

Activity No. <3>	
<Hands-on Activity 3.1 Linked Lists>	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: Sep 27, 2024
Section: CPE21S4	Date Submitted: Sep 30, 2024
Name(s): Danica T. Guariño	Instructor: Ma'am Sayo
6. Output	

Screenshot

C++ shell

```
1  #include <iostream>
2  using namespace std;
3
4  // Define the node structure
5  struct Node {
6      char data;
7      Node* next;
8  };
9
10 // Function to create a new node
11 Node* createNode(char data) {
12     Node* newNode = new Node();
13     newNode->data = data;
14     newNode->next = nullptr;
15     return newNode;
16 }
17
18 // Function to print the linked list
19 void printList(Node* head) {
20     Node* temp = head;
21     while (temp != nullptr) {
22         cout << temp->data << " -> ";
23         temp = temp->next;
24     }
25     cout << "NULL" << endl;
26 }
27
28 int main() {
29
30     Node* head = createNode('C');
```

```

20     Node* temp = head;
21     while (temp != nullptr) {
22         cout << temp->data << " -> ";
23         temp = temp->next;
24     }
25     cout << "NULL" << endl;
26 }
27
28 int main() {
29
30     Node* head = createNode('C');
31     head->next = createNode('P');
32     head->next->next = createNode('E');
33     head->next->next->next = createNode('0');
34     head->next->next->next->next = createNode('1');
35     head->next->next->next->next->next = createNode('0');
36
37
38     printList(head);
39
40
41     Node* temp;
42     while (head != nullptr) {
43         temp = head;
44         head = head->next;
45         delete temp;
46     }
47
48     return 0;
49 }

```

Link to this code: [\[copy\]](#)

**options** **compilation** **execution**

C -> P -> E -> 0 -> 1 -> 0 -> NULL

## Discussion

The linked list implementation described generates nodes for each character in the string "CPE010" and connects them, concluding with "NULL" to indicate the list's end. Each character is stored in its own node, which is generated dynamically in the heap. The nodes are linked by setting each node's next pointer to the next node, and the last node's next pointer to NULL. The 'printList' function iterates through the list from the head to the last node, printing each node's data followed by "-> " and at last "NULL".

To further improve the implementation, you can:

1. Implement error handling for memory allocation.
2. Separate linked list operations (insertion, deletion, and traversal) into separate functions.
3. Use a class to encapsulate linked list operations, making your code more organized and object-oriented.

**Table 3-1. Output of Initial/Simple Implementation**

Operation	Screenshot
Traversal	<pre> // Traversal void traverse() {     Node* current = head;     while (current != nullptr) {         std::cout &lt;&lt; current-&gt;data &lt;&lt; " -&gt; ";         current = current-&gt;next;     }     std::cout &lt;&lt; "NULL" &lt;&lt; std::endl; } </pre>
Insertion at head	<pre> // Insertion at head void insert_at_head(int data) {     Node* new_node = new Node(data);     new_node-&gt;next = head;     head = new_node; } </pre>
Insertion at any part of the list	<pre> // Insertion at any part of the list void insert_at_position(int data, int position) {     if (position == 0) {         insert_at_head(data);         return;     }     Node* new_node = new Node(data);     Node* current = head;     for (int i = 0; i &lt; position - 1; ++i) {         if (current == nullptr) {             throw std::out_of_range("Position out of bounds");         }         current = current-&gt;next;     }     new_node-&gt;next = current-&gt;next;     current-&gt;next = new_node; } </pre>
Insertion at the end	<pre> // Insertion at the end void insert_at_end(int data) {     Node* new_node = new Node(data);     if (head == nullptr) {         head = new_node;         return;     }     Node* current = head;     while (current-&gt;next != nullptr) {         current = current-&gt;next;     }     current-&gt;next = new_node; } </pre>

### Deletion of a node

```
// Deletion of a node
void delete_node(int key) {
    Node* current = head;
    Node* prev = nullptr;
    if (current != nullptr && current->data == key) {
        head = current->next;
        delete current;
        return;
    }
    while (current != nullptr && current->data != key) {
        prev = current;
        current = current->next;
    }
    if (current == nullptr) return;
    prev->next = current->next;
    delete current;
}
};
```

Table 3-2. Code for the List Operations

a.	Source Code	<pre> 1  #include&lt;iostream&gt; 2  #include&lt;utility&gt; 3  using namespace std; 4  class Node { 5  public: 6      char data; 7      Node *next; 8  }; 9 10 int main() { 11     //step 1 12     Node *head = NULL; 13     Node *second = NULL; 14     Node *third = NULL; 15     Node *fourth = NULL; 16     Node *fifth = NULL; 17     Node *last = NULL; 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     //step 3 26     head-&gt;data = 'C'; 27     head-&gt;next = second; 28     second-&gt;data = 'P'; 29     second-&gt;next = third; 30     third-&gt;data = 'E'; 31     third-&gt;next = fourth; 32     fourth-&gt;data = '0'; 33     fourth-&gt;next = fifth; 34     fifth-&gt;data = '1'; 35     fifth-&gt;next = last; 36     //step 4 37     last-&gt;data = '0'; 38     last-&gt;next = nullptr; 39 40     Node *current = head; 41     while (current != nullptr){ 42         cout &lt;&lt; current-&gt;data &lt;&lt; "-&gt;"; 43         current = current-&gt;next; 44     } 45     cout &lt;&lt; "end of linked list"; 46 }</pre>
	Console	<div>options   compilation   execution</div> <pre> The elements in the list are: C -&gt; P -&gt; E -&gt; 1 -&gt; 0 -&gt; 1</pre>

b.

## Source Code

```

1  #include <iostream>
2  using namespace std;
3
4  class Node {
5  public:
6      char data;
7      Node *next;
8  };
9
10 void traverseList(Node *head) {
11     Node *temp = head;
12     printf("\n\nThe elements in the list are: \n");
13     while (temp != NULL) {
14         printf("%c", temp->data);
15         temp = temp->next;
16         if (temp != NULL) {
17             printf(" -> ");
18         }
19     }
20 }
21 int main() {
22     // Step 1
23     Node *head = NULL;
24     Node *second = NULL;
25     Node *third = NULL;
26     Node *fourth = NULL;
27     Node *fifth = NULL;
28     Node *last = NULL;
29
30     // Step 2
31     head = new Node;
32     second = new Node;
33     third = new Node;
34     fourth = new Node;
35     fifth = new Node;
36     last = new Node;
37
38     // Step 3
39     head->data = 'C';
40     head->next = second;
41
42     second->data = 'P';
43     second->next = third;
44
45     third->data = 'E';
46     third->next = fourth;
47
48     fourth->data = '1';
49     fourth->next = fifth;
50
51     fifth->data = '0';
52     fifth->next = last;
53
54     last->data = '1';
55     last->next = NULL;
56
57     // Insertion at the head
58     Node *newNode = new Node;
59     newNode->data = 'G';
60     newNode->next = head;
61     head = newNode;
62
63     traverseList(head);
64     return 0;
65 }

```

	<div>Console</div>	<div> <div>options</div> <div>compilation</div> <div>execution</div> </div> <div>The elements in the list are: G -&gt; C -&gt; P -&gt; E -&gt; 1 -&gt; 0 -&gt; 1</div>
c.	Source Code	<pre> 1  #include &lt;iostream&gt; 2  using namespace std; 3 4  class Node { 5  public: 6      char data; 7      Node *next; 8  }; 9 10 void traverseList(Node *head) { 11     Node *temp = head; 12     printf("\n\nThe elements in the list are: \n"); 13     while (temp != NULL) { 14         printf("%c", temp-&gt;data); 15         temp = temp-&gt;next; 16         if (temp != NULL) { 17             printf(" -&gt; "); 18         } 19     } 20 } 21 int main() { 22     // Step 1 23     Node *head = NULL; 24     Node *second = NULL; 25     Node *third = NULL; 26     Node *fourth = NULL; 27     Node *fifth = NULL; 28     Node *last = NULL; 29 30     // Step 2 31     head = new Node; 32     second = new Node; 33     third = new Node; 34     fourth = new Node; 35     fifth = new Node; 36     last = new Node; 37 38     // Step 3 39     head-&gt;data = 'C'; 40     head-&gt;next = second; 41 42     second-&gt;data = 'P'; 43     second-&gt;next = third; 44 </pre>

```
44
45     third->data = 'E';
46     third->next = fourth;
47
48     fourth->data = '1';
49     fourth->next = fifth;
50
51     fifth->data = '0';
52     fifth->next = last;
53
54     last->data = '1';
55     last->next = NULL;
56
57     // Insertion at the head
58     Node *newNode = new Node;
59     newNode->data = 'G';
60     newNode->next = head;
61     head = newNode;
62
63     traverseList(head);
64     return 0;
65 }
```

## Console

options compilation execution

The elements in the list are:  
G -> C -> P -> E -> 1 -> 0 -> 1



d.

## Source Code

```

1  #include <iostream>
2  using namespace std;
3
4  class Node {
5  public:
6      char data;
7      Node *next;
8  };
9
10 void traverseList(Node *head) {
11     Node *temp = head;
12     printf("\n\nThe elements in the list are: \n");
13     while (temp != NULL) {
14         printf("%c", temp->data);
15         temp = temp->next;
16         if (temp != NULL) {
17             printf(" -> ");
18         }
19     }
20 }
21 int main() {
22     // Step 1
23     Node *head = NULL;
24     Node *second = NULL;
25     Node *third = NULL;
26     Node *fourth = NULL;
27     Node *fifth = NULL;
28     Node *last = NULL;
29
30     // Step 2
31     head = new Node;
32     second = new Node;
33     third = new Node;
34     fourth = new Node;
35     fifth = new Node;
36     last = new Node;
37
38     // Step 3
39     head->data = 'C';
40     head->next = second;
41
42     second->data = 'P';
43     second->next = third;
44
45     third->data = 'E';
46     third->next = fourth;
47
48     fourth->data = '1';
49     fourth->next = fifth;
50
51     fifth->data = '0';
52     fifth->next = last;
53
54     last->data = '1';
55     last->next = NULL;
56
57     // Insertion at the head
58     Node *newNode = new Node;
59     newNode->data = 'G';
60     newNode->next = head;
61     head = newNode;
62
63     traverseList(head);
64     return 0;
65 }

```

	Console	<div> <div>options</div> <div>compilation</div> <div>execution</div> </div> <pre> The elements in the list are: G -&gt; C -&gt; P -&gt; E -&gt; 1 -&gt; 0 -&gt; 1 </pre>
e.	Source Code	<pre> #include &lt;iostream&gt; using namespace std;  class Node { public:     char data;     Node *next; };  void traverseList(Node *head) {     Node *temp = head;     printf("\n\nThe elements in the list are: \n");     while (temp != NULL) {         printf("%c", temp-&gt;data);         temp = temp-&gt;next;         if (temp != NULL) {             printf(" -&gt; ");         }     } }  int main() {     // Step 1     Node *head = NULL;     Node *second = NULL;     Node *third = NULL;     Node *fourth = NULL;     Node *fifth = NULL;     Node *last = NULL;      // Step 2     head = new Node;     second = new Node;     third = new Node;     fourth = new Node;     fifth = new Node; </pre>

```
Node *fourth = NULL;
Node *fifth = NULL;
Node *last = NULL;

// Step 2
head = new Node;
second = new Node;
third = new Node;
fourth = new Node;
fifth = new Node;
last = new Node;

// Step 3
head->data = 'C';
head->next = second;

second->data = 'P';
second->next = third;

third->data = 'E';
third->next = fourth;

fourth->data = '1';
fourth->next = fifth;

fifth->data = '0';
fifth->next = last;

last->data = '1';
last->next = NULL;

// Insert at head
Node *newNode = new Node;
newNode->data = 'G';
newNode->next = head;
head = newNode;
```

```
// Insert E
Node *newNode2 = new Node;
newNode2->data = 'E';


Node *temp = head;
while (temp != NULL) {
    if (temp->data == 'P') {

        newNode2->next = temp->next;
        temp->next = newNode2;
        break;
    }
    temp = temp->next;
}

traverseList(head);

return 0;
```

Console

LINK TO THIS CODE:  [\[copy\]](#)

options compilation **execution**

The elements in the list are:  
G -> E -> E -> 1 -> 0 -> 1

f.	Source Code	<pre>#include &lt;iostream&gt;  using namespace std;  class Node { public:     char data;     Node* next; };  void traverseList(Node* head) {     Node* temp = head;     cout &lt;&lt; "\n\nThe elements in the list are: \n";     while (temp != NULL) {         cout &lt;&lt; temp-&gt;data;         temp = temp-&gt;next;         if (temp != NULL) {             cout &lt;&lt; " -&gt; ";         }     } }  void insertAtStart(Node*&amp; head, char data) {     Node* newNode = new Node;     newNode-&gt;data = data;     newNode-&gt;next = head;     head = newNode; }  void insertAfter(Node* prevNode, char data) {     if (prevNode == NULL) {         cout &lt;&lt; "Previous node cannot be NULL." &lt;&lt; endl;         return;     }      Node* newNode = new Node;     newNode-&gt;data = data;     newNode-&gt;next = prevNode-&gt;next;     prevNode-&gt;next = newNode; }</pre>
----	-------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

void deleteNode(Node*& head, char data) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }

    if (head->data == data) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* temp = head;

```

```

    while (temp->next != NULL && temp->next->data != data) {
        temp = temp->next;
    }

    if (temp->next == NULL) {
        cout << "Node with data '" << data << "' not found." <<
endl;
        return;
    }

    Node* nodeToDelete = temp->next;
    temp->next = nodeToDelete->next;
    delete nodeToDelete;
}

int main() {
    Node* head = NULL;
    Node* second = NULL;
    Node* third = NULL;
    Node* fourth = NULL;
    Node* fifth = NULL;
    Node* last = NULL;

    head = new Node;
    second = new Node;
    third = new Node;
    fourth = new Node;
    fifth = new Node;
    last = new Node;

    head->data = 'C';
    head->next = second;

    second->data = 'P';
    second->next = third;

```

```
head = new Node;
second = new Node;
third = new Node;
fourth = new Node;
fifth = new Node;
last = new Node;

head->data = 'C';
head->next = second;

second->data = 'P';
second->next = third;

third->data = 'E';
third->next = fourth;

fourth->data = '1';
fourth->next = fifth;

fifth->data = '0';
fifth->next = last;

last->data = '1';
last->next = NULL;

// Insert 'G' at the start
insertAtStart(head, 'G');

// Insert 'E' after 'P'
Node* temp = head;
while (temp != NULL && temp->data != 'P') {
    temp = temp->next;
```

		<pre>     }     if (temp != NULL) {         insertAfter(temp, 'E');     }      // Delete 'C'     deleteNode(head, 'C');      // Delete 'P'     deleteNode(head, 'P');      // Traverse the list     traverseList(head);      return 0; } </pre>
	Console	 <p>The elements in the list are: G -&gt; E -&gt; E -&gt; 1 -&gt; 0 -&gt; 1</p>

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

Screenshot(s)	Analysis
	<ul style="list-style-type: none"> <li>• In the node there is a prev pointer added</li> <li>• The &lt;-&gt; displays the lists double linked list</li> </ul>



```
#include <iostream>
using namespace std;

class Node {
public:
    char data;
    Node *next;
    Node *prev;
};

void traverseList(Node *head) {
    Node *temp = head;
    printf("\n\nThe elements in the list
are: \n");
    while (temp != NULL) {
        printf("%c", temp->data);
        temp = temp->next;
    }
}
```

```
        if (temp != NULL) {  
            printf(" <-> ");  
        }  
    }  
}
```

```
int main() {  
    Node *head = NULL;  
    Node *second = NULL;  
    Node *third = NULL;  
    Node *fourth = NULL;  
    Node *fifth = NULL;  
    Node *last = NULL;  
  
    head = new Node;  
    second = new Node;  
    third = new Node;  
    fourth = new Node;  
    fifth = new Node;  
    last = new Node;  
  
    head->data = 'C';  
    head->next = second;  
    head->prev = NULL;  
  
    second->data = 'P';  
    second->next = third;  
    second->prev = head;  
  
    third->data = 'E';  
    third->next = fourth;  
    third->prev = second;  
  
    fourth->data = '0';  
    fourth->next = fifth;  
    fourth->prev = third;  
  
    fifth->data = '1';  
    fifth->next = last;  
    fifth->prev = fourth;
```

```
third->data = 'E';  
third->next = fourth;  
third->prev = second;
```

```
fourth->data = '0';  
fourth->next = fifth;  
fourth->prev = third;
```

```
fifth->data = '1';  
fifth->next = last;  
fifth->prev = fourth;
```

```
last->data = '0';  
last->next = NULL;  
last->prev = fifth;
```

```
traverseList(head);
```

```
return 0;
```

```
}
```

```
#include <iostream>
using namespace std;

class Node {
public:
    char data;
    Node *next;
    Node *prev;
};

void traverseList(Node *head) {
    Node *temp = head;
    printf("\n\nThe elements in the list
are: - \n");
    while (temp != NULL) {
        printf("%c", temp->data);
        temp = temp->next;
        if (temp != NULL) {
            printf(" <-> ");
        }
    }
}

int main() {
    Node *head = NULL;
    Node *second = NULL;
    Node *third = NULL;
    Node *fourth = NULL;
    Node *fifth = NULL;
    Node *last = NULL;

    head = new Node;
    second = new Node;
    third = new Node;
    fourth = new Node;
    fifth = new Node;
    last = new Node;

    head->data = 'C';
    head->next = second;
    head->prev = NULL;
```

```
second->data = 'P';  
second->next = third;  
second->prev = head;
```

```
third->data = 'E';  
third->next = fourth;  
third->prev = second;
```

```
fourth->data = '0';  
fourth->next = fifth;  
fourth->prev = third;
```

```
fifth->data = '1';
```

```
fifth->next = last;  
fifth->prev = fourth;
```

```
last->data = '0';  
last->next = NULL;  
last->prev = fifth;
```

```
Node *newNode = new Node;  
newNode->data = 'B';  
newNode->next = head;  
newNode->prev = NULL;  
head->prev = newNode;  
head = newNode;
```

```
traverseList(head);
```

```
return 0;
```

```
}
```

```

#include <iostream>
using namespace std;

class Node {
public:
    char data;
    Node *next;
    Node *prev;
};

void traverseList(Node *head) {
    Node *temp = head;
    printf("\n\nThe elements in the list
are: - \n");
    while (temp != NULL) {
        printf("%c", temp->data);
        temp = temp->next;
        if (temp != NULL) {
            printf(" <-> ");
        }
    }
}

int main() {
    Node *head = NULL;
    Node *second = NULL;
    Node *third = NULL;
    Node *fourth = NULL;
    Node *fifth = NULL;

```

- In the node there is a prev pointer added
- The <-> displays the lists double linked list

```
Node *last = NULL;

head = new Node;
second = new Node;
third = new Node;
fourth = new Node;
fifth = new Node;
last = new Node;

head->data = 'C';
head->next = second;
head->prev = NULL;

second->data = 'P';
second->next = third;
second->prev = head;

third->data = 'E';
third->next = fourth;
third->prev = second;

fourth->data = '0';
fourth->next = fifth;
fourth->prev = third;

fifth->data = '1';
fifth->next = last;
fifth->prev = fourth;

last->data = '0';
last->next = NULL;
last->prev = fifth;

Node *newNode = new Node;
newNode->data = 'B';
newNode->next = head;
newNode->prev = NULL;
head->prev = newNode;
head = newNode;
```

```

int position = 4;
newNode = new Node;
newNode->data = 'E';

if (position == 1) {
    newNode->next = head;
    newNode->prev = NULL;
    head->prev = newNode;
    head = newNode;
} else {
    Node *temp = head;
    for (int i = 2; i < position;
i++) {
        if (temp->next != NULL) {
            temp = temp->next;

```

```

        } else {
            cout << "Previous node
cannot be null." << endl;
            delete newNode;
            return 1;
        }
    }
    newNode->next = temp->next;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

traverseList(head);

return 0;
}

```



```

#include <iostream>
using namespace std;

class Node {
public:
    char data;
    Node *next;
    Node *prev;
};

int main() {
    Node *head = NULL;
    Node *second = NULL;
    Node *third = NULL;
    Node *fourth = NULL;
    Node *fifth = NULL;
    Node *last = NULL;

    head = new Node;
    second = new Node;
    third = new Node;
    fourth = new Node;
    fifth = new Node;
    last = new Node;

    head->data = 'C';
    head->next = second;
    head->prev = NULL;

```

- In the node there is a prev pointer added
- The <-> displays the lists double linked list

```
second->data = 'P';
second->next = third;
second->prev = head;

third->data = 'E';
third->next = fourth;
third->prev = second;

fourth->data = '0';
fourth->next = fifth;
fourth->prev = third;

fifth->data = '1';
fifth->next = last;
fifth->prev = fourth;

last->data = '0';
last->next = NULL;
last->prev = fifth;

Node *newNode = new Node;
newNode->data = '1';
newNode->next = NULL;
newNode->prev = NULL;

if (head == NULL) {
    head = newNode;
} else {
    Node *temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
```

```
        nextNode->prev = temp;
    }

    Node *temp = head;
    printf("\n\nThe elements in the list
are: - \n");
    while (temp != NULL) {
        printf("%c", temp->data);
        temp = temp->next;
        if (temp != NULL) {
            printf(" <-> ");
        }
    }

    temp = head;
    while (temp != NULL) {
        Node *nextNode = temp->next;
        delete temp;
        temp = nextNode;
    }

    return 0;
```

```

#include <iostream>
using namespace std;

class Node {
public:
    char data;
    Node *next;
    Node *prev;
};

int main() {
    Node *head = NULL;
    Node *second = NULL;
    Node *third = NULL;
    Node *fourth = NULL;
    Node *fifth = NULL;
    Node *last = NULL;

    head = new Node;
    second = new Node;
    third = new Node;
    fourth = new Node;
    fifth = new Node;
    last = new Node;

    head->data = 'C';
    head->next = second;
    head->prev = NULL;

    second->data = 'P';
    second->next = third;
    second->prev = head;

    third->data = 'E';
    third->next = fourth;
    third->prev = second;

    fourth->data = '0';
    fourth->next = fifth;
    fourth->prev = third;

```

- In the node there is a prev pointer added
- The <-> displays the lists double linked list

```
third->data = 'E';  
third->next = fourth;  
third->prev = second;
```

```
fourth->data = '0';  
fourth->next = fifth;  
fourth->prev = third;
```

```
fifth->data = '1';  
fifth->next = last;  
fifth->prev = fourth;
```

```

last->data = '0';
last->next = NULL;
last->prev = fifth;

int position = 3;
Node *temp = head;

if (position == 1) {
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    delete temp;
} else {
    for (int i = 2; i < position;
i++) {
        temp = temp->next;
    }
    Node *nodeToDelete = temp->next;
    if (nodeToDelete != NULL) {
        temp->next =
nodeToDelete->next;
        if (nodeToDelete->next !=
NULL) {
            nodeToDelete->next->prev
= temp;
        }
        delete nodeToDelete;
    }
}

Node *printTemp = head;
printf("\n\nThe elements in the list
are: - \n");
while (printTemp != NULL) {
    printf("%c", printTemp->data);
    printTemp = printTemp->next;
    if (printTemp != NULL) {
        printf(" <-> ");
    }
}

return 0;
}

```

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

## 7. Supplementary Activity

ILO B: Solve given problems utilizing linked lists in C++  
 Problem Title: Implementing a Song Playlist using Linked List  
 Source: Packt Publishing

#### Problem Description:

In this activity, we'll look at some applications for which a singly linked list is not enough or not convenient. We will build a tweaked version that fits the application. We often encounter cases where we have to customize default implementations, such as when looping songs in a music player or in games where multiple players take a turn one by one in a circle.

These applications have one common property – we traverse the elements of the sequence in a circular fashion. Thus, the node after the last node will be the first node while traversing the list. This is called a circular linked list.

We'll take the use case of a music player. It should have following functions supported:

- Create a playlist using multiple songs.
- Add songs to the playlist.
- Remove a song from the playlist.
- Play songs in a loop (for this activity, we will print all the songs once).

Here are the steps to solve the problem:

- Design the basic structure that supports circular data representation.
- After that, implement the insert and delete functions in the structure.
- Implement a function for traversing the playlist.

The driver function should allow for common operations on a playlist such as: next, previous, play all songs, insert and remove

# C++ shell

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6
7  struct Node {
8      string songName;
9      Node* next;
10 };
11
12
13 Node* createNode(string songName) {
14     Node* newNode = new Node;
15     newNode->songName = songName;
16     newNode->next = nullptr;
17     return newNode;
18 }
19
20 // Function to insert a song at the end of the playlist
21 void insertSong(Node** headRef, string songName) {
22     Node* newNode = createNode(songName);
23
24     if (*headRef == nullptr) {
25         *headRef = newNode;
26         newNode->next = nullptr;
27         return;
28     }
29
30     Node* lastNode = *headRef;
31     while (lastNode->next != nullptr) {
32         lastNode = lastNode->next;
33     }
34
35     lastNode->next = newNode;
36     newNode->next = nullptr;
37 }
38
39 // Function to remove a song from the playlist
40 void removeSong(Node** headRef, string songName) {
41     if (*headRef == nullptr) {
42         return; // Empty list
43     }
44
45     if ((*headRef)->songName == songName && (*headRef)->next == nullptr) {
46         delete *headRef;
47         *headRef = nullptr;
48         return;
49     }
50
51     Node* current = *headRef;
52     Node* previous = nullptr;
53
54     while (current != nullptr) {
55         if (current->songName == songName) {
56             if (previous == nullptr) {
57                 *headRef = current->next;
58             } else {
59                 previous->next = current->next;
60             }
61             delete current;
62             current = current->next;
63         } else {
64             previous = current;
65             current = current->next;
66         }
67     }
68 }
```



```

58
59 ▾ do {
60     previous = current;
61     current = current->next;
62
63 ▾     if (current->songName == songName) {
64
65         previous->next = current->next;
66         delete current;
67
68
69 ▾         if (current == *headRef) {
70             *headRef = previous;
71         }
72
73         return;
74     }
75
76 } while (current != *headRef);
77
78 cout << "Song not found in playlist" << endl;
79 }
80
81 // Function to play all songs in the playlist
82 ▾ void playSongs(Node* head) {
83 ▾     if (head == nullptr) {
84         cout << "Playlist is empty" << endl;
85         return;
86     }
87

```

```

86     }
87
88     Node* current = head;
89     do {
90         cout << "\nPlaying: " << current->songName << endl;
91         current = current->next;
92     } while (current != head);
93 }
94
95 // Function to get the next song in the playlist
96 Node* nextSong(Node* current) {
97     return current->next;
98 }
99
100 // Function to get the previous song in the playlist
101 Node* previousSong(Node* current, Node* head) {
102     if (current == head) {
103         return head; // No previous song
104     }
105     Node* previous = head;
106     while (previous->next != current) {
107         previous = previous->next;
108     }
109     return previous;
110 }
111
112 int main() {
113     Node* head = nullptr;
114
115     // Add songs to the playlist

```

```

112 int main() {
113     Node* head = nullptr;
114
115     // Add songs to the playlist
116     insertSong(&head, "Goodluck Babe! by Chappel Roan");
117     insertSong(&head, "Please Please Please by Sabrina Carpenter");
118     insertSong(&head, "love. by Wave to Earth");
119
120     // Playing all songs
121     cout << "Playing all songs:" << endl;
122     playSongs(head);
123
124     // Removing of a song
125     removeSong(&head, "Please Please Please by Sabrina Carpenter");
126     cout << "\nSong removed: Please Please Please by Sabrina Carpenter " << endl;
127     cout << "\nPlaying remaining songs: " << endl;
128     playSongs(head);
129
130     // Get next and previous songs
131     Node* currentSong = head; // Start from the beginning
132     cout << "\nCurrent song: " << currentSong->songName << endl;
133
134     currentSong = nextSong(currentSong); // Get the next song
135     cout << "\nNext song: " << currentSong->songName << endl;
136
137     currentSong = previousSong(currentSong, head); // Get the previous song
138     cout << "\nPrevious song: " << currentSong->songName << endl;
139
140     return 0;
141 }

```

## Output:

```
Options | Compilation | Execution |
Playing all songs:
Playing: Goodluck Babe! by Chappel Roan
Playing: Please Please Please by Sabrina Carpenter
Playing: love. by Wave to Earth
Song removed: Please Please Please by Sabrina Carpenter
Playing remaining songs:
Playing: Goodluck Babe! by Chappel Roan
Playing: love. by Wave to Earth
Current song: Goodluck Babe! by Chappel Roan
Next song: love. by Wave to Earth
Previous song: Goodluck Babe! by Chappel Roan
```

## 8. Conclusion

In conclusion, this activity is challenging since there are so many codings are needed to be programmed. It's also quite confusing analyzing the outputs of each program and I got easily confuse with the process of programming. I have executed the various applications of linked lists like a singly linked list where every node contains data and a pointer to the next node, doubly linked lists where traversal can be possible in both directions, and circular linked lists that allow one to loop continuously with the list. With this process, I have been able to create and understand the playlist for managing songs along with operations like node insertion and deletion, traversal, and so on. This exercise definitely made me understand better the concepts of linked lists. However, I realize that there is still room for improvement. To become a better programmer, I must practice more and hone my skills so that I will be able to apply what I have learned accordingly.

## 9. Assessment Rubric