

Activity No. 3	
LINKED LISTS	
Course Code: CPE010	Program: BS Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 09 / 27 / 2024
Section: CPE21S4	Date Submitted: 09 / 27 / 2024
Name(s): ROALLOS, Jean Gabriel Vincent G.	Instructor: Prof. Maria Rizette Sayo

6. Output

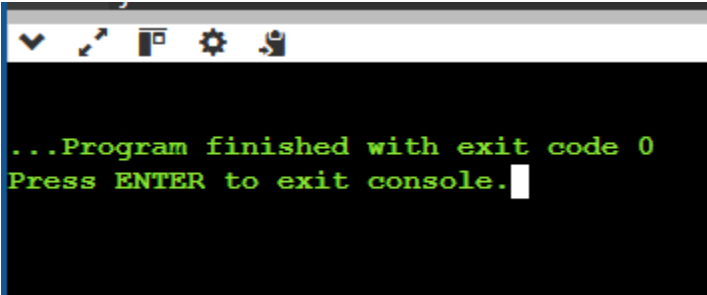
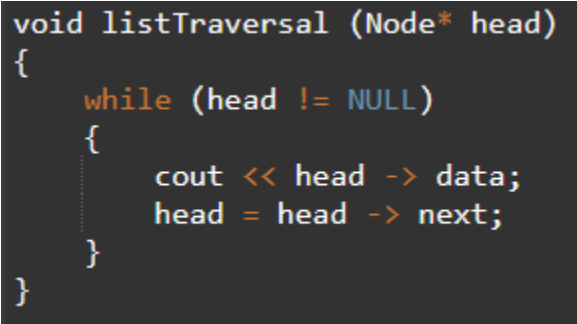
Screenshot	
Discussion	<p>We can implement the code snippet:</p> <pre>void listTraversal (Node* head) { while (head != NULL) { cout << head -> data; head = head -> next; } }</pre> <p>To output the contents of the nodes and display 'CPE010' This process is used for the linked list traversal.</p>

Table 3-1. Output of Initial/Simple Implementation

Operation	Screenshot
Traversal	

Insertion at head

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

Insertion at any part of the list

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

    if (position == 1) {
        newNode -> next = head;
        head = newNode;
        return;
    }

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

    if (temp == nullptr) {
        cout << "Error: Not in range." << endl;
        return;
    }

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

Insertion at the end

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

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

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

    temp->next = newNode;
}
```

Deletion of a node

```
void deleteNode(Node *&head, char data)
{
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }

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

    Node *temp = head;
    while (temp -> next != nullptr) {
        if (temp -> next -> data == data) {
            Node *nodeToDelete = temp -> next;
            temp -> next = temp -> next -> next;
            delete nodeToDelete;
            return;
        }
        temp = temp -> next;
    }
}
```

Table 3-2. Code for the List Operations

a.	Source Code	<pre>void listTraversal (Node* head) { while (head != NULL) { cout << head -> data; head = head -> next; } }</pre>
	Console	<pre>Original Linked List, displayed using list traversal: CPE101</pre>
b.	Source Code	<pre>void insertHead(Node *&head, char data) { Node *newNode = new Node; newNode -> data = data; newNode -> next = head; head = newNode; }</pre>
	Console	<pre>Inserted a new node as head. Updated Linked List: YCPE101</pre>

C.	Source Code	<pre> void insertList(Node *&head, char data, int position) { Node *newNode = new Node; newNode -> data = data; if (position == 1) { newNode -> next = head; head = newNode; return; } Node *temp = head; for (int i = 1; i < position - 1 && temp != nullptr; i++) { temp = temp -> next; } if (temp == nullptr) { cout << "Error: Not in range." << endl; return; } newNode->next = temp->next; temp->next = newNode; } </pre>
	Console	<pre> Inserted a new item between the list. Updated Linked List: YCPTE101 </pre>
d.	Source Code	<pre> void insertEnd(Node *&head, char data) { Node *newNode = new Node; newNode -> data = data; newNode -> next = nullptr; if (head == nullptr) { head = newNode; return; } Node *temp = head; while (temp->next != nullptr) { temp = temp->next; } temp->next = newNode; } </pre>
	Console	<pre> Inserted a new item the end of the list. Updated Linked List: YCPTE1012 </pre>

e.	Source Code	<pre>void deleteNode(Node *&head, char data) { if (head == nullptr) { cout << "List is empty" << endl; return; } if (head -> data == data) { Node *temp = head; head = head -> next; delete temp; return; } Node *temp = head; while (temp -> next != nullptr) { if (temp -> next -> data == data) { Node *nodeDelete = temp -> next; temp -> next = temp -> next -> next; delete nodeDelete; return; } temp = temp -> next; } }</pre>
	Console	<pre>Deleted an item in the list. Updated Linked List: YCPTE112</pre>

f.	Source Code	<pre> int main() { Node *head = NULL; insertEnd(head, 'C'); insertEnd(head, 'P'); insertEnd(head, 'E'); insertEnd(head, '1'); insertEnd(head, '0'); insertEnd(head, '1'); cout << "Original Linked List, displayed using list traversal: "; listTraversal(head); cout << endl; insertHead(head, 'G'); cout << "\nInserted 'G' as new head node: "; listTraversal(head); cout << endl; insertList(head, 'E', 4); cout << "\nInserted 'E' within the list: "; listTraversal(head); cout << endl; deleteNode(head, 'C'); deleteNode(head, 'P'); cout << "\nDeleted items 'C' and 'P' from the list: "; listTraversal(head); cout << endl; /*↔*/ return 0; } </pre>
	Console	<pre> Original Linked List, displayed using list traversal: CPE101 Inserted 'G' as new head node: GCPE101 Inserted 'E' within the list: GCPEE101 Deleted items 'C' and 'P' from the list: GEE101 </pre>

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

Screenshot(s)	Analysis
<pre> class Node { public: char data; Node *next; Node *prev; }; </pre>	<p>This modification handles the type of link each item has within the list. <i>Node *prev</i> gives permission to access previous items in the list.</p>
<pre> void insertHead(Node *&head, char data) { Node *newNode = new Node; newNode -> data = data; newNode -> next = head; head = newNode; if (head != nullptr) { head -> prev = newNode; } } </pre>	<p>All functions have a node initialization process. This helps in creating new nodes to manipulate such as adding and deleting elements.</p>
<pre> void insertlist(Node *&head, char data, int position) { Node *newNode = new Node; newNode -> data = data; if (position == 1) { newNode -> next = head; head = newNode; return; } Node *temp = head; for (int i = 1; i < position - 1 && temp != nullptr; i++) { temp = temp -> next; } if (temp == nullptr) { cout << "Error: Not in range." << endl; return; } newNode -> next = temp -> next; temp -> next -> prev = newNode; } </pre>	<p>With the help of the for loop finding the position to where the new node should be, the last two lines of the function reassign values within the list.</p>

```

void insertEnd(Node *&head, char data)
{
    Node *newNode = new Node;
    newNode -> data = data;
    newNode -> next = nullptr;

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

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

    temp->next = newNode;
    newNode -> prev = temp;
}

```

The last two lines of code in this function gives the characteristic of having a doubly-linked list.

Table 3-4. Modified Operations for Doubly Linked Lists

7. Supplementary Activity

```

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

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

void AddSong(string new_song)
{
    Node* new_node = new Node(new_song);
    if (head == nullptr)
    {
        head = new_node;
        new_node->next = head; // Pointing to itself
    }
    else
    {
        Node* temp = head;
        while (temp->next != head)
        {
            temp = temp->next; // Traverse to the last node
        }
        temp->next = new_node;
        new_node->next = head; // Make it circular
    }
    cout << "Added: " << new_song << endl;
}

```



```
void makePlaylist(string songs[], int n)
{
    for (int i = 0; i < n; i++)
    {
        addSong(songs[i]);
    }
}
```

Incomplete code

8. Conclusion

I have learned more about pointers and nodes and all possible operations. I have also learned about the possible perks of using linked lists in specific use cases. I had a hard time understanding nodes and pointers so I had trouble doing the laboratory activity. Modifying the operations to accommodate for doubly linked lists was hard as the output differed from the expected output. Even though I had grasped a deeper understanding of the main laboratory work, I had little time left for the supplementary activity, thus not completing a working code. I will still try to finish the supplementary activity to hone my skill.

9. Assessment Rubric