linked Lists>	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 8/14/25**
Section: CPE21S4	**Date Submitted: 8/14/25**
Name(s): Quioyo, Angelo M.	**Instructor: Engr. Jimlord Quejado**

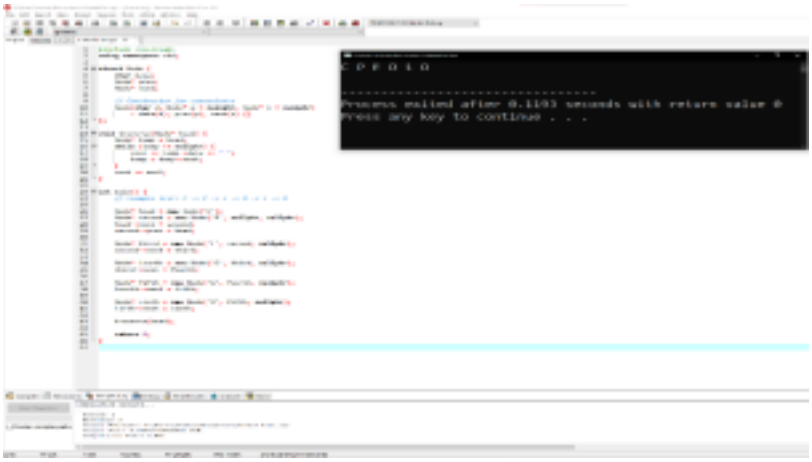
6. Output:



Discussion: I implemented a linked list where I can store characters and display it on a sequence order by linking the nodes correctly.

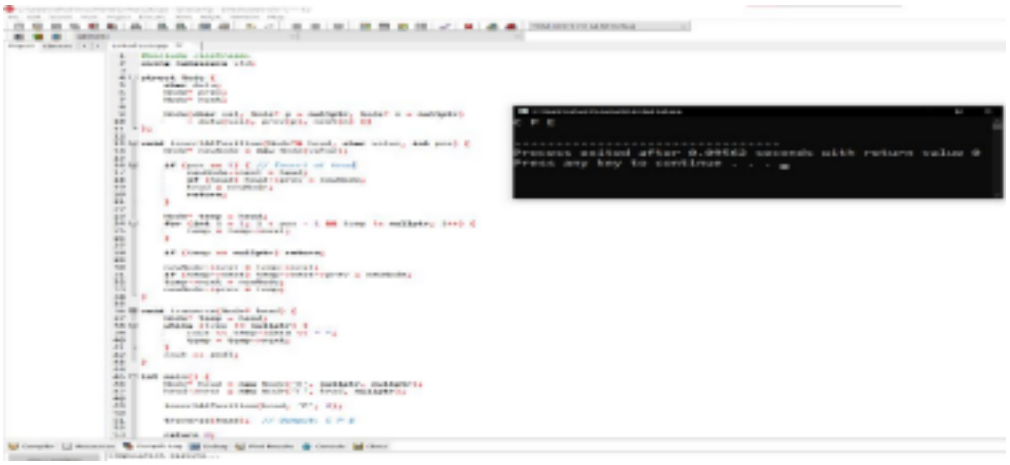
Operation:

Traversal:

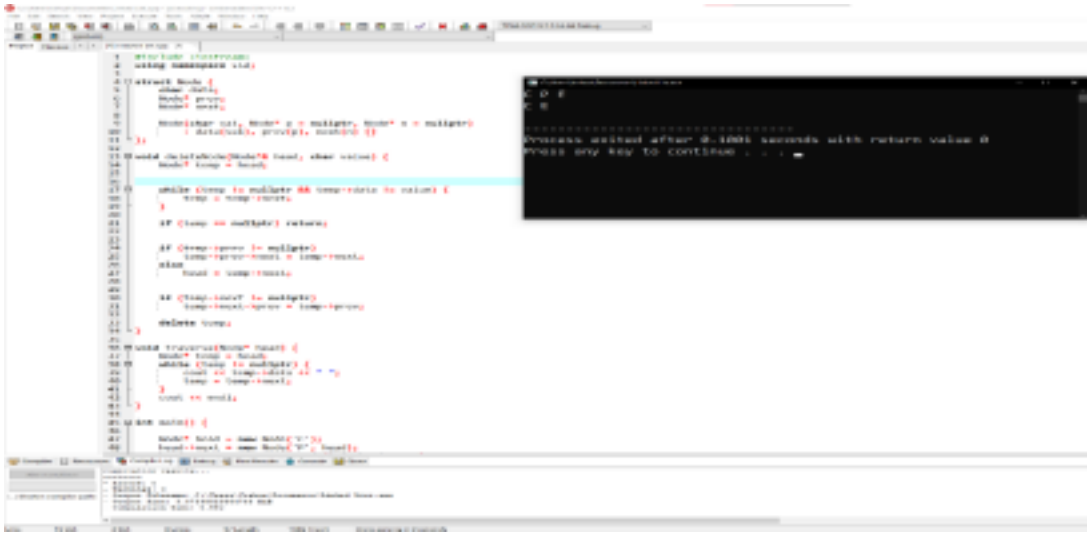


Insertion at Head:

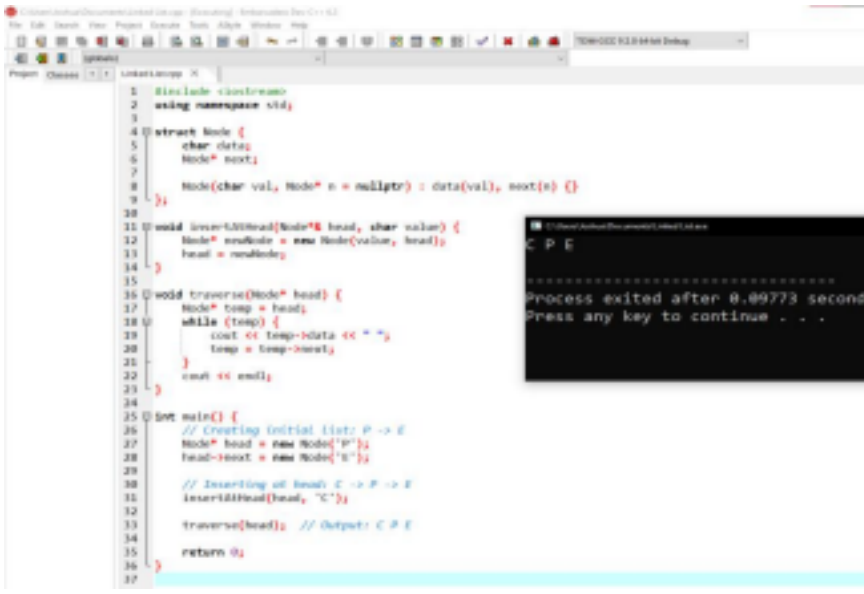
Insertion at the end:



Deletion of a node:



Source Code: Source Code:



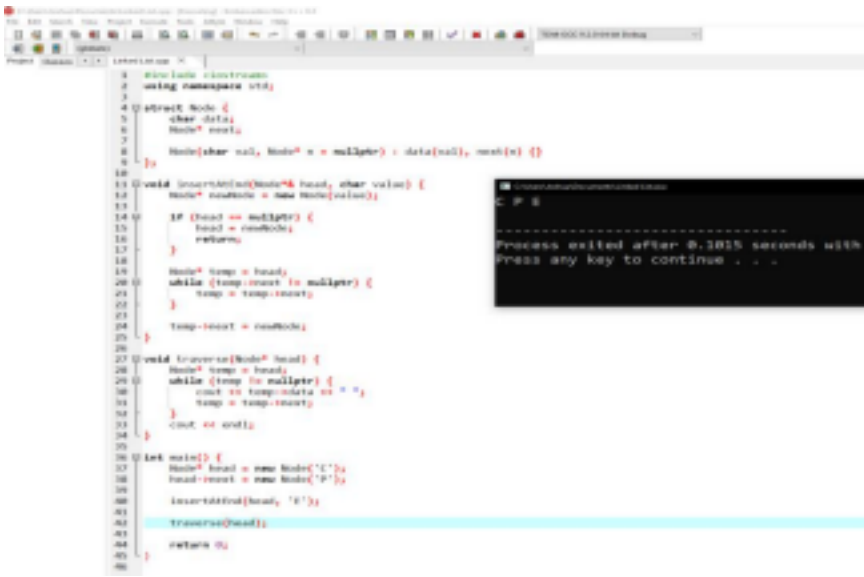
```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;

    Node(char val, Node* n = nullptr) : data(val), next(n) {}
};

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() {
    // Creating initial list: P -> E
    Node* head = new Node('P');
    head->next = new Node('E');

    // Inserting at head: C -> P -> E
    insertAtHead(head, 'C');

    traverse(head);  // Output: C P E

    return 0;
}
```

```
C P E

--------------------------------
Process exited after 0.09773 second
Press any key to continue . . .
```

Source Code:

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

struct Node {
    char data;
    Node* next;

    Node(char val, Node* n = nullptr) : data(val), next(n) {}
};

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

int main() {
    // Creating the list manually: C -> P -> E
    Node* head = new Node('C');
    head->next = new Node('P');
    head->next->next = new Node('E');

    traverse(head);  // Output: C P E

    return 0;
}
```

```
C P E

--------------------------------
Process exited after 0.09
Press any key to continue
```



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

struct Node {
    char data;
    Node* next;

    Node(char val, Node* n = nullptr) : data(val), next(n) {}
};

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

    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 != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

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

    insertAtHead(head, 'C');

    traverse(head);

    return 0;
}
```

```
C P E

--------------------------------
Process exited after 0.1015 seconds with
Press any key to continue . . .
```

Source Code:

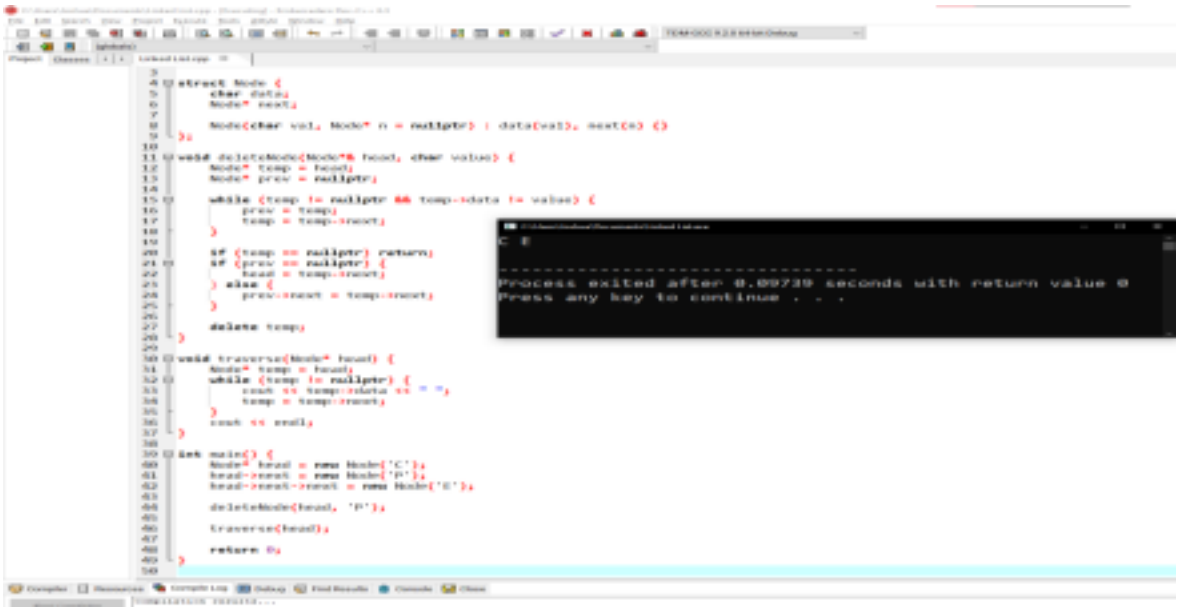


Table 3-3. Code and Analysis for Singly Linked Lists

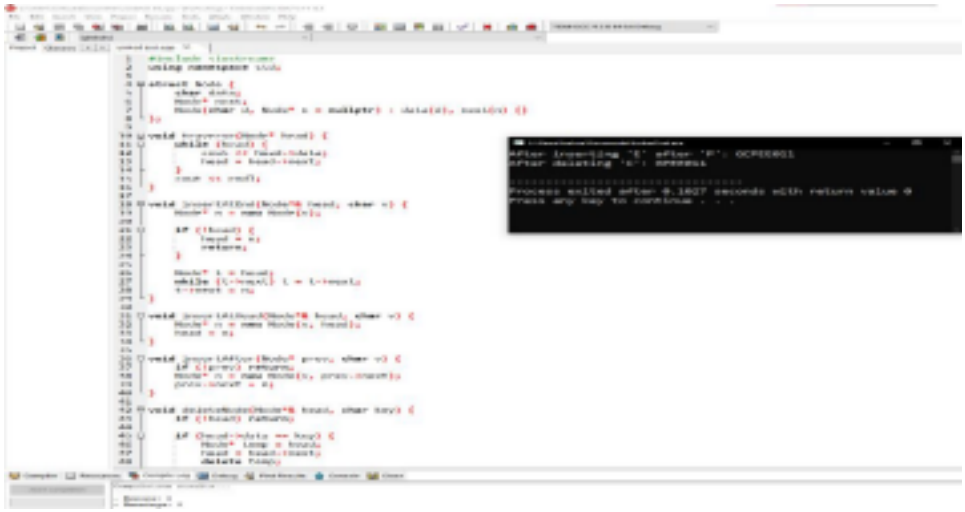
Analysis:

Traversing the list by making the head pointer pass. Function walks from node to node and prints the stored characters.

Analysis:

Insert a head creates a new node wherein next points to the previous head, then reassigns head.

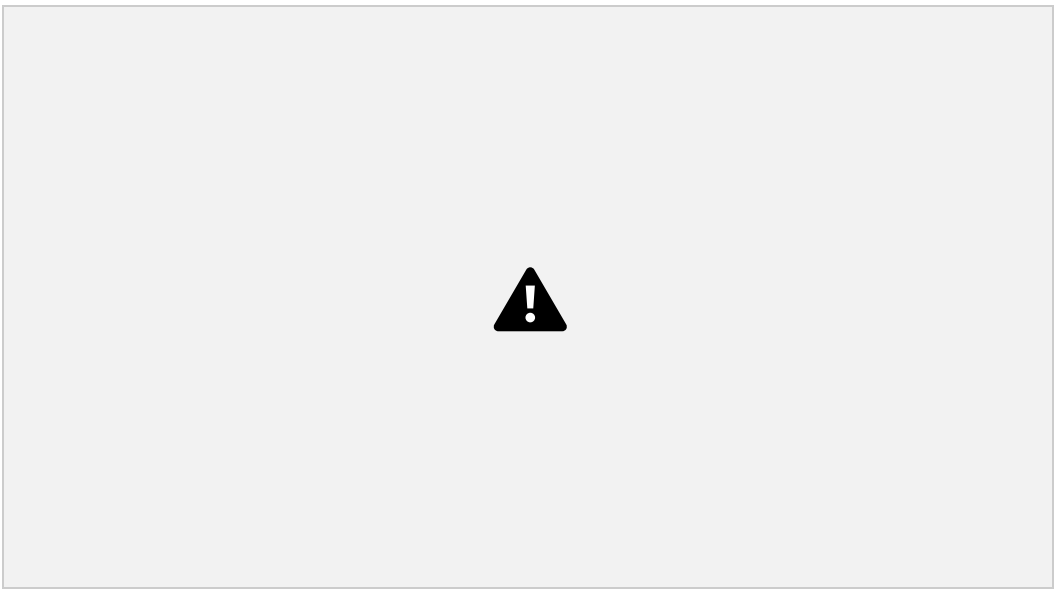
Analysis: It finds node “P”, allocates new node and adjust the next pointers.



Analysis: Delete a node by locating the node before the desired target. Changing its next to skip the target

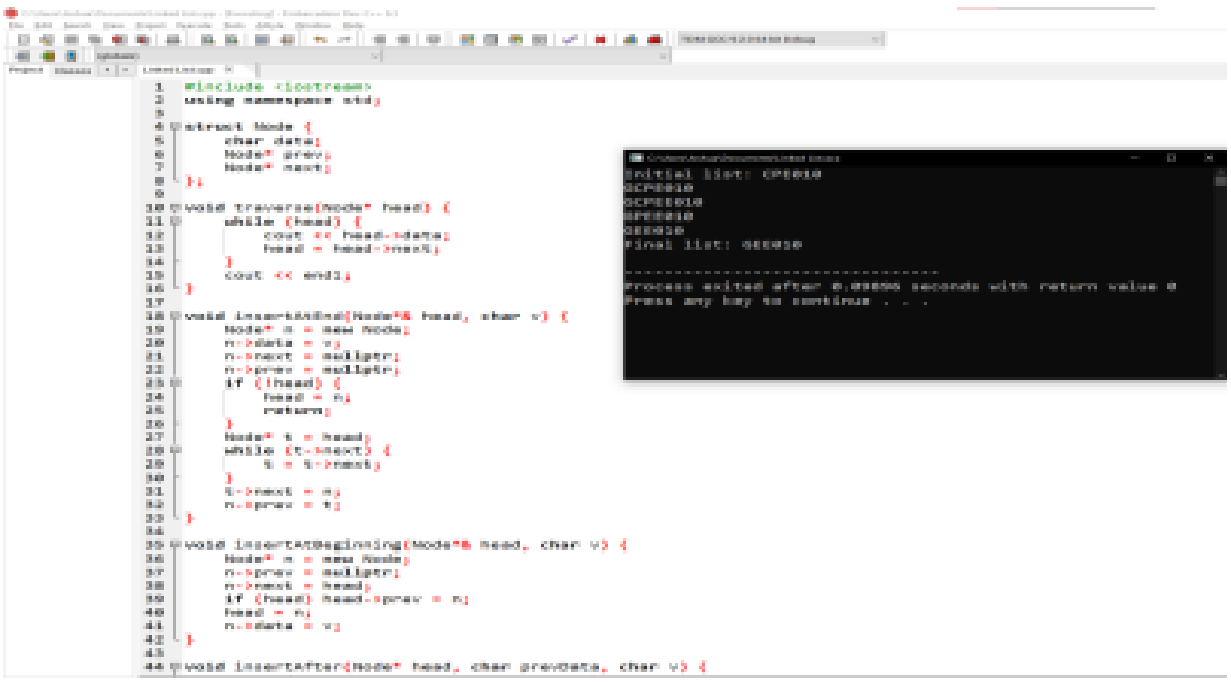
node.

Analysis: Applying deletion again to remove “P”.



Analysis: Displaying the result after all operations.

Table 3-4. Modified Operations for Doubly Linked Lists



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

struct Node {
    char data;
    Node* prev;
    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;
    n->prev = nullptr;
    if (!head) {
        head = n;
        return;
    }
    Node* t = head;
    while (t->next) {
        t = t->next;
    }
    t->next = n;
    n->prev = t;
}

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

void insertAfter(Node* head, char prevData, char v) {
```

```
Initial list: CPE01A
CCPE01A
CCPE101A
CPPE101A
CEE101A
Final list: CEE101A

--------------------------------
Process exited after 0.08006 seconds with return value 0
Press any key to continue . . .
```

7. Supplementary Activity:

Untitled3.cpp

```cpp
#include <iostream>
#include <string>

// This is like a single song in our pla
struct SongNode {
    std::string songTitle;
    SongNode* next; // This points to th
};

// This is the whole playlist, it's a ci
class Playlist {
private:
    SongNode* last; // We'll keep a poin

public:
    Playlist() {
        last = NULL; // The playlist sta
    }

    // Function to add a song
    void addSong(const std::string& song
        SongNode* newSong = new SongNode
        newSong->songTitle = songTitle;
```

```
C:\Users\TIPQC\Documents\L
--- End of playlist ---

Removed 'Hotel California' from the playlist.

--- Playing all songs in the playlist ---
Bohemian Rhapsody
Stairway to Heaven
Billie Jean
--- End of playlist ---

Added 'Imagine' to the playlist.
Removed 'Bohemian Rhapsody' from the playlist.

--- Playing all songs in the playlist ---
Stairway to Heaven
Billie Jean
Imagine
--- End of playlist ---

Removed 'Imagine' from the playlist.
Removed 'Stairway to Heaven' from the playlist.
Removed 'Billie Jean' from the playlist.

The playlist is empty.

Playlist is empty. Cannot remove.

--------------------------------
Process exited after 0.02141 seconds with return value 0
Press any key to continue . . .
```

8. Conclusion: In this activity 3, I understand how to revise single linked lists with different connections to doubly linked lists, which gave us ideas on how pointers work. We need to be careful as we move to the next and previous pointers.Furthermore, I need to study and learn more to improve my skills to organize my code just to look clean and neat so that it can understand it easily.

9. Assessment Rubric