Activity No. 3	
<Hands-on Activity 3.1 Linked Lists>	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 27/09/24**
Section: BSCPE21S1	**Date Submitted:29/09/24**
Name(s): Anduque, Kurt Gabriel A.	**Instructor: Mrs. Maria Rizette Sayo**

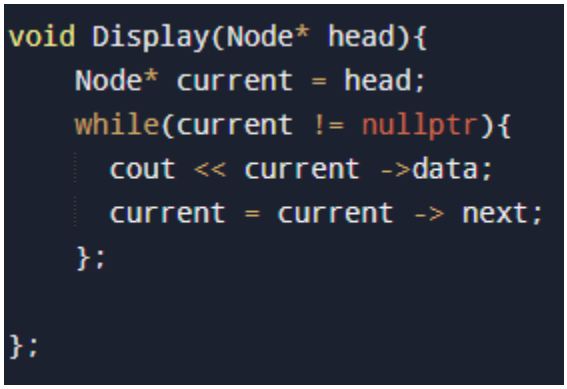
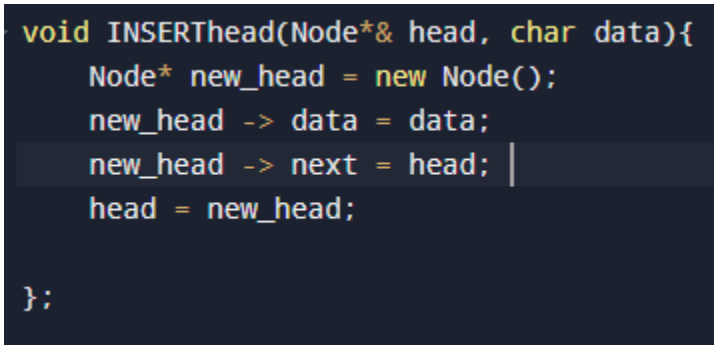
6. Output

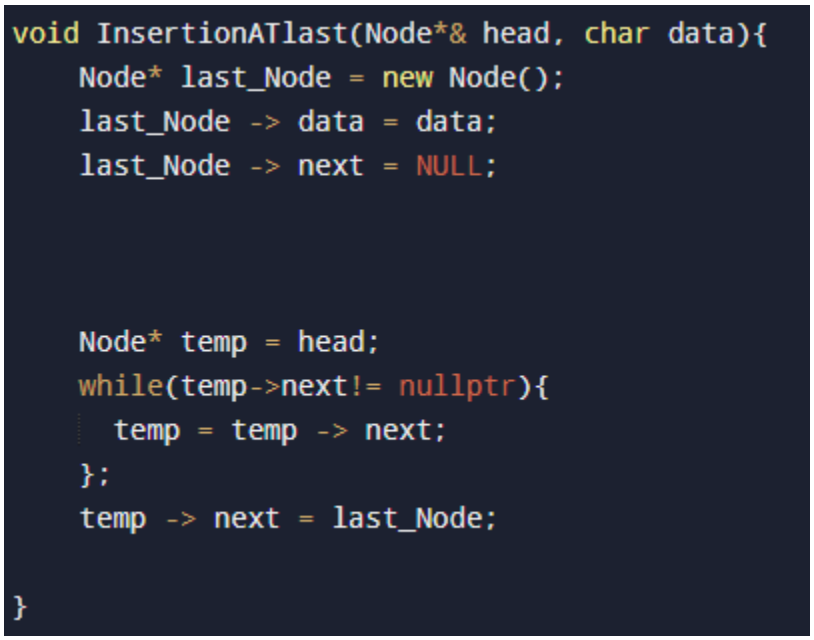
Screenshot

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

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

void traversal(Node* temp){
        Node* current = temp;
        while(current!=nullptr){
        cout << current -> data;
        current = current-> next;
        };
};



int main() {
        /// I combine step 1 and step 2

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

        Node* temp = head;

        head -> data = 'C';
        second -> data = 'P';
        third -> data = 'E';
        fourth -> data = '0';
        fifth -> data = '1';
        last -> data = '0';
```

	head -> next = second; second -> next = third; third -> next = fourth; fourth -> next = fifth; fifth -> next = last; last -> next = NULL; traversal(temp); return 0; }
Discussion	The program's output will show the characters stored in each node of the linked list, which are CPE010. This is because each node has a single character and is linked to the next node, resulting in a chain of values. The program walks the linked list beginning with head and printing each character until it reaches the last node, which points to nullptr.

Table 3-1. Output of Initial/Simple Implementation

OPERATION	SCREENSHOT
Traversal	
Insertion at the head	

Insertion at any part of the list	
Insertion at the end	

```
void InsertionANYpart(Node*& head, char data, int position){
    Node* new_Node = new Node();
    new_Node -> data = data;

    Node* current = head;
    for (int counter = 1 ; counter < position -1 && current != nullptr;counter++){
        current = current -> next;
    };
        if (current ==  nullptr){
          cout << "position entered is out of ranged"<< endl;
          return;
        };

        new_Node->next = current->next;
        current->next = new_Node;


};
```

```
void InsertionATlast(Node*& head, char data){
    Node* last_Node = new Node();
    last_Node -> data = data;
    last_Node -> next = NULL;



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

}
```

Deletion of node	

```
void DeleteNode(Node*& head, char data) {

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

    previous->next = current->next;
    delete current;
}
```

Table 3-2. Code for the List Operations

A.	**Source Code**	`cout << "Traversal of Data"<< endl;` `Display(head);` `cout << endl;` `cout << endl;`
	Console	`Traversal of Data` `CPE010`
B.	**Source Code**	`cout << "Insertion at the Head"<<endl;` `INSERThead(head,'G');` `Display(head);` `cout << endl;` `cout << endl;`
	Console	`Insertion at the Head` `GCPE010`

C.	**Source Code**	`cout << "Insertion at any part of the linked list"<<endl;` `InsertionANYpart(head,'E',4);` `InsertionANYpart(head,'C',6);` `Display(head);` `cout<<endl;` `cout <<endl;`
	Console	`Insertion at any part of the linked list` `GCPEEC010`
D.	**Source Code**	`cout << "Insert at last part"<<endl;` `InsertionATlast(head,'Y');` `Display(head);` `cout << endl;` `cout << endl;`
	Console	`Insert at last part` `GCPEEC010Y`
E.	**Source Code**	`cout << "Delete A node"<< endl;` `cout << "Delete Node C"<<endl;` `DeleteNode(head,'C');` `DeleteNode(head,'C');` `cout <<"Delete Node P"<<endl;` `DeleteNode(head,'P');`
	Console	`Delete A node` `Delete Node C` `Delete Node P` `Delete Node Y`

F.	**Source Code**	`cout<<"Final OUTPUT"<<endl;` `Display(head);`
	Console	`Final OUTPUT` `GEE101`

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

ScreenShots(s)	Analysis
`void Display(Node* head) {` `  Node* current = head;` `  while (current != nullptr) {` `  cout << current->data;` `  current = current->next;` `  }` `  cout << endl;` `}`	This code demonstrates the method required to iterate through a list with two links. It is evident that this is comparable to the head node's traversal of a single link list.
`void INSERThead(Node*& head, char data) {` `  Node* new_head = new Node(data);` `  new_head->next = head;` `  if (head != nullptr) {` `  head->prev = new_head;` `  }` `  head = new_head;` `}`	This image illustrates a function that adds a node and value to the beginning of a double linked list. This isn't the same as a list with just one link. In order to place ourselves at the head, we must track the head node and the preceding node.

```
void InsertionANYpart(Node*& head, char data, int position) {
    Node* new_Node = new Node(data);

    if (position == 1) {
        INSERThead(head, data);
        return;
    }

    Node* current = head;
    for (int counter = 1; counter < position - 1 && current != nullptr; counter++) {
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Position entered is out of range" << endl;
        return;
    }

    new_Node->next = current->next;
    new_Node->prev = current;

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

    current->next = new_Node;
}
```

This code demonstrates the creation of a function that appends a node and value to the end of the list. Because the nodes in a double linked list are connected, we must maintain track of the previous and next nodes. After determining the end of the list with a temporary node, we inserted a node that was discovered by the while condition.

```
void InsertionATlast(Node*& head, char data) {
    Node* last_Node = new Node(data);

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

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

    temp->next = last_Node;
    last_Node->prev = temp;
}
```

The function for adding at any place in a double linked list is depicted in the figure. There are some similarities between a single linked list and this one, however in order to correctly locate the correct place and add a new node to the list, we must maintain track of and update the previous and current nodes correspondingly.

`` void DeleteNode(Node*& head, char data) { Node* current = head; if (current != nullptr && current->data == data) { head = current->next; if (head != nullptr) { head->prev = nullptr; } delete current; return; } while (current != nullptr && current->data != data) { current = current->next; } if (current == nullptr) { cout << "Node with data " << data << " not found." << endl; return; } if (current->next != nullptr) { current->next->prev = current->prev; } if (current->prev != nullptr) { current->prev->next = current->next; } delete current; } ```	In order to remove a node from a double link list, we must search through the link list until we locate the chosen node. After a node in the list is deleted, we examine the if statements that are used to verify and then perform the appropriate actions to remove and relink the nodes. In order to avoid causing a segmentation fault, we must update it correctly.

Table 3-4. Modified Operations for Doubly Linked Lists

7. Supplementary Activity

Source Code:

```cpp
#include <iostream>
#include <utility>
#include <string>
#include <thread>
#include <chrono>
using namespace std;

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


void InsertHEAD(Node*& head, string songs){
    Node* new_song = new Node();
    new_song -> songs = songs;
    new_song -> next = head;
    head = new_song;

};



void PlaySong(Node* head, int Song_counter){
    if (head == nullptr){
      cout << "PLAYLIST IS EMPTY";
    };

    Node* current = head;
    int Counter = Song_counter;
    while(current != nullptr){
      cout << Counter<<". "<<current ->songs <<endl;
      Counter--;
      current = current ->next;
    };
```

```
void PlaySong1(Node* head){

    if (head == nullptr){
      cout << "PLAYLIST IS EMPTY";
    };

    Node* current = head;
    while(current!=nullptr){
      cout <<"Playing song: "<< current ->songs<<endl;

      current = current->next;
      if (current!=nullptr){
          cout <<"next"<<endl;
          this_thread::sleep_for(chrono::seconds(2));
      };



    };
    cout<<"DONE PLAYING ALL SONGS"<<endl;
};
```

```
void DeleteSong(Node*& head, string songs){

    if (head == nullptr){
      cout << "PLAYLIST IS EMPTY";
    };

    Node* current = head;
    Node* previous = nullptr;

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

    while(current != nullptr && current-> songs != songs){
        previous = current;
        current = current -> next;

    };

    previous -> next = current ->next;
    delete current;
};
```

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


  string song;
  string insert;
  string deletion;
  int Song_counter = 1;
  int choice;

  cout << "Enter your first song to insert: ";
  getline(cin,song);


  head -> songs = song;
  head -> next = nullptr;

  cout<<endl;
  cout<<endl;
  while (true){

      cout << "=================== SONG PLAYER ==================="<<endl;
      cout <<"[1] = play your playlist"<<endl;
      cout <<"[2] = insert a new song"<<endl;
      cout <<"[3] = delete a song in playlist"<<endl;
      cout <<"[4] = End the streaming"<<endl;
      cout<< "choice: ";
      cin>>choice;
      cout << "================================================="<<endl;


      cout <<endl;
      cout <<endl;
```

```
if (choice == 1){

    cout << "============= Playing Line up Pls Enjoy :> =============="<<endl;
    PlaySong1(head);
    cout << endl;
    cout << endl;
    cout << "============================================================"<<endl;
    cout <<endl;



}else if (choice == 2){

    cout << "==================== SONG DOWNLOADER ====================="<<endl;
    cout<<endl;
    cout << "Enter the new song to download: ";
    cin.ignore();

    getline(cin,insert);
    Song_counter++;

    cout <<endl;
    InsertHEAD(head,insert);
    cout << "New playlist: "<<endl;
    PlaySong(head,Song_counter);
    cout << "============================================================"<<endl;

    cout<<endl;
    cout<<endl;
```

```
    }else if (choice == 3){

        cin.ignore();
        cout << "==================== SONG DELETION ====================="<<endl;

        cout <<endl;
        cout<< "Choose a song from playlist to delete"<<endl;

        PlaySong(head,Song_counter);

        cout<<endl;

        cout << "Enter the song to delete in playlist: ";
        getline(cin,deletion);
        Song_counter--;


        cout <<endl;
        cout <<endl;

        cout <<"Updated Playlist"<<endl;
        DeleteSong(head,deletion);
        PlaySong(head,Song_counter);
        cout<<endl;
        cout << "============================================================"<<endl;

    }else if (choice == 4){
      cout <<"============================ ENDING STREAM ============================"
          <<endl;
      break;
    };


};
```

Output:

```
Enter your first song to insert: Ikaw


================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 2
=================================================


================== SONG DOWNLOADER ==================

Enter the new song to download: Escolta

New playlist:
2. Escolta
1. Ikaw
=====================================================


================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 2
=================================================
```

```
=================== SONG DOWNLOADER ====================

Enter the new song to download: Escolta

New playlist:
2. Escolta
1. Ikaw
=========================================================



=================== SONG PLAYER ====================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 2
=========================================================



=================== SONG DOWNLOADER ====================

Enter the new song to download: Dance

New playlist:
3. Dance
2. Escolta
1. Ikaw
=========================================================
```

```
================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 2
=================================================


=================== SONG DOWNLOADER ===================

Enter the new song to download: Where you from part 1

New playlist:
4. Where you from part 1
3. Dance
2. Escolta
1. Ikaw
=======================================================
```

```
================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 2
=================================================


=================== SONG DOWNLOADER ===================

Enter the new song to download: Where you from part 2

New playlist:
5. Where you from part 2
4. Where you from part 1
3. Dance
2. Escolta
1. Ikaw
=======================================================
```

```
================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 2
=================================================



=================== SONG DOWNLOADER ====================

Enter the new song to download: Where you from part 3

New playlist:
6. Where you from part 3
5. Where you from part 2
4. Where you from part 1
3. Dance
2. Escolta
1. Ikaw
========================================================
```

```
================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 2
=================================================



==================== SONG DOWNLOADER ====================

Enter the new song to download: Oras

New playlist:
7. Oras
6. Where you from part 3
5. Where you from part 2
4. Where you from part 1
3. Dance
2. Escolta
1. Ikaw
=========================================================
```

```
================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 3
=================================================



================== SONG DELETION ==================

Choose a song from playlist to delete
7. Oras
6. Where you from part 3
5. Where you from part 2
4. Where you from part 1
3. Dance
2. Escolta
1. Ikaw

Enter the song to delete in playlist: Ikaw


Updated Playlist
6. Oras
5. Where you from part 3
4. Where you from part 2
3. Where you from part 1
2. Dance
1. Escolta

=================================================
```

```
================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 1
=================================================


============= Playing Line up Pls Enjoy :> ==============
Playing song: Oras
next
Playing song: Where you from part 3
next
Playing song: Where you from part 2
next
Playing song: Where you from part 1
next
Playing song: Dance
next
Playing song: Escolta
DONE PLAYING ALL SONGS


=========================================================
```

```
================== SONG PLAYER ==================
[1] = play your playlist
[2] = insert a new song
[3] = delete a song in playlist
[4] = End the streaming
choice: 4
=================================================


========================== ENDING STREAM ==========================
=================== Thank you for listening!!!!! ===================


=== Code Execution Successful ===
```

8. Conclusion

- I was able to learn about C++ syntax and linked list creation after completing this exercise. One type of linear data structure is the linked list. I was able to comprehend the structure of a linked list with this exercise. I discovered that linked lists are made up of pointers and are dynamic. The activity began by describing how the list was initialized and went on to describe the various operations. I've worked on fundamental linked list operations, such iterating through the list to show the songs, appending a new song to the top of the list, and removing a song. I learned how to handle pointers and comprehend the list's fundamental structure from this encounter

9. Assessment Rubric