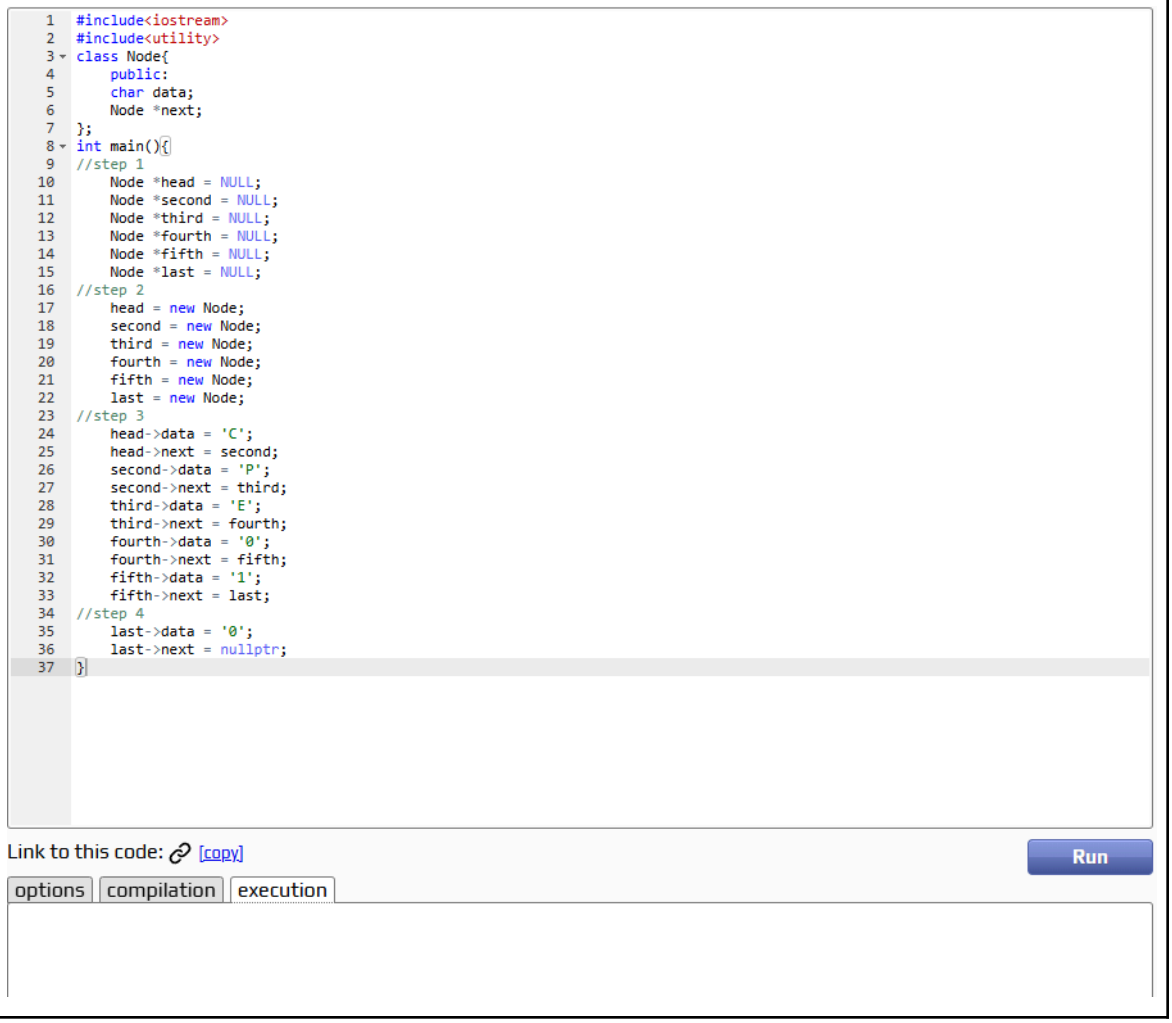
Activity No. 3	
Linked List	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 09/27/2024**
Section: CPE21S4	**Date Submitted:**
Name(s): Santos, Ma. Kassandra Nicole D	**Instructor: Ms. Rizette Sayo**
6. Output	

Screenshot (FIRST OUTPUT WITHOUT MODIFYING IT TO BE FURTHER IMPROVED)	

```cpp
#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;
}
```

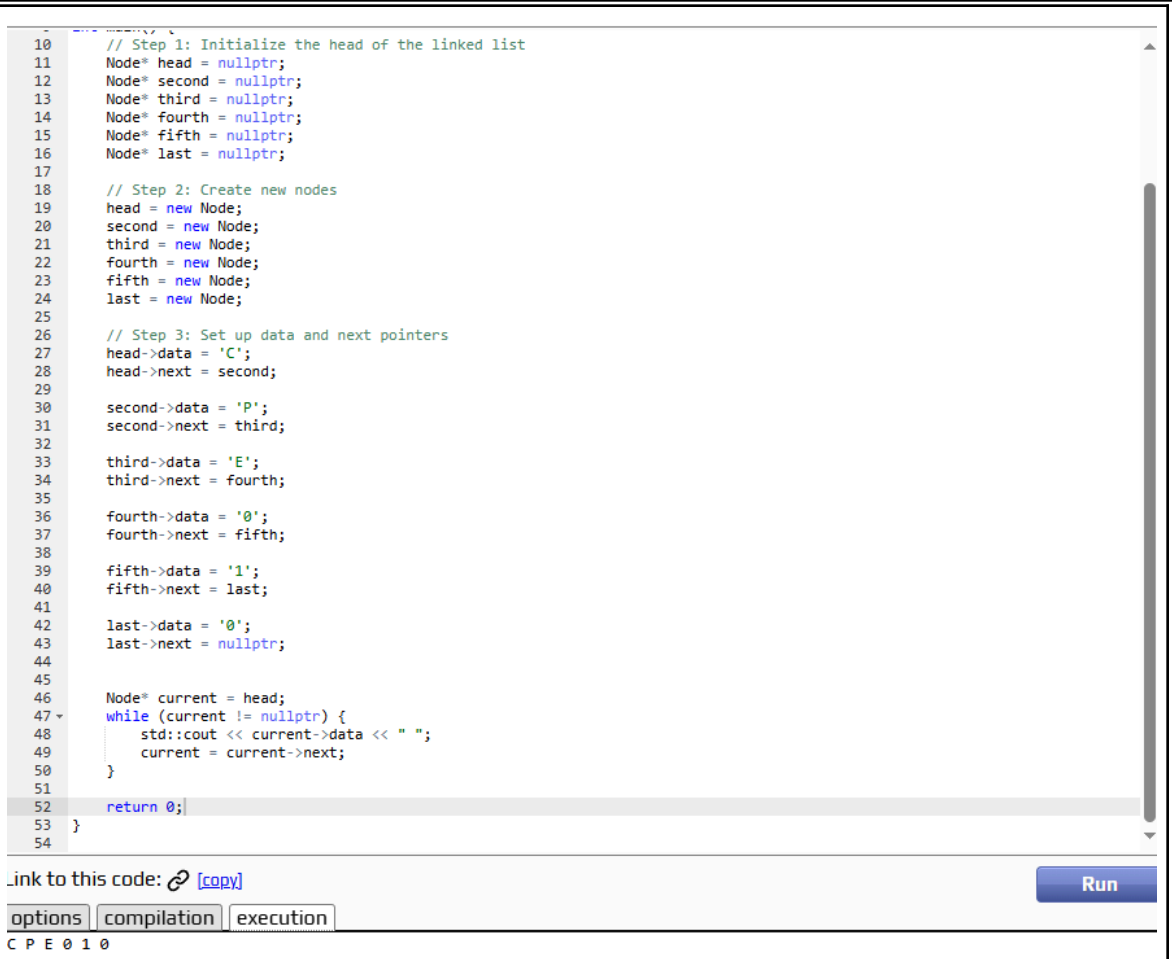
Screenshot (SECOND OUTPUT AFTER MODIFYING IT TO BE FURTHER IMPROVED)	
Discussion	In the first screenshot, we see that the program works just fine with no error, however, the supposed “CPE010” did not appear in the execution, this means that though there is nothing wrong in the program, there is indeed something missing in the program that would’ve been the component for the output to be displayed. To fix the issue, I have placed a character output (std cout) to display the CPE010, however, it would be tedious to manually display every input there for I have placed a while loop in order to print out every letter or number within CPE010

TABLE 3-1. Output of Initial/Simple Implementation

Operation	Procedure
Traversal	C/C++ `#include <iostream>` `struct Node {` `char data;` `Node* next;`

	`` }; void insertAtHead(Node*& head, char newData) { Node* newNode = new Node; newNode->data = newData; newNode->next = head; head = newNode; } void ListTraversal(Node* n) { while (n != nullptr) { std::cout << n->data << " "; n = n->next; } std::cout << std::endl; } int main() { Node* head = nullptr; insertAtHead(head, '0'); insertAtHead(head, '1'); insertAtHead(head, '0'); insertAtHead(head, 'E'); insertAtHead(head, 'P'); insertAtHead(head, 'C'); ListTraversal(head); return 0; } ```
Insertion at head	C/C++ ``` #include <iostream> struct Node { char data; Node* next; }; void insertAtHead(Node*& head, char newData) { Node* newNode = new Node; newNode->data = newData; newNode->next = head; head = newNode; } int main() { Node* head = nullptr; ```

	`` insertAtHead(head, '0'); insertAtHead(head, '1'); insertAtHead(head, '0'); insertAtHead(head, 'E'); insertAtHead(head, 'P'); insertAtHead(head, 'C'); Node* current = head; while (current != nullptr) { std::cout << current->data << " "; current = current->next; } return 0; } ```
Insertion at any part of the list	C/C++ ``` void insertAfter(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; newNode->prev = prevNode; if (newNode->next != nullptr) { newNode->next->prev = newNode; } ```
Insertion at the end	C/C++ ``` #include <iostream> struct Node { char data; Node* next; ```

```
};

void insertAtTail(Node*& head, char newData) {
    head = (head == nullptr) ? new Node{newData, nullptr} :
(insertAtTail(head->next, newData), head);
}

int main() {
    Node* head = nullptr;

    insertAtTail(head, 'C');
    insertAtTail(head, 'P');
    insertAtTail(head, 'E');
    insertAtTail(head, '0');
    insertAtTail(head, '1');
    insertAtTail(head, '0');

    for (Node* current = head; current != nullptr; current =
current->next) {
        std::cout << current->data << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Deletion of a code

C/C++

```
#include <iostream>

struct Node {
    char data;
    Node* next;
};

void insertAtTail(Node*& head, char newData) {
    if (head == nullptr) {
        head = new Node{newData, nullptr};
        return;
    }
    insertAtTail(head->next, newData);
}
Node* deleteNode(Node* head, char key) {
    if (head == nullptr) return nullptr;
    if (head->data == key) return head->next;
    head->next = deleteNode(head->next, key);
    return head;
}
```

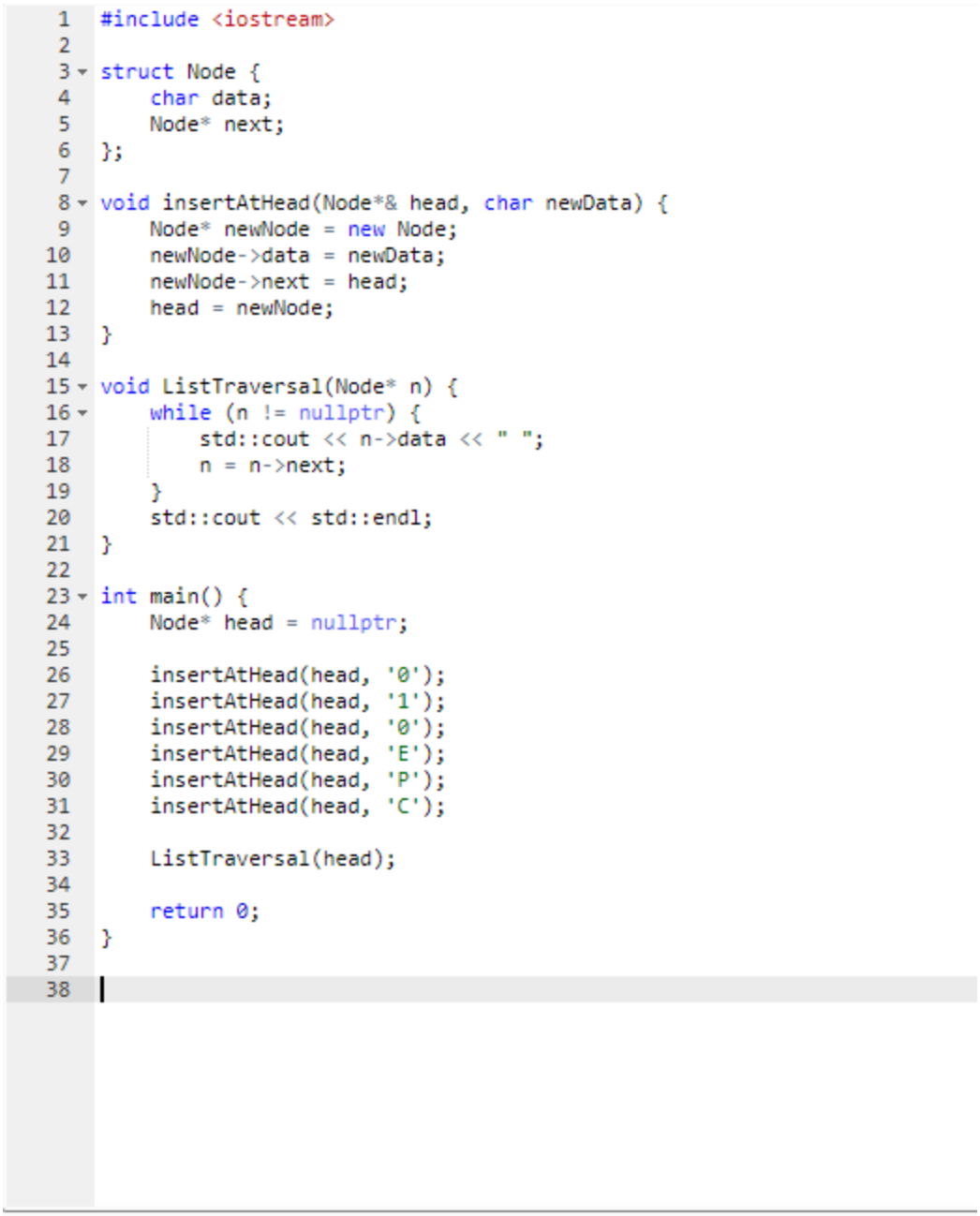
```
int main() {
    Node* head = nullptr;

    insertAtTail(head, 'C');
    insertAtTail(head, 'P');
    insertAtTail(head, 'E');
    insertAtTail(head, '0');
    insertAtTail(head, '1');
    insertAtTail(head, '0');

    head = deleteNode(head, 'E');

    std::cout << "List after deleting 'E' from CPE010: ";
    for (Node* current = head; current != nullptr; current =
current->next) {
        std::cout << current->data << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

a	**Souce Code**	 Link to this code: [copy] options compilation execution C P E 0 1 0
	Console	Using the traverse code in the first activity, this would be the output

```cpp
#include <iostream>

struct Node {
    char data;
    Node* next;
};

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

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

int main() {
    Node* head = nullptr;

    insertAtHead(head, '0');
    insertAtHead(head, '1');
    insertAtHead(head, '0');
    insertAtHead(head, 'E');
    insertAtHead(head, 'P');
    insertAtHead(head, 'C');

    ListTraversal(head);

    return 0;
}
```

b	**Souce Code**	

```cpp
#include <iostream>

struct Node {
    char data;
    Node* next;
};

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

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

int main() {
    Node* head = nullptr;

    insertAtHead(head, '0');
    insertAtHead(head, '1');
    insertAtHead(head, '0');
    insertAtHead(head, 'E');
    insertAtHead(head, 'P');
    insertAtHead(head, 'C');
    insertAtHead(head, 'G');

    ListTraversal(head);

    return 0;
}
```

Link to this code: [copy]

options | compilation | execution

```
G C P E 0 1 0
```

	Console	

options | compilation | execution

```
G C P E 0 1 0
```

C	**Souce Code**	

```cpp
#include <iostream>

struct Node {
    char data;
    Node* next;
};

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

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

int main() {
    Node* head = nullptr;

    insertAtHead(head, '0');
    insertAtHead(head, '1');
    insertAtHead(head, '0');
    insertAtHead(head, 'E');
    insertAtHead(head, 'E');
    insertAtHead(head, 'P');
    insertAtHead(head, 'C');
    insertAtHead(head, 'G');

    ListTraversal(head);

    return 0;
}
```

Link to this code: [copy]

options | compilation | execution

```
G C P E E 0 1 0
```

Console

Link to this code: [copy]

options | compilation | execution

```
G C P E E 0 1 0
```

<table>
<tr><td rowspan="2">d</td><td>Souce Code</td><td>

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

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

void insertAtHEad(Node*& head, char newData) {
    if (head == nullptr) {
        head = new Node{newData, nullptr};
        return;
    }
    insertAtHead(head->next, newData);
}
Node* deleteNode(Node* head, char key) {
    if (head == nullptr) return nullptr;
    if (head->data == key) return head->next;
    head->next = deleteNode(head->next, key);
    return head;
}


int main() {
    Node* head = nullptr;

    insertAtHead(head, '0');
    insertAtHead(head, '1');
    insertAtHead(head, '0');
    insertAtHead(head, 'E');
    insertAtHead(head, 'E');
    insertAtHead(head, 'P');
    insertAtHead(head, 'C');
    insertAtHead(head, 'G');

    head = deleteNode(head, 'C');
    ListTraversal(head);

    return 0;
}
```

Link to this code: [copy] Run

options | compilation | execution

```
G P E E 0 1 0

Normal program termination. Exit status: 0
```

</td></tr>
<tr><td>Console</td><td>Through the deletion method in the prior activity, I managed to remove the C and P in the final output</td></tr>
<tr><td rowspan="2">e</td><td>Souce Code</td><td>options | compilation
G E E 0 1 0</td></tr>
<tr><td>Console</td><td>Through the deletion method in the prior activity, I managed to remove the C and P in the final output</td></tr>
</table>

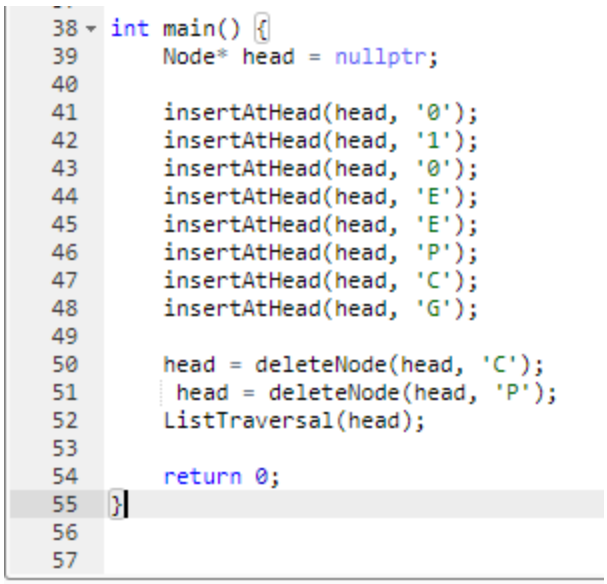
f	**Souce Code**	
	Console	

Table 3.3 singly linked list

CODE (Screenshot) *technically its not a screenshot since it wont fit or it is not very clear	ANALYSIS
SIMPLE CODE WITH DOUBLE LINK IMPLEMENTED C/C++ `#include <iostream>` `using namespace std;` `struct Node {` `char data;` `Node* next;` `Node* prev;` `};` `int main(){` `Node *head;` `Node *one = nullptr;` `Node *two = nullptr;` `Node *three = nullptr;` `Node *four = nullptr;` `Node *five = nullptr;`	In this code, the nodes are now connected from both the previous and next node which can be seen in the int main ex. `one->next = two;` `one->prev = nullptr;` `two->next = three;` `two->prev = one;` since the first starts with no input, there will be no previous value that it needs to be connected to but it can connect to “two”. The code is like a domino effect, the following nodes will connect to one another until the last one will have nothing to connect to ex.

```
    Node *six = nullptr;


    one = new Node();
    two = new Node();
    three = new Node();
    four = new Node();
    five = new Node();
    six = new Node();


    one->data = 'C';
    two->data = 'P';
    three->data = 'E';
    four->data = '0';
    five->data = '1';
    six->data = '0';


    one->next = two;
    one->prev = nullptr;

    two->next = three;
    two->prev = one;

    three->next = four;
    three->prev = two;

    four->next = five;
    four->prev = three;

    five->next = six;
    five->prev = four;

    six->next = nullptr;
    six->prev = five;

    head = one;


    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }

    delete one;
    delete two;
    delete three;
    delete four;
    delete five;
    delete six;

    return 0;
}
```

```
six->next = nullptr;
    six->prev = five;

    head = one;
```

INSERTING NODE ANYWHERE

C/C++

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

struct Node {
    char data;
    Node* next;
    Node* prev;
};

// Function to insert a new node after a given
previous node
void insertAfter(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;
    newNode->prev = prevNode;

    if (newNode->next != nullptr) {
        newNode->next->prev = newNode;
    }
}

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

int main() {
    Node *head;
    Node *one = nullptr;
    Node *two = nullptr;
    Node *three = nullptr;
    Node *four = nullptr;
    Node *five = nullptr;
    Node *six = nullptr;

    one = new Node();
    two = new Node();
    three = new Node();
    four = new Node();
    five = new Node();
    six = new Node();

    one->data = 'C';
```

In this code, I made the code in a way where you wll be the one who will add the node in anywhere you want not in a way where the system will be the one who will add it.

ex.

```
insertAfter(two, 'E');
```

the two will serve where the location of the E will be, it can be changed depending on which position you want it to be in

```
    two->data = 'P';
    three->data = 'E';
    four->data = '0';
    five->data = '1';
    six->data = '0';

    // Connect nodes
    one->next = two;
    one->prev = nullptr;

    two->next = three;
    two->prev = one;

    three->next = four;
    three->prev = two;

    four->next = five;
    four->prev = three;

    five->next = six;
    five->prev = four;

    six->next = nullptr;
    six->prev = five;

    head = one;


    cout << "Before insertion:" << endl;
    printList(head);


    insertAfter(two, 'E');


    cout << "After inserting 'E' after the second
node:" << endl;
    printList(head);


    delete one;
    delete two;
    delete three;
    delete four;
    delete five;
    delete six;

    return 0;
}
```

Adding NODE at the END

```
C/C++
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = new Node;
    newNode->data = data;
    newNode->next = NULL;

    struct Node* temp = *head;
    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        return;
    }

    while (temp->next != NULL)
        temp = temp->next;
   >next = newNode;


    newNode->prev = temp;
}

// I have removed the int main () because it was
getting long but the int main is still the same
from the prior
```

In this code, I added inserting and then connected the previous last node to the new tail. as seen here:

```
if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        return;
    }
```

DELETION

```
C/C++
void deleteNode(Node** head, Node* delNode) {
    if (*head == nullptr || delNode == nullptr) {
        cout << "Node to be deleted cannot be
null." << endl;
        return;
    }
    if (*head == delNode) {
        *head = delNode->next;
    }

    if (delNode->next != nullptr) {
        delNode->next->prev = delNode->prev;
    }
    if (delNode->prev != nullptr) {
        delNode->prev->next = delNode->next;

    delete delNode;
}

// I have removed the int main () because it was
getting long but the int main is still the same
from the prior
```

The deletion function was implemented in a way that the pointers of the previous and next nodes will be able to free the memory (deleting) of the node that is why there commands such as:

```
delNode->next->prev = delNode->prev;
 *head = delNode->next;
```


7. Supplementary Activity

incomplete work 😕 sorry Ma'am Sayo I had a hard time doing this haha </3 I tried my best, I'll do better next time nalang po ma'am sorry po 😭

```C/C++
#include <iostream>
using namespace std;

struct Node {
    string songName;
    Node* next;
};


class Playlist {
private:
    Node* last;
public:
    Playlist() {
        last = nullptr;
    }


    void addSong(string song) {
        Node* newNode = new Node();
        newNode->songName = song;


        if (last == nullptr) {
            newNode->next = newNode;
            last = newNode;
        } else {
            newNode->next = last->next;
            last->next = newNode;
            last = newNode;
        }
    }

    void removeSong(string song) {
        if (last == nullptr) {
            cout << "Playlist is empty.\n";
            return;
        }
```

8. Conclusion

In conclusion, we how the behavior of a linked list varies depending on the desired output you need. In this lesson, we talked about how a linked list operates by creating a variety of C++ codes, we learned how to insert values in different

ways such as placing the value at the head, tail, or insertion at any point, we also learned how to drop a node through deleting it. In this activity, I think that I did a good job at making the codes, I would know if I did a good job if I could easily understand the procedure which I did. However, the areas that I would need to improve would be enhancing my ability to analyze and easily solve errors because I noticed how it takes me a long time to be able to fix a code even if the code is simple
9. Assessment Rubric