

Activity No. <n>	
<Replace with Title>	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 09 / 27 / 2024
Section: CPE21S4	Date Submitted: 09 / 27 / 2024
Name(s):LeoJ Jeam B. Tandayu	Instructor: Ms. Maria Rizette Sayo

6. Output

Output	
Discussion	The code can be improved by adding an output syntax to output what is inside the linked list since the original code has no output syntax.

Table 3-1. Output of Initial/Simple Implementation

Operation	Screenshot
Traversal	<pre> void traverse(Node *head) {     int ll_count = 1;     while(head != NULL) {         cout &lt;&lt; "[000" &lt;&lt; ll_count &lt;&lt; "   " &lt;&lt; head-&gt;data &lt;&lt; " ] -&gt;";         head = head-&gt;next;         ll_count++;     }     cout &lt;&lt; "null" &lt;&lt; endl; // Indicate the end of the list } </pre>
Insertion at head	<pre> void insertAtHead(Node *&amp;head, char newData) {     Node *newNode = new Node;     newNode-&gt;data = newData;     newNode-&gt;next = head;     head = newNode; } </pre>
Insertion at any part of the list	<pre> void insertAtPosition(Node *&amp;head, char newData, int position) {     Node *newNode = new Node;     newNode-&gt;data = newData;      if (position == 0) {         newNode-&gt;next = head;         head = newNode;         return;     }      Node *current = head;     for (int i = 0; i &lt; position - 1 &amp;&amp; current != nullptr; i++) {         current = current-&gt;next;     }      if (current == nullptr) {         cout &lt;&lt; "Position out of bounds, inserting at end instead." &lt;&lt; endl;         delete newNode;         return;     }      newNode-&gt;next = current-&gt;next;     current-&gt;next = newNode; } </pre>

Insertion at the end

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

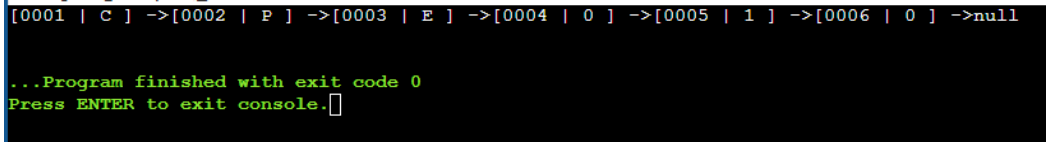
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node *current = head;
    while (current->next != nullptr) {
        current = current->next;
    }

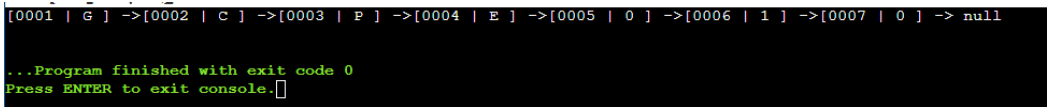
    current->next = newNode;
}
```

Deletion of a node	<pre> void deleteAtPosition(Node *&amp;head, int position) {     if (head == nullptr) {         cout &lt;&lt; "List is empty. No nodes to delete." &lt;&lt; endl;         return;     }      Node *temp = head;      if (position == 0) {         head = temp-&gt;next;         delete temp;         return;     }      for (int i = 0; temp != nullptr &amp;&amp; i &lt; position - 1; i++) {         temp = temp-&gt;next;     }      if (temp == nullptr    temp-&gt;next == nullptr) {         cout &lt;&lt; "Position out of bounds." &lt;&lt; endl;         return;     }      Node *next = temp-&gt;next-&gt;next;     delete temp-&gt;next;     temp-&gt;next = next; } </pre>
--------------------	--

Table 3-2. Code for the List Operations

a.	<p><b>Source Code</b></p> <pre> #include &lt;iostream&gt; #include &lt;utility&gt; using namespace std;  class Node { public:     char data;     Node *next; };  void traverse(Node *head) {     int ll_count = 1;     while (head != NULL) {         cout &lt;&lt; "[000" &lt;&lt; ll_count &lt;&lt; "   " &lt;&lt; head-&gt;data &lt;&lt; " ] -&gt;";         head = head-&gt;next;     } } </pre>
----	--

		<pre>         ll_count++;     }     cout &lt;&lt; "null" &lt;&lt; endl; } int main() {     Node *head = new Node;     Node *second = new Node;     Node *third = new Node;     Node *fourth = new Node;     Node *fifth = new Node;     Node *last = new Node;      head-&gt;data = 'C';     head-&gt;next = second;     second-&gt;data = 'P';     second-&gt;next = third;     third-&gt;data = 'E';     third-&gt;next = fourth;     fourth-&gt;data = 'O';     fourth-&gt;next = fifth;     fifth-&gt;data = '1';     fifth-&gt;next = last;     last-&gt;data = '0';     last-&gt;next = nullptr;      traverse(head);      return 0; } </pre>
	Console	 <pre> [0001   C ] -&gt;[0002   P ] -&gt;[0003   E ] -&gt;[0004   O ] -&gt;[0005   1 ] -&gt;[0006   0 ] -&gt;null  ...Program finished with exit code 0 Press ENTER to exit console. </pre>
b.	Source Code	<pre> #include &lt;iostream&gt; #include &lt;utility&gt; using namespace std;  class Node { public:     char data;     Node *next; };  void traverse(Node *head) {     int ll_count = 1;     while (head != NULL) {         cout &lt;&lt; "[000" &lt;&lt; ll_count &lt;&lt; "   " &lt;&lt; head-&gt;data &lt;&lt; " ] -&gt;";     } } </pre>

		<pre>         head = head-&gt;next;         ll_count++;     }     cout &lt;&lt; " null" &lt;&lt; endl; }  void insertAtHead(Node *&amp;head, char newData) {     Node *newNode = new Node;     newNode-&gt;data = newData;     newNode-&gt;next = head;     head = newNode; }  int main() {      Node *head = new Node;     Node *second = new Node;     Node *third = new Node;     Node *fourth = new Node;     Node *fifth = new Node;     Node *last = new Node;      head-&gt;data = 'C';     head-&gt;next = second;     second-&gt;data = 'P';     second-&gt;next = third;     third-&gt;data = 'E';     third-&gt;next = fourth;     fourth-&gt;data = 'O';     fourth-&gt;next = fifth;     fifth-&gt;data = '1';     fifth-&gt;next = last;     last-&gt;data = '0';     last-&gt;next = nullptr;      insertAtHead(head, 'G');      traverse(head);      return 0; } </pre>
	<b>Console</b>	
c.	<b>Source Code</b>	<pre> #include &lt;iostream&gt; #include &lt;utility&gt; using namespace std; </pre>

```

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

void traverse(Node *head) {
    int ll_count = 1;
    while (head != NULL) {
        cout << "[000" << ll_count << " | " << head->data << " ] ->";
        head = head->next;
        ll_count++;
    }
    cout << " null" << endl;
}

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

void deleteAtPosition(Node *&head, int position) {
    if (head == nullptr) {
        cout << "List is empty. No nodes to delete." << endl;
        return;
    }

    Node *temp = head;

    if (position == 0) {
        head = temp->next;
        delete temp;
        return;
    }

    for (int i = 0; temp != nullptr && i < position - 1; i++) {
        temp = temp->next;
    }

    if (temp == nullptr || temp->next == nullptr) {
        cout << "Position out of bounds." << endl;
        return;
    }

    Node *next = temp->next->next;
    delete temp->next;
    temp->next = next;
}

```

```

}

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

    if (position == 0) {
        newNode->next = head;
        head = newNode;
        return;
    }

    Node *current = head;
    for (int i = 0; i < position - 1 && current != nullptr; i++) {
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Position out of bounds, inserting at end instead." << endl;
        delete newNode;
        return;
    }

    newNode->next = current->next;
    current->next = newNode;
}

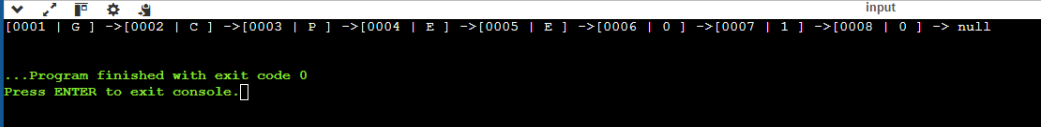
int main()
{
    Node *head = new Node;
    Node *second = new Node;
    Node *third = new Node;
    Node *fourth = new Node;
    Node *fifth = new Node;
    Node *last = new Node;

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

    insertAtHead(head, 'G');
}

```



		<pre> insertAtPosition(head, 'E' , 3);  traverse(head);  return 0; } </pre>
	<b>Console</b>	 <pre> input [0001   G ] -&gt; [0002   C ] -&gt; [0003   P ] -&gt; [0004   E ] -&gt; [0005   E ] -&gt; [0006   0 ] -&gt; [0007   1 ] -&gt; [0008   0 ] -&gt; null ...Program finished with exit code 0 Press ENTER to exit console. </pre>
d.	<b>Source Code</b>	<pre> #include &lt;iostream&gt; #include &lt;utility&gt; using namespace std;  class Node { public:     char data;     Node *next; };  void traverse(Node *head) {     int ll_count = 1;     while (head != NULL) {         cout &lt;&lt; "[000" &lt;&lt; ll_count &lt;&lt; "   " &lt;&lt; head-&gt;data &lt;&lt; " ] -&gt;";         head = head-&gt;next;         ll_count++;     }     cout &lt;&lt; " null" &lt;&lt; endl; }  void insertAtHead(Node *&amp;head, char newData) {     Node *newNode = new Node;     newNode-&gt;data = newData;     newNode-&gt;next = head;     head = newNode; }  void deleteAtPosition(Node *&amp;head, int position) {     if (head == nullptr) {         cout &lt;&lt; "List is empty. No nodes to delete." &lt;&lt; endl;         return;     }      Node *temp = head;      if (position == 0) {         head = temp-&gt;next;     } } </pre>

```

        delete temp;
        return;
    }

    for (int i = 0; temp != nullptr && i < position - 1; i++) {
        temp = temp->next;
    }

    if (temp == nullptr || temp->next == nullptr) {
        cout << "Position out of bounds." << endl;
        return;
    }

    Node *next = temp->next->next;
    delete temp->next;
    temp->next = next;
}

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

    head->data = 'C';
    head->next = second;
    second->data = 'P';
    second->next = third;
    third->data = 'E';
    third->next = fourth;
    fourth->data = 'O';
    fourth->next = fifth;
    fifth->data = 'I';
    fifth->next = last;
    last->data = 'O';
    last->next = nullptr;

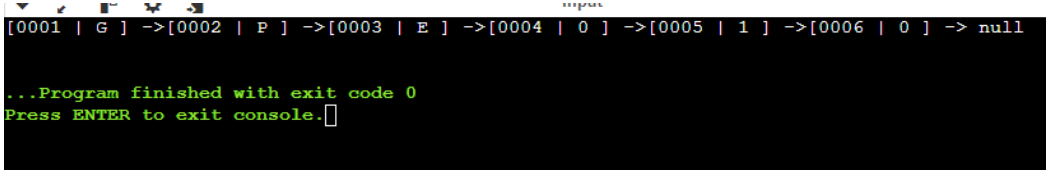
    insertAtHead(head, 'G');

    deleteAtPosition(head, 1);

    traverse(head);

    return 0;
}

```

	<div data-bbox="261 184 367 218" data-label="Section-Header"> <h3>Console</h3> </div>	
e.	<div data-bbox="261 390 423 424" data-label="Section-Header"> <h3>Source Code</h3> </div>	<pre> #include &lt;iostream&gt; #include &lt;utility&gt; using namespace std;  class Node { public:     char data;     Node *next; };  void traverse(Node *head) {     int ll_count = 1;     while (head != NULL) {         cout &lt;&lt; "[000" &lt;&lt; ll_count &lt;&lt; "   " &lt;&lt; head-&gt;data &lt;&lt; " ] -&gt;";         head = head-&gt;next;         ll_count++;     }     cout &lt;&lt; " null" &lt;&lt; endl; }  void insertAtHead(Node *&amp;head, char newData) {     Node *newNode = new Node;     newNode-&gt;data = newData;     newNode-&gt;next = head;     head = newNode; }  void deleteAtPosition(Node *&amp;head, int position) {     if (head == nullptr) {         cout &lt;&lt; "List is empty. No nodes to delete." &lt;&lt; endl;         return;     }      Node *temp = head;      if (position == 0) {         head = temp-&gt;next;         delete temp;         return;     }      for (int i = 0; temp != nullptr &amp;&amp; i &lt; position - 1; i++) {         temp = temp-&gt;next;     } </pre>

```
}

if (temp == nullptr || temp->next == nullptr) {
    cout << "Position out of bounds." << endl;
    return;
}
```

```
Node *next = temp->next->next;
delete temp->next;
temp->next = next;
}
```

```
int main()
{
```

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

```
    head->data = 'C';
    head->next = second;
    second->data = 'P';
    second->next = third;
    third->data = 'E';
    third->next = fourth;
    fourth->data = 'O';
    fourth->next = fifth;
    fifth->data = 'I';
    fifth->next = last;
    last->data = '0';
    last->next = nullptr;
```

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

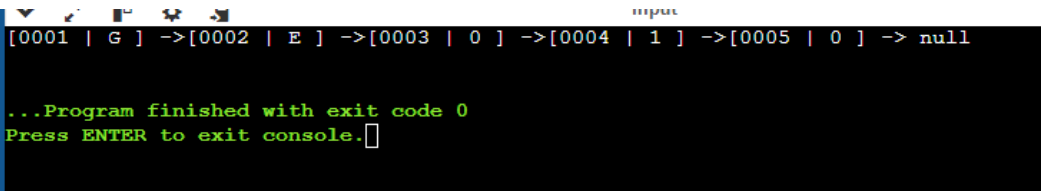
```
    deleteAtPosition(head, 1);
```

```
    deleteAtPosition(head, 1);
```

```
    traverse(head);
```

```
    return 0;
```

```
}
```

	Console	 <pre> [0001   G ] -&gt;[0002   E ] -&gt;[0003   0 ] -&gt;[0004   1 ] -&gt;[0005   0 ] -&gt; null  ...Program finished with exit code 0 Press ENTER to exit console. </pre>
f.	Source Code	<pre> #include &lt;iostream&gt; #include &lt;utility&gt; using namespace std;  class Node { public:     char data;     Node *next; };  void traverse(Node *head) {     int ll_count = 1;     while (head != NULL) {         cout &lt;&lt; "[000" &lt;&lt; ll_count &lt;&lt; "   " &lt;&lt; head-&gt;data &lt;&lt; " ] -&gt;";         head = head-&gt;next;         ll_count++;     }     cout &lt;&lt; " null" &lt;&lt; endl; }  void insertAtHead(Node *&amp;head, char newData) {     Node *newNode = new Node;     newNode-&gt;data = newData;     newNode-&gt;next = head;     head = newNode; }  void deleteAtPosition(Node *&amp;head, int position) {     if (head == nullptr) {         cout &lt;&lt; "List is empty. No nodes to delete." &lt;&lt; endl;         return;     }      Node *temp = head;      if (position == 0) {         head = temp-&gt;next;         delete temp;         return;     }      for (int i = 0; temp != nullptr &amp;&amp; i &lt; position - 1; i++) {         temp = temp-&gt;next;     } </pre>

```

if (temp == nullptr || temp->next == nullptr) {
    cout << "Position out of bounds." << endl;
    return;
}

Node *next = temp->next->next;
delete temp->next;
temp->next = next;
}

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

    if (position == 0) {
        newNode->next = head;
        head = newNode;
        return;
    }

    Node *current = head;
    for (int i = 0; i < position - 1 && current != nullptr; i++) {
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Position out of bounds, inserting at end instead." << endl;
        delete newNode;
        return;
    }

    newNode->next = current->next;
    current->next = newNode;
}

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

    head->data = 'C';
    head->next = second;
    second->data = 'P';
    second->next = third;
    third->data = 'E';

```

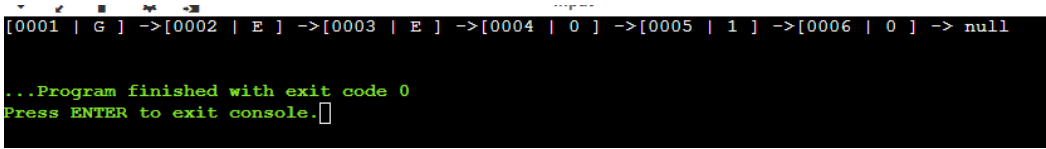
	<pre> third-&gt;next = fourth; fourth-&gt;data = '0'; fourth-&gt;next = fifth; fifth-&gt;data = '1'; fifth-&gt;next = last; last-&gt;data = '0'; last-&gt;next = nullptr;  insertAtHead(head, 'G');  deleteAtPosition(head, 1);  deleteAtPosition(head, 1);  insertAtPosition(head, 'E' , 1);  traverse(head);  return 0; } S </pre>
Console	 <pre> [0001   G ] -&gt;[0002   E ] -&gt;[0003   E ] -&gt;[0004   0 ] -&gt;[0005   1 ] -&gt;[0006   0 ] -&gt; null  ...Program finished with exit code 0 Press ENTER to exit console. </pre>

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

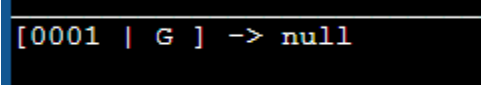
Screenshot(s)	Analysis
 <pre> [0001   G ] -&gt; null </pre>	<p>When I used the traverse function, the head is now the tail which makes the code output only "G".</p>

Table 3-4. Modified Operations for Doubly Linked List

## 7. Supplementary Activity

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

class Song {
public:
    string title;
    Song* next;

    Song(string t) : title(t), next(nullptr) {}
};

class Playlist {
private:
    Song* head;

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

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

    void removeSong(string title) {
        if (!head) {
            cout << "Playlist is empty." << endl;
            return;
        }

        Song* current = head;
        Song* previous = nullptr;

        do {
            if (current->title == title) {
                if (previous) {
                    previous->next = current->next;
                } else {
                    Song* last = head;
```



```

        while (last->next != head) {
            last = last->next;
        }
        last->next = current->next;
        head = current->next;
    }
    delete current;
    cout << "Removed: " << title << endl;
    return;
}
previous = current;
current = current->next;
} while (current != head);

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

void playAll() {
    if (!head) {
        cout << "Playlist is empty." << endl;
        return;
    }

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

};

int main() {
    Playlist myPlaylist;

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

    myPlaylist.playAll();

    myPlaylist.removeSong("Song 2");

    myPlaylist.playAll();

    return 0;
}

```

```
Added: Song 1
Added: Song 2
Added: Song 3
Playing: Song 1
Playing: Song 2
Playing: Song 3
Removed: Song 2
Playing: Song 1
Playing: Song 3
```

## 8. Conclusion

Provide the following:

- Summary of lessons learned
  - There are many ways to utilize linked lists such as insertion and deletion of the data inside an existing list. We can also use a double linked list which functions in both ways of the list in a way that the head can be the tail and the tail can be the head.
- Analysis of the procedure
  -
- Analysis of the supplementary activity
  - It is similar to the sample activity which utilize adding and deleting items from the linked list.
- Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?
  - I would say I did an average job in this activity, with the help of my friends I was able to accomplish it. For improvement I would say handling larger scale of data and how to use the double linked list properly as it is a little bit confusing for me.

## 9. Assessment Rubric