Hands-on Activity 3.1	
Linked Lists	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:8/14/25**
Section:CPE21S4	**Date Submitted:8/14/25**
Name(s):Crishen Luper S. Pulgado	**Instructor: Jimlord Quejado**

Output

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

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

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

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

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

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

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

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

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

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

    std::cout << head->data;
    std::cout << second->data;
    std::cout << third->data;
    std::cout << fourth->data;
    std::cout << fifth->data;
    std::cout << last->data;
    std::cout << std::endl;
```

```
STDIN
Input for th

Output:
CPE010
```

Discussion:

The output came by simply pointing the node to the next node. First is we determining a class that functions as the initializing the data type and pointing it to the next node. Then we put null to each node so that we can put a data then we assign a new node for each. Lastly, we determine the data in each node and point it to the next new node. To improve the output, we could use a doubly in which we can traverse from head to tail or tail to head.

Table 3-2:

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

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

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

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

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


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

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

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

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

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

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

STDIN

Input for the pr

Output:

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

```cpp
        std::cout << n->data << " ";
        n = n->next;
    }
    std::cout << std::endl;
}

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

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


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

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

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

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

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

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

    Node *newNode = new Node();
    newNode->data = 'G';
    newNode->next = head;
    head = newNode;

    ListTraversal(head);

    return 0;
}
```

STDIN

Input for the p

Output:

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

```cpp

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

    Node *newNode = new Node();
    newNode->data = 'G';
    newNode->next = head;
    head = newNode;

    Node* previous = head;
    while (previous != nullptr && previous->data != 'P') {
        previous = previous->next;
    }

    if (previous == nullptr) {
        std::cout << "Previous node cannot be null." << std::endl;
    } else {
        Node* newNodeE = new Node();
        newNodeE->data = 'E';
        newNodeE->next = previous->next;
        previous->next = newNodeE;
    }


    Node* prev = nullptr;
    Node* curr = head;
    while (curr != nullptr && curr->data != 'C') {
        prev = curr;
        curr = curr->next;
    }
    if (curr != nullptr) {
        if (prev == nullptr) {
            head = curr->next;
        } else {
            prev->next = curr->next;
        }
        delete curr;
    }

    prev = nullptr;
    curr = head;
    while (curr != nullptr && curr->data != 'P') {
        prev = curr;
        curr = curr->next;
    }
    if (curr != nullptr) {
        if (prev == nullptr) {
            head = curr->next;
        } else {
            prev->next = curr->next;
        }
        delete curr;
```

STDIN

Input for the pr

Output:

```
G E E 1 0 1
```

Table 3-3

NewFile1.cpp +

43tnj8wen

```cpp
    Node *last = NULL;

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

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

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

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

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

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

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

    cout << "Forward traversal: ";
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;

    cout << "Backward traversal: ";
    temp = last;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->previous;
    }
    cout << endl;
```

STDIN

Input for the program (Optional)

Output:

```
Forward traversal: C P E 0 1 0
Backward traversal: 0 1 0 E P C
```

Here, we add a pointer to the previous node in the class so that it will have 2 addresses for next and the previous node. We also use while loop to print the linked list and stops if it encounters a pointer.
Supplementary:

main.cpp

Share Run

```cpp
#include <iostream>
#include <bits/stdc++.h>

class Node {
public:
    std::string song;
    Node* next;
    Node* prev;
};

int main() {
    Node* song1 = new Node();
    Node* song2 = new Node();
    Node* song3 = new Node();

    song1->song = "Multo";
    song2->song = "Panaginip";
    song3->song = "Ligaya";

    song1->next = song2;
    song2->next = song3;
    song3->next = song1;

    song1->prev = song3;
    song2->prev = song1;
    song3->prev = song2;

    Node* head = song1;
    Node* current = song1;

    int choice;
    std::string delSong;

    while (true) {
        std::cout << "\nPlaylist Menu:\n";
        std::cout << "1. Play Current Song\n";
        std::cout << "2. Next Song\n";
        std::cout << "3. Previous Song\n";
        std::cout << "4. Remove Song\n";
        std::cout << "5. Exit\n";
        std::cout << "Enter choice: ";
        std::cin >> choice;
        std::cin.ignore();

        if (!head && choice != 5) {
            std::cout << "Playlist is empty.\n";
            continue;
        }

        if (choice == 1) {
            std::cout << "Now playing: " << current->song << std::endl;
        }
        else if (choice == 2) {
            current = current->next;
            std::cout << "Now playing: " << current->song << std::endl;
        }
        else if (choice == 3) {
            current = current->prev;
            std::cout << "Now playing: " << current->song << std::endl;
        }
        else if (choice == 4) {
            std::cout << "Enter song title to remove: ";
            std::getline(std::cin, delSong);
```

Output

```
Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 1
Now playing: Multo

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 2
Now playing: Panaginip

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 3
Now playing: Multo

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 4
Enter song title to remove: Ligaya
```

```
    else if (choice == 2) {
        current = current->next;
        std::cout << "Now playing: " << current->song << std::endl;
    }
    else if (choice == 3) {
        current = current->prev;
        std::cout << "Now playing: " << current->song << std::endl;
    }
    else if (choice == 4) {
        std::cout << "Enter song title to remove: ";
        std::getline(std::cin, delSong);

        Node* temp = head;
        bool found = false;

        do {
            if (temp->song == delSong) {
                if (temp->next == temp) {
                    delete temp;
                    head = nullptr;
                    current = nullptr;
                } else {
                    if (temp == head) head = head->next;
                    temp->prev->next = temp->next;
                    temp->next->prev = temp->prev;
                    if (temp == current) current = temp->next;
                    delete temp;
                }
                std::cout << "Removed: " << delSong << std::endl;
                found = true;
                break;
            }
            temp = temp->next;
        } while (temp != head && head != nullptr);
```

```
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 2
Now playing: Panaginip

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 3
Now playing: Multo

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 4
Enter song title to remove: Ligaya
Removed: Ligaya

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 5
Exiting...
```

```
                temp->next->prev = temp->prev;
                if (temp == current) current = temp->next;
                delete temp;
            }
            std::cout << "Removed: " << delSong << std::endl;
            found = true;
            break;
        }
        temp = temp->next;
    } while (temp != head && head != nullptr);

    if (!found) std::cout << "Song not found.\n";
}
else if (choice == 5) {
    std::cout << "Exiting...\n";
    break;
}
else {
    std::cout << "Invalid choice.\n";
}
}


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

return 0;
}
```

```
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 2
Now playing: Panaginip

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 3
Now playing: Multo

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 4
Enter song title to remove: Liga
Removed: Ligaya

Playlist Menu:
1. Play Current Song
2. Next Song
3. Previous Song
4. Remove Song
5. Exit
Enter choice: 5
Exiting...
```

Conclusion:

To conclude, linked list is a dynamic allocation that has the feature of insertion of a new node and deleting a node. It has much more advantage than arrays since it can insert new nodes while arrays have fixed size. The procedure has demonstrated what a linked list can do and a linked has data and points to the next node. The importance of making linked list is that you should know the sequence of the node and where it is pointing next. In the supplementary, using a

loop and if and else statement is the key for making the options and implementing the adding of new nodes and deleting it. In this activity, I think I had a hard time in applying the linked list but I have grasped the concepts of the linked list well.