

# Hands-on Activity 3.1 Linked Lists

## LINKED LISTS

**Course Code:** CPE010

**Program:** Computer Engineering

**Course Title:** Data Structures and Algorithms

**Date Performed:** SEPTEMBER 27, 2024

**Section:** CPE21S1

**Date Submitted:** SEPTEMBER 30, 2024

**Name(s):** GASPAR, AARON ROWEN O.

**Instructor:** MA'AM MARIA RIZETTE SAYO

## 6. Output

```
main.cpp x +
main.cpp | main
1 #include <iostream>
2 using namespace std;
3 class Node {
4 public:
5     char data;
6     Node *next;
7 };
8
9 int main() {
10     // Step 1
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     // Step 2
19     head = new Node;
20     second = new Node;
21     third = new Node;
22     fourth = new Node;
23     fifth = new Node;
24     last = new Node;
25
26     // Step 3
27     head->data = 'C';
28     head->next = second;
29     second->data = 'P';
30     second->next = third;
31     third->data = 'E';
32     third->next = fourth;
33     fourth->data = 'O';
34     fourth->next = fifth;
35     fifth->data = 'I';
36     fifth->next = last;
37
38     // Step 4
39     last->data = '0';
40     last->next = nullptr;
41
42     // Print the linked list
43     Node *current = head;
44     while (current != nullptr) {
45         cout << current->data << " ";
46         current = current->next;
47     }
48     cout << endl;
49 }
```

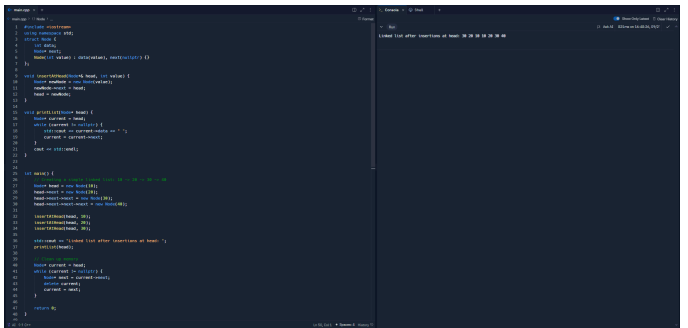
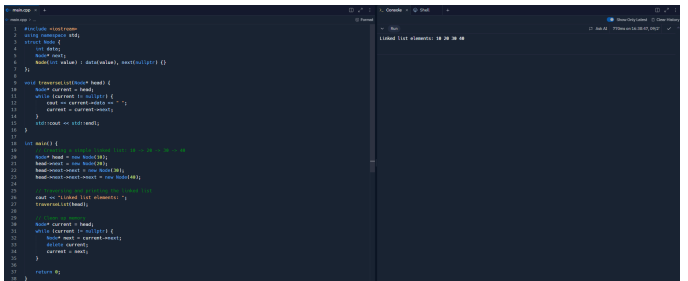
Discussion: The following code is a simple implementation of linked list. The output gives nothing, so I just added some code to print the output of the program.

Operation

Transversal

Insertion at head

Screenshot



## Insertion at any part of the list

```
1 // Insertion at any part of the list
2 #include <iostream>
3 using namespace std;
4
5 class Node {
6 public:
7     char data;
8     Node *next;
9 };
10
11 void traverseList(Node *head) {
12     Node *current = head;
13     while (current != NULL) {
14         cout << current->data << " ";
15         current = current->next;
16     }
17 }
18
19 int main() {
20     // Step 1
21     Node *head = nullptr;
22     Node *second = nullptr;
23     Node *third = nullptr;
24     Node *fourth = nullptr;
25     Node *fifth = nullptr;
26     Node *last = nullptr;
27
28     // Step 2
29     head = new Node;
30     second = new Node;
31     third = new Node;
32     fourth = new Node;
33     fifth = new Node;
34     last = new Node;
35
36     // Step 3
37     head->data = 'C';
38     head->next = second;
39     second->data = 'P';
40     second->next = third;
41     third->data = 'E';
42     third->next = fourth;
43     fourth->data = 'O';
44     fourth->next = fifth;
45     fifth->data = '1';
46     fifth->next = last;
47     last->data = '0';
48     last->next = nullptr;
49
50     traverseList(head);
51
52     return 0;
53 }
```

## Insertion at the end

```
1 // Insertion at the end
2 #include <iostream>
3 using namespace std;
4
5 class Node {
6 public:
7     char data;
8     Node *next;
9 };
10
11 void traverseList(Node *head) {
12     Node *current = head;
13     while (current != NULL) {
14         cout << current->data << " ";
15         current = current->next;
16     }
17 }
18
19 void insertAtEnd(Node *head, char value) {
20     Node *newNode = new Node;
21     newNode->data = value;
22     newNode->next = nullptr;
23
24     if (head == nullptr) {
25         head = newNode;
26     } else {
27         Node *temp = head;
28         while (temp->next != nullptr) {
29             temp = temp->next;
30         }
31         temp->next = newNode;
32     }
33 }
34
35 int main() {
36     // Step 1
37     Node *head = nullptr;
38     Node *second = nullptr;
39     Node *third = nullptr;
40     Node *fourth = nullptr;
41     Node *fifth = nullptr;
42     Node *last = nullptr;
43
44     // Step 2
45     head = new Node;
46     second = new Node;
47     third = new Node;
48     fourth = new Node;
49     fifth = new Node;
50     last = new Node;
51
52     // Step 3
53     head->data = 'C';
54     head->next = second;
55     second->data = 'P';
56     second->next = third;
57     third->data = 'E';
58     third->next = fourth;
59     fourth->data = 'O';
60     fourth->next = fifth;
61     fifth->data = '1';
62     fifth->next = last;
63     last->data = '0';
64     last->next = nullptr;
65
66     traverseList(head);
67
68     // Step 4
69     insertAtEnd(head, 'X');
70
71     traverseList(head);
72
73     return 0;
74 }
```

## Deletion of node

```
1 // Deletion of node
2 #include <iostream>
3 using namespace std;
4
5 class Node {
6 public:
7     char data;
8     Node *next;
9 };
10
11 void traverseList(Node *head) {
12     Node *current = head;
13     while (current != NULL) {
14         cout << current->data << " ";
15         current = current->next;
16     }
17 }
18
19 void deleteNode(Node *head, int index) {
20     if (index == 1) {
21         head = head->next;
22     } else {
23         Node *temp = head;
24         for (int i = 1; i < index; i++) {
25             temp = temp->next;
26         }
27         if (temp->next != nullptr) {
28             temp->next = temp->next->next;
29         }
30     }
31 }
32
33 int main() {
34     // Step 1
35     Node *head = nullptr;
36     Node *second = nullptr;
37     Node *third = nullptr;
38     Node *fourth = nullptr;
39     Node *fifth = nullptr;
40     Node *last = nullptr;
41
42     // Step 2
43     head = new Node;
44     second = new Node;
45     third = new Node;
46     fourth = new Node;
47     fifth = new Node;
48     last = new Node;
49
50     // Step 3
51     head->data = 'C';
52     head->next = second;
53     second->data = 'P';
54     second->next = third;
55     third->data = 'E';
56     third->next = fourth;
57     fourth->data = 'O';
58     fourth->next = fifth;
59     fifth->data = '1';
60     fifth->next = last;
61     last->data = '0';
62     last->next = nullptr;
63
64     traverseList(head);
65
66     // Step 4
67     deleteNode(head, 3);
68
69     traverseList(head);
70
71     return 0;
72 }
```

a

```
1 // main.cpp
2 #include <iostream>
3 using namespace std;
4
5 class Node {
6 public:
7     char data;
8     Node *next;
9 };
10
11 void traverseList(Node *head) {
12     Node *current = head;
13     while (current != NULL) {
14         cout << current->data << " ";
15         current = current->next;
16     }
17 }
18
19 int main() {
20     // Step 1
21     Node *head = nullptr;
22     Node *second = nullptr;
23     Node *third = nullptr;
24     Node *fourth = nullptr;
25     Node *fifth = nullptr;
26     Node *last = nullptr;
27
28     // Step 2
29     head = new Node;
30     second = new Node;
31     third = new Node;
32     fourth = new Node;
33     fifth = new Node;
34     last = new Node;
35
36     // Step 3
37     head->data = 'C';
38     head->next = second;
39     second->data = 'P';
40     second->next = third;
41     third->data = 'E';
42     third->next = fourth;
43     fourth->data = 'O';
44     fourth->next = fifth;
45     fifth->data = '1';
46     fifth->next = last;
47     last->data = '0';
48     last->next = nullptr;
49
50     traverseList(head);
51
52     return 0;
53 }
```

b

```
main.cpp x +
main.cpp > f main
1 #include<iostream>
2
3 class Node {
4 public:
5     char data;
6     Node *next;
7 };
8
9 // Function to insert a new node at the head of the list
10 Node* insertAtHead(Node* head, char new_data) {
11     // Step 1: Create a new node
12     Node* new_node = new Node();
13     new_node->data = new_data;
14
15     // Step 2: Make the next of the new node point to the current head
16     new_node->next = head;
17
18     // Step 3: Update the head to be the new node
19     head = new_node;
20
21     return head;
22 }
23
24 // Function to traverse and print the linked list
25 void traverseList(Node* head) {
26     Node* current = head;
27     while (current != nullptr) {
28         std::cout << current->data << " ";
29         current = current->next;
30     }
31     std::cout << std::endl;
32 }
33
34 int main() {
35     // Step 1
36     Node *head = nullptr;
37     Node *second = nullptr;
38
39     // Step 2
40     head = insertAtHead(head, 'G');
41     head = insertAtHead(head, 'C');
42     head = insertAtHead(head, 'P');
43     head = insertAtHead(head, 'E');
44     head = insertAtHead(head, '1');
45
46     traverseList(head);
47 }
```

Ln 65, Col 33 • Spaces: 4 History

Linked List after insertion at head: G C P E 1 0

C

```
main.cpp x +
main.cpp > ...
1 #include<iostream>
2
3 class Node {
4 public:
5     char data;
6     Node *next;
7 };
8
9 // Function to insert a new node after a given node
10 void insertAfter(Node* prev_node, char new_data) {
11     if (prev_node == nullptr) {
12         std::cout << "The given previous node cannot be NULL" << std::endl;
13         return;
14     }
15
16     // Step 1: Create a new node
17     Node* new_node = new Node();
18     new_node->data = new_data;
19
20     // Step 2: Make the next of the new node point to the next of prev_node
21     new_node->next = prev_node->next;
22
23     // Step 3: Make the next of prev_node point to the new node
24     prev_node->next = new_node;
25 }
26
27 // Function to insert a new node at the head of the list
28 Node* insertAtHead(Node* head, char new_data) {
29     // Step 1: Create a new node
30     Node* new_node = new Node();
31     new_node->data = new_data;
32
33     // Step 2: Make the next of the new node point to the current head
34     new_node->next = head;
35
36     // Step 3: Update the head to be the new node
37     head = new_node;
38
39     return head;
40 }
```

Ln 65, Col 33 • Spaces: 4 History

Linked List after insertion at head: G C P E 1 0

d

```

1 #include<iostream>
2
3 class Node {
4 public:
5     char data;
6     Node *next;
7 };
8
9 // Function to delete a node with a given key
10 Node* deleteNode(Node* head, char key) {
11     // Store head node
12     Node* temp = head;
13     Node* prev = nullptr;
14
15     // If head node itself holds the key to be deleted
16     if (temp != nullptr && temp->data == key) {
17         head = temp->next; // Changed head
18         delete temp; // Free old head
19         return head;
20     }
21
22     // Search for the key to be deleted, keep track of the previous node
23     while (temp != nullptr && temp->data != key) {
24         prev = temp;
25         temp = temp->next;
26     }
27
28     // If key was not present in linked list
29     if (temp == nullptr) return head;

```

Console Output:

```

Linked List before deletion: G C P E E 0 1 0
Linked List after deletion: G P E E 0 1 0

```

e

```

1 #include<iostream>
2
3 class Node {
4 public:
5     char data;
6     Node *next;
7 };
8
9 // Function to delete a node with a given key
10 Node* deleteNode(Node* head, char key) {
11     // Store head node
12     Node* temp = head;
13     Node* prev = nullptr;
14
15     // If head node itself holds the key to be deleted
16     if (temp != nullptr && temp->data == key) {
17         head = temp->next; // Changed head
18         delete temp; // Free old head
19         return head;
20     }
21
22     // Search for the key to be deleted, keep track of the previous node
23     while (temp != nullptr && temp->data != key) {
24         prev = temp;
25         temp = temp->next;
26     }
27
28     // If key was not present in linked list
29     if (temp == nullptr) return head;

```

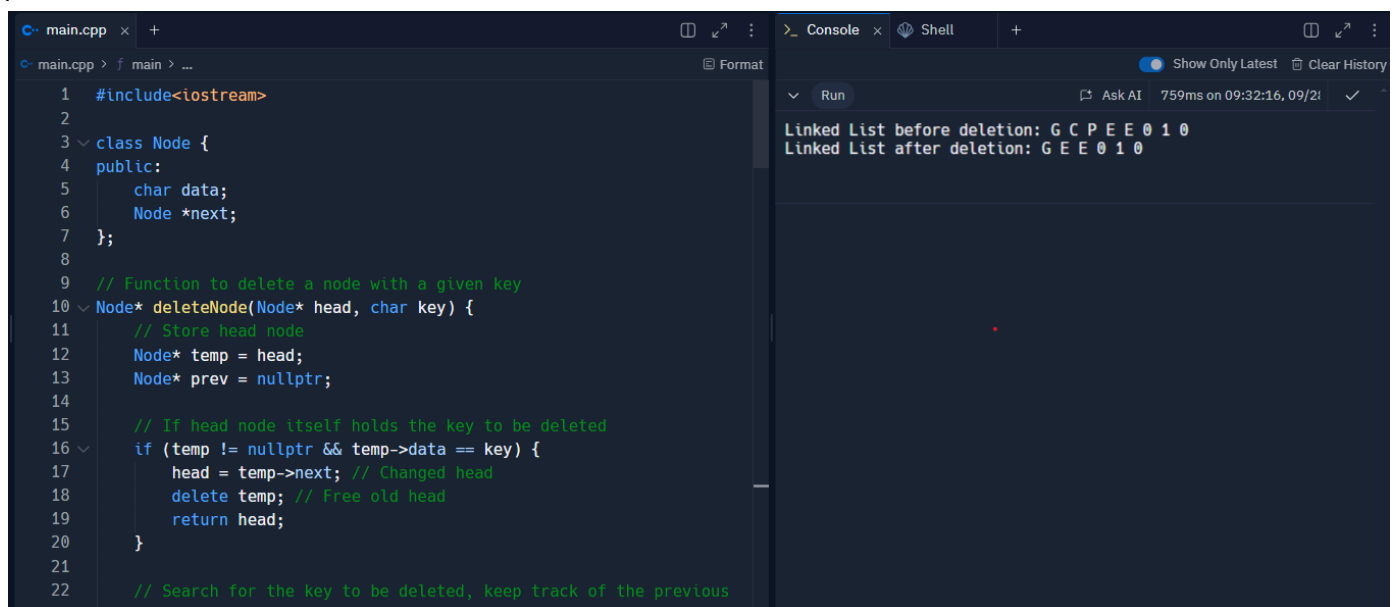
Console Output:

```

Linked List before deletion: G C P E E 0 1 0
Linked List after deletion: G E E 0 1 0

```

f



```
1 #include<iostream>
2
3 class Node {
4 public:
5     char data;
6     Node *next;
7 };
8
9 // Function to delete a node with a given key
10 Node* deleteNode(Node* head, char key) {
11     // Store head node
12     Node* temp = head;
13     Node* prev = nullptr;
14
15     // If head node itself holds the key to be deleted
16     if (temp != nullptr && temp->data == key) {
17         head = temp->next; // Changed head
18         delete temp; // Free old head
19         return head;
20     }
21
22     // Search for the key to be deleted, keep track of the previous
```

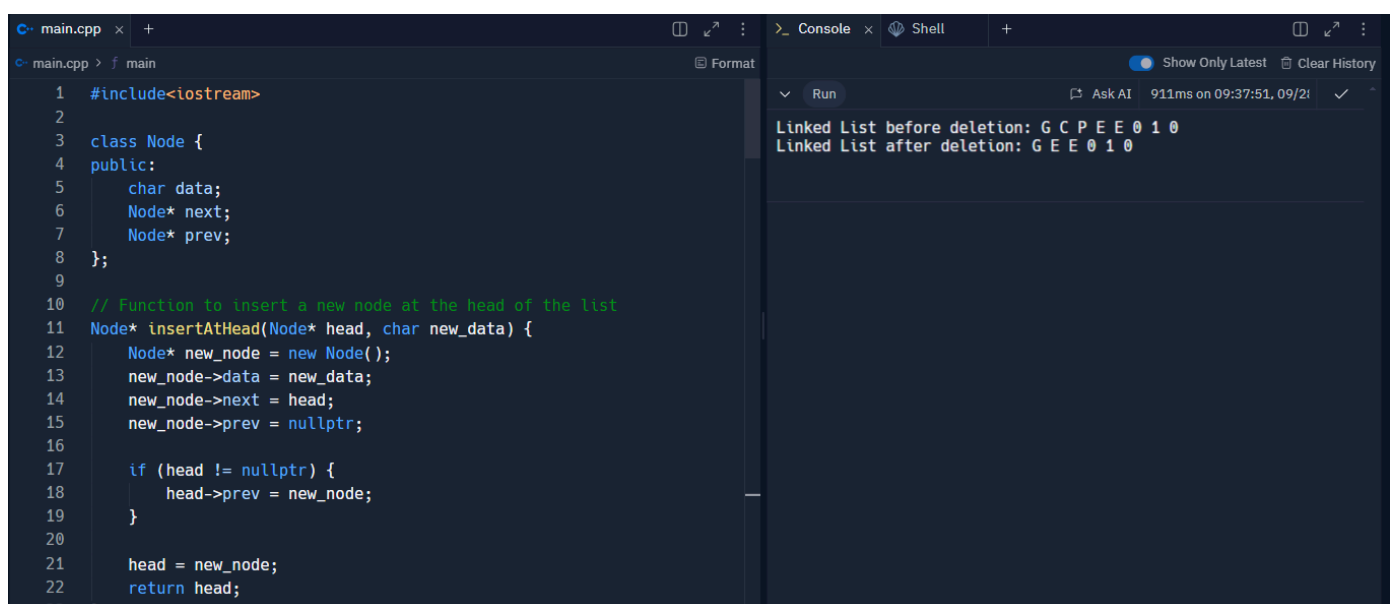
Console

Run

Show Only Latest Clear History

Ask AI 759ms on 09:32:16, 09/21

Linked List before deletion: G C P E E 0 1 0  
Linked List after deletion: G E E 0 1 0



```
1 #include<iostream>
2
3 class Node {
4 public:
5     char data;
6     Node* next;
7     Node* prev;
8 };
9
10 // Function to insert a new node at the head of the list
11 Node* insertAtHead(Node* head, char new_data) {
12     Node* new_node = new Node();
13     new_node->data = new_data;
14     new_node->next = head;
15     new_node->prev = nullptr;
16
17     if (head != nullptr) {
18         head->prev = new_node;
19     }
20
21     head = new_node;
22     return head;
23 }
```

Console

Run

Show Only Latest Clear History

Ask AI 911ms on 09:37:51, 09/21

Linked List before deletion: G C P E E 0 1 0  
Linked List after deletion: G E E 0 1 0

### Analysis:

The Node class is defined with three members: data, next, and prev. This allows each node to point to both the next and previous nodes in the list, enabling bidirectional traversal. The insertAtHead function inserts a new node at the beginning of the list. The insertAfter function inserts a new node after a specified node. The deleteNode function deletes a node with a specified key. The traverseList function prints the data of each node in the list. Overall, the code efficiently handles the basic operations of a doubly linked list.

## 7. Supplementary Activity

## Program:

```
114 int main() {
115     Playlist myPlaylist;
116
117     myPlaylist.addSong("Song 1");
118     myPlaylist.addSong("Song 2");
119     myPlaylist.addSong("Song 3");
120
121     std::cout << "Playing all songs in the playlist:" << endl;
122     myPlaylist.playAllSongs();
123
124     std::cout << "\nPlaying next song:" << endl;
125     myPlaylist.nextSong();
126
127     std::cout << "\nPlaying previous song:" << endl;
128     myPlaylist.previousSong();
129
130     std::cout << "\nRemoving 'Song 2' from the playlist." << endl;
131     myPlaylist.removeSong("Song 2");
132
133     std::cout << "\nPlaying all songs in the playlist:" << endl;
134     myPlaylist.playAllSongs();
135
136     return 0;
137 }
```

```
75 // Function to play the next song
76 void nextSong() {
77     if (current) {
78         current = current->next;
79         std::cout << "Playing: " << current->song << endl;
80     }
81 }
82
83 // Function to play the previous song
84 void previousSong() {
85     if (current) {
86         current = current->prev;
87         std::cout << "Playing: " << current->song << endl;
88     }
89 }
90
91 // Function to play all songs in the playlist
92 void playAllSongs() const {
93     if (!head) return;
94
95     Node* temp = head;
96     do {
97         std::cout << temp->song << std::endl;
98         temp = temp->next;
99     } while (temp != head);
100 }
101
102 ~Playlist() {
103     if (!head) return;
104
105     Node* temp = head;
106     do {
107         Node* next = temp->next;
108         delete temp;
109         temp = next;
110     } while (temp != head);
111 }
112 };
```

```

37 // Function to remove a song from the playlist
38 void removeSong(const string& songName) {
39     if (!head) return;
40
41     Node* temp = head;
42
43     // If the song to be deleted is the head
44     if (head->song == songName) {
45         if (head == head->next) {
46             delete head;
47             head = nullptr;
48             current = nullptr;
49         } else {
50             Node* tail = head->prev;
51             head = head->next;
52             tail->next = head;
53             head->prev = tail;
54             delete temp;
55             current = head;
56         }
57         return;
58     }
59
60     // Search for the song to be deleted
61     do {
62         if (temp->song == songName) {
63             temp->prev->next = temp->next;
64             temp->next->prev = temp->prev;
65             if (current == temp) {
66                 current = temp->next;
67             }
68             delete temp;
69             return;
70         }
71         temp = temp->next;
72     } while (temp != head);
73 }

```



```

main.cpp x +
main.cpp > f main
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Node {
6      string song;
7      Node* next;
8      Node* prev; // Adding a previous pointer for doubly linked list
9  };
10
11 class Playlist {
12 private:
13     Node* head;
14     Node* current; // Pointer to keep track of the current song
15
16 public:
17     Playlist() : head(nullptr), current(nullptr) {}
18
19     // Function to add a song to the playlist
20     void addSong(const string& songName) {
21         Node* newNode = new Node();
22         newNode->song = songName;
23         if (!head) {
24             head = newNode;
25             head->next = head;
26             head->prev = head;
27             current = head;
28         } else {
29             Node* tail = head->prev;
30             tail->next = newNode;
31             newNode->prev = tail;
32             newNode->next = head;
33             head->prev = newNode;
34         }
35     }
36

```

Output:

```

>_ Console x Shell +
Show Only Latest Clear History
Run Ask AI 1s on 16:27:15, 09/27 ✓
Playing all songs in the playlist:
Song 1
Song 2
Song 3

Playing next song:
Playing: Song 2

Playing previous song:
Playing: Song 1

Removing 'Song 2' from the playlist.

Playing all songs in the playlist:
Song 1
Song 3

```

## 8. Conclusion

In conclusion, learning linked lists in C++ is indeed a crucial milestone for any aspiring programmer. It offers a hands-on approach to understanding dynamic data structures, which are essential for managing data efficiently. By working with linked lists, you gain valuable insights into memory management and pointer manipulation—skills that are fundamental in C++ programming. Mastering linked lists not only sets the stage for tackling more advanced topics in computer science, such as trees, graphs, and complex algorithms, but it also equips you with the practical know-how to solve real-world problems. Whether you're optimizing a music playlist, managing a dynamic set of data, or implementing a game loop, the principles you learn from linked lists are directly applicable.