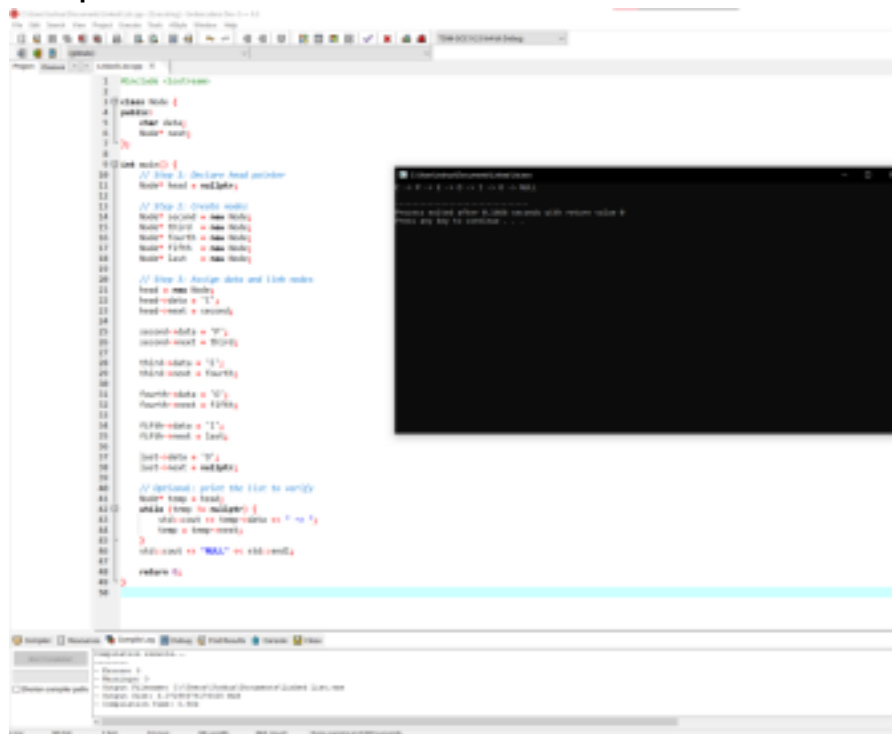


<b>Activity No. &lt;3&gt;</b>	
<b>&lt;Hands-on Activity 3.1 Linked Lists&gt;</b>	
<b>Course Code:</b> CPE010	<b>Program:</b> Computer Engineering
<b>Course Title:</b> Data Structures and Algorithms	<b>Date Performed:</b> 8/14/25
<b>Section:</b> CPE21S4	<b>Date Submitted:</b> 8/14/25
<b>Name(s):</b> Quioyo,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

## 6. Output:

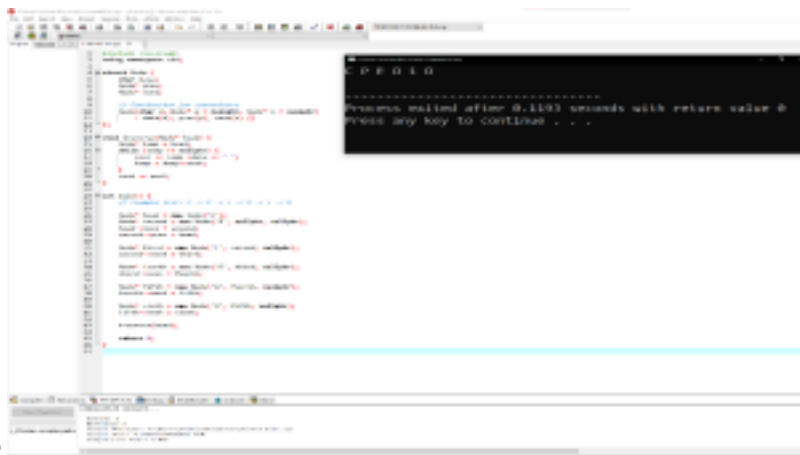


The screenshot shows a C++ IDE with a source code editor on the left and a terminal window on the right. The code implements a linked list with a `Node` structure and a `LinkedList` class. The `main` function creates a linked list with five nodes containing the characters 'H', 'E', 'L', 'L', 'O'. The terminal output shows the characters 'H E L L O' printed in sequence, followed by a message indicating the program ended successfully.

```
1 // Node structure
2
3 struct Node {
4     char data;
5     Node* next;
6 };
7
8 // LinkedList class
9
10 class LinkedList {
11 public:
12     Node* head;
13     Node* tail;
14
15     // Step 1: Create nodes
16     Node* createNode(char data) {
17         Node* newNode = new Node;
18         newNode->data = data;
19         newNode->next = NULL;
20         return newNode;
21     }
22
23     // Step 2: Insert nodes at the end
24     void insertAtEnd(char data) {
25         Node* newNode = createNode(data);
26         if (head == NULL) {
27             head = newNode;
28             tail = newNode;
29         } else {
30             tail->next = newNode;
31             tail = newNode;
32         }
33     }
34
35     // Step 3: Display the linked list
36     void display() {
37         Node* temp = head;
38         while (temp != NULL) {
39             cout << temp->data << " ";
40             temp = temp->next;
41         }
42         cout << endl;
43     }
44 };
45
46 int main() {
47     LinkedList ll;
48     ll.insertAtEnd('H');
49     ll.insertAtEnd('E');
50     ll.insertAtEnd('L');
51     ll.insertAtEnd('L');
52     ll.insertAtEnd('O');
53     ll.display();
54     return 0;
55 }
```

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

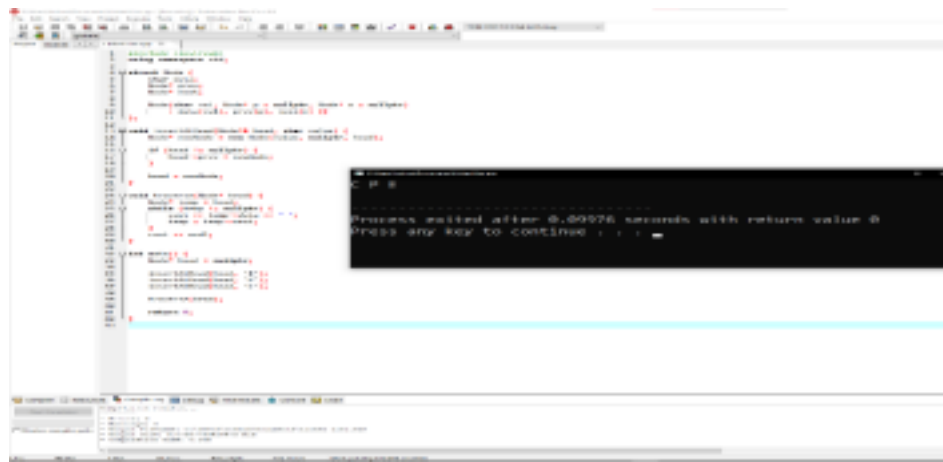
## Operation:



The screenshot shows a C++ IDE with a source code editor on the left and a terminal window on the right. The code implements a linked list with a `Node` structure and a `LinkedList` class. The `main` function creates a linked list with five nodes containing the characters 'H', 'E', 'L', 'L', 'O'. The terminal output shows the characters 'H E L L O' printed in sequence, followed by a message indicating the program ended successfully.

```
1 // Node structure
2
3 struct Node {
4     char data;
5     Node* next;
6 };
7
8 // LinkedList class
9
10 class LinkedList {
11 public:
12     Node* head;
13     Node* tail;
14
15     // Step 1: Create nodes
16     Node* createNode(char data) {
17         Node* newNode = new Node;
18         newNode->data = data;
19         newNode->next = NULL;
20         return newNode;
21     }
22
23     // Step 2: Insert nodes at the end
24     void insertAtEnd(char data) {
25         Node* newNode = createNode(data);
26         if (head == NULL) {
27             head = newNode;
28             tail = newNode;
29         } else {
30             tail->next = newNode;
31             tail = newNode;
32         }
33     }
34
35     // Step 3: Display the linked list
36     void display() {
37         Node* temp = head;
38         while (temp != NULL) {
39             cout << temp->data << " ";
40             temp = temp->next;
41         }
42         cout << endl;
43     }
44 };
45
46 int main() {
47     LinkedList ll;
48     ll.insertAtEnd('H');
49     ll.insertAtEnd('E');
50     ll.insertAtEnd('L');
51     ll.insertAtEnd('L');
52     ll.insertAtEnd('O');
53     ll.display();
54     return 0;
55 }
```

## Traversal:



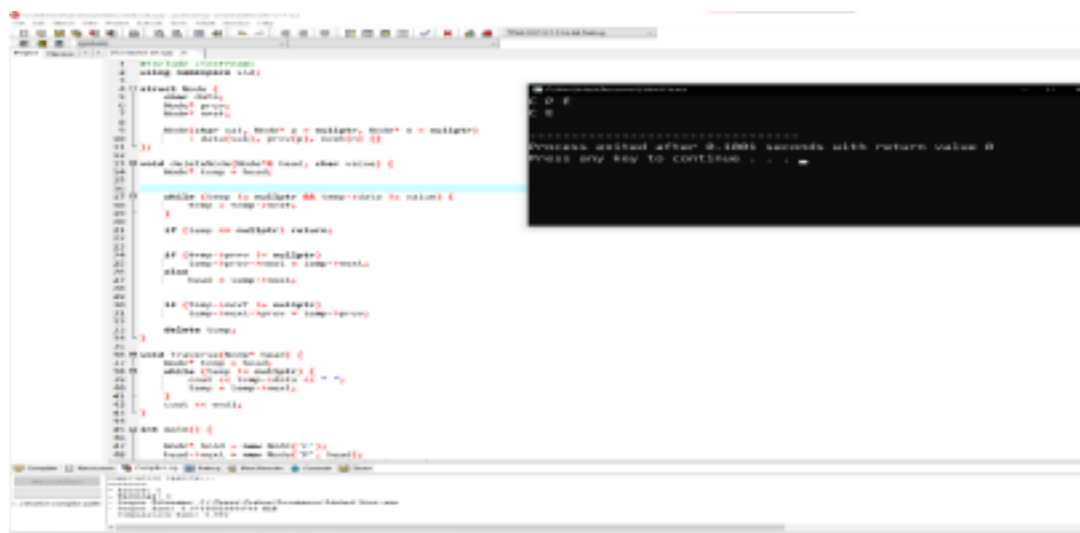
Insertion at Head:

Insertion at



the end:

Deletion of a



node:

Source Code: Source Code:

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char val, Node* n = nullptr) : data(val), next(n) {}
9
10 void insertAtHead(Node*& head, char value) {
11     Node* newNode = new Node(value, head);
12     head = newNode;
13 }
14
15 void traverse(Node* head) {
16     Node* temp = head;
17     while (temp) {
18         cout << temp->data << " ";
19         temp = temp->next;
20     }
21     cout << endl;
22 }
23
24 int main() {
25     // Creating initial list: P -> E
26     Node* head = new Node('P');
27     head->next = new Node('E');
28
29     // Inserting at head: C -> P -> E
30     insertAtHead(head, 'C');
31
32     traverse(head); // Output: C P E
33
34     return 0;
35 }
```

```
C P E
.....
Process exited after 0.09773 seconds
Press any key to continue . . .
```

Source Code:

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char val, Node* n = nullptr) : data(val), next(n) {}
9
10 void traverse(Node* head) {
11     Node* temp = head;
12     while (temp != nullptr) {
13         cout << temp->data << " ";
14         temp = temp->next;
15     }
16     cout << endl;
17 }
18
19 int main() {
20     // Creating the list manually: C -> P -> E
21     Node* head = new Node('C');
22     head->next = new Node('P');
23     head->next->next = new Node('E');
24
25     traverse(head); // Output: C P E
26
27     return 0;
28 }
```

```
C P E
.....
Process exited after 0.09
Press any key to continue
```

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char val, Node* n = nullptr) : data(val), next(n) {}
9
10 void insertAtHead(Node*& head, char value) {
11     Node* newNode = new Node(value);
12
13     if (head == nullptr) {
14         head = newNode;
15         return;
16     }
17
18     Node* temp = head;
19     while (temp->next != nullptr) {
20         temp = temp->next;
21     }
22     temp->next = newNode;
23 }
24
25 void traverse(Node* head) {
26     Node* temp = head;
27     while (temp != nullptr) {
28         cout << temp->data << " ";
29         temp = temp->next;
30     }
31     cout << endl;
32 }
33
34 int main() {
35     Node* head = new Node('C');
36     head->next = new Node('P');
37
38     insertAtHead(head, 'E');
39
40     traverse(head);
41
42     return 0;
43 }
```

```
C P E
.....
Process exited after 0.1015 seconds with
Press any key to continue . . .
```

```

1
2
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8
9 Node* char_val, Node* n = nullptr; ; data[val], next[n] {}
10
11 void deleteNode(Node* head, char value) {
12     Node* temp = head;
13     Node* prev = nullptr;
14
15     while (temp != nullptr && temp->data != value) {
16         prev = temp;
17         temp = temp->next;
18     }
19
20     if (temp == nullptr) return;
21     if (prev == nullptr) {
22         head = temp->next;
23     } else {
24         prev->next = temp->next;
25     }
26     delete temp;
27 }
28
29 void traverse(Node* head) {
30     Node* temp = head;
31     while (temp != nullptr) {
32         cout << temp->data << " ";
33         temp = temp->next;
34     }
35     cout << endl;
36 }
37
38 int main() {
39     Node* head = new Node('C');
40     head->next = new Node('P');
41     head->next->next = new Node('E');
42     deleteNode(head, 'P');
43     traverse(head);
44     return 0;
45 }

```

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

```

1
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     char data;
7     Node* next;
8 }
9
10 Node* char_val, Node* n = nullptr; ; data[val], next[n] {}
11
12 void traverse(Node* head) {
13     while (head) {
14         cout << head->data;
15         head = head->next;
16     }
17     cout << endl;
18 }
19
20 void insertAtHead(Node* head, char s) {
21     Node* n = new Node(s);
22     if (head == nullptr) {
23         head = n;
24     } else {
25         n->next = head;
26         head = n;
27     }
28 }
29
30 int main() {
31     Node* head = nullptr;
32
33     insertAtHead(head, 'C');
34     insertAtHead(head, 'P');
35     insertAtHead(head, 'E');
36     insertAtHead(head, 'C');
37
38     cout << "Initial list: ";
39     traverse(head);
40     return 0;
41 }

```

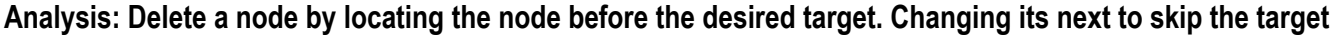
Analysis:  
Traversing the list by making the head pointer pass. Function walks from node to node and prints the stored characters.

```

1
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     char data;
7     Node* next;
8 }
9
10 Node* char_val, Node* n = nullptr; ; data[val], next[n] {}
11
12 void traverse(Node* head) {
13     while (head) {
14         cout << head->data;
15         head = head->next;
16     }
17     cout << endl;
18 }
19
20 void insertAtHead(Node* head, char s) {
21     Node* n = new Node(s);
22     if (head == nullptr) {
23         head = n;
24     } else {
25         n->next = head;
26         head = n;
27     }
28 }
29
30 int main() {
31     Node* head = nullptr;
32
33     insertAtHead(head, 'C');
34     insertAtHead(head, 'P');
35     insertAtHead(head, 'E');
36     insertAtHead(head, 'C');
37
38     cout << "Initial list: ";
39     traverse(head);
40     return 0;
41 }

```

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



**Analysis: Applying deletion again to remove “P”.**



**Analysis: Displaying the result after all operations.**

### Table 3-4. Modified Operations for Doubly Linked Lists

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node* prev;
8 }
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 void insertAtBeginning(Node*& head, char v) {
19     Node* n = new Node;
20     n->data = v;
21     n->next = head;
22     n->prev = nullptr;
23     if (!head) {
24         head = n;
25         return;
26     }
27     Node* t = head;
28     while (t->next) {
29         t = t->next;
30     }
31     t->next = n;
32     n->prev = t;
33 }
34
35 void insertAtBeginning(Node*& head, char v) {
36     Node* n = new Node;
37     n->prev = nullptr;
38     n->next = head;
39     if (head) head->prev = n;
40     head = n;
41     n->data = v;
42 }
43
44 void insertAfter(Node* head, char prevData, char v) {

```

## 7. Supplementary Activity:

Untitled3.cpp

```

1  #include <iostream>
2  #include <string>
3
4  // This is like a single song in our playlist
5  struct SongNode {
6      std::string songTitle;
7      SongNode* next; // This points to the next song
8  };
9
10 // This is the whole playlist, it's a class
11 class Playlist {
12 private:
13     SongNode* last; // We'll keep a pointer to the last song
14
15 public:
16     Playlist() {
17         last = NULL; // The playlist starts empty
18     }
19
20     // Function to add a song
21     void addSong(const std::string& songTitle) {
22         SongNode* newSong = new SongNode;
23         newSong->songTitle = songTitle;
24     }

```

```

C:\Users\TIPQC\Documents\L  X  +  v
--- 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