

## 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

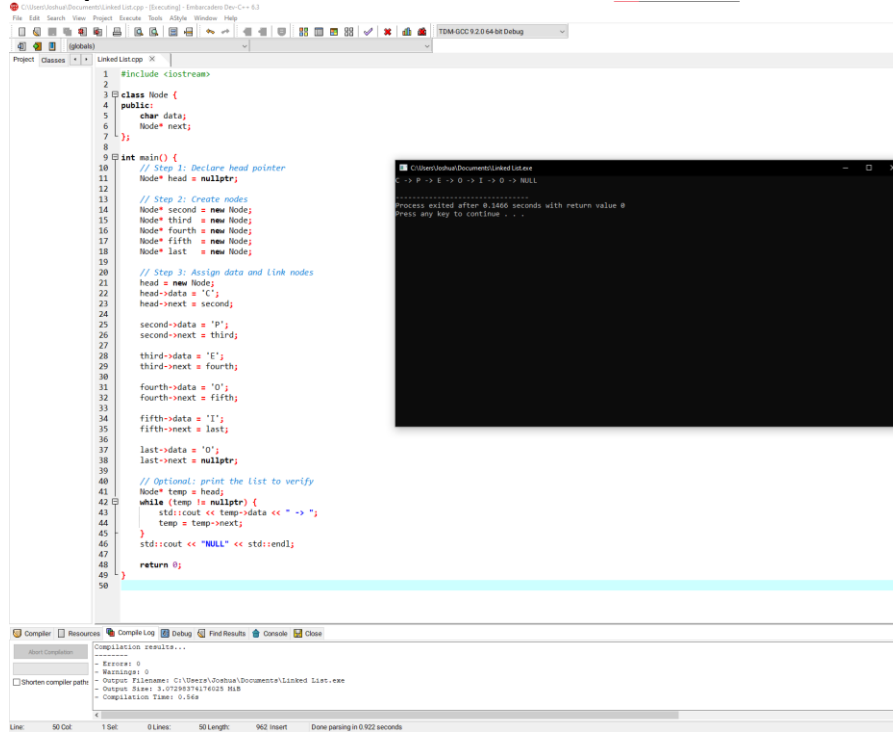
Section: CPE21S4

Date Submitted: 8/14/25

Name(s): Alcantara, Jason P.

Instructor: Engr. Jimlord Quejado

#### 6. Output:



The screenshot shows a C++ IDE with a file named 'Linked List.cpp'. The code implements a linked list with five nodes. The 'main' function follows three steps: 1. Declaring a head pointer, 2. Creating five nodes with data 'C', 'P', 'E', '0', and '1', and 3. Assigning data and linking the nodes. A while loop prints the list contents, resulting in 'C P E 0 1'. The console output shows the program exiting after 0.1466 seconds.

```
1 #include <iostream>
2
3 class Node {
4 public:
5     char data;
6     Node* next;
7 };
8
9 int main() {
10     // Step 1: Declare head pointer
11     Node* head = nullptr;
12
13     // Step 2: Create nodes
14     Node* second = new Node;
15     Node* third = new Node;
16     Node* fourth = new Node;
17     Node* fifth = new Node;
18     Node* last = new Node;
19
20     // Step 3: Assign data and link nodes
21     head = new Node;
22     head->data = 'C';
23     head->next = second;
24
25     second->data = 'P';
26     second->next = third;
27
28     third->data = 'E';
29     third->next = fourth;
30
31     fourth->data = '0';
32     fourth->next = fifth;
33
34     fifth->data = '1';
35     fifth->next = last;
36
37     last->data = '0';
38     last->next = nullptr;
39
40     // Optional: print the list to verify
41     Node* temp = head;
42     while (temp != nullptr) {
43         std::cout << temp->data << " ";
44         temp = temp->next;
45     }
46     std::cout << "NULL" << std::endl;
47
48     return 0;
49 }
```

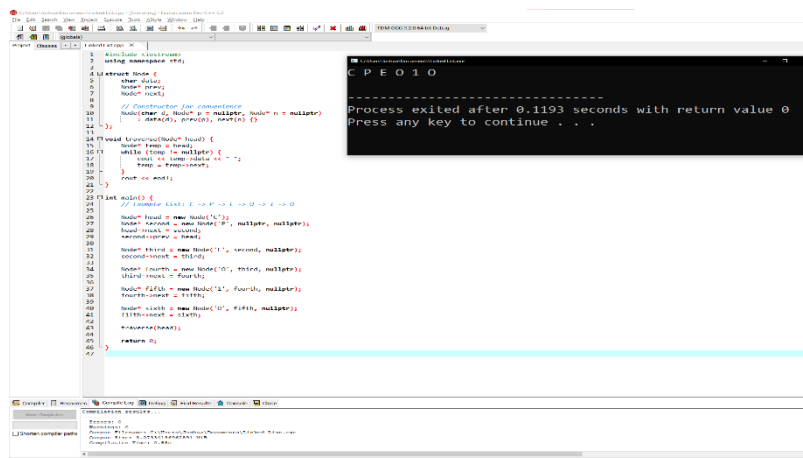
Console Output:

```
C:\Users\jason\Documents\Linked List.exe
C P E 0 1
Process exited after 0.1466 seconds with return value 0
Press any key to continue . . .
```

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

#### Operation:

#### Traversal:



The screenshot shows a C++ IDE with a file named 'Linked List.cpp'. The code implements a linked list with five nodes. The 'main' function follows three steps: 1. Declaring a head pointer, 2. Creating five nodes with data 'C', 'P', 'E', '0', and '1', and 3. Assigning data and linking the nodes. A while loop prints the list contents, resulting in 'C P E 0 1'. The console output shows the program exiting after 0.1193 seconds.

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 };
8
9 // Constructor for constructor
10 Node* createNode(char* data, Node* n = nullptr) {
11     Node* new_node = new Node;
12     new_node->data = *data;
13     new_node->next = n;
14 }
15
16 // void traverse(Node* head) {
17     Node* temp = head;
18     while (temp != nullptr) {
19         cout << temp->data << " ";
20         temp = temp->next;
21     }
22     cout << endl;
23 }
24
25 // int main() {
26     // Sample list: C -> P -> E -> 0 -> 1 -> 0
27
28     Node* head = new Node("C");
29     Node* second = new Node("P", nullptr, nullptr);
30     head->next = second;
31     second->next = third;
32
33     Node* third = new Node("E", second, nullptr);
34     second->next = third;
35
36     Node* fourth = new Node("0", third, nullptr);
37     third->next = fourth;
38
39     Node* fifth = new Node("1", fourth, nullptr);
40     fourth->next = fifth;
41
42     Node* sixth = new Node("0", fifth, nullptr);
43     fifth->next = sixth;
44
45     traverse(head);
46
47     return 0;
48 }
```

Console Output:

```
C P E 0 1
Process exited after 0.1193 seconds with return value 0
Press any key to continue . . .
```

## Insertion at Head:

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

Process exited after 0.09976 seconds with return value 0  
Press any key to continue . . .

## Insertion at the end:

```
1 // Insertion at the end
2 #include <iostream>
3 using namespace std;
4 struct Node {
5     char data;
6     Node* prev;
7     Node* next;
8 };
9 Node(char val, Node* p = nullptr, Node* n = nullptr) {
10     data(val); prev(p); next(n);
11 }
12 void insertAtPosition(Node* head, char value, int pos) {
13     Node* newNode = new Node(value);
14     if (pos == 1) { // Insert at head
15         newNode->prev = head;
16         if (head) head->prev = newNode;
17         head = newNode;
18         return;
19     }
20     Node* temp = head;
21     for (int i = 1; i < pos - 1 && temp != nullptr; i++) {
22         temp = temp->next;
23     }
24     if (temp == nullptr) return;
25     newNode->prev = temp->next;
26     if (temp->next) temp->next->prev = newNode;
27     temp->next = newNode;
28     newNode->next = temp->next;
29 }
30 void traverse(Node* head) {
31     Node* temp = head;
32     while (temp != nullptr) {
33         cout << temp->data << " ";
34         temp = temp->next;
35     }
36     cout << endl;
37 }
38 int main() {
39     Node* head = new Node('C', nullptr, nullptr);
40     head->next = new Node('E', head, nullptr);
41     insertAtPosition(head, 'P', 2);
42     traverse(head); // Output: C P E
43     return 0;
44 }
```

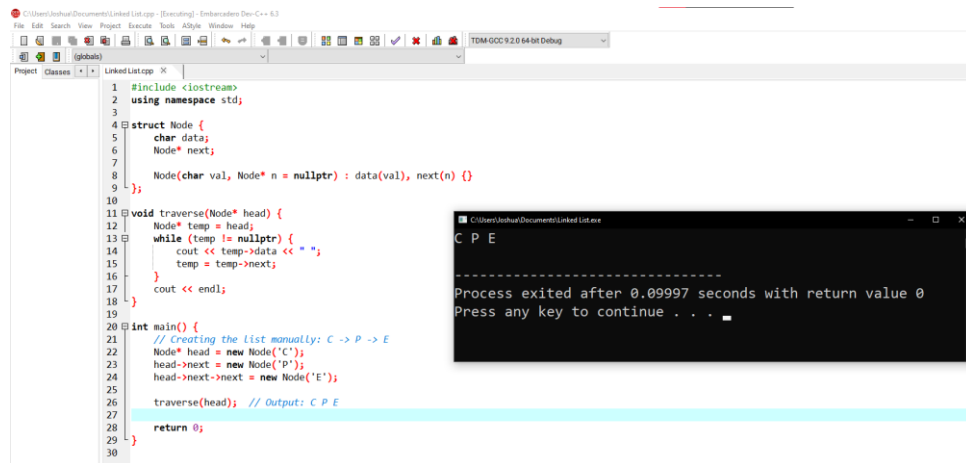
Process exited after 0.09562 seconds with return value 0  
Press any key to continue . . .

## Deletion of a node:

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

Process exited after 0.1001 seconds with return value 0  
Press any key to continue . . .

## Source Code:

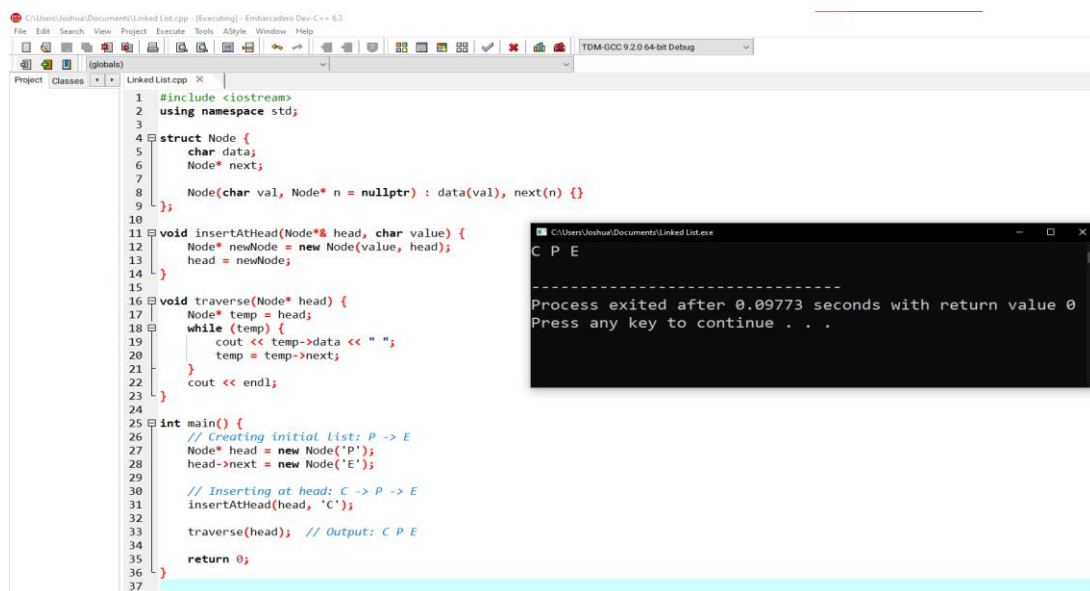


```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7
8     Node(char val, Node* n = nullptr) : data(val), next(n) {}
9 };
10
11 void traverse(Node* head) {
12     Node* temp = head;
13     while (temp != nullptr) {
14         cout << temp->data << "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

Terminal Output:

```
C P E
-----
Process exited after 0.09997 seconds with return value 0
Press any key to continue . . .
```

## Source Code:

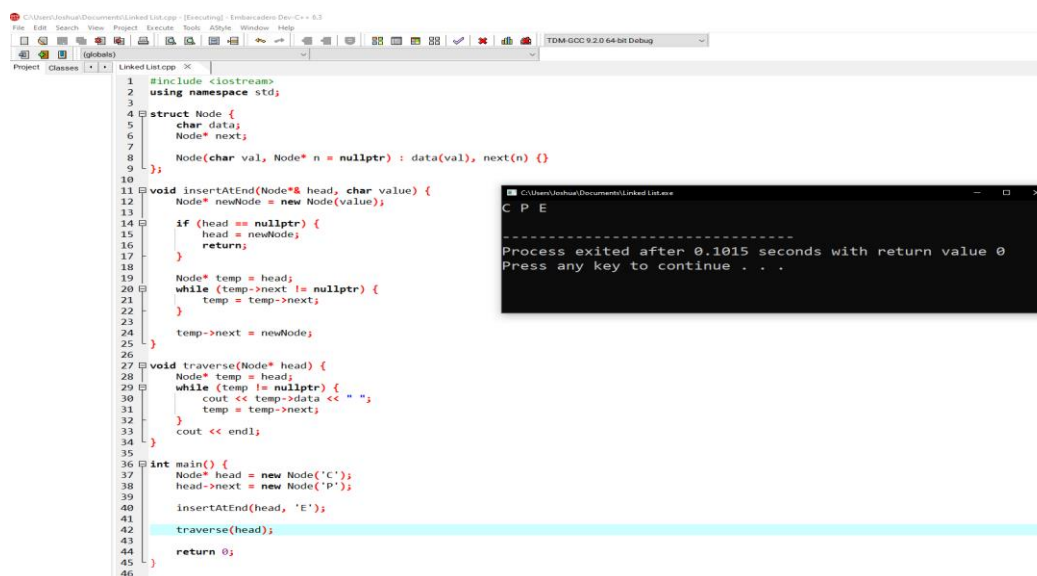


```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7
8     Node(char val, Node* n = nullptr) : data(val), next(n) {}
9 };
10
11 void insertAtHead(Node*& head, char value) {
12     Node* newNode = new Node(value, head);
13     head = newNode;
14 }
15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

Terminal Output:

```
C P E
-----
Process exited after 0.09773 seconds with return value 0
Press any key to continue . . .
```

## Source Code:

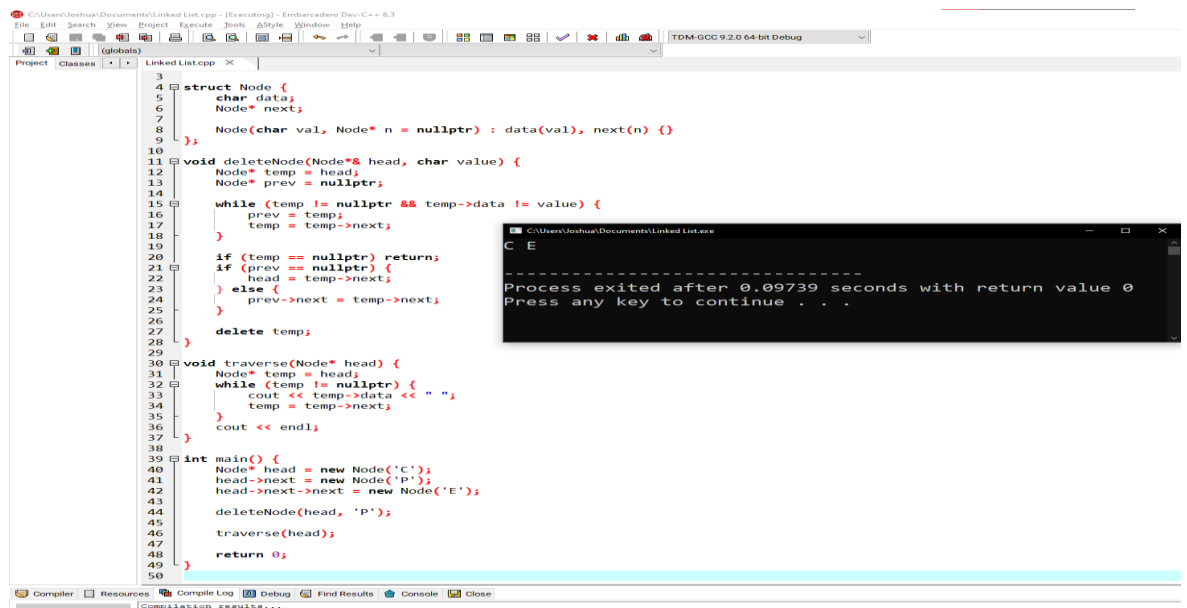


```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7
8     Node(char val, Node* n = nullptr) : data(val), next(n) {}
9 };
10
11 void insertAtEnd(Node*& head, char value) {
12     Node* newNode = new Node(value);
13
14     if (head == nullptr) {
15         head = newNode;
16         return;
17     }
18
19     Node* temp = head;
20     while (temp->next != nullptr) {
21         temp = temp->next;
22     }
23
24     temp->next = newNode;
25 }
26
27 void traverse(Node* head) {
28     Node* temp = head;
29     while (temp != nullptr) {
30         cout << temp->data << " ";
31         temp = temp->next;
32     }
33     cout << endl;
34 }
35
36 int main() {
37     Node* head = new Node('C');
38     head->next = new Node('P');
39
40     insertAtEnd(head, 'E');
41
42     traverse(head);
43
44     return 0;
45 }
46
```

Terminal Output:

```
C P E
-----
Process exited after 0.1015 seconds with return value 0
Press any key to continue . . .
```

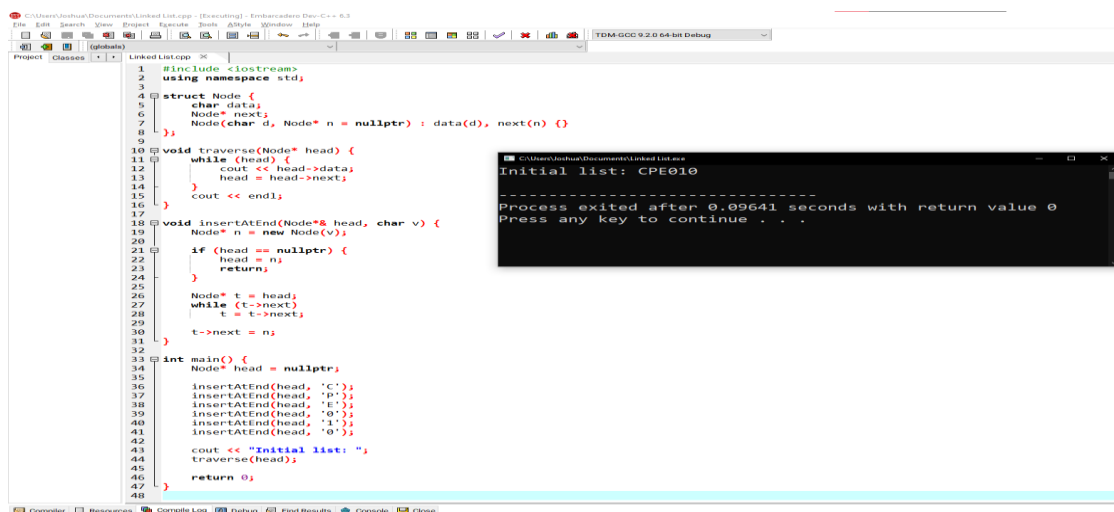
## Source Code:



```
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char val, Node* n = nullptr) : data(val), next(n) {}
9
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
30 void traverse(Node* head) {
31     Node* temp = head;
32     while (temp != nullptr) {
33         cout << temp->data << " ";
34         temp = temp->next;
35     }
36     cout << endl;
37 }
38
39 int main() {
40     Node* head = new Node('C');
41     head->next = new Node('P');
42     head->next->next = new Node('E');
43     deleteNode(head, 'P');
44     traverse(head);
45     return 0;
46 }
47
48
49
50
```

Process exited after 0.09739 seconds with return value 0  
Press any key to continue . . .

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

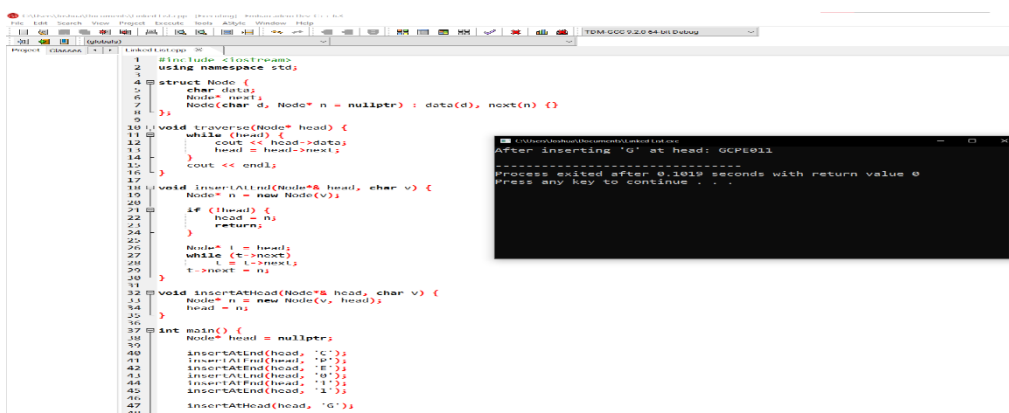


```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char d, Node* n = nullptr) : data(d), next(n) {}
9
10
11 void traverse(Node* head) {
12     while (head) {
13         cout << head->data;
14         head = head->next;
15     }
16     cout << endl;
17 }
18
19 void insertAtEnd(Node*& head, char v) {
20     Node* n = new Node(v);
21
22     if (head == nullptr) {
23         head = n;
24     }
25
26     Node* t = head;
27     while (t->next) {
28         t = t->next;
29     }
30     t->next = n;
31 }
32
33 int main() {
34     Node* head = nullptr;
35
36     insertAtEnd(head, 'C');
37     insertAtEnd(head, 'P');
38     insertAtEnd(head, 'E');
39     insertAtEnd(head, '0');
40     insertAtEnd(head, '1');
41     insertAtEnd(head, '0');
42
43     cout << "Initial list: ";
44     traverse(head);
45     return 0;
46 }
47
48
49
```

Initial list: CPE010  
Process exited after 0.09641 seconds with return value 0  
Press any key to continue . . .

### Analysis:

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



```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char d, Node* n = nullptr) : data(d), next(n) {}
9
10
11 void traverse(Node* head) {
12     while (head) {
13         cout << head->data;
14         head = head->next;
15     }
16     cout << endl;
17 }
18
19 void insertAtEnd(Node*& head, char v) {
20     Node* n = new Node(v);
21
22     if (head == nullptr) {
23         head = n;
24     }
25
26     Node* t = head;
27     while (t->next) {
28         t = t->next;
29     }
30     t->next = n;
31 }
32
33 void insertAtHead(Node*& head, char v) {
34     Node* n = new Node(v, head);
35     head = n;
36 }
37
38 int main() {
39     Node* head = nullptr;
40
41     insertAtEnd(head, 'C');
42     insertAtEnd(head, 'P');
43     insertAtEnd(head, 'E');
44     insertAtEnd(head, '0');
45     insertAtEnd(head, '1');
46     insertAtHead(head, 'G');
47
48     traverse(head);
49 }
```

After inserting 'G' at head: GCPE011  
Process exited after 0.1019 seconds with return value 0  
Press any key to continue . . .

### Analysis:

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

Insert a head

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node(char d, Node* n = nullptr) : data(d), next(n) {}
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 insertAtEnd(Node*& head, char v) {
19     Node* n = new Node(v);
20     if (!head) {
21         head = n;
22         return;
23     }
24     Node* t = head;
25     while (t->next)
26         t = t->next;
27     t->next = n;
28 }
29
30 void insertAtHead(Node*& head, char v) {
31     Node* n = new Node(v, head);
32     head = n;
33 }
34
35 void insertAfter(Node* prev, char v) {
36     if (!prev) return;
37     Node* n = new Node(v, prev->next);
38     prev->next = n;
39 }
40
41 int main() {
42     Node* head = nullptr;
43     insertAtEnd(head, 'C');
44     insertAtEnd(head, 'P');
45     insertAtEnd(head, 'E');
46 }
```

After inserting 'E' after 'P': GCPEE011  
Process exited after 0.09641 seconds with return value 0  
Press any key to continue . . .

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

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node(char d, Node* n = nullptr) : data(d), next(n) {}
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 insertAtEnd(Node*& head, char v) {
19     Node* n = new Node(v);
20     if (!head) {
21         head = n;
22         return;
23     }
24     Node* t = head;
25     while (t->next) t = t->next;
26     t->next = n;
27 }
28
29 void insertAtHead(Node*& head, char v) {
30     Node* n = new Node(v, head);
31     head = n;
32 }
33
34 void insertAfter(Node* prev, char v) {
35     if (!prev) return;
36     Node* n = new Node(v, prev->next);
37     prev->next = n;
38 }
39
40 void deleteNode(Node*& head, char key) {
41     if (!head) return;
42     if (head->data == key) {
43         Node* temp = head;
44         head = head->next;
45         delete temp;
46     }
47 }
48
49 int main() {
50     Node* head = nullptr;
51     insertAtEnd(head, 'C');
52     insertAtEnd(head, 'P');
53     insertAtEnd(head, 'E');
54     deleteNode(head, 'C');
55 }
```

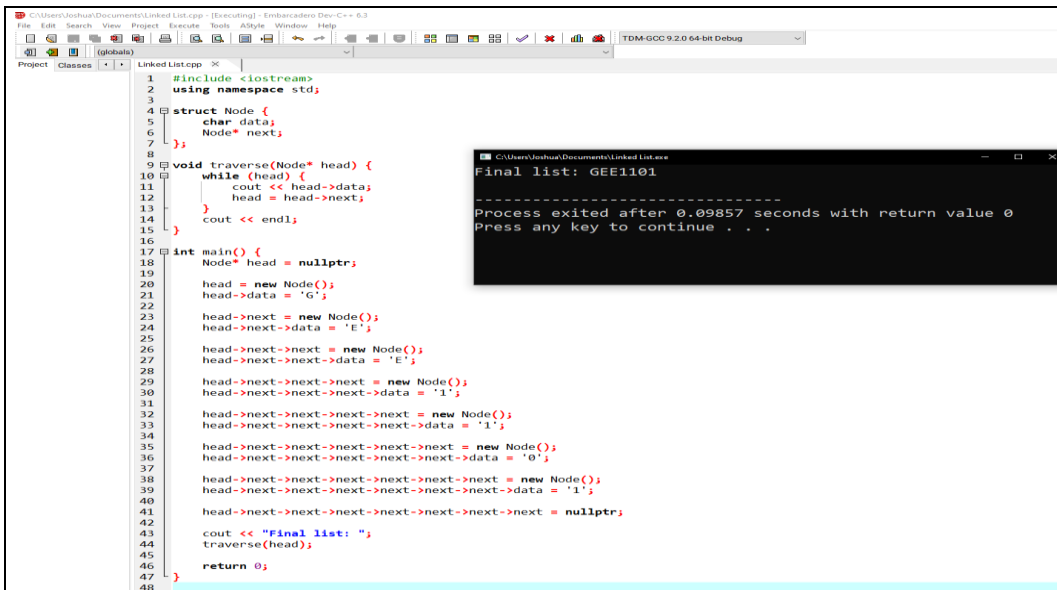
After inserting 'E' after 'P': GCPEE011  
After deleting 'C': GPEEE011  
Process exited after 0.1027 seconds with return value 0  
Press any key to continue . . .

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

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node(char d, Node* n = nullptr) : data(d), next(n) {}
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 insertAtEnd(Node*& head, char v) {
19     Node* n = new Node(v, nullptr);
20     if (!head) {
21         head = n;
22         return;
23     }
24     Node* t = head;
25     while (t->next) t = t->next;
26     t->next = n;
27 }
28
29 void insertAtHead(Node*& head, char v) {
30     Node* n = new Node(v, head);
31     head = n;
32 }
33
34 void insertAfter(Node* prev, char v) {
35     if (!prev) return;
36     Node* n = new Node(v, prev->next);
37     prev->next = n;
38 }
39
40 void deleteNode(Node*& head, char key) {
41     if (!head) return;
42     if (head->data == key) {
43         Node* temp = head;
44         head = head->next;
45         delete temp;
46     }
47 }
48
49 int main() {
50     Node* head = nullptr;
51     insertAtEnd(head, 'C');
52     insertAtEnd(head, 'P');
53     insertAtEnd(head, 'E');
54     deleteNode(head, 'P');
55 }
```

After inserting 'E' after 'P': GCPEE011  
After deleting 'P': GCCEE011  
Process exited after 0.1046 seconds with return value 0  
Press any key to continue . . .

Analysis: Applying deletion again to remove “P”.



The screenshot shows a C++ IDE with a file named 'LinkedList.cpp'. The code defines a 'Node' struct with 'data' and 'next' pointers. A 'traverse' function prints the data of each node in the list. The 'main' function creates a linked list with nodes containing 'G', 'E', 'E', '1', '1', '0', and '1'. It then calls 'traverse' to display the list. The output window shows 'Final list: GEE1101' and a message indicating the process exited after 0.09857 seconds.

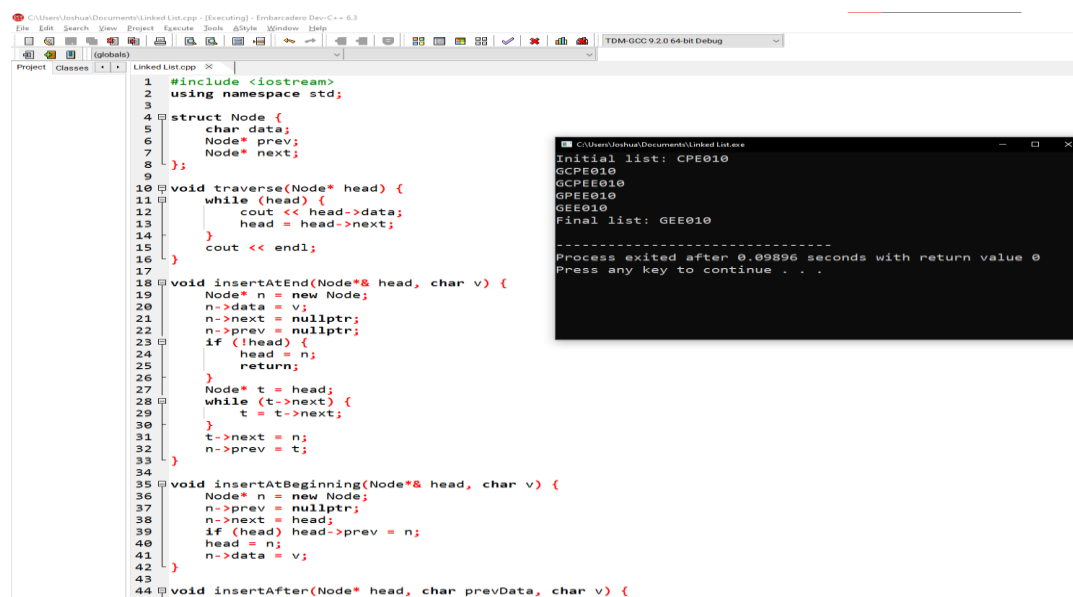
```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 };
8
9 void traverse(Node* head) {
10     while (head) {
11         cout << head->data;
12         head = head->next;
13     }
14     cout << endl;
15 }
16
17 int main() {
18     Node* head = nullptr;
19
20     head = new Node();
21     head->data = 'G';
22
23     head->next = new Node();
24     head->next->data = 'E';
25
26     head->next->next = new Node();
27     head->next->next->data = 'E';
28
29     head->next->next->next = new Node();
30     head->next->next->next->data = '1';
31
32     head->next->next->next->next = new Node();
33     head->next->next->next->next->data = '1';
34
35     head->next->next->next->next->next = new Node();
36     head->next->next->next->next->next->data = '0';
37
38     head->next->next->next->next->next->next = new Node();
39     head->next->next->next->next->next->next->data = '1';
40
41     head->next->next->next->next->next->next->next = nullptr;
42
43     cout << "Final list: ";
44     traverse(head);
45
46     return 0;
47 }
```

Final list: GEE1101

Process exited after 0.09857 seconds with return value 0  
Press any key to continue . . .

Analysis: Displaying the result after all operations.

Table 3-4. Modified Operations for Doubly Linked Lists



The screenshot shows a C++ IDE with a file named 'LinkedList.cpp'. The code defines a 'Node' struct with 'data', 'prev', and 'next' pointers. It includes functions for traversing the list, inserting a node at the end, inserting a node at the beginning, and inserting a node after a given node. The 'main' function demonstrates these operations. The output window shows the initial list 'CPE010', the insertion of 'GCPE010', 'GPPE010', and 'GEE010', and the final list 'GEE010'. It also shows the process exiting after 0.09896 seconds.

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* prev;
7     Node* next;
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 insertAtEnd(Node*& head, char v) {
19     Node* n = new Node;
20     n->data = v;
21     n->next = nullptr;
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) {
```

Initial list: CPE010  
GCPE010  
GPPE010  
GEE010  
Final list: GEE010

Process exited after 0.09896 seconds with return value 0  
Press any key to continue . . .

## 7. Supplementary Activity:

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Song {
6      string title;
7      Song* next;
8      Song* prev;
9  };
10
11 void addSong(Song*& head, const string& title) {
12     Song* newSong = new Song(title, nullptr, nullptr);
13     if (!head) {
14         head = newSong;
15         head->next = head;
16         head->prev = head;
17         return;
18     }
19     Song* tail = head->prev;
20     tail->next = newSong;
21     newSong->prev = tail;
22     newSong->next = head;
23     head->prev = newSong;
24 }
25
26 void removeSong(Song*& head, const string& title) {
27     if (!head) return;
28     Song* curr = head;
29     do {
30         if (curr->title == title) {
31             if (curr->next == curr) {
32                 delete curr;
33                 head = nullptr;
34                 return;
35             }
36             curr->prev->next = curr->next;
37             curr->next->prev = curr->prev;
38             if (curr == head) head = curr->next;
39             delete curr;
40             return;
41         }
42         curr = curr->next;
43     } while (curr != head);
44 }
45

```

```

41     }
42     curr = curr->next;
43     } while (curr != head);
44 }
45
46 void playAll(Song* head) {
47     if (!head) {
48         cout << "Playlist is empty.\n";
49         return;
50     }
51     Song* curr = head;
52     do {
53         cout << "Playing: " << curr->title << endl;
54         curr = curr->next;
55     } while (curr != head);
56 }
57
58 Song* nextSong(Song* curr) {
59     if (!curr) return nullptr;
60     return curr->next;
61 }
62
63 Song* prevSong(Song* curr) {
64     if (!curr) return nullptr;
65     return curr->prev;
66 }
67
68 int main() {
69     Song* playlist = nullptr;
70
71     addSong(playlist, "Song A");
72     addSong(playlist, "Song B");
73     addSong(playlist, "Song C");
74     addSong(playlist, "Song D");
75     addSong(playlist, "Song E");
76
77     cout << "\nInitial Playlist:\n";
78     playAll(playlist);
79
80     cout << "\nRemoving Song B...\n";
81     removeSong(playlist, "Song B");
82     playAll(playlist);
83
84     Song* current = playlist;
85     cout << "\nCurrently playing: " << current->title << endl;
86     current = nextSong(current);
87     cout << "Next song: " << current->title << endl;
88     current = prevSong(current);
89     cout << "Previous song: " << current->title << endl;
90
91     return 0;
92 }
93

```

```

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 A

-----
Process exited after 0.1066 seconds with return value 0
Press any key to continue . . .

```

**8. Conclusion:** In this activity 3.1 Linked Lists, I understand how to revise the single linked lists with connection to doubly linked lists, which gave us an idea how did pointers work. We need to be careful in moving to the next and previous pointers. In general I need to study and learn more to improve my skills to organize my codes just to look clean and neat so it that it can understand it easily.

## 9. Assessment Rubric