

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

```
main.cpp
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     char data;
7     Node* next;
8 };
9
10 int main() {
11     Node* head = NULL;
12     Node* second = NULL;
13     Node* third = NULL;
14     Node* fourth = NULL;
15     Node* fifth = NULL;
16     Node* last = NULL;
17
18     head = new Node;
19     second = new Node;
20     third = new Node;
21     fourth = new Node;
22     fifth = new Node;
23     last = new Node;
24
25     head->data = 'C';
26     head->next = second;
27
28     second->data = 'P';
29     second->next = third;
30
31     third->data = 'E';
32     third->next = fourth;
33
34     fourth->data = 'O';
35     fourth->next = fifth;
36
37     fifth->data = 'I';
38     fifth->next = last;
39
40     last->data = 'L';
41     last->next = nullptr;
```

CPE010
==== Code Execution Successful ====

Analysis:

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

Traversal:

```
main.cpp
```

```
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     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     // Example List: C -> P -> E -> O -> I -> 0
21     Node* head = new Node{'C', nullptr, nullptr};
22     Node* second = new Node{'P', head, nullptr};
23     head->next = second;
24     Node* third = new Node{'E', second, nullptr};
25     second->next = third;
26     Node* fourth = new Node{'O', third, nullptr};
27     third->next = fourth;
28     Node* fifth = new Node{'I', fourth, nullptr};
29     fourth->next = fifth;
30     Node* sixth = new Node{'0', fifth, nullptr};
31     fifth->next = sixth;
32
33     traverse(head);
34 }
```



Output

C P E O I 0

== Code Execution Successful ==

Insertion at Head:

```
main.cpp
```

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



Output

C P E O I 0

== Code Execution Successful ==

Insertion at any part of the list:

```
main.cpp | Run | Output
```

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

Insertion at the End:

```
main.cpp
```

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

C P E
== Code Execution Successful ==

Deletion of a Node:

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

Source Code:

main.cpp



Share

Run

Output

```
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    Node* temp = head;
11    while (temp != nullptr) {
12        cout << temp->data << " ";
13        temp = temp->next;
14    }
15    cout << endl;
16 }
17
18 int main() {
19     Node* head = new Node{'C', nullptr};
20     head->next = new Node{'P', nullptr};
21     head->next->next = new Node{'E', nullptr};
22
23     traverse(head);
24
25     return 0;
26 }
```

C P E

== Code Execution Successful ==

Source Code:

main.cpp



Share

Run

Output

C P E

== Code Execution Successful ==

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

Source Code:

The screenshot shows a code editor interface with a dark theme. The left panel contains the source code for `main.cpp`. The right panel shows the output of the code execution.

Source Code (main.cpp):

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 };
8
9 void insertAfter(Node* prevNode, char value) {
10 if (prevNode == nullptr) {
11     cout << "Previous node cannot be null." << endl;
12     return;
13 }
14 Node* newNode = new Node{value, prevNode->next};
15 prevNode->next = newNode;
16 }
17
18 void traverse(Node* head) {
19     Node* temp = head;
20     while (temp) {
21         cout << temp->data << " ";
22         temp = temp->next;
23     }
24     cout << endl;
25 }
26
27 int main() {
28     Node* head = new Node{'C', nullptr};
29     head->next = new Node{'E', nullptr};
30
31     insertAfter(head, 'P');
32
33     traverse(head);
34 }
```

Output:

```
C P E
== Code Execution Successful ==
```

Source Code:

```
main.cpp | Run | Output
```

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

C P E
== Code Execution Successful ==

Source Code:

main.cpp

```
4 struct Node {
5     char data;
6     Node* next;
7 };
8
9 void deleteNode(Node*& head, char value) {
10    Node* temp = head;
11    Node* prev = nullptr;
12
13    while (temp != nullptr && temp->data != value) {
14        prev = temp;
15        temp = temp->next;
16    }
17
18    if (temp == nullptr) return;
19
20    if (prev == nullptr) {
21        head = temp->next;
22    } else {
23        prev->next = temp->next;
24    }
25
26    delete temp;
27 }
28
29 void traverse(Node* head) {
30    Node* temp = head;
31    while (temp) {
32        cout << temp->data << " ";
33        temp = temp->next;
34    }
35    cout << endl;
36 }
37
38 int main() {
39    Node* head = new Node{'C', nullptr};
40    head->next = new Node{'P', nullptr};
41    head->next->next = new Node{'E', nullptr};
42
43    deleteNode(head, 'P');
44    traverse(head);
45
46    return 0;
47 }
48
```

Output

CE

== Code Execution Successful ==

Screenshots:

```
main.cpp
```

Run	Output
	Initial list: CPE101 ==== Code Execution Successful ===

```
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 void insertAtEnd(Node*& head, char v) {
18     Node* n = new Node;
19     n->data = v;
20     n->next = nullptr;
21     if (!head) {
22         head = n;
23         return;
24     }
25     Node* t = head;
26     while (t->next) {
27         t = t->next;
28     }
29     t->next = n;
30 }
31
32 int main() {
33     Node* head = nullptr;
34     insertAtEnd(head, 'C');
35     insertAtEnd(head, 'P');
36     insertAtEnd(head, 'E');
37     insertAtEnd(head, 'I');
38     insertAtEnd(head, 'O');
39     insertAtEnd(head, 'I');
40     cout << "Initial list: ";
41     traverse(head);
42     return 0;
43 }
```

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:

The screenshot shows a code editor window with the following details:

- Title Bar:** The title bar displays "main.cpp".
- Toolbar:** The toolbar includes icons for file operations, a search bar, and buttons for "Share" and "Run".
- Code Area:** The main area contains C++ code for a singly linked list. It defines a `Node` structure, implements a `traverse` function to print the list, and provides two insertion functions: `insertAtEnd` and `insertAtHead`. The `main` function demonstrates inserting nodes with characters 'C', 'P', 'E', 'I', 'O', and 'G' at the head.
- Output Area:** The output panel shows the result of the execution: "After inserting 'G' at head: GCPE101" and "==== Code Execution Successful ===".

```
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 void insertAtEnd(Node*& head, char v) {
18     Node* n = new Node;
19     n->data = v;
20     n->next = nullptr;
21     if (!head) {
22         head = n;
23         return;
24     }
25     Node* t = head;
26     while (t->next) {
27         t = t->next;
28     }
29     t->next = n;
30 }
31
32 void insertAtHead(Node*& head, char v) {
33     Node* n = new Node;
34     n->data = v;
35     n->next = head;
36     head = n;
37 }
38
39 int main() {
40     Node* head = nullptr;
41     insertAtEnd(head, 'C');
42     insertAtEnd(head, 'P');
43     insertAtEnd(head, 'E');
44     insertAtEnd(head, 'I');
45     insertAtEnd(head, 'O');
46     insertAtHead(head, 'G');
}
```

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 | Run | Output
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 void insertAtEnd(Node*& head, char v) {
18     Node* n = new Node;
19     n->data = v;
20     n->next = nullptr;
21     if (!head) {
22         head = n;
23         return;
24     }
25     Node* t = head;
26     while (t->next) {
27         t = t->next;
28     }
29     t->next = n;
30 }
31
32 void insertAtHead(Node*& head, char v) {
33     Node* n = new Node;
34     n->data = v;
35     n->next = head;
36     head = n;
37 }
38
39 void insertAfter(Node* prev, char v) {
40     if (!prev) return;
41     Node* n = new Node;
42     n->data = v;
43     n->next = prev->next;
44     prev->next = n;
45 }
```

After inserting 'E' after 'P': GPPEE101

== 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:

The screenshot shows a C++ IDE interface with the following details:

- Title Bar:** The title bar displays "main.cpp".
- Toolbar:** The toolbar includes icons for file operations, a search bar, and buttons for "Run" and "Output".
- Code Editor:** The main area contains the source code for a linked list implementation. The code includes functions for traversing the list, inserting nodes at the end, beginning, and after a specific node, and deleting a node by value.
- Output Panel:** The right panel shows the output of the program's execution. It displays the message "After deleting 'C': GCPEE101" followed by "---- Code Execution Successful ----".

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 | After deleting 'C': GCPEE101
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 void insertAtEnd(Node*& head, char v) {
18    Node* n = new Node;
19    n->data = v;
20    n->next = nullptr;
21    if (!head) {
22        head = n;
23        return;
24    }
25    Node* t = head;
26    while (t->next) {
27        t = t->next;
28    }
29    t->next = n;
30 }
31
32 void insertAtHead(Node*& head, char v) {
33    Node* n = new Node;
34    n->data = v;
35    n->next = head;
36    head = n;
37 }
38
39 void insertAfter(Node* prev, char v) {
40    if (!prev) return;
41    Node* n = new Node;
42    n->data = v;
43    n->next = prev->next;
44    prev->next = n;
45 }
46
47 void deleteNode(Node*& head, char key) {
48    if (head == nullptr) return;
49
50    if (head->data == key) {
51        Node* tmp = head;
52        head = head->next;
53        delete tmp;
54        return;
55    }
56
57    Node* cur = head;
58    while (cur->next && cur->next->data != key) {
59        cur = cur->next;
60    }
61
62    if (cur->next) {
63        Node* tmp = cur->next;
64        cur->next = tmp->next;
65        delete tmp;
66    }
67 }
68
69 int main() {
70    Node* head = nullptr;
71    insertAtEnd(head, 'C');
72    cout << "After deleting 'C': GCPEE101"
73 }
```

Analysis:

Applying deletion again to remove “P”

Screenshot:

```
main.cpp | Run | Output
```

```
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     head = new Node;
20     head->data = 'G';
21     head->next = new Node;
22     head->next->data = 'C';
23     head->next->next = new Node;
24     head->next->next->data = 'E';
25     head->next->next->next = new Node;
26     head->next->next->next->data = 'I';
27     head->next->next->next->next = new Node;
28     head->next->next->next->next->data = 'O';
29     head->next->next->next->next->next = new Node;
30     head->next->next->next->next->next->data = 'I';
31     head->next->next->next->next->next->next = nullptr;
32
33     cout << "Final list: ";
34     traverse(head);
35     return 0;
36 }
```

Final list: GCEIOI

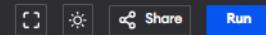
== Code Execution Successful ==

Analysis:

Displaying the result after all operations

7. Supplementary Activity

```
main.cpp
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) {
39                 head = curr->next;
40             }
41             delete curr;
42             return;
43         }
44         curr = curr->next;
45     } while (curr != head);
46 }
```



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 ==

```

4     return;
5     curr = curr->next;
6 } while (curr != head);
7 }

8 void playAll(Song* head) {
9     if (!head) {
10         cout << "Playlist is empty." << endl;
11         return;
12     }
13     Song* curr = head;
14     do {
15         cout << "Playing: " << curr->title << endl;
16         curr = curr->next;
17     } while (curr != head);
18 }

19 Song* nextSong(Song* curr) {
20     if (!curr) return nullptr;
21     return curr->next;
22 }

23 Song* prevSong(Song* curr) {
24     if (!curr) return nullptr;
25     return curr->prev;
26 }

27 int main() {
28     Song* playlist = nullptr;
29     addSong(playlist, "Song A");
30     addSong(playlist, "Song B");
31     addSong(playlist, "Song C");
32     addSong(playlist, "Song D");
33     addSong(playlist, "Song E");
34

35     cout << "\nInitial Playlist:\n";
36     playAll(playlist);
37

38     cout << "\nRemoving Song B...\n";
39     removeSong(playlist, "Song B");
40     playAll(playlist);
41

42     Song* current = playlist;
43     cout << "\nCurrently playing: " << current->title << endl;
44     current = nextSong(current);
45     cout << "Next song: " << current->title << endl;
46     current = prevSong(current);
47     cout << "Previous song: " << current->title << endl;
48

49     return 0;
50 }
51
52

```

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