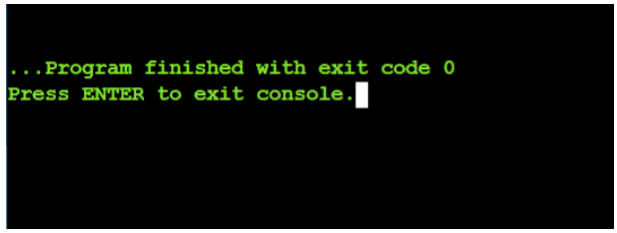
Activity No. <3.1>		
<Hands-on Activity 3.1 Linked Lists>		
Course Code: CPE010		**Program:** Computer Engineering
Course Title: Data Structures and Algorithms		**Date Performed: Sept 27,2024**
Section:CPE21S4		**Date Submitted:Sept 27, 2024**
Name(s): Kenn Jie Valleser		**Instructor: Ma. Rizette Sayo**

6. Output

Screenshot	

Discussion	The CPE010 was Inputted in each node which starts at head and ends with null. To improve the code i insert a loop for the code to display in the output :
	while(head!=NULL){
	cout<<head->data;
	head = head -> next;
	}

Table 3-1. Output of Initial/Simple Implementation

Operations	Screenshots

Traversal	`void Display(Node* node) {`
	`while (node != NULL) {`
	`cout << node->data;`
	`node = node->next;`
	`}`
	`cout << endl;`
	`}`

Insertion at head	`void InNodeHead(Node*& head, int newData) {`
	`Node* newNode = new Node();`
	`newNode->data = newData;`
	`newNode->next = head;`
	`head = newNode;`
	`}`

Insertion at any part of the list	
Insertion at the end	
Deletion of a node	

```
void InNodeAny(Node*& prevNode, int newData) {
    if (prevNode == NULL) {
        cout << "Previous node cannot be null.\n";
        return;
    }
    Node* newNode = new Node;
    newNode->data = newData;
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}
```

```
void InNodeEnd(Node** head, int newData) {
    Node* newNode = new Node;
    newNode->data = newData;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

    Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}
```

```
void DelNode(Node** head, int 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;
}
```

Table 3-2. Code for the List Operations

a) Source Code	

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

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

```
void Display(Node* node) {
        while (node != NULL) {
                cout << node->data;
                node = node->next;
        }
        cout << endl;
}
int main() {
        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;
    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;
//step 4
    last->data = '1';
    last->next = nullptr;

        Display(head);

        return 0;
}
```

Console

```
CPE101


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

b) Source Code	

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

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

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

void Display(Node* node) {
	while (node != NULL) {
		cout << node->data;
		node = node->next;
	}
	cout << endl;
}
int main() {
	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;
   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;
//step 4
   last->data = '1';
   last->next = nullptr;
```

```
InNodeHead(head, 'G');

        Display(head);

        return 0;
}
```

Console

```
GCPE101


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

c) Source Code

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

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

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

void Display(Node* node) {
        while (node != NULL) {
                cout << node->data;
                node = node->next;
        }
        cout << endl;
}
void InNodeAny(Node*& prevNode, char newData) {
        if (prevNode == NULL) {
                cout << "Previous node cannot be null.\n";
                return;
        }
        Node* newNode = new Node;
        newNode->data = newData;
        newNode->next = prevNode->next;
        prevNode->next = newNode;
}

int main() {
        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;
    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;
//step 4
    last->data = '1';
    last->next = nullptr;
        InNodeHead(head, 'G');

        InNodeAny(second, 'E');

        Display(head);

        return 0;
}
```

Console

```
GCPEE101


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

d) Source Code

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

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

void InNodeHead(Node*& head, char newData) {
        Node* newNode = new Node;
```

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

void Display(Node* node) {
	while (node != NULL) {
		cout << node->data;
		node = node->next;
	}
	cout << endl;
}
void InNodeAny(Node*& prevNode, char newData) {
	if (prevNode == NULL) {
		cout << "Previous node cannot be null.\n";
		return;
	}
	Node* newNode = new Node;
	newNode->data = newData;
	newNode->next = prevNode->next;
	prevNode->next = newNode;
}
void DelNode(Node** head, char key) {
	Node* temp = *head;
	Node* prev = nullptr;

	if (temp != nullptr && temp->data == key) { // 1. Find previous node of the node
to be deleted.
		*head = temp->next; // 2. Change the next of previous node.
		delete temp; // 3. Free memory for the node to be deleted.
		return;
	}

	while (temp != nullptr && temp->data != key) {
		prev = temp;
		temp = temp->next;
	}

	if (temp == nullptr) return;

	prev->next = temp->next; // 2. Change the next of previous node.
	delete temp; // 3. Free memory for the node to be deleted.
}

int main() {
	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;
    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;
//step 4
    last->data = '1';
    last->next = nullptr;
        InNodeHead(head, 'G');

        InNodeAny(second, 'E');
        DelNode(&head,'C');
        Display(head);

        return 0;
}
```

Console

```
GPEE101


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

e) Source Code

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

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

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

```
}

void Display(Node* node) {
	while (node != NULL) {
		cout << node->data;
		node = node->next;
	}
	cout << endl;
}
void InNodeAny(Node*& prevNode, char newData) {
	if (prevNode == NULL) {
		cout << "Previous node cannot be null.\n";
		return;
	}
	Node* newNode = new Node;
	newNode->data = newData;
	newNode->next = prevNode->next;
	prevNode->next = newNode;
}
void DelNode(Node** head, char key) {
	Node* temp = *head;
	Node* prev = nullptr;

	if (temp != nullptr && temp->data == key) { // 1. Find previous node of the node
to be deleted.
		*head = temp->next; // 2. Change the next of previous node.
		delete temp; // 3. Free memory for the node to be deleted.
		return;
	}

	while (temp != nullptr && temp->data != key) {
		prev = temp;
		temp = temp->next;
	}

	if (temp == nullptr) return;

	prev->next = temp->next; // 2. Change the next of previous node.
	delete temp; // 3. Free memory for the node to be deleted.
}

int main() {
	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;
    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;
//step 4
    last->data = '1';
    last->next = nullptr;
        InNodeHead(head, 'G');

        InNodeAny(second, 'E');
        DelNode(&head,'C');
DelNode(&head,'P');
        Display(head);

        return 0;
}
```

Console

```
GEE101


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

f) Source Code

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

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

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

void Display(Node* node) {
        while (node != NULL) {
```

```
		cout << node->data;
		node = node->next;
	}
	cout << endl;
}
void InNodeAny(Node*& prevNode, char newData) {
	if (prevNode == NULL) {
		cout << "Previous node cannot be null.\n";
		return;
	}
	Node* newNode = new Node;
	newNode->data = newData;
	newNode->next = prevNode->next;
	prevNode->next = newNode;
}
void DelNode(Node** head, char key) {
	Node* temp = *head;
	Node* prev = nullptr;

	if (temp != nullptr && temp->data == key) { // 1. Find previous node of the node
to be deleted.
		*head = temp->next; // 2. Change the next of previous node.
		delete temp; // 3. Free memory for the node to be deleted.
		return;
	}

	while (temp != nullptr && temp->data != key) {
		prev = temp;
		temp = temp->next;
	}

	if (temp == nullptr) return;

	prev->next = temp->next; // 2. Change the next of previous node.
	delete temp; // 3. Free memory for the node to be deleted.
}

int main() {
	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;
```

	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;
	//step 4
	last->data = '1';
	last->next = nullptr;
	InNodeHead(head, 'G');
	
	InNodeAny(second, 'E');
	DelNode(&head,'C');
	DelNode(&head,'P');
	Display(head);
	
	return 0;
	}

Console	GEE101 ...Program finished with exit code 0 Press ENTER to exit console.

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

Screenshot	Analysis

`void InNodeHead(Node*& head, char newData) {`
`  Node* newNode = new Node;`
`  newNode->data = newData;`
`  newNode->next = head;`
`  newNode->prev = nullptr;`
 	
`  if (head != nullptr) {`
`  head->prev = newNode;`
`  }`
`  head = newNode;`
`}`
	function was updated to set the prev pointer of the new node to nullptr and adjust the prev pointer of the old head if it exists.

`class Node {`
`public:`
`  char data;`
`  Node* next;`
`  Node* prev;`
`};`
	The Node class was modified to include a prev pointer alongside next. This enables traversal in both directions

``
void InNodeAny(Node*& prevNode, char newData) {
    if (prevNode == nullptr) {
        cout << "Previous node cannot be null.\n";
        return;
    }
    Node* newNode = new Node;
    newNode->data = newData;
    newNode->next = prevNode->next;
    newNode->prev = prevNode;
 	
    if (prevNode->next != nullptr) {
        prevNode->next->prev = newNode;
    }
    prevNode->next = newNode;
}
``
Adjusted to manage both next and prev pointers when inserting a new node in the middle. It also updates the prev pointer of the next node if it exists.	

``
void InNodeEnd(Node** head, char newData) {
    Node* newNode = new Node;
    newNode->data = newData;
    newNode->next = nullptr;
    newNode->prev = nullptr;
 	
    if (*head == nullptr) {
        *head = newNode;
        return;
    }
 	
    Node* last = *head;
    while (last->next != nullptr) {
        last = last->next;
    }
    last->next = newNode;
    newNode->prev = last;
}
``
function now sets the prev pointer of the new node to the last node, allowing proper linkage.	

``
void DelNode(Node** head, char key) {
    Node* temp = *head;
 	
    if (temp != nullptr && temp->data == key) {
        *head = temp->next;
        if (*head != nullptr) {
            (*head)->prev = nullptr;
        }
        delete temp;
        return;
    }
 	
    while (temp != nullptr && temp->data != key) {
        temp = temp->next;
    }
 	
    if (temp == nullptr) return;
 	
    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    }
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }
    delete temp;
}
``
Modified to ensure that both next and prev pointers are correctly updated when a node is deleted. It checks if the node to be deleted is the head and handles the necessary adjustments for both pointers.	

Table 3-4. Modified Operations for Doubly Linked Lists

7. Supplementary Activity

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

class SongNode {
public:
    string songTitle;
    SongNode* next;

    SongNode(string title) : songTitle(title), next(nullptr) {}
};

class Playlist {
private:
    SongNode* head;

public:
    Playlist() : head(nullptr) {}

    void addSong(const string& title) {
        SongNode* newSong = new SongNode(title);
        if (!head) {
            head = newSong;
            head->next = head;
        } else {
            SongNode* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            temp->next = newSong;
            newSong->next = head;
        }
        cout << "Added song: " << title << endl;
    }

    void removeSong(const string& title) {
        if (!head) return;

        SongNode* current = head;
        SongNode* prev = nullptr;

        do {
            if (current->songTitle == title) {
                if (prev) {
                    prev->next = current->next;
                } else {
                    SongNode* tail = head;
                    while (tail->next != head) {
                        tail = tail->next;
                    }
                    tail->next = current->next;
```

```
                head = current->next;
            }
            delete current;
            cout << "Removed song: " << title << endl;
            return;
        }
        prev = current;
        current = current->next;
    } while (current != head);

    cout << "Song not found: " << title << endl;
}

void playAllSongs() const {
    if (!head) {
        cout << "No songs in the playlist." << endl;
        return;
    }

    SongNode* current = head;
    do {
        cout << "Playing: " << current->songTitle << endl;
        current = current->next;
    } while (current != head);
}

void nextSong() {
    if (!head) {
        cout << "No songs to play." << endl;
        return;
    }
    head = head->next;
    cout << "Now playing: " << head->songTitle << endl;
}

void previousSong() {
    if (!head) {
        cout << "No songs to play." << endl;
        return;
    }
    SongNode* current = head;
    while (current->next != head) {
        current = current->next;
    }
    head = current;
    cout << "Now playing: " << head->songTitle << endl;
}
};

int main() {
    Playlist myPlaylist;
```

```
    myPlaylist.addSong("Song 1");
    myPlaylist.addSong("Song 2");
    myPlaylist.addSong("Song 3");

    myPlaylist.playAllSongs();

    myPlaylist.nextSong();
    myPlaylist.previousSong();

    myPlaylist.removeSong("Song 2");
    myPlaylist.playAllSongs();

    return 0;
}
```

8. Conclusion

Provide the following:

• Summary of lessons learned

I learned how to implement a linked list, which allows for efficient traversal and manipulation of elements in a continuous manner.

• Analysis of the procedure

The procedure implements a singly linked list with functions for inserting nodes at the head, a specific position, and the end, as well as for deleting nodes and displaying the list. Each function ensures proper linkage between nodes while maintaining the integrity of the list.

• Analysis of the supplementary activity

The supplementary activity emphasized practical application by requiring the integration of playlist operations, which reinforced the understanding of linked list operations and their real-world use cases.

• Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

I conclude that I believe I did well in this activity by successfully implementing the required conditions however, I found out the need to improve my skills in memory management.

9. Assessment Rubric