

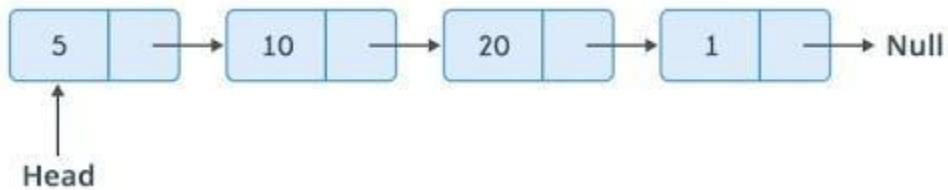
### **ACTIVITY NO. 3**

#### **LINKED LISTS**

<b>Course Code:</b> CPE010	<b>Program:</b> Computer Engineering
<b>Course Title:</b> Data Structures and Algorithms	<b>Date Performed:</b>
<b>Section:</b> CPE21S4	<b>Date Submitted:</b>
<b>Name:</b> Cruz, Axl Waltz E.	<b>Instructor:</b> Engr. Jimlord Quejado
<b>1. Objective(s)</b>	
<ul style="list-style-type: none"><li>• Implement the list ADT using singly and doubly linked lists</li><li>• Define operations based on list ADT from the module discussion</li></ul>	
<b>2. Intended Learning Outcomes (ILOs)</b>	
After this activity, the student should be able to: <ol style="list-style-type: none"><li>a. Construct C++ code for a singly and doubly linked list in C++</li><li>b. Solve given problems utilizing linked lists in C++</li></ol>	
<b>3. Discussion</b>	

## PART A: What is a linked list?

Linked Lists are a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.



*Image Source: Hackearth.com*

### Why Linked List?

- Arrays are useful but have the following limitations:
- Fixed size.
- Allocated memory remains to be the upper limit.
- Expensive operations (such as insertion) which requires movement of all existing elements and creation of room for new elements.

### Advantages over Arrays

- Dynamic size
- Ease of insertion/deletion

### Drawbacks

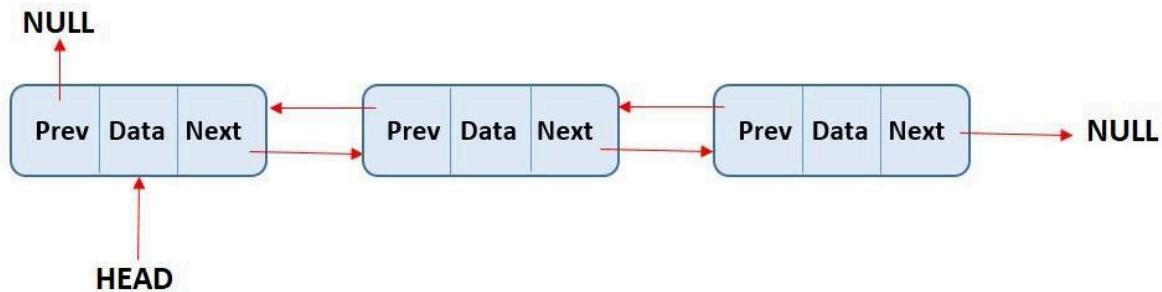
- Random access is not allowed. We have to access elements sequentially starting from the first node
- Extra memory space for a pointer is required with each element of the list.
- Not cache friendly. No locality reference.

### Representation:

- A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.
- Each node in a list consists of at least two parts:
  - Data
  - Pointer (Or Reference) to the next node

## **PART B: Doubly Linked Lists**

**Doubly Linked Lists** are traversed in either direction. It is a linked list in which every node has a next pointer and a backpointer.



*Image Source: AlphaCodingSkills*

Every node contains address of next node (except the last node). Every node contains address of previous node (except the first node).

## **PART C: Common Operations on**

### **Linked Lists Typical operations:**

- Initialize the list
- Destroy the list
- Determine if list empty
- Search list for a given item
- Insert an item
- Delete an item, and so on

## **4. Materials and Equipment**

Personal Computer with C++ IDE

Recommended IDE:

- CLion (must use TIP email to download)
- DevC++ (use the embarcadero fork or configure to C++17)

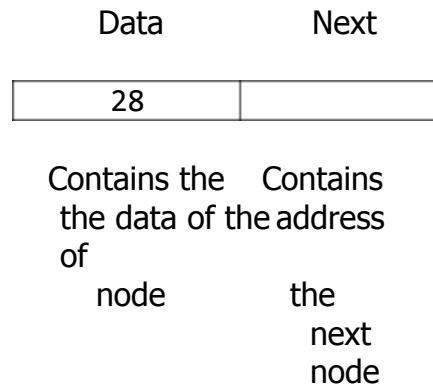
## **5. Procedure**



## ILO A: Construct C++ code for a singly and doubly linked list in C++

### A.1. Singly Linked List

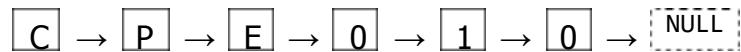
To start, we will do a simple implementation of a linked list. We must keep in mind the visual representation of the individual nodes that will make up our linked list:



Every node in a singly linked list will have 2 compartments, the data and the pointer to the next. The data contains the element of a single type that we want it to contain. The link will point to the next node in our list. In C++, the node is defined as:

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

We will implement a list to represent the string “CPE010” that looks like the figure below:



Implementation will follow the given steps:

1. Create the node pointers and initialize as NULL
2. Create new instances of the node class and allocate them in the heap
3. Define every node in the list
4. Point the last node to null

to null Simple

implementation:

```
#include<iostream>
#include<utility>

class Node{
public:
    char data;
    Node *next;
```

```
};

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;
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 = '0';
fourth->next = fifth;

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

//step 4
last->data = '0';
last->next = nullptr;
}

```

Although we have created the linked list, this is an implementation that is useless and meant only to show what we want to happen for the given output. **Run your code and screenshot the output, then briefly discuss how the output came to be and how could it be improved (table 3-1)?**

Imagine if we had to make multiple items in the list in the hundreds or thousands! This would be too tedious, ineffective, and inefficient. So, we must implement certain methods. These methods/operations associated with the linked lists are:

- Traversal
- Insertion at head
- Insertion at any part of the list
- Insertion at the end
- Deletion of a

node Linked List

Traversal

Algorithm: ListTraversal (parameter: pointer to node n)

```
WHILE n IS NOT EQUAL TO null
    PRINT data OF n
    GO TO NEXT NODE n := next
ENDWHILE
```



The screenshot shows a debugger interface with several windows. The assembly window displays the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     char data;
7     Node* next;
8     Node(char d, Node* n = nullptr) : data(d), next(n) {}
9 }
10
11 void traverse(Node* head) {
12     while (head != nullptr) {
13         cout << head->data;
14         head = head->next;
15     }
16 }
17
18 void insertFront(Node*& head, char v) {
19     Node* n = new Node(v);
20     n->next = head;
21     head = n;
22 }
23
24 void insertAfter(Node*& head, char v) {
25     Node* L = head;
26     while (L->next != nullptr) {
27         L = L->next;
28     }
29     L->next = new Node(v);
30 }
31
32 void insertBefore(Node*& head, char v) {
33     Node* n = new Node(v, head);
34     head = n;
35 }
36
37 void insertAtPos(Node*& head, char v) {
38     Node* n = new Node(v, prev->next);
39     prev->next = n;
40 }
41
42 void deleteNode(Node*& head, char key) {
43     if (head == nullptr)
44         return;
45     if (head->data == key) {
46         Node* temp = head;
47         head = head->next;
48         delete temp;
49     }
50 }
```

The memory dump window shows the state of memory after inserting 'E' after 'P':

Address	Value
0x0000000000401000	45
0x0000000000401004	45
0x0000000000401008	45
0x000000000040100C	45
0x0000000000401010	45
0x0000000000401014	45
0x0000000000401018	45
0x000000000040101C	45
0x0000000000401020	45
0x0000000000401024	45
0x0000000000401028	45
0x000000000040102C	45
0x0000000000401030	45
0x0000000000401034	45
0x0000000000401038	45
0x000000000040103C	45
0x0000000000401040	45
0x0000000000401044	45
0x0000000000401048	45
0x000000000040104C	45
0x0000000000401050	45
0x0000000000401054	45
0x0000000000401058	45
0x000000000040105C	45
0x0000000000401060	45
0x0000000000401064	45
0x0000000000401068	45
0x000000000040106C	45
0x0000000000401070	45
0x0000000000401074	45
0x0000000000401078	45
0x000000000040107C	45
0x0000000000401080	45
0x0000000000401084	45
0x0000000000401088	45
0x000000000040108C	45
0x0000000000401090	45
0x0000000000401094	45
0x0000000000401098	45
0x000000000040109C	45
0x00000000004010A0	45
0x00000000004010A4	45
0x00000000004010A8	45
0x00000000004010AC	45
0x00000000004010B0	45
0x00000000004010B4	45
0x00000000004010B8	45
0x00000000004010BC	45
0x00000000004010C0	45
0x00000000004010C4	45
0x00000000004010C8	45
0x00000000004010CC	45
0x00000000004010D0	45
0x00000000004010D4	45
0x00000000004010D8	45
0x00000000004010DC	45
0x00000000004010E0	45
0x00000000004010E4	45
0x00000000004010E8	45
0x00000000004010EC	45
0x00000000004010F0	45
0x00000000004010F4	45
0x00000000004010F8	45
0x00000000004010FC	45
0x0000000000401100	45
0x0000000000401104	45
0x0000000000401108	45
0x000000000040110C	45
0x0000000000401110	45
0x0000000000401114	45
0x0000000000401118	45
0x000000000040111C	45
0x0000000000401120	45
0x0000000000401124	45
0x0000000000401128	45
0x000000000040112C	45
0x0000000000401130	45
0x0000000000401134	45
0x0000000000401138	45
0x000000000040113C	45
0x0000000000401140	45
0x0000000000401144	45
0x0000000000401148	45
0x000000000040114C	45
0x0000000000401150	45
0x0000000000401154	45
0x0000000000401158	45
0x000000000040115C	45
0x0000000000401160	45
0x0000000000401164	45
0x0000000000401168	45
0x000000000040116C	45
0x0000000000401170	45
0x0000000000401174	45
0x0000000000401178	45
0x000000000040117C	45
0x0000000000401180	45
0x0000000000401184	45
0x0000000000401188	45
0x000000000040118C	45
0x0000000000401190	45
0x0000000000401194	45
0x0000000000401198	45
0x000000000040119C	45
0x00000000004011A0	45
0x00000000004011A4	45
0x00000000004011A8	45
0x00000000004011AC	45
0x00000000004011B0	45
0x00000000004011B4	45
0x00000000004011B8	45
0x00000000004011BC	45
0x00000000004011C0	45
0x00000000004011C4	45
0x00000000004011C8	45
0x00000000004011CC	45
0x00000000004011D0	45
0x00000000004011D4	45
0x00000000004011D8	45
0x00000000004011DC	45
0x00000000004011E0	45
0x00000000004011E4	45
0x00000000004011E8	45
0x00000000004011EC	45
0x00000000004011F0	45
0x00000000004011F4	45
0x00000000004011F8	45
0x00000000004011FC	45
0x00000000004011F0	45
0x00000000004011F4	45
0x00000000004011F8	45
0x00000000004011FC	45
0x0000000000401200	45
0x0000000000401204	45
0x0000000000401208	45
0x000000000040120C	45
0x0000000000401210	45
0x0000000000401214	45
0x0000000000401218	45
0x000000000040121C	45
0x0000000000401220	45
0x0000000000401224	45
0x0000000000401228	45
0x000000000040122C	45
0x0000000000401230	45
0x0000000000401234	45
0x0000000000401238	45
0x000000000040123C	45
0x0000000000401240	45
0x0000000000401244	45
0x0000000000401248	45
0x000000000040124C	45
0x0000000000401250	45
0x0000000000401254	45
0x0000000000401258	45
0x000000000040125C	45
0x0000000000401260	45
0x0000000000401264	45
0x0000000000401268	45
0x000000000040126C	45
0x0000000000401270	45
0x0000000000401274	45
0x0000000000401278	45
0x000000000040127C	45
0x0000000000401280	45
0x0000000000401284	45
0x0000000000401288	45
0x000000000040128C	45
0x0000000000401290	45
0x0000000000401294	45
0x0000000000401298	45
0x000000000040129C	45
0x00000000004012A0	45
0x00000000004012A4	45
0x00000000004012A8	45
0x00000000004012AC	45
0x00000000004012B0	45
0x00000000004012B4	45
0x00000000004012B8	45
0x00000000004012BC	45
0x00000000004012C0	45
0x00000000004012C4	45
0x00000000004012C8	45
0x00000000004012CC	45
0x00000000004012D0	45
0x00000000004012D4	45
0x00000000004012D8	45
0x00000000004012DC	45
0x00000000004012E0	45
0x00000000004012E4	45
0x00000000004012E8	45
0x00000000004012EC	45
0x00000000004012F0	45
0x00000000004012F4	45
0x00000000004012F8	45
0x00000000004012FC	45
0x0000000000401300	45
0x0000000000401304	45
0x0000000000401308	45
0x000000000040130C	45
0x0000000000401310	45
0x0000000000401314	45
0x0000000000401318	45
0x000000000040131C	45
0x0000000000401320	45
0x0000000000401324	45
0x0000000000401328	45
0x000000000040132C	45
0x0000000000401330	45
0x0000000000401334	45
0x0000000000401338	45
0x000000000040133C	45
0x0000000000401340	45
0x0000000000401344	45
0x0000000000401348	45
0x000000000040134C	45
0x0000000000401350	45
0x0000000000401354	45
0x0000000000401358	45
0x000000000040135C	45
0x0000000000401360	45
0x0000000000401364	45
0x0000000000401368	45
0x000000000040136C	45
0x0000000000401370	45
0x0000000000401374	45
0x0000000000401378	45
0x000000000040137C	45
0x0000000000401380	45
0x0000000000401384	45
0x0000000000401388	45
0x000000000040138C	45
0x0000000000401390	45
0x0000000000401394	45
0x0000000000401398	45
0x000000000040139C	45
0x00000000004013A0	45
0x00000000004013A4	45
0x00000000004013A8	45
0x00000000004013AC	45
0x00000000004013B0	45
0x00000000004013B4	45
0x00000000004013B8	45
0x00000000004013BC	45
0x00000000004013C0	45
0x00000000004013C4	45
0x00000000004013C8	45
0x00000000004013CC	45
0x00000000004013D0	45
0x00000000004013D4	45
0x00000000004013D8	45
0x00000000004013DC	45
0x00000000004013E0	45
0x00000000004013E4	45
0x00000000004013E8	45
0x00000000004013EC	45
0x00000000004013F0	45
0x00000000004013F4	45
0x00000000004013F8	45
0x00000000004013FC	45
0x0000000000401400	45
0x0000000000401404	45
0x0000000000401408	45
0x000000000040140C	45
0x0000000000401410	45
0x0000000000401414	45
0x0000000000401418	45
0x000000000040141C	45
0x0000000000401420	45
0x0000000000401424	45
0x0000000000401428	45
0x000000000040142C	45
0x0000000000401430	45
0x0000000000401434	45
0x0000000000401438	45
0x000000000040143C	45
0x0000000000401440	45
0x0000000000401444	45
0x0000000000401448	45
0x000000000040144C	45
0x0000000000401450	45
0x0000000000401454	45
0x0000000000401458	45
0x000000000040145C	45
0x0000000000401460	45
0x0000000000401464	45
0x0000000000401468	45
0x000000000040146C	45
0x0000000000401470	45
0x0000000000401474	45
0x0000000000401478	45
0x000000000040147C	45
0x0000000000401480	45
0x0000000000401484	45
0x0000000000401488	45
0x000000000040148C	45
0x0000000000401490	45
0x0000000000401494	45
0x0000000000401498	45
0x000000000040149C	45
0x00000000004014A0	45
0x00000000004014A4	45
0x00000000004014A8	45
0x00000000004014AC	45
0x00000000004014B0	45
0x00000000004014B4	45
0x00000000004014B8	45
0x00000000004014BC	45
0x00000000004014C0	45
0x00000000004014C4	45
0x00000000004014C8	45
0x00000000004014CC	45
0x00000000004014D0	45
0x00000000004014D4	45
0x00000000004014D8	45
0x00000000004014DC	45
0x00000000004014E0	45
0x00000000004014E4	45
0x00000000004014E8	45
0x00000000004014EC	45
0x00000000004014F0	45
0x00000000004014F4	45
0x00000000004014F8	45
0x00000000004014FC	45
0x0000000000401500	45
0x0000000000401504	45
0x0000000000401508	45
0x000000000040150C	45
0x0000000000401510	45
0x0000000000401514	45
0x0000000000401518	45
0x000000000040151C	45
0x0000000000401520	45
0x0000000000401524	45
0x0000000000401528	45
0x000000000040152C	45
0x0000000000401530	45
0x0000000000401534	45
0x0000000000401538	45
0x000000000040153C	45
0x0000000000401540	45
0x0000000000401544	45
0x0000000000401548	45
0x000000000040154C	45
0x0000000000401550	45
0x0000000000401554	45
0x0000000000401558	45
0x000000000040155C	45
0x0000000000401560	45
0x0000	

```
PRINT next line
END
```

Sometimes we have a linked list, and we need to insert a node somewhere other than at the end of the list. We will look at a couple of different ways to insert a node into an existing list.

To insert a node at the head:

1. Allocate memory for the new node
2. Put our data into the new node
3. Set Next of the new node to point to the previous Head
4. Reset Head to point to the new node

To insert a node at any location between the head and tail:

1. Check if it is the head node (previous node is null)
2. If null, print "Previous node cannot be null."
3. Allocate a new node
4. Store data in the new node
5. Point new node to the node previous node was pointing to
6. Point previous node to the new node

To insert a node at the end:

1. Allocate new node
2. Dereference to the head node
3. Store data in new node
4. Point next of new node to NULL
5. Traverse the list until next of the node is null
6. Point the next of the current node to the new node

To delete a node from linked list:

1. Find previous node of the node to be deleted.
2. Change the next of previous node.
3. Free memory for the node to be deleted.

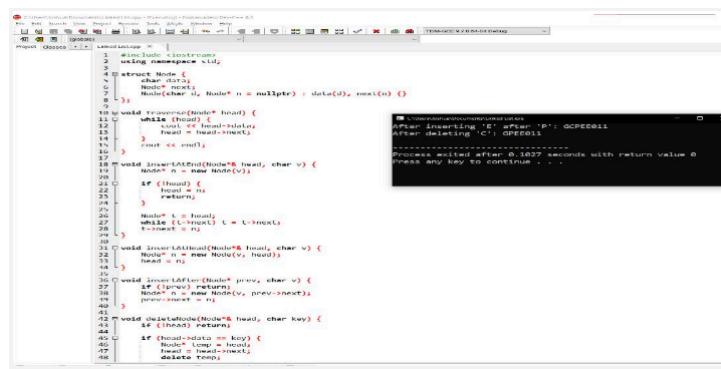
**Create code for each of the pseudocode given for all list operations above. Provide screenshots in table 3-2.**

The screenshot shows a C++ development environment with several windows open. On the left, there is a code editor with C++ code for linked list operations. The code includes functions for inserting nodes at the head, tail, and after a specific node, as well as deleting nodes. On the right, there is a terminal window showing the output of the program. The terminal window displays the initial state of the list, the insertion of a node with value 'C', and the final state of the list after the insertion. The code in the editor is as follows:

```
1 // Linked List
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 insertAtHead(Node*& head, char v) {
18     Node* n = new Node();
19     n->data = v;
20     n->next = head;
21     head = n;
22 }
23
24 void insertAtTail(Node*& head, char v) {
25     Node* t = head;
26     while (t->next != NULL) t = t->next;
27     t->next = new Node(v);
28 }
29
30 void insertAfter(Node*& head, char v) {
31     Node* n = new Node(v);
32     n->next = head->next;
33     head->next = n;
34 }
35
36 void deleteNode(Node*& head, char key) {
37     if (!head) return;
38     Node* temp = head->next;
39     head->next = temp->next;
40     delete temp;
41 }
42
43 void printList() {
44     Node* temp = head;
45     while (temp) {
46         cout << temp->data;
47         temp = temp->next;
48     }
49 }
```

**In your driver function, show the use of each list operation and show the output in table 3-3 found in section 6 with a descriptive caption for each. The tasks you have to perform are as follows:**

- a. Traverse the list by passing the head of the created list into the function
- b. Insert the element 'G' at the start of the list to replace the current node. Output should now show "GCPE101"
- c. Insert an element "E" with the previous node element being "P". Output should now show "GCPEE101".
- d. Delete the node containing the element C.



The screenshot shows a C++ development environment with the following code in a file named `list.h`:

```
1 // C:\Users\Aman\Documents\Visual Studio 2010\Projects\List\List\list.h
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     char data;
7     Node* next;
8 };
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 void insertAtHead(Node*& head, char v) {
19     Node* n = new Node(v);
20     if (!head)
21         head = n;
22     else {
23         n->next = head;
24         head = n;
25     }
26 }
27
28 void insertAtAfter(Node*& head, char v) {
29     Node* t = head;
30     while (t->next != NULL) t = t->next;
31     t->next = new Node(v, head);
32     head = t;
33 }
34
35 void insertAtBefore(Node*& prev, char v) {
36     Node* t = prev->next;
37     if (!t)
38         return;
39     Node* n = new Node(v, t);
40     prev->next = n;
41 }
42
43 void deleteNode(Node*& head, char key) {
44     Node* temp;
45     if (head->data == key) {
46         Node* temp = head;
47         head = head->next;
48         delete temp;
49     }
50 }
```

The terminal window shows the execution of the program:

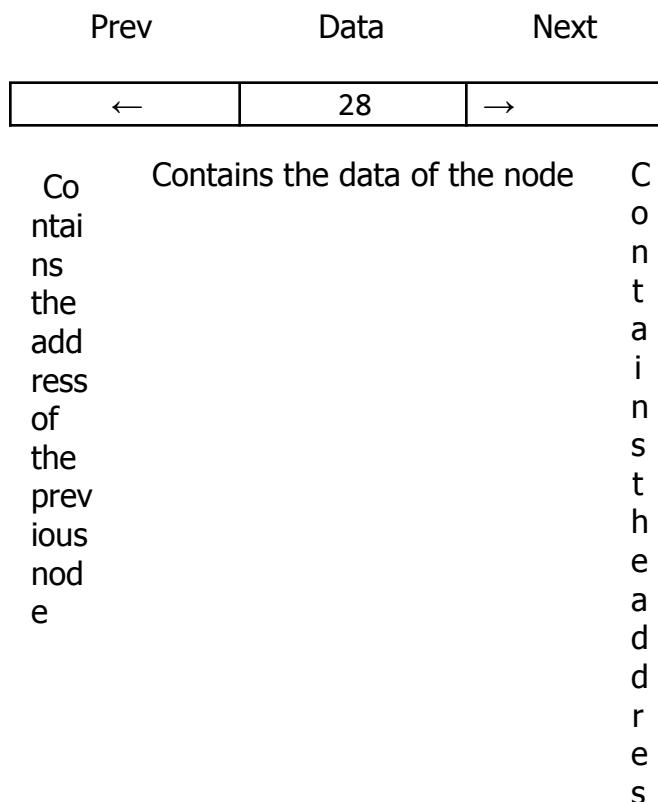
```
C:\Users\Aman\Documents\Visual Studio 2010\Projects\List\List> list
List
After inserting character 'B': GCPEE101
After deleting 'C': GCPEE01
Press any key to continue . . .
```

- e. Delete the node containing the element P.
  - f. Show the elements in the list. Output should be "GEE101".

## A.2. Doubly Linked List

The singly linked list allows for direct access from a list node only to the next node in the list. A doubly linked list allows convenient access from a list node to the next node and also to the preceding node on the list.

The doubly linked list node accomplishes this in the obvious way by storing two pointers: one to the node following it (as in the singly linked list), and a second pointer to the node preceding it.





The screenshot shows a C++ development environment with the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 };
8
9 void insertFront(Node*& head, char v) {
10    while (head != NULL) {
11        Node* temp = head->next;
12        head->data = v;
13        head = temp;
14    }
15    cout << endl;
16}
17
18 void insertAtEnd(Node*& head, char v) {
19    Node* n = new Node(v);
20    if (head == NULL) {
21        head = n;
22    } else {
23        Node* temp = head;
24        while (temp->next != NULL) {
25            temp = temp->next;
26        }
27        temp->next = n;
28    }
29}
30
31 void insertAtMid(Node*& head, char v) {
32    Node* n = new Node(v);
33    head = n;
34}
35
36 void insertAfter(Node* prev, char v) {
37    Node* n = new Node(v);
38    n->next = prev->next;
39    prev->next = n;
40}
41
42 void deleteNode(Node*& head, char key) {
43    if (!head) return;
44    if (head->data == key) {
45        Node* temp = head;
46        head = head->next;
47        delete temp;
48    }
49}
```

The status bar at the bottom right indicates "Press any key to continue .".

e  
n  
e  
x  
t

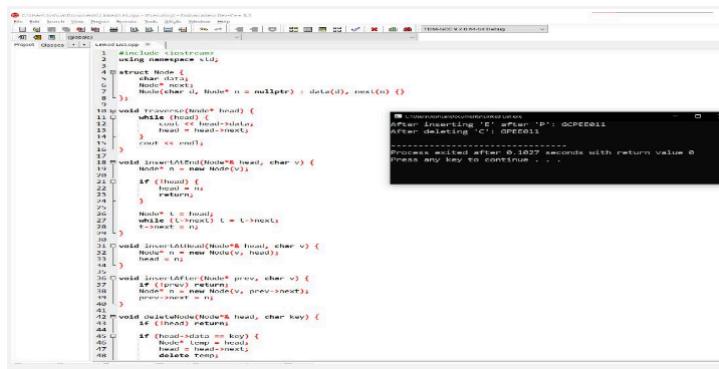
n  
c  
e

This means that in addition to our implementation of the singly linked list, we'll have addition compartment.

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

**Modify the given operations used in the singly linked lists to work on the new construct of a doubly linked list. Provide a screenshot of your code and analysis in table 3-4.**

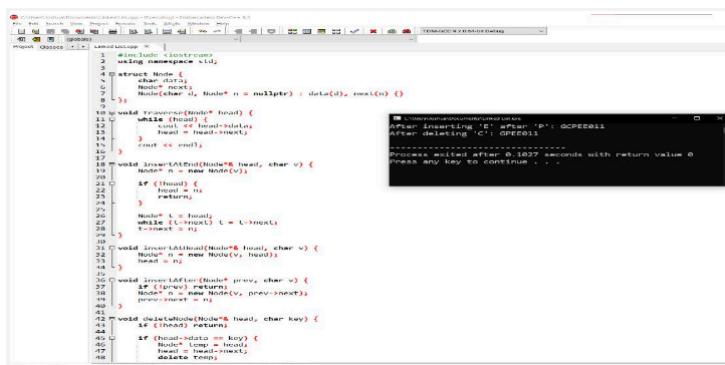
## 6. Output



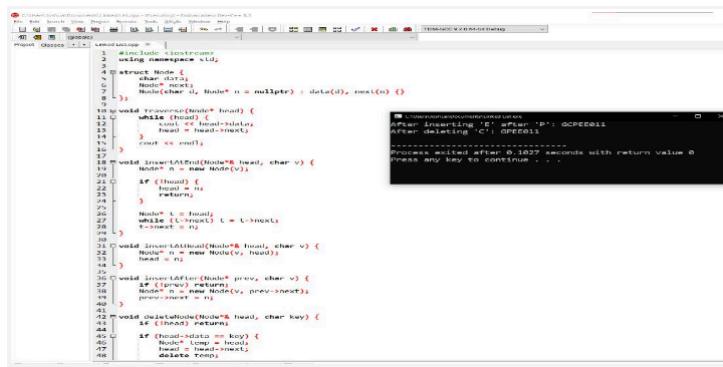
<h2>Screenshot</h2>	<pre>#include &lt;iostream&gt; using namespace std;  // Only define Node once class Node { public:     char data;     Node *next; };  int main() {     // Step 1     Node *head = NULL;     Node *second = NULL;     Node *third = NULL;     Node *fourth = NULL;     Node *fifth = NULL;     Node *last = NULL;      // Step 2     head-&gt;data = 'C'; head-&gt;next = second;     second-&gt;data = 'P'; second-&gt;next = third;     third-&gt;data = 'E'; third-&gt;next = fourth;     fourth-&gt;data = 'R'; fourth-&gt;next = fifth;     fifth-&gt;data = 'T'; fifth-&gt;next = last;      // Step 3     last-&gt;data = 'O'; last-&gt;next = nullptr;      // Optional: Print to verify (not required if you don't want changes)     Node* temp = head;     while (temp != nullptr) {         cout &lt;&lt; temp-&gt;data &lt;&lt; " ";         temp = temp-&gt;next;     }     cout &lt;&lt; endl; }  return 0; }</pre>	<p>STDIN Input for the program (Optional)</p> <p>Output: C P E O T O</p>
<h2>Discussion</h2>	<p>This list is properly constructed, each node is correctly linked and the data forms the string when read sequentially, to improve the code encapsulation with a link list could do and memory management which is a destructor to allocate memory and never freeing it.</p>	

Table 3-1. Output of Initial/Simple Implementation

Operation	Screenshot
-----------	------------



Traversal	<pre>// 1. Traversal void traverse(Node* head) {     Node* current = head;     while (current != nullptr) {         cout &lt;&lt; current-&gt;data &lt;&lt; " ";         current = current-&gt;next;     }     cout &lt;&lt; endl; }</pre>
Insertion at head	<pre>// 2. Insertion at head void insertAtHead(Node*&amp; head, char value) {     Node* newNode = new Node;     newNode-&gt;data = value;     newNode-&gt;next = head;     head = newNode; }</pre>
Insertion at any part of the list	<pre>// 3. Insertion at any position (0-based) void insertAtPosition(Node*&amp; head, int position, char value) {     if (position == 0) {         insertAtHead(head, value);         return;     }     -</pre>



```

1 // C++ program to demonstrate singly linked list
2
3 #include <iostream>
4 using namespace std;
5
6 struct Node {
7     char data;
8     Node* next;
9 };
10
11 void traverse(Node* head) {
12     while (head) {
13         cout << head->data << " ";
14         head = head->next;
15     }
16     cout << endl;
17 }
18
19 void insertAtHead(Node*& head, char v) {
20     Node* n = new Node;
21     n->data = v;
22     n->next = head;
23     head = n;
24 }
25
26 void insertAtPosition(Node*& head, int pos, char v) {
27     Node* t = head;
28     while (pos > 1) {
29         t = t->next;
30         pos--;
31     }
32     Node* l = new Node;
33     l->data = v;
34     l->next = t->next;
35     t->next = l;
36 }
37
38 void insertAfter(Node*& prev, char v) {
39     if (!prev) return;
40     Node* n = new Node(v, prev->next);
41     prev->next = n;
42 }
43
44 void deleteNode(Node*& head, char key) {
45     Node* temp;
46     if (head->data == key) {
47         temp = head;
48         head = head->next;
49         delete temp;
50     }
51     else {
52         Node* t = head;
53         while (t->next) {
54             if (t->next->data == key) {
55                 temp = t->next;
56                 t->next = t->next->next;
57                 delete temp;
58             }
59             t = t->next;
60         }
61     }
62 }
```

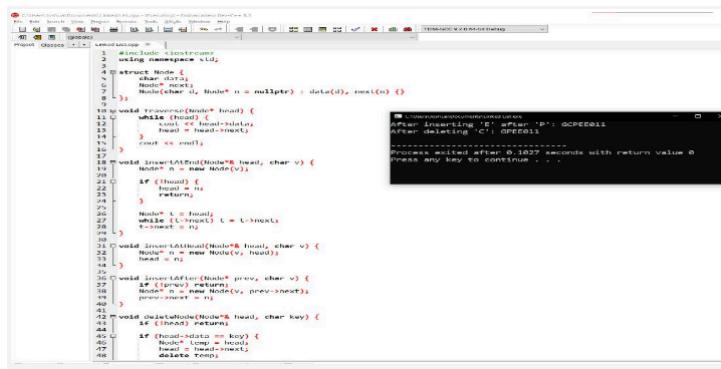
## Insertion at the end

```
// 4. Insertion at end
void insertAtEnd(Node*& head, char value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
        return;
    }

    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    temp->next = newNode;
}
```



## Deletion of a node

```
// 5. Deletion of a node by value
void deleteNode(Node*& head, char value) {
    if (head == nullptr) return;

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

    Node* prev = head;
    Node* curr = head->next;

    while (curr != nullptr && curr->data != value) {
        prev = curr;
        curr = curr->next;
    }

    if (curr == nullptr) {
        cout << "Value not found." << endl;
        return;
    }

    prev->next = curr->next;
    delete curr;
}
```

Table 3-2. Code for the List Operations



The screenshot shows a debugger interface with assembly code and memory dump panes. The assembly pane displays C++ code for linked list operations. The memory dump pane shows the state of memory after inserting a node 'C' at address 0x10000000. The stack dump pane shows the state of the stack after the operation.

```
1 //include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 };
8
9 Node* insert(Node* head, char v) {
10    Node* newnode = new Node();
11    newnode->data = v;
12    newnode->next = head;
13    head = newnode;
14    cout << "Data inserted" << endl;
15    return head;
16 }
17
18 void traverse(Node* head) {
19     while (head != NULL) {
20         cout << head->data;
21         head = head->next;
22     }
23     cout << endl;
24 }
25
26 void printList(Node* head) {
27     if (head == NULL) {
28         cout << "List is empty" << endl;
29     } else {
30         Node* t = head;
31         while (t->next != NULL) {
32             cout << t->data << " ";
33             t = t->next;
34         }
35         cout << t->data << endl;
36     }
37 }
38
39 void insertAfter(Node*& head, char v) {
40     Node* newnode = new Node(v, head);
41     head = newnode;
42 }
43
44 void insertBefore(Node*& prev, char v) {
45     Node* newnode = new Node(v, prev->next);
46     prev->next = newnode;
47 }
48
49 void deleteNode(Node*& head, char key) {
50
51     Node* temp = head;
52
53     if (temp->data == key) {
54         Node* temp1 = head;
55         head = head->next;
56         delete temp1;
57     }
58 }
```

Creates a new node with its next pointer referencing the current head, then updates the head to this new node.

Analysis: It locates the node “P,” creates a new node, a

```

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 != nullptr) {
11        cout << head->data;
12        head = head->next;
13    }
14    cout << endl;
15 }
16
17 void insertAfter(Node*& head, char v) {
18    if (head == nullptr) {
19        cout << "Initial list: ";
20        traverse(head);
21        cout << endl;
22        return;
23    }
24
25    Node* t = head;
26    while (t->next != nullptr) {
27        t = t->next;
28    }
29    t->next = new Node(v);
30
31 }
32
33 int main() {
34     Node* head = nullptr;
35
36     insertAfter(head, 'C');
37     insertAfter(head, 'D');
38     insertAfter(head, 'E');
39     insertAfter(head, 'F');
40
41     cout << "Initial list: ";
42     traverse(head);
43     cout << endl;
44
45     return 0;
46 }

```

```

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 != nullptr) {
11        cout << head->data;
12        head = head->next;
13    }
14    cout << endl;
15 }
16
17 void insertAfter(Node*& head, char v) {
18    if (head == nullptr) {
19        cout << "Initial list: ";
20        traverse(head);
21        cout << endl;
22        return;
23    }
24
25    Node* t = head;
26    while (t->next != nullptr) {
27        t = t->next;
28    }
29    t->next = new Node(v);
30
31 }
32
33 void insertAtHead(Node*& head, char v) {
34    Node* n = new Node(v, head);
35    head = n;
36
37 }
38
39 void insertBefore(Node*& prev, char v) {
40    if (!prev) return;
41    Node* n = new Node(v, prev->next);
42    prev->next = n;
43
44 }
45
46 void deleteNode(Node*& head, char key) {
47    if (!head) return;
48
49    if (head->data == key) {
50        Node* temp = head;
51        head = head->next;
52        delete temp;
53        return;
54    }
55
56    Node* t = head;
57    while (t->next != head) {
58        t = t->next;
59    }
60    t->next = head->next;
61
62 }
63
64 int main() {
65     Node* head = nullptr;
66
67     insertAtHead(head, 'C');
68     insertBefore(head, 'D');
69     insertBefore(head, 'E');
70
71     cout << "Initial list: ";
72     traverse(head);
73     cout << endl;
74
75     cout << "After inserting 'E' after 'P': ";
76     insertAfter(head, 'E');
77
78     cout << "After deleting 'P': ";
79     deleteNode(head, 'P');
80
81     cout << endl;
82
83     cout << "Final list: ";
84     traverse(head);
85     cout << endl;
86
87     return 0;
88 }

```

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

The function moves through the list by advancing

```

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 != nullptr) {
11        cout << head->data;
12        head = head->next;
13    }
14    cout << endl;
15 }
16
17 void insertAfter(Node*& head, char v) {
18    if (head == nullptr) {
19        cout << "Initial list: ";
20        traverse(head);
21        cout << endl;
22        return;
23    }
24
25    Node* t = head;
26    while (t->next != nullptr) {
27        t = t->next;
28    }
29    t->next = new Node(v);
30
31 }
32
33 int main() {
34     Node* head = nullptr;
35
36     insertAfter(head, 'C');
37     insertAfter(head, 'D');
38     insertAfter(head, 'E');
39     insertAfter(head, 'F');
40
41     cout << "Initial list: ";
42     traverse(head);
43     cout << endl;
44
45     return 0;
46 }

```

```

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 != nullptr) {
11        cout << head->data;
12        head = head->next;
13    }
14    cout << endl;
15 }
16
17 void insertAfter(Node*& head, char v) {
18    if (head == nullptr) {
19        cout << "Initial list: ";
20        traverse(head);
21        cout << endl;
22        return;
23    }
24
25    Node* t = head;
26    while (t->next != nullptr) {
27        t = t->next;
28    }
29    t->next = new Node(v);
30
31 }
32
33 void insertAtHead(Node*& head, char v) {
34    Node* n = new Node(v, head);
35    head = n;
36
37 }
38
39 void insertBefore(Node*& prev, char v) {
40    if (!prev) return;
41    Node* n = new Node(v, prev->next);
42    prev->next = n;
43
44 }
45
46 void deleteNode(Node*& head, char key) {
47    if (!head) return;
48
49    if (head->data == key) {
50        Node* temp = head;
51        head = head->next;
52        delete temp;
53        return;
54    }
55
56    Node* t = head;
57    while (t->next != head) {
58        t = t->next;
59    }
60    t->next = head->next;
61
62 }
63
64 int main() {
65     Node* head = nullptr;
66
67     insertAtHead(head, 'C');
68     insertBefore(head, 'D');
69     insertBefore(head, 'E');
70
71     cout << "Initial list: ";
72     traverse(head);
73     cout << endl;
74
75     cout << "After inserting 'E' after 'P': ";
76     insertAfter(head, 'E');
77
78     cout << "After deleting 'P': ";
79     deleteNode(head, 'P');
80
81     cout << endl;
82
83     cout << "Final list: ";
84     traverse(head);
85     cout << endl;
86
87     return 0;
88 }

```

```

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 != nullptr) {
11        cout << head->data;
12        head = head->next;
13    }
14    cout << endl;
15 }
16
17 void insertAfter(Node*& head, char v) {
18    if (head == nullptr) {
19        cout << "Initial list: ";
20        traverse(head);
21        cout << endl;
22        return;
23    }
24
25    Node* t = head;
26    while (t->next != nullptr) {
27        t = t->next;
28    }
29    t->next = new Node(v);
30
31 }
32
33 int main() {
34     Node* head = nullptr;
35
36     insertAfter(head, 'C');
37     insertAfter(head, 'D');
38     insertAfter(head, 'E');
39     insertAfter(head, 'F');
40
41     cout << "Initial list: ";
42     traverse(head);
43     cout << endl;
44
45     return 0;
46 }

```

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 };
8
9 void traverse(Node* head) {
10     while (head) {
11         cout << head->data;
12         head = head->next;
13     }
14     cout << endl;
15 }
16
17 int main() {
18     Node* head = nullptr;
19
20     head = new Node();
21     head->data = 'G';
22
23     head->next = new Node();
24     head->next->data = 'E';
25
26     head->next->next = new Node();
27     head->next->next->data = 'E';
28
29     head->next->next->next = new Node();
30     head->next->next->next->data = 'I';
31
32     head->next->next->next->next = new Node();
33     head->next->next->next->next->data = 'L';
34
35     head->next->next->next->next->next = new Node();
36     head->next->next->next->next->next->data = 'O';
37
38     head->next->next->next->next->next->next = new Node();
39     head->next->next->next->next->next->next->data = 'P';
40
41     head->next->next->next->next->next->next->next = new Node();
42     head->next->next->next->next->next->next->next->data = 'I';
43
44     cout << "Final list: ";
45     traverse(head);
46
47     return 0;
48 }

```

Final list: GEEILOI  
Process exited after 0.09857 seconds with return value 0  
Press any key to continue . . .

Table 3-4. Modified Operations for Doubly Linked Lists

```

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* n = head;
12     while (head) {
13         cout << head->data;
14         head = head->next;
15     }
16     cout << endl;
17 }
18
19 void insertAtEnd(Node*& head, char v) {
20     Node* n = new Node();
21     n->data = v;
22     n->next = nullptr;
23     n->prev = nullptr;
24     if (head) {
25         head->next = n;
26         return;
27     }
28     Node* t = head;
29     while (t->next) {
30         t = t->next;
31     }
32     t->next = n;
33     n->prev = t;
34 }
35
36 void insertAtBeginning(Node*& head, char v) {
37     Node* n = new Node();
38     n->prev = nullptr;
39     n->next = head;
40     if (head) head->prev = n;
41     head = n;
42     n->data = v;
43 }
44
45 void insertAfter(Node*& head, char prevData, char v) {

```

Initial list: GPEEIOI  
GCPFEIOI  
GPEEIOI  
GEEIOI  
Final list: GEEBIOI  
Process exited after 0.09896 seconds with return value 0  
Press any key to continue . . .

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node* prev;
8 };
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 void insertAtEnd(Node*& head, char v) {
19     Node* n = new Node();
20     n->data = v;
21     if (head) {
22         head->next = n;
23         n->prev = head;
24     }
25     else {
26         head = n;
27         n->prev = nullptr;
28     }
29 }
30
31 void insertAtBeginning(Node*& head, char v) {
32     Node* n = new Node();
33     n->data = v;
34     if (head) {
35         head->prev = n;
36         n->next = head;
37         head = n;
38     }
39 }
40
41 void insertBefore(Node*& prev, char v) {
42     if (!prev) return;
43     Node* n = new Node();
44     n->data = v;
45     n->next = prev->next;
46     n->prev = prev;
47 }
48
49 void deleteNode(Node*& head, char key) {
50     Node* temp;
51     if (head->data == key) {
52         temp = head;
53         head = head->next;
54         delete temp;
55     }
56     else {
57         Node* t = head;
58         while (t->next) {
59             if (t->next->data == key) {
60                 temp = t->next;
61                 t->next = t->next->next;
62                 t->next->prev = t;
63                 delete temp;
64             }
65         }
66     }
67 }

```

Initial list: GPEEIOI  
After deleting 'C': GPEEIOI  
Process exited after 0.09861 seconds with return value 0  
Press any key to continue . . .

## **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.

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 struct Node {
 5     char data;
 6     Node* next;
 7 };
 8
 9 Node* insertFront(Node* head, char v) {
10     Node* n = new Node();
11     n->data = v;
12     n->next = head;
13     head = n;
14     return head;
15 }
16
17 void traverse(Node* head) {
18     while (head != NULL) {
19         cout << head->data << endl;
20         head = head->next;
21     }
22 }
23
24 void insertATEnd(Node*& head, char v) {
25     if (!head) {
26         head = new Node();
27         head->data = v;
28         head->next = NULL;
29     } else {
30         Node* l = head;
31         while (l->next != NULL)
32             l = l->next;
33         l->next = new Node(v);
34     }
35 }
36
37 void insertATHead(Node*& head, char v) {
38     Node* n = new Node(v);
39     n->next = head;
40     head = n;
41 }
42
43 void deleteLastNode(Node*& head, char key) {
44     Node* temp;
45
46     if (head->data == key) {
47         Node* temp = head;
48         head = head->next;
49         delete temp;
50     }
```

```

#include <string>
using namespace std;

struct Song {
    string title;
    Song* next;
    Song* prev;
};

void addSong(Song*& head, const string& title) {
    Song* newSong = new Song{title, nullptr, nullptr};
    if (!head) {
        newSong->next = newSong;
        newSong->prev = newSong;
    } else {
        newSong->next = head;
        newSong->prev = head->prev;
        head->prev = newSong;
        head = newSong;
    }
}

#include <string>
using namespace std;

struct Song {
    string title;
    Song* next;
    Song* prev;
};

void addSong(Song*& head, const string& title) {
    Song* newSong = new Song{title, nullptr, nullptr};
    if (!head) {
        newSong->next = newSong;
        newSong->prev = newSong;
        head = newSong;
        return;
    }
    Song* tail = head->prev;
    tail->next = newSong;
    newSong->prev = tail;
    newSong->next = head;
    head->prev = newSong;
}

```

```

1 // C:\Users\Hans\Documents\Visual Studio 2015\Projects\List\List\main.cpp
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     char data;
7     Node* next;
8     Node* prev;
9 };
10
11 void traverse(Node* head) {
12     while (head) {
13         cout << head->data;
14         head = head->next;
15     }
16     cout << endl;
17 }
18
19 void insertAtHead(Node*& head, char v) {
20     Node* n = new Node{v, head, head};
21     if (head == n)
22         head = n;
23     else
24         n->next = head;
25     head->prev = n;
26     head = n;
27 }
28
29 void insertAtTail(Node*& head, char v) {
30     Node* n = new Node{v, head, head};
31     if (head == n)
32         head = n;
33     else
34         n->prev = head;
35     head->next = n;
36 }
37
38 void insertAfter(Node*& prev, char v) {
39     if (!prev) return;
40     Node* n = new Node{v, prev, prev->next};
41     prev->next = n;
42 }
43
44 void deleteNode(Node*& head, char key) {
45     if (head->data == key) {
46         Node* temp = head;
47         head = head->next;
48         delete temp;
49     }
50 }

```

Output from terminal window:

```

C:\Users\Hans\Documents\Visual Studio 2015\Projects\List\List> g++ main.cpp
C:\Users\Hans\Documents\Visual Studio 2015\Projects\List\List> ./list
Press any key to continue . .

```

```

void removeSong(Song*& head, const string& title) {
    if (!head) return;
    Song* curr = head;
    do {
        if (curr->title == title) {
            if (curr->next == curr) {
                delete curr;
                head = nullptr;
                return;
            }
            curr->prev->next = curr->next;
            curr->next->prev = curr->prev;
            if (curr == head) head = curr->next;
            delete curr;
            return;
        }
        curr = curr->next;
    } while (curr != head);
}

void displayPlaylist(Song* head) {
    if (!head) {
        cout << "Playlist is empty.\n";
        return;
    }
    Song* curr = head;
    do {
        cout << "Playing: " << curr->title << endl;
        curr = curr->next;
    } while (curr != head);
}

int main() {
    Song* playlist = nullptr;

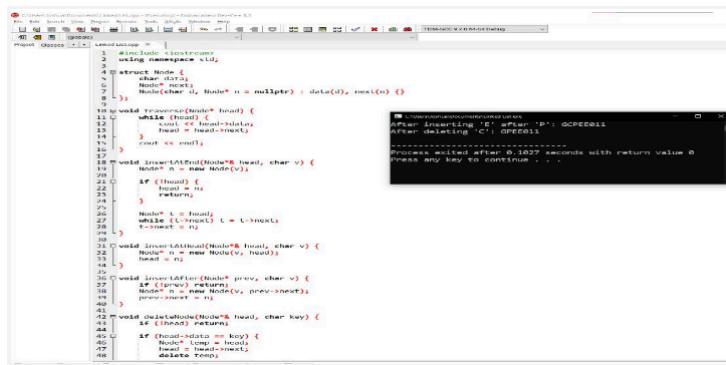
    addSong(playlist, "Song A");
    addSong(playlist, "Song B");
    addSong(playlist, "Song C");
    addSong(playlist, "Song D");

    cout << "Initial Playlist:\n";
    displayPlaylist(playlist);

    cout << "\nRemoving Song B...\n";
    removeSong(playlist, "Song B");

    cout << "Updated Playlist:\n";
    displayPlaylist(playlist);
}

```



```
39     delete curr;
40     return;
41   }
42   curr = curr->next;
43 } while (curr != head);
44 }

45 void printList() {
46   cout << "Initial Playlist:" << endl;
47   Node* curr = head;
48   while (curr != NULL) {
49     cout << "Playing: " << curr->data << endl;
50     curr = curr->next;
51   }
52   cout << "Removing Song B..." << endl;
53
54   curr = head;
55   while (curr != NULL) {
56     if (curr->data == 'B') {
57       curr = curr->next;
58     } else {
59       curr->next = curr->next->next;
60     }
61   }
62
63   cout << "Updated Playlist:" << endl;
64   curr = head;
65   while (curr != NULL) {
66     cout << "Playing: " << curr->data << endl;
67     curr = curr->next;
68   }
69
70   cout << "-----" << endl;
71
72   cout << "Process exited after 0.01568 seconds with return value 0" << endl;
73   cout << "Press any key to continue . . ." << endl;
74
75 }

76 }

77 }
```

Resources    Compile Log    Debug    Filter Results    Close

Message

re\TIPQC\Documents\Untitled1.cpp

In function 'void addSong(Node\*& head, const string&)':

## 8. Conclusion

I learned about the link list on how nodes are made up in data structure where it stores data and a pointer to the next node allowing dynamic memory usage and easy insertion or deletion. Unlike arrays they didn't store elements in contiguous memory like link list do.

Provide the following:

- Summary of lessons learned
- Analysis of the procedure
- Analysis of the supplementary activity
- Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

## 9. Assessment Rubric

```
1 // C:\Users\TIPQC\Documents\Untitled1.cpp
2 // Microsoft Visual Studio Community 2015
3 // Version 14.0.25123.0
4 // Copyright (c) Microsoft Corporation. All rights reserved.
5 // Licensed under the Apache License, Version 2.0 (the "License");
6 // you may not use this file except in compliance with the License.
7 // You may obtain a copy of the License at
8 // http://www.apache.org/licenses/LICENSE-2.0
9 // Unless required by applicable law or agreed to in writing, software
10 // distributed under the License is distributed on an "AS IS" BASIS,
11 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 // See the License for the specific language governing permissions and
13 // limitations under the License.
14
15 #include <iostream>
16
17 using namespace std;
18
19 struct Node {
20   char data;
21   Node* next;
22 };
23
24 Node* head;
25
26 void traverse(Node* head) {
27   while (head != NULL) {
28     cout << head->data << " ";
29     head = head->next;
30   }
31   cout << endl;
32 }
33
34 void insertAtHead(Node*& head, char v) {
35   Node* n = new Node();
36   n->data = v;
37   n->next = head;
38   head = n;
39 }
40
41 void insertAtTail(Node*& head, char v) {
42   Node* n = new Node();
43   n->data = v;
44   Node* t = head;
45   while (t->next != NULL) t = t->next;
46   t->next = n;
47 }
48
49 void insertAfter(Node*& prev, char v) {
50   Node* n = new Node();
51   n->data = v;
52   n->next = prev->next;
53   prev->next = n;
54 }
55
56 void deleteNode(Node*& head, char key) {
57   Node* curr = head;
58   Node* temp;
59   if (curr->data == key) {
60     temp = curr;
61     curr = curr->next;
62     delete temp;
63   }
64   curr = curr->next;
65   while (curr != head) {
66     if (curr->data == key) {
67       temp = curr;
68       curr = curr->next;
69       delete temp;
70     }
71     curr = curr->next;
72   }
73 }
```

100% of the code has been executed.

After deleting 'C': GPCED01

-----

Process exited after 0.01568 seconds with return value 0

Press any key to continue . . .



The screenshot shows a debugger interface with several windows. The assembly window displays the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     char data;
7     Node* next;
8     Node(char d, Node* n = nullptr) : data(d), next(n) {}
9 }
10
11 void traverse(Node* head) {
12     while (head != nullptr) {
13         cout << head->data;
14         head = head->next;
15     }
16 }
17
18 void insertFront(Node*& head, char v) {
19     Node* n = new Node(v);
20     n->next = head;
21     head = n;
22 }
23
24 void insertAfter(Node*& head, char v) {
25     Node* L = head;
26     while (L->next != nullptr) {
27         L = L->next;
28     }
29     L->next = new Node(v);
30 }
31
32 void insertBefore(Node*& head, char v) {
33     Node* n = new Node(v, head);
34     head = n;
35 }
36
37 void insertAtPos(Node*& head, char v) {
38     Node* n = new Node(v, prev->next);
39     prev->next = n;
40 }
41
42 void deleteNode(Node*& head, char key) {
43     if (head == nullptr)
44         return;
45     if (head->data == key) {
46         Node* temp = head;
47         head = head->next;
48         delete temp;
49     }
50 }
```

The memory dump window shows the state of memory after inserting 'E' after 'P':

Address	Value
0x0000000000401000	45
0x0000000000401004	45
0x0000000000401008	45
0x000000000040100C	45
0x0000000000401010	45
0x0000000000401014	45
0x0000000000401018	45
0x000000000040101C	45
0x0000000000401020	45
0x0000000000401024	45
0x0000000000401028	45
0x000000000040102C	45
0x0000000000401030	45
0x0000000000401034	45
0x0000000000401038	45
0x000000000040103C	45
0x0000000000401040	45
0x0000000000401044	45
0x0000000000401048	45
0x000000000040104C	45
0x0000000000401050	45
0x0000000000401054	45
0x0000000000401058	45
0x000000000040105C	45
0x0000000000401060	45
0x0000000000401064	45
0x0000000000401068	45
0x000000000040106C	45
0x0000000000401070	45
0x0000000000401074	45
0x0000000000401078	45
0x000000000040107C	45
0x0000000000401080	45
0x0000000000401084	45
0x0000000000401088	45
0x000000000040108C	45
0x0000000000401090	45
0x0000000000401094	45
0x0000000000401098	45
0x000000000040109C	45
0x00000000004010A0	45
0x00000000004010A4	45
0x00000000004010A8	45
0x00000000004010AC	45
0x00000000004010B0	45
0x00000000004010B4	45
0x00000000004010B8	45
0x00000000004010BC	45
0x00000000004010C0	45
0x00000000004010C4	45
0x00000000004010C8	45
0x00000000004010CC	45
0x00000000004010D0	45
0x00000000004010D4	45
0x00000000004010D8	45
0x00000000004010DC	45
0x00000000004010E0	45
0x00000000004010E4	45
0x00000000004010E8	45
0x00000000004010EC	45
0x00000000004010F0	45
0x00000000004010F4	45
0x00000000004010F8	45
0x00000000004010FC	45
0x0000000000401100	45
0x0000000000401104	45
0x0000000000401108	45
0x000000000040110C	45
0x0000000000401110	45
0x0000000000401114	45
0x0000000000401118	45
0x000000000040111C	45
0x0000000000401120	45
0x0000000000401124	45
0x0000000000401128	45
0x000000000040112C	45
0x0000000000401130	45
0x0000000000401134	45
0x0000000000401138	45
0x000000000040113C	45
0x0000000000401140	45
0x0000000000401144	45
0x0000000000401148	45
0x000000000040114C	45
0x0000000000401150	45
0x0000000000401154	45
0x0000000000401158	45
0x000000000040115C	45
0x0000000000401160	45
0x0000000000401164	45
0x0000000000401168	45
0x000000000040116C	45
0x0000000000401170	45
0x0000000000401174	45
0x0000000000401178	45
0x000000000040117C	45
0x0000000000401180	45
0x0000000000401184	45
0x0000000000401188	45
0x000000000040118C	45
0x0000000000401190	45
0x0000000000401194	45
0x0000000000401198	45
0x000000000040119C	45
0x00000000004011A0	45
0x00000000004011A4	45
0x00000000004011A8	45
0x00000000004011AC	45
0x00000000004011B0	45
0x00000000004011B4	45
0x00000000004011B8	45
0x00000000004011BC	45
0x00000000004011C0	45
0x00000000004011C4	45
0x00000000004011C8	45
0x00000000004011CC	45
0x00000000004011D0	45
0x00000000004011D4	45
0x00000000004011D8	45
0x00000000004011DC	45
0x00000000004011E0	45
0x00000000004011E4	45
0x00000000004011E8	45
0x00000000004011EC	45
0x00000000004011F0	45
0x00000000004011F4	45
0x00000000004011F8	45
0x00000000004011FC	45
0x00000000004011F0	45
0x00000000004011F4	45
0x00000000004011F8	45
0x00000000004011FC	45
0x0000000000401200	45
0x0000000000401204	45
0x0000000000401208	45
0x000000000040120C	45
0x0000000000401210	45
0x0000000000401214	45
0x0000000000401218	45
0x000000000040121C	45
0x0000000000401220	45
0x0000000000401224	45
0x0000000000401228	45
0x000000000040122C	45
0x0000000000401230	45
0x0000000000401234	45
0x0000000000401238	45
0x000000000040123C	45
0x0000000000401240	45
0x0000000000401244	45
0x0000000000401248	45
0x000000000040124C	45
0x0000000000401250	45
0x0000000000401254	45
0x0000000000401258	45
0x000000000040125C	45
0x0000000000401260	45
0x0000000000401264	45
0x0000000000401268	45
0x000000000040126C	45
0x0000000000401270	45
0x0000000000401274	45
0x0000000000401278	45
0x000000000040127C	45
0x0000000000401280	45
0x0000000000401284	45
0x0000000000401288	45
0x000000000040128C	45
0x0000000000401290	45
0x0000000000401294	45
0x0000000000401298	45
0x000000000040129C	45
0x00000000004012A0	45
0x00000000004012A4	45
0x00000000004012A8	45
0x00000000004012AC	45
0x00000000004012B0	45
0x00000000004012B4	45
0x00000000004012B8	45
0x00000000004012BC	45
0x00000000004012C0	45
0x00000000004012C4	45
0x00000000004012C8	45
0x00000000004012CC	45
0x00000000004012D0	45
0x00000000004012D4	45
0x00000000004012D8	45
0x00000000004012DC	45
0x00000000004012E0	45
0x00000000004012E4	45
0x00000000004012E8	45
0x00000000004012EC	45
0x00000000004012F0	45
0x00000000004012F4	45
0x00000000004012F8	45
0x00000000004012FC	45
0x0000000000401300	45
0x0000000000401304	45
0x0000000000401308	45
0x000000000040130C	45
0x0000000000401310	45
0x0000000000401314	45
0x0000000000401318	45
0x000000000040131C	45
0x0000000000401320	45
0x0000000000401324	45
0x0000000000401328	45
0x000000000040132C	45
0x0000000000401330	45
0x0000000000401334	45
0x0000000000401338	45
0x000000000040133C	45
0x0000000000401340	45
0x0000000000401344	45
0x0000000000401348	45
0x000000000040134C	45
0x0000000000401350	45
0x0000000000401354	45
0x0000000000401358	45
0x000000000040135C	45
0x0000000000401360	45
0x0000000000401364	45
0x0000000000401368	45
0x000000000040136C	45
0x0000000000401370	45
0x0000000000401374	45
0x0000000000401378	45
0x000000000040137C	45
0x0000000000401380	45
0x0000000000401384	45
0x0000000000401388	45
0x000000000040138C	45
0x0000000000401390	45
0x0000000000401394	45
0x0000000000401398	45
0x000000000040139C	45
0x00000000004013A0	45
0x00000000004013A4	45
0x00000000004013A8	45
0x00000000004013AC	45
0x00000000004013B0	45
0x00000000004013B4	45
0x00000000004013B8	45
0x00000000004013BC	45
0x00000000004013C0	45
0x00000000004013C4	45
0x00000000004013C8	45
0x00000000004013CC	45
0x00000000004013D0	45
0x00000000004013D4	45
0x00000000004013D8	45
0x00000000004013DC	45
0x00000000004013E0	45
0x00000000004013E4	45
0x00000000004013E8	45
0x00000000004013EC	45
0x00000000004013F0	45
0x00000000004013F4	45
0x00000000004013F8	45
0x00000000004013FC	45
0x0000000000401400	45
0x0000000000401404	45
0x0000000000401408	45
0x000000000040140C	45
0x0000000000401410	45
0x0000000000401414	45
0x0000000000401418	45
0x000000000040141C	45
0x0000000000401420	45
0x0000000000401424	45
0x0000000000401428	45
0x000000000040142C	45
0x0000000000401430	45
0x0000000000401434	45
0x0000000000401438	45
0x000000000040143C	45
0x0000000000401440	45
0x0000000000401444	45
0x0000000000401448	45
0x000000000040144C	45
0x0000000000401450	45
0x0000000000401454	45
0x0000000000401458	45
0x000000000040145C	45
0x0000000000401460	45
0x0000000000401464	45
0x0000000000401468	45
0x000000000040146C	45
0x0000000000401470	45
0x0000000000401474	45
0x0000000000401478	45
0x000000000040147C	45
0x0000000000401480	45
0x0000000000401484	45
0x0000000000401488	45
0x000000000040148C	45
0x0000000000401490	45
0x0000000000401494	45
0x0000000000401498	45
0x000000000040149C	45
0x00000000004014A0	45
0x00000000004014A4	45
0x00000000004014A8	45
0x00000000004014AC	45
0x00000000004014B0	45
0x00000000004014B4	45
0x00000000004014B8	45
0x00000000004014BC	45
0x00000000004014C0	45
0x00000000004014C4	45
0x00000000004014C8	45
0x00000000004014CC	45
0x00000000004014D0	45
0x00000000004014D4	45
0x00000000004014D8	45
0x00000000004014DC	45
0x00000000004014E0	45
0x00000000004014E4	45
0x00000000004014E8	45
0x00000000004014EC	45
0x00000000004014F0	45
0x00000000004014F4	45
0x00000000004014F8	45
0x00000000004014FC	45
0x0000000000401500	45
0x0000000000401504	45
0x0000000000401508	45
0x000000000040150C	45
0x0000000000401510	45
0x0000000000401514	45
0x0000000000401518	45
0x000000000040151C	45
0x0000000000401520	45
0x0000000000401524	45
0x0000000000401528	45
0x000000000040152C	45
0x0000000000401530	45
0x0000000000401534	45
0x0000000000401538	45
0x000000000040153C	45
0x0000000000401540	45
0x0000000000401544	45
0x0000000000401548	45
0x000000000040154C	45
0x0000000000401550	45
0x0000000000401554	45
0x0000000000401558	45
0x000000000040155C	45
0x0000000000401560	45
0x0000	