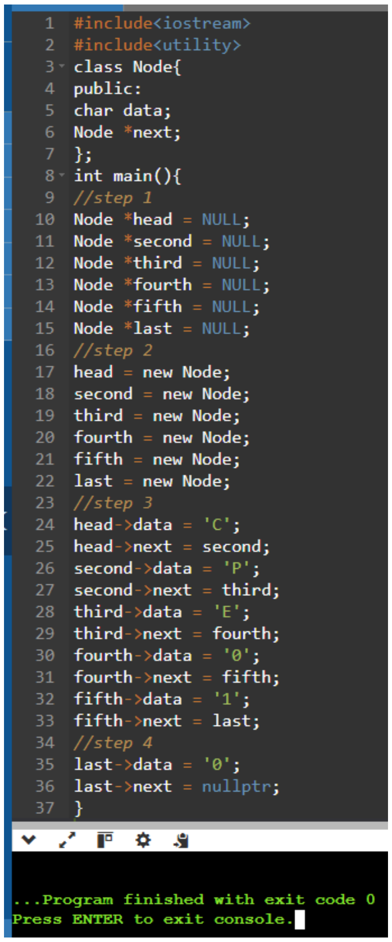
Activity No. 1	
Hands-on Activity 3.1 Linked Lists	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** September 27, 2024
Section: CPE21S4	**Date Submitted:** September 27, 2024
Name(s): Don Eleazar T. Fernandez	**Instructor:** Maria Rizette Sayo
6. Output	

Screenshot	
Discussion	The program resulted in no output as there was no character output implemented within the program. What could be done to improve this is to output each of the linked list data. A part to integrate within the program to achieve it is: while (temp != NULL) { cout << temp -> data << " "; temp = temp -> next; }.

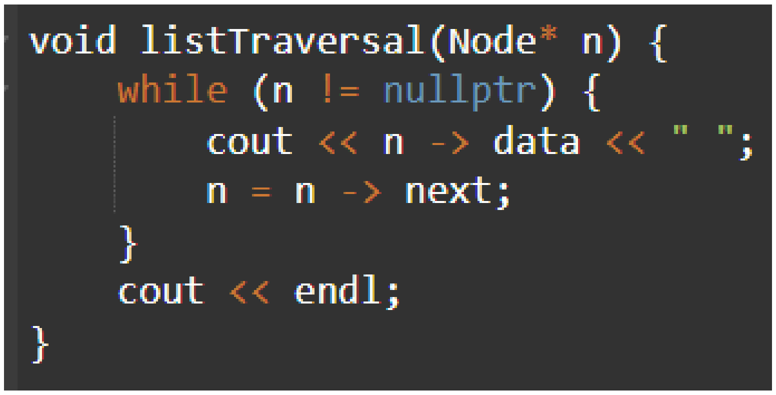
```
#include<iostream>
#include<utility>
class Node{
public:
char data;
Node *next;
};
int main(){
//step 1
Node *head = NULL;
Node *second = NULL;
Node *third = NULL;
Node *fourth = NULL;
Node *fifth = NULL;
Node *last = NULL;
//step 2
head = new Node;
second = new Node;
third = new Node;
fourth = new 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;
}
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

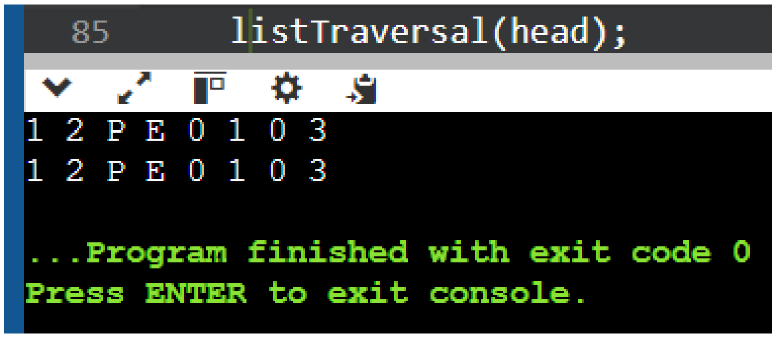
Table 3 - 1. Output of Initial/Simple Implementation

Operation	**Screenshot**

Transversal



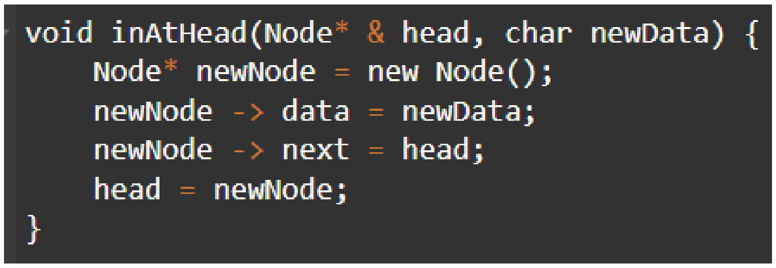
```
void listTraversal(Node* n) {
    while (n != nullptr) {
        cout << n -> data << " ";
        n = n -> next;
    }
    cout << endl;
}
```



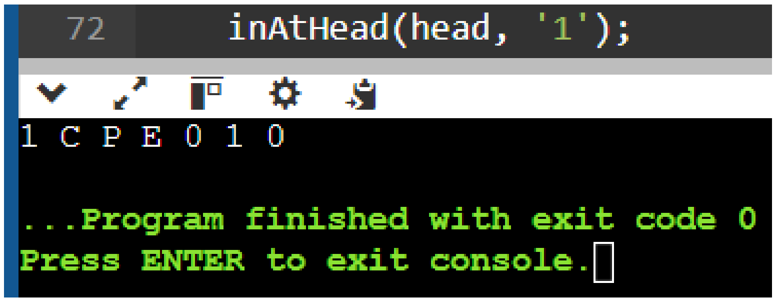
```
        listTraversal(head);
```

```
1 2 P E 0 1 0 3
1 2 P E 0 1 0 3

...Program finished with exit code 0
Press ENTER to exit console.
```

Insertion at head



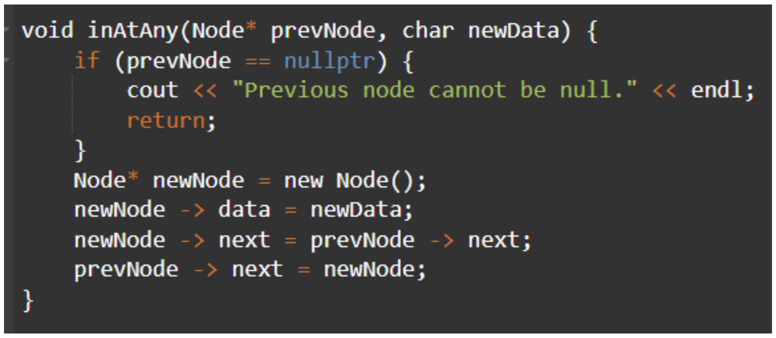
```
void inAtHead(Node* & head, char newData) {
    Node* newNode = new Node();
    newNode -> data = newData;
    newNode -> next = head;
    head = newNode;
}
```



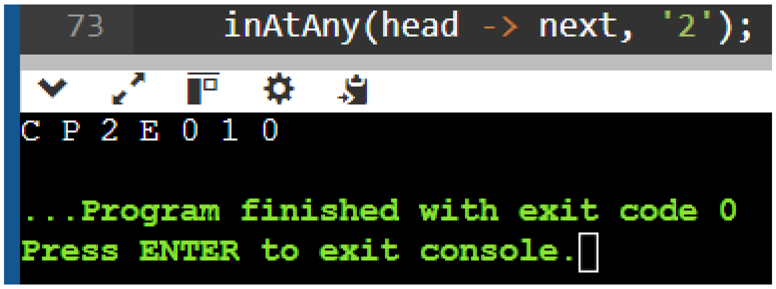
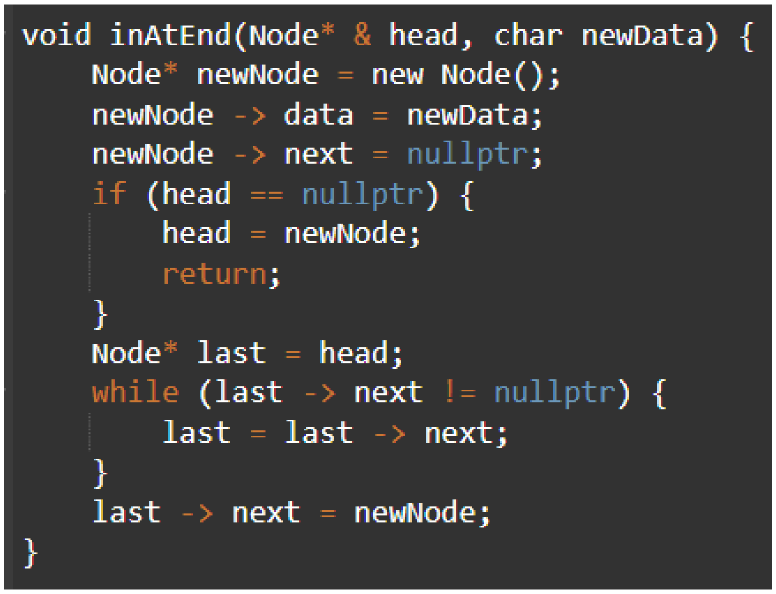
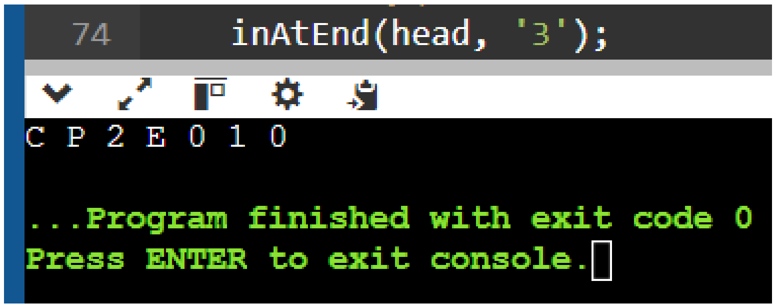
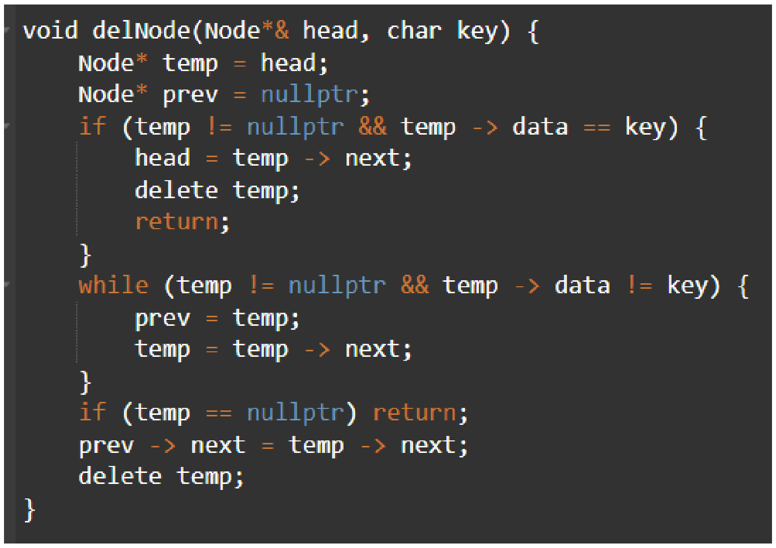
```
        inAtHead(head, '1');
```

```
1 C P E 0 1 0

...Program finished with exit code 0
Press ENTER to exit console.
```

Insertion at any part of the list



```
void inAtAny(Node* prevNode, char newData) {
    if (prevNode == nullptr) {
        cout << "Previous node cannot be null." << endl;
        return;
    }
    Node* newNode = new Node();
    newNode -> data = newData;
    newNode -> next = prevNode -> next;
    prevNode -> next = newNode;
}
```

	
Insertion at the end	 
Deletion of a node	

```
        inAtAny(head -> next, '2');
```

```
C P 2 E 0 1 0

...Program finished with exit code 0
Press ENTER to exit console.
```

```cpp
void inAtEnd(Node* & head, char newData) {
    Node* newNode = new Node();
    newNode -> data = newData;
    newNode -> next = nullptr;
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* last = head;
    while (last -> next != nullptr) {
        last = last -> next;
    }
    last -> next = newNode;
}
```

```
        inAtEnd(head, '3');
```

```
C P 2 E 0 1 0

...Program finished with exit code 0
Press ENTER to exit console.
```

```cpp
void delNode(Node*& head, char key) {
    Node* temp = head;
    Node* prev = nullptr;
    if (temp != nullptr && temp -> data == key) {
        head = temp -> next;
        delete temp;
        return;
    }
    while (temp != nullptr && temp -> data != key) {
        prev = temp;
        temp = temp -> next;
    }
    if (temp == nullptr) return;
    prev -> next = temp -> next;
    delete temp;
}
```

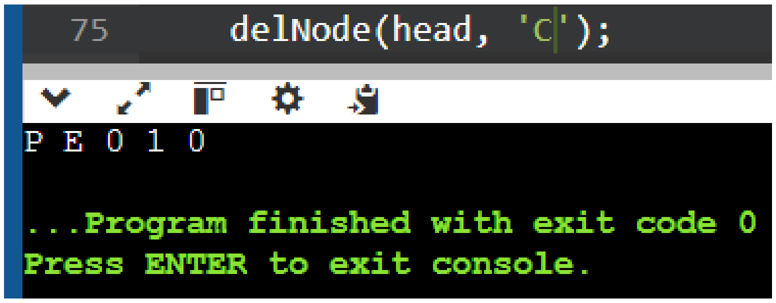
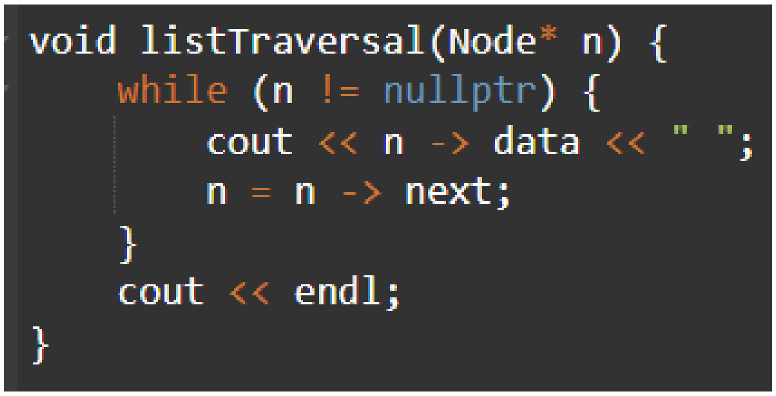
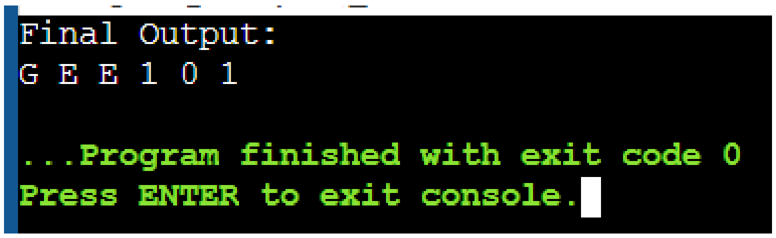
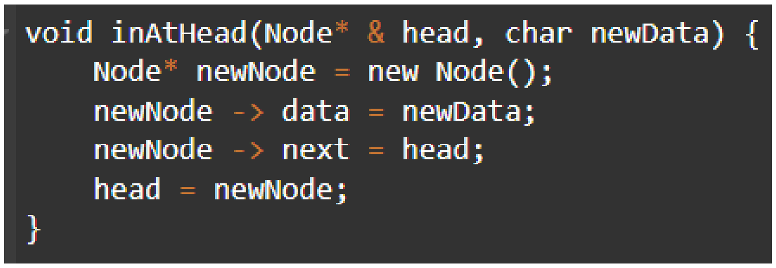
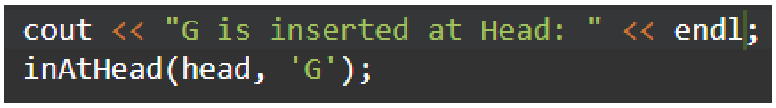
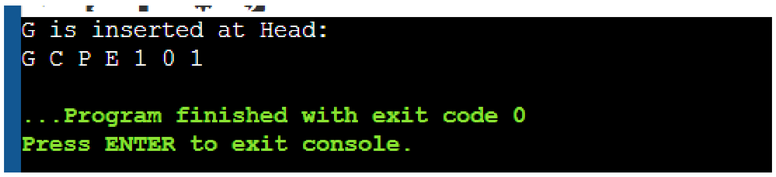
	 `delNode(head, 'C');`
	`P E 0 1 0`
	
	`...Program finished with exit code 0`
	`Press ENTER to exit console.`

Table 3 - 2. Code for the List Operations

a.	**Source Code**	 `void listTraversal(Node* n) {`
		`    while (n != nullptr) {`
		`        cout << n -> data << " ";`
		`        n = n -> next;`
		`    }`
		`    cout << endl;`
		`}`

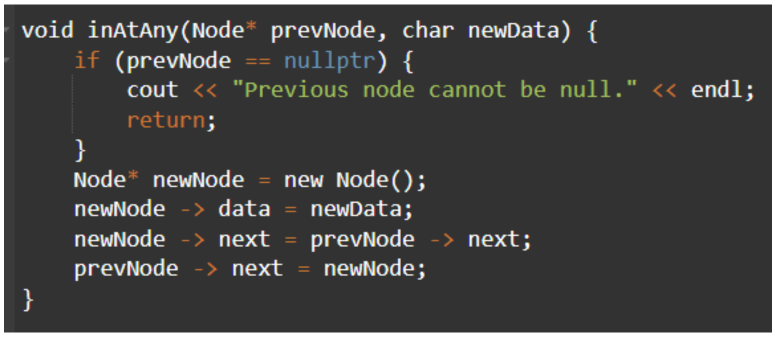
	Console	 `Final Output:`
		`G E E 1 0 1`
		
		`...Program finished with exit code 0`
		`Press ENTER to exit console.`

b.	**Source Code**	 `void inAtHead(Node* & head, char newData) {`
		`    Node* newNode = new Node();`
		`    newNode -> data = newData;`
		`    newNode -> next = head;`
		`    head = newNode;`
		`}`
		 `cout << "G is inserted at Head: " << endl;`
		`inAtHead(head, 'G');`

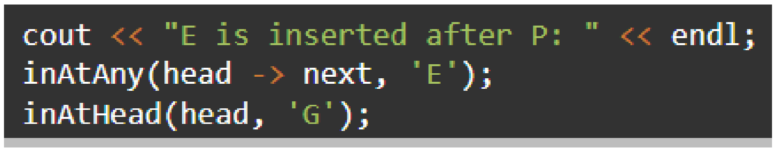
	Console	 `G is inserted at Head:`
		`G C P E 1 0 1`
		
		`...Program finished with exit code 0`
		`Press ENTER to exit console.`

c.

Source Code

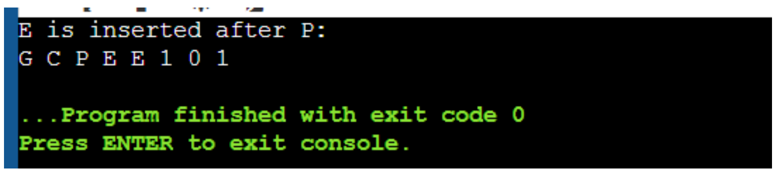


```
void inAtAny(Node* prevNode, char newData) {
    if (prevNode == nullptr) {
        cout << "Previous node cannot be null." << endl;
        return;
    }
    Node* newNode = new Node();
    newNode -> data = newData;
    newNode -> next = prevNode -> next;
    prevNode -> next = newNode;
}
```



```
cout << "E is inserted after P: " << endl;
inAtAny(head -> next, 'E');
inAtHead(head, 'G');
```

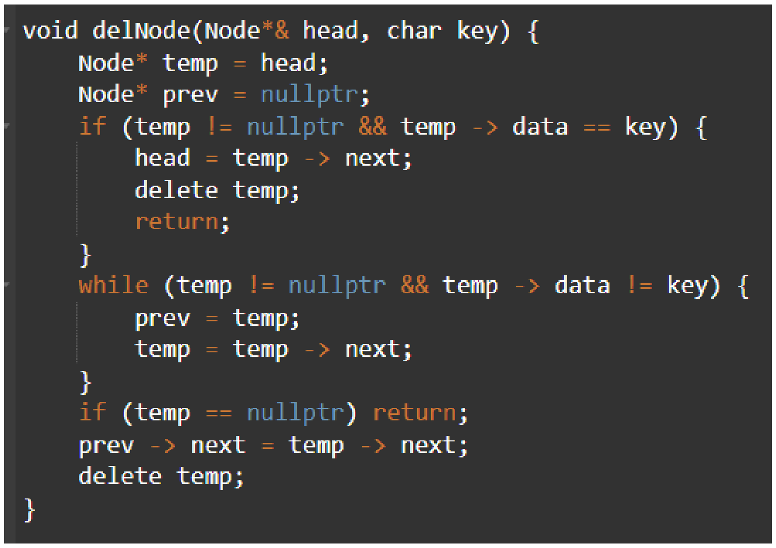
Console



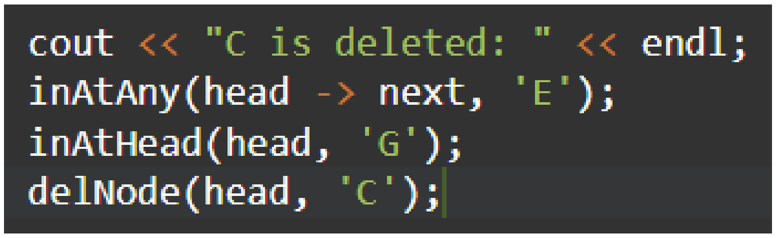
```
E is inserted after P:
G C P E E 1 0 1

...Program finished with exit code 0
Press ENTER to exit console.
```

d.

Source Code

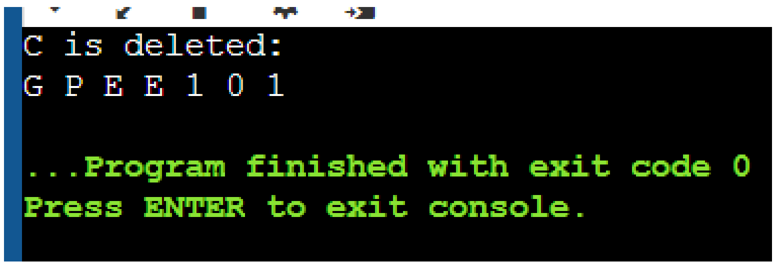


```
void delNode(Node*& head, char key) {
    Node* temp = head;
    Node* prev = nullptr;
    if (temp != nullptr && temp -> data == key) {
        head = temp -> next;
        delete temp;
        return;
    }
    while (temp != nullptr && temp -> data != key) {
        prev = temp;
        temp = temp -> next;
    }
    if (temp == nullptr) return;
    prev -> next = temp -> next;
    delete temp;
}
```



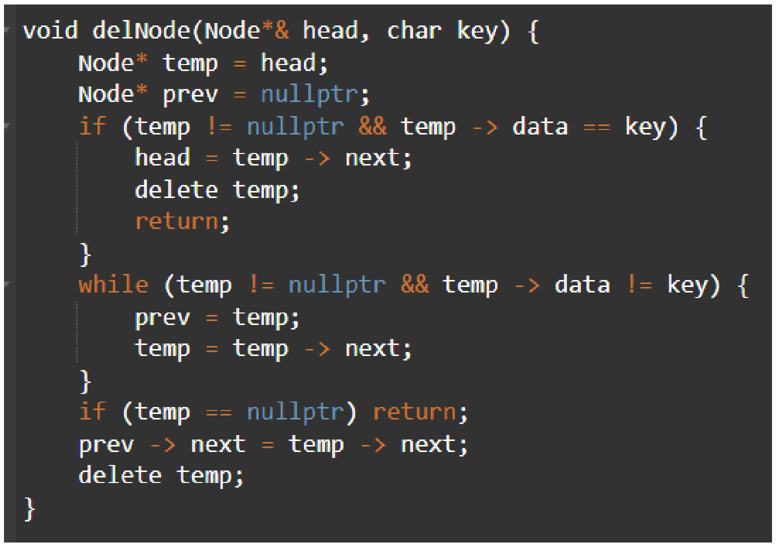
```
cout << "C is deleted: " << endl;
inAtAny(head -> next, 'E');
inAtHead(head, 'G');
delNode(head, 'C');
```

Console

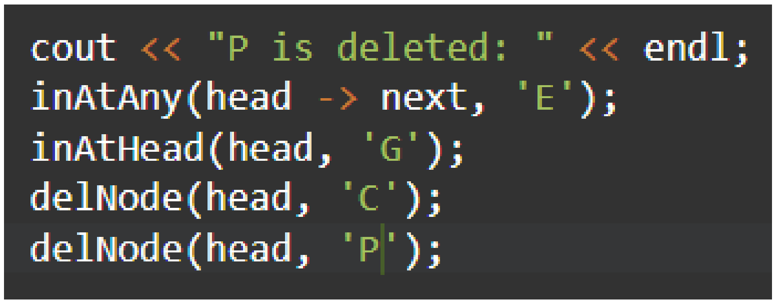


```
C is deleted:
G P E E 1 0 1

...Program finished with exit code 0
Press ENTER to exit console.
```

e. **Source Code**

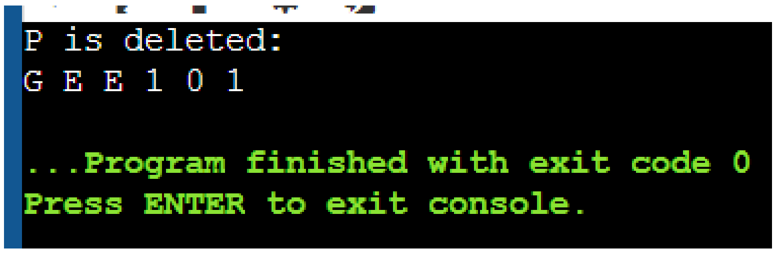


```
void delNode(Node*& head, char key) {
    Node* temp = head;
    Node* prev = nullptr;
    if (temp != nullptr && temp -> data == key) {
        head = temp -> next;
        delete temp;
        return;
    }
    while (temp != nullptr && temp -> data != key) {
        prev = temp;
        temp = temp -> next;
    }
    if (temp == nullptr) return;
    prev -> next = temp -> next;
    delete temp;
}
```



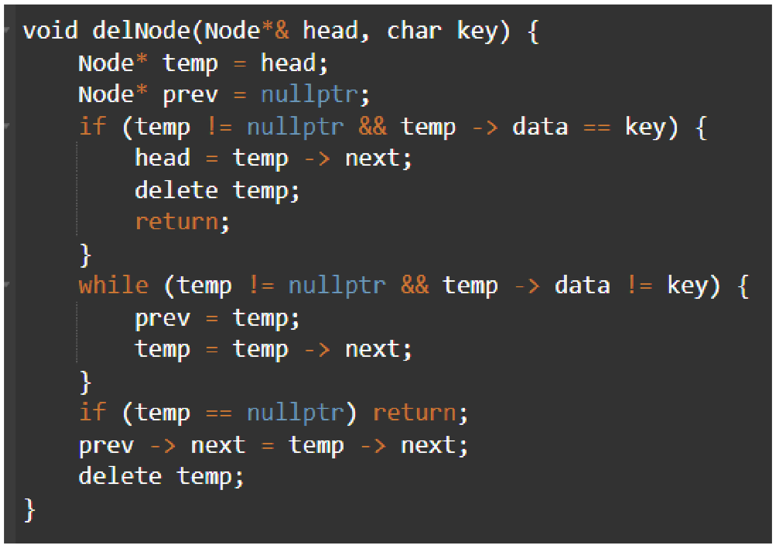
```
cout << "P is deleted: " << endl;
inAtAny(head -> next, 'E');
inAtHead(head, 'G');
delNode(head, 'C');
delNode(head, 'P');
```

Console



```
P is deleted:
G E E 1 0 1

...Program finished with exit code 0
Press ENTER to exit console.
```

f. **Source Code**



```
void delNode(Node*& head, char key) {
    Node* temp = head;
    Node* prev = nullptr;
    if (temp != nullptr && temp -> data == key) {
        head = temp -> next;
        delete temp;
        return;
    }
    while (temp != nullptr && temp -> data != key) {
        prev = temp;
        temp = temp -> next;
    }
    if (temp == nullptr) return;
    prev -> next = temp -> next;
    delete temp;
}
```

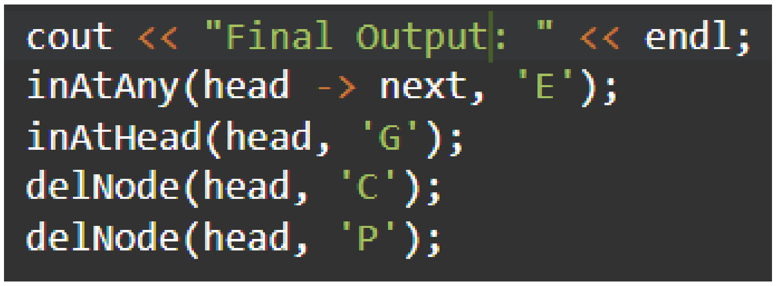
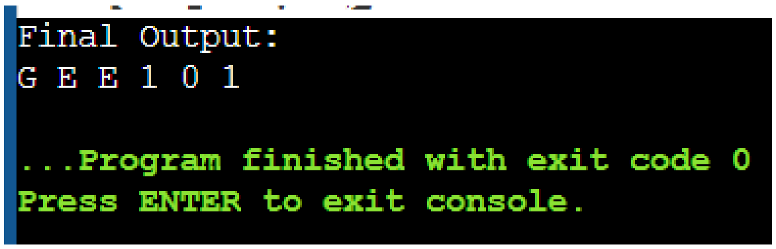
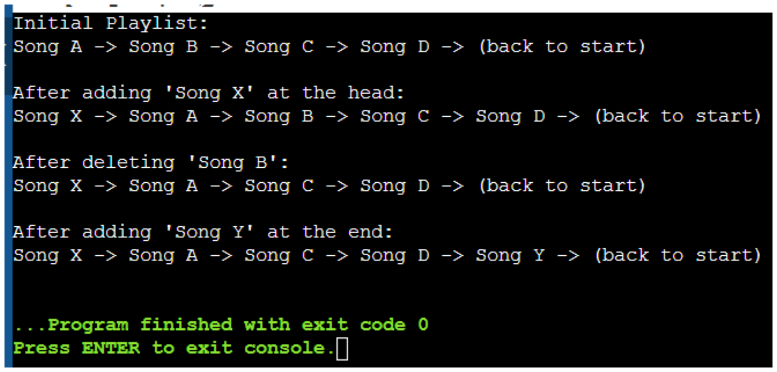
		 ```cout << "Final Output: " << endl;``` ```inAtAny(head -> next, 'E');``` ```inAtHead(head, 'G');``` ```delNode(head, 'C');``` ```delNode(head, 'P');```
	Console	 ```Final Output:``` ```G E E 1 0 1``` ```...Program finished with exit code 0``` ```Press ENTER to exit console.```

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

7. Supplementary Activity

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

class Node {
public:
   string songName;
   Node *next;
};

void listTraversal(Node* head) {
   if (head == nullptr) {
      cout << "Playlist is empty!" << endl;
      return;
   }
   Node* temp = head;
   do {
      cout << temp -> songName << " -> ";
      temp = temp -> next;
   } while (temp != head);
   cout << "(back to start)" << endl;
}

void inAtHead(Node* &head, string newSong) {
   Node* newNode = new Node();
   newNode -> songName = newSong;
   if (head == nullptr) {
      newNode -> next = newNode;
      head = newNode;
   } else {
      Node* temp = head;
      while (temp -> next != head) {
         temp = temp -> next;
```

```
        }
        newNode -> next = head;
        temp -> next = newNode;
        head = newNode;
    }
}

void inAtEnd(Node* &head, string newSong) {
    Node* newNode = new Node();
    newNode -> songName = newSong;
    if (head == nullptr) {
        newNode -> next = newNode;
        head = newNode;
    } else {
        Node* temp = head;
        while (temp -> next != head) {
            temp = temp -> next;
        }
        temp -> next = newNode;
        newNode -> next = head;
    }
}

void delNode(Node*& head, string songName) {
    if (head == nullptr) return;

    Node *temp = head, *prev = nullptr;

    if (temp != nullptr && temp -> songName == songName) {
        if (temp -> next == head) {
            delete temp;
            head = nullptr;
            return;
        }

        while (temp -> next != head) {
            temp = temp-> next;
        }
        temp -> next = head -> next;
        delete head;
        head = temp -> next;
        return;
    }

    do {
        prev = temp;
        temp = temp -> next;
    } while (temp != head && temp -> songName != songName);

    if (temp == head) return;
    prev -> next = temp -> next;
    delete temp;
}

int main() {
    Node *head = nullptr;
    inAtEnd(head, "Song A");
    inAtEnd(head, "Song B");
    inAtEnd(head, "Song C");
    inAtEnd(head, "Song D");
    cout << "Initial Playlist: " << endl;
    listTraversal(head);

    inAtHead(head, "Song X");
    cout << "\nAfter adding 'Song X' at the head: " << endl;
    listTraversal(head);

    delNode(head, "Song B");
    cout << "\nAfter deleting 'Song B': " << endl;
    listTraversal(head);

    inAtEnd(head, "Song Y");
    cout << "\nAfter adding 'Song Y' at the end: " << endl;
    listTraversal(head);

    return 0;
}
```



```
Initial Playlist:
Song A -> Song B -> Song C -> Song D -> (back to start)

After adding 'Song X' at the head:
Song X -> Song A -> Song B -> Song C -> Song D -> (back to start)

After deleting 'Song B':
Song X -> Song A -> Song C -> Song D -> (back to start)

After adding 'Song Y' at the end:
Song X -> Song A -> Song C -> Song D -> Song Y -> (back to start)


...Program finished with exit code 0
Press ENTER to exit console.
```

8. Conclusion

This exercise improved our knowledge of linked lists by practicing insertion, deletion, and traversal. It also demonstrated the benefits of dynamic data structures over arrays, especially when making frequent changes. This made it clear that effective node management was necessary, and the circular playlist experiment demonstrated how flexible linked lists are when used for certain purposes. In my case, I think I did well enough on this activity. What I could do to improve is to try it all over again without a guide from the laboratory manual.

9. Assessment Rubric

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.