

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:

The screenshot shows a Java development environment with a code editor and a terminal window.

Code Editor Content:

```
1 package com.simplilearn;
2
3 public class Main {
4     public static void main(String[] args) {
5         Node head = null;
6         Node curr = head;
7         Node temp;
8
9         // Step 1: Create nodes
10        Node node1 = new Node();
11        Node node2 = new Node();
12        Node node3 = new Node();
13        Node node4 = new Node();
14        Node node5 = new Node();
15
16        // Step 2: Assign next
17        head.next = node1;
18        node1.next = node2;
19        node2.next = node3;
20        node3.next = node4;
21        node4.next = node5;
22        node5.next = null;
23
24        // Step 3: Assign data and Link nodes
25        head.data = "T";
26        head.next = record;
27
28        record.data = "W";
29        record.next = record;
30
31        record.data = "Q";
32        record.next = record;
33
34        record.data = "W";
35        record.next = record;
36
37        record.data = "T";
38        record.next = record;
39
40        curr = head;
41
42        // operation : print the list to verify
43        while (temp != null) {
44            System.out.println(temp.data);
45            temp = temp.next;
46        }
47    }
48 }
```

Terminal Window:

```
Process: maind[1193] seconds with return value 0
Press any key to continue . . .
```

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 Java development environment with a code editor and a terminal window.

Code Editor Content:

```
1 package com.simplilearn;
2
3 public class Main {
4     public static void main(String[] args) {
5         Node head = null;
6         Node curr = head;
7         Node temp;
8
9         // Step 1: Create nodes
10        Node node1 = new Node();
11        Node node2 = new Node();
12        Node node3 = new Node();
13        Node node4 = new Node();
14        Node node5 = new Node();
15
16        // Step 2: Assign next
17        head.next = node1;
18        node1.next = node2;
19        node2.next = node3;
20        node3.next = node4;
21        node4.next = node5;
22        node5.next = null;
23
24        // Step 3: Assign data and Link nodes
25        head.data = "P";
26        head.next = record;
27
28        record.data = "F";
29        record.next = record;
30
31        record.data = "O";
32        record.next = record;
33
34        record.data = "D";
35        record.next = record;
36
37        record.data = "P";
38        record.next = record;
39
40        curr = head;
41
42        // operation : print the list to verify
43        while (temp != null) {
44            System.out.println(temp.data);
45            temp = temp.next;
46        }
47    }
48 }
```

Terminal Window:

```
Process: maind[1193] seconds with return value 0
Press any key to continue . . .
```

Traversal:

```

1 // Insert at Head
2
3 #include <iostream>
4
5 struct Node {
6     int data;
7     Node* next;
8 }
9
10 void insertHead(Node** head, int value) {
11     Node* newNode = new Node();
12     newNode->data = value;
13     newNode->next = *head;
14     *head = newNode;
15 }
16
17 void printList(Node* head) {
18     while (head != NULL) {
19         std::cout << head->data << " ";
20         head = head->next;
21     }
22 }
23
24 int main() {
25     Node* head = NULL;
26     insertHead(&head, 1);
27     insertHead(&head, 2);
28     insertHead(&head, 3);
29     insertHead(&head, 4);
30
31     printList(head);
32 }

```

Process exited after 0.00026 seconds with return value 0
Press any key to continue . . .

Insertion at Head:

Insertion at

```

1 // Insert at End
2
3 #include <iostream>
4
5 struct Node {
6     int data;
7     Node* next;
8 }
9
10 void insertEnd(Node** head, int value) {
11     Node* newNode = new Node();
12     newNode->data = value;
13     if (*head == NULL) {
14         *head = newNode;
15         return;
16     }
17     Node* temp = *head;
18     while (temp->next != NULL) {
19         temp = temp->next;
20     }
21     temp->next = newNode;
22 }
23
24 void printList(Node* head) {
25     while (head != NULL) {
26         std::cout << head->data << " ";
27         head = head->next;
28     }
29 }
30
31 int main() {
32     Node* head = NULL;
33     insertEnd(&head, 1);
34     insertEnd(&head, 2);
35     insertEnd(&head, 3);
36     insertEnd(&head, 4);
37
38     printList(head);
39 }

```

Process exited after 0.00026 seconds with return value 0
Press any key to continue . . .

the end:

Deletion of a

```

1 // Delete Node
2
3 #include <iostream>
4
5 struct Node {
6     int data;
7     Node* next;
8 }
9
10 void deleteNode(Node** head, int value) {
11     Node* temp = *head;
12     Node* prev = NULL;
13
14     while (temp != NULL) {
15         if (temp->data == value) {
16             if (prev == NULL) {
17                 *head = temp->next;
18             } else {
19                 prev->next = temp->next;
20             }
21             delete temp;
22             return;
23         }
24         prev = temp;
25         temp = temp->next;
26     }
27 }
28
29 void printList(Node* head) {
30     while (head != NULL) {
31         std::cout << head->data << " ";
32         head = head->next;
33     }
34 }
35
36 int main() {
37     Node* head = NULL;
38     insertEnd(&head, 1);
39     insertEnd(&head, 2);
40     insertEnd(&head, 3);
41     insertEnd(&head, 4);
42
43     printList(head);
44     deleteNode(&head, 2);
45
46     printList(head);
47 }

```

Process exited after 0.100 seconds with return value 0
Press any key to continue . . .

node:

Source Code: Source Code:

The screenshot shows a C++ IDE interface with two panes. The left pane displays the source code for a linked list implementation. The right pane shows the execution results, including the output of the program and the time taken.

```
1 #include <iostream>
2 using namespace std;
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 insertHead(Node*& head, char value) {
12     Node* node = new Node(value, head);
13     head = node;
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 }
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     insertHead(head, 'C');
31
32     traverse(head); // Output: C P E
33
34     return 0;
35 }
36
37
```

Output window:
C P E
Process exited after 0.09773 seconds
Press any key to continue . . .

Source Code:

The screenshot shows a C++ IDE interface with two panes. The left pane displays the source code for a linked list implementation. The right pane shows the execution results, including the output of the program and the time taken.

```
1 #include <iostream>
2 using namespace std;
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 insertHead(Node*& head, char value) {
12     Node* node = new Node(value);
13
14     if (head == nullptr) {
15         head = node;
16         return;
17     }
18
19     Node* temp = head;
20     while (temp->next != nullptr) {
21         temp = temp->next;
22     }
23
24     temp->next = node;
25 }
26
27 void traverse(Node* head) {
28     Node* temp = head;
29     while (temp->next != nullptr) {
30         cout << temp->data << " ";
31         temp = temp->next;
32     }
33
34 }
35
36 int main() {
37     // creating one list: C -> P -> E
38     Node* head = new Node('C');
39     head->next = new Node('P');
40     head->next->next = new Node('E');
41
42     traverse(head); // Output: C P E
43
44     return 0;
45 }
```

Output window:
C P E
Process exited after 0.09
Press any key to continue . . .

The screenshot shows a C++ IDE interface with two panes. The left pane displays the source code for a linked list implementation. The right pane shows the execution results, including the output of the program and the time taken.

```
1 #include <iostream>
2 using namespace std;
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 insertHead(Node*& head, char value) {
12     Node* node = new Node(value);
13
14     if (head == nullptr) {
15         head = node;
16         return;
17     }
18
19     Node* temp = head;
20     while (temp->next != nullptr) {
21         temp = temp->next;
22     }
23
24     temp->next = node;
25 }
26
27 void traverse(Node* head) {
28     Node* temp = head;
29     while (temp->next != nullptr) {
30         cout << temp->data << " ";
31         temp = temp->next;
32     }
33
34 }
35
36 int main() {
37     // creating one list: C -> P -> E
38     Node* head = new Node('C');
39     head->next = new Node('P');
40
41     insertHead(head, 'E');
42
43     traverse(head); // Output: C P E
44
45 }
```

Output window:
C P E
Process exited after 0.1015 seconds
Press any key to continue . . .

```

1 // Singly Linked List Implementation in C
2
3 #include <stdio.h>
4
5 struct Node {
6     char data;
7     struct Node* next;
8 }
9
10 void insertHead(struct Node** head, char value) {
11     struct Node* temp = head;
12     struct Node* prev = NULL;
13
14     while (temp != NULL && temp->data != value) {
15         prev = temp;
16         temp = temp->next;
17     }
18
19     if (temp == NULL) return;
20     if (prev == NULL) {
21         head = temp->next;
22     } else {
23         prev->next = temp->next;
24     }
25
26     temp->data = value;
27 }
28
29 void traverseList(struct Node* head) {
30     struct Node* temp = head;
31
32     while (temp != NULL) {
33         printf("%c ", temp->data);
34         temp = temp->next;
35     }
36
37 }
38
39 int main() {
40     struct Node* head = NULL;
41     head = insert('C');
42     head = insert('P');
43     head = insert('E');
44
45     insertHead(&head, 'P');
46
47     traverseList(head);
48
49     return 0;
50 }

```

Process exited after 0.09238 seconds with return value 0
Press any key to continue . . .

Source Code:

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

```

1 // Singly Linked List Implementation in C
2
3 #include <stdio.h>
4
5 struct Node {
6     char data;
7     struct Node* next;
8 }
9
10 void insertHead(struct Node** head, char value) {
11     struct Node* temp = head;
12
13     if (temp == NULL) {
14         *head = malloc(sizeof(struct Node));
15         (*head)->data = value;
16         (*head)->next = NULL;
17     } else {
18         struct Node* newnode = malloc(sizeof(struct Node));
19         newnode->data = value;
20         newnode->next = temp;
21
22         (*head) = newnode;
23     }
24 }
25
26 void insertEnd(struct Node** head, char value) {
27     struct Node* temp = head;
28
29     if (temp == NULL) {
30         *head = malloc(sizeof(struct Node));
31         (*head)->data = value;
32         (*head)->next = NULL;
33     } else {
34         while (temp->next != NULL) {
35             temp = temp->next;
36         }
37
38         temp->next = malloc(sizeof(struct Node));
39         temp = temp->next;
40         temp->data = value;
41         temp->next = NULL;
42     }
43 }
44
45 void printList(struct Node* head) {
46     struct Node* temp = head;
47
48     while (temp != NULL) {
49         printf("%c ", temp->data);
50         temp = temp->next;
51     }
52 }
53
54 int main() {
55     struct Node* head = NULL;
56
57     insert('C');
58     insert('P');
59     insert('E');
60
61     insertHead(&head, 'P');
62
63     printList(head);
64
65     return 0;
66 }

```

Initial list: CPE
Process exited after 0.09642 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 // Singly Linked List Implementation in C
2
3 #include <stdio.h>
4
5 struct Node {
6     char data;
7     struct Node* next;
8 }
9
10 void insertHead(struct Node** head, char value) {
11     struct Node* temp = head;
12
13     if (temp == NULL) {
14         *head = malloc(sizeof(struct Node));
15         (*head)->data = value;
16         (*head)->next = NULL;
17     } else {
18         struct Node* newnode = malloc(sizeof(struct Node));
19         newnode->data = value;
20         newnode->next = temp;
21
22         (*head) = newnode;
23     }
24 }
25
26 void insertEnd(struct Node** head, char value) {
27     struct Node* temp = head;
28
29     if (temp == NULL) {
30         *head = malloc(sizeof(struct Node));
31         (*head)->data = value;
32         (*head)->next = NULL;
33     } else {
34         while (temp->next != NULL) {
35             temp = temp->next;
36         }
37
38         temp->next = malloc(sizeof(struct Node));
39         temp = temp->next;
40         temp->data = value;
41         temp->next = NULL;
42     }
43 }
44
45 void printList(struct Node* head) {
46     struct Node* temp = head;
47
48     while (temp != NULL) {
49         printf("%c ", temp->data);
50         temp = temp->next;
51     }
52 }
53
54 int main() {
55     struct Node* head = NULL;
56
57     insert('C');
58     insert('P');
59     insert('E');
60
61     insertHead(&head, 'P');
62
63     printList(head);
64
65     return 0;
66 }

```

Initial list: CPE
Process exited after 0.09642 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. void insert(char data) {
2.     Node* head;
3.     Node* node;
4.     Node* newnode;
5.     if(head == NULL) {
6.         newnode = (Node*)malloc(sizeof(Node));
7.         newnode->data = data;
8.         newnode->next = NULL;
9.         head = newnode;
10.    } else {
11.        node = head;
12.        while(node->next != NULL) {
13.            if(node->data == data) {
14.                newnode = (Node*)malloc(sizeof(Node));
15.                newnode->data = data;
16.                newnode->next = node->next;
17.                node->next = newnode;
18.                break;
19.            }
20.            node = node->next;
21.        }
22.    }
23. }
24.
25. void insertAfter(char data, char key) {
26.     Node* head;
27.     Node* node;
28.     Node* newnode;
29.     if(head == NULL) {
30.         newnode = (Node*)malloc(sizeof(Node));
31.         newnode->data = data;
32.         newnode->next = NULL;
33.         head = newnode;
34.     } else {
35.         node = head;
36.         while(node->next != NULL) {
37.             if(node->data == key) {
38.                 newnode = (Node*)malloc(sizeof(Node));
39.                 newnode->data = data;
40.                 newnode->next = node->next;
41.                 node->next = newnode;
42.                 break;
43.             }
44.             node = node->next;
45.         }
46.     }
47. }
```

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

```
1. void insert(char data) {
2.     Node* head;
3.     Node* node;
4.     Node* newnode;
5.     if(head == NULL) {
6.         newnode = (Node*)malloc(sizeof(Node));
7.         newnode->data = data;
8.         newnode->next = NULL;
9.         head = newnode;
10.    } else {
11.        node = head;
12.        while(node->next != NULL) {
13.            if(node->data == data) {
14.                newnode = (Node*)malloc(sizeof(Node));
15.                newnode->data = data;
16.                newnode->next = node->next;
17.                node->next = newnode;
18.                break;
19.            }
20.            node = node->next;
21.        }
22.    }
23. }
24.
25. void insertAfter(char data, char key) {
26.     Node* head;
27.     Node* node;
28.     Node* newnode;
29.     if(head == NULL) {
30.         newnode = (Node*)malloc(sizeof(Node));
31.         newnode->data = data;
32.         newnode->next = NULL;
33.         head = newnode;
34.     } else {
35.         node = head;
36.         while(node->next != NULL) {
37.             if(node->data == key) {
38.                 newnode = (Node*)malloc(sizeof(Node));
39.                 newnode->data = data;
40.                 newnode->next = node->next;
41.                 node->next = newnode;
42.                 break;
43.             }
44.             node = node->next;
45.         }
46.     }
47. }
```

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

```
1. #include <conio.h>
2. using namespace std;
3.
4. struct Node {
5.     char data;
6.     Node* next;
7. };
8.
9. void insert(char data) {
10.    Node* head;
11.    Node* node;
12.    Node* newnode;
13.    if(head == NULL) {
14.        newnode = (Node*)malloc(sizeof(Node));
15.        newnode->data = data;
16.        newnode->next = NULL;
17.        head = newnode;
18.    } else {
19.        node = head;
20.        while(node->next != NULL) {
21.            if(node->data == data) {
22.                newnode = (Node*)malloc(sizeof(Node));
23.                newnode->data = data;
24.                newnode->next = node->next;
25.                node->next = newnode;
26.                break;
27.            }
28.            node = node->next;
29.        }
30.    }
31. }
32.
33. void insertAfter(char data, char key) {
34.    Node* head;
35.    Node* node;
36.    Node* newnode;
37.    if(head == NULL) {
38.        newnode = (Node*)malloc(sizeof(Node));
39.        newnode->data = data;
40.        newnode->next = NULL;
41.        head = newnode;
42.    } else {
43.        node = head;
44.        while(node->next != NULL) {
45.            if(node->data == key) {
46.                newnode = (Node*)malloc(sizeof(Node));
47.                newnode->data = data;
48.                newnode->next = node->next;
49.                node->next = newnode;
50.                break;
51.            }
52.            node = node->next;
53.        }
54.    }
55. }
56.
57. void insertBefore(Node* head, char key) {
58.    Node* node;
59.    Node* newnode;
60.    if(head == NULL) {
61.        cout << "List is empty" << endl;
62.    } else {
63.        node = head;
64.        while(node->next != NULL) {
65.            if(node->data == key) {
66.                newnode = (Node*)malloc(sizeof(Node));
67.                newnode->data = data;
68.                newnode->next = node;
69.                node->next = newnode;
70.                break;
71.            }
72.            node = node->next;
73.        }
74.    }
75. }
```

node.

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 header files
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     char data;
7     Node* prev;
8     Node* next;
9 };
10
11 void traverse(Node* head) {
12     while (head != NULL) {
13         cout << head->data;
14         head = head->next;
15     }
16 }
17
18 void insertAtHead(Node*& head, char v) {
19     Node* n = new Node();
20     n->data = v;
21     n->prev = NULL;
22     n->next = head;
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 insertAtEnd(Node*& head, char v) {
36     Node* n = new Node();
37     n->data = v;
38     n->prev = NULL;
39     n->next = head;
40     if (!head) head = n;
41     else {
42         Node* t = head;
43         while (t->next) {
44             t = t->next;
45         }
46         t->next = n;
47         n->prev = t;
48     }
49 }
50
51 void insertAfterNode(Node*& head, char prevData, char v) {
52 }
```

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 end
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& song) {
22         SongNode* newSong = new SongNode;
23         newSong->songTitle = song;
24     }
}
C:\Users\TIPQC\Documents\Untitled3.cpp + - 
--- 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