

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: 9/27/24

Section: CPE21S4

Date Submitted: 9/27/24

Name(s): Santos, Andrei R.

Instructor: Professor Maria Rizette Sayo

6. Output

Screenshot

```
main.cpp
1  #include <iostream>
2  #include <utility>
3
4  class Node
5  {
6      public:
7          char data;
8          Node *next;
9  };
10
11 int main()
12 // step 1
13 {
14     Node *head = NULL;
15     Node *second = NULL;
16     Node *third = NULL;
17     Node *fourth = NULL;
18     Node *fifth = NULL;
19     Node *last = NULL;
20
21 //step 2
22     head = new Node;
23     second = new Node;
24     third = new Node;
25     fourth = new Node;
26     fifth = new Node;
27     last = new Node;
28
29 //step 3
30     head -> data = 'C';
31     head -> next = second;
32
33     second -> data = 'P';
34     second -> next = third;
35
36     third -> data = 'E';
37     third -> next = fourth;
38
39     fourth -> data = '0';
40     fourth -> next = fifth;
41
42     fifth -> data = '1';
43     fifth -> next = last;
44
45 //step 4
46     last -> data = '0';
47     last -> next = nullptr;
48 }
49
...Program finished with exit code 0
Press ENTER to exit console.
```

Discussion	After I completed coding the program, it just needs a simple output since we can see there is no cout wherein the cout helps to show the output of the program after compiling.
------------	---

Table 3-1. Output of Initial/Simple Implementation

Operation	Screenshot
Traversal	<pre> 1 #include <iostream> 2 using namespace std; 3 4 class Node { 5 public: 6 char data; 7 Node *next; 8 }; 9 10 void ListTraversal(Node *n) 11 { 12 // while n is not equal to 13 while (n != nullptr) 14 { 15 // Print n 16 cout << n->data << " "; 17 // Go to next node n := next 18 n = n->next; 19 } 20 // print 21 cout << endl; 22 } 23 </pre>
Insertion at head	<pre> 24 void InsAtHead(Node *&head, char data) 25 { 26 // Allocate memory for the new node 27 Node *newNode = new Node; 28 29 // Put our data into the new node by applying it to the new data 30 newNode->data = data; 31 32 // Set Next of the new node to point to the previous Head 33 newNode->next = head; 34 35 // Change the head now to make it the point to the new node 36 head = newNode; 37 } 38 </pre>

Insertion at any part of the list

```
40 void InsAnyPart(Node *&head, char data, int position)
41 {
42     Node *newNode = new Node;
43     newNode->data = data;
44
45     // deciding what position could the any part in the output
46     if (position == 1)
47     {
48         newNode->next = head;
49         head = newNode;
50         return;
51     }
52
53     Node *temp = head;
54     for (int i = 1; i < position - 1 && temp != nullptr; i++)
55     {
56         temp = temp->next;
57     }
58
59     if (temp == nullptr)
60     {
61         cout << "Previous node cannot be null" << endl;
62         return;
63     }
64
65     newNode->next = temp->next;
66     temp->next = newNode;
67 }
68
```

Insertion at the end

```
69 // Entering the insertion at end of the output
70 void InsAtEnd(Node *&head, char data)
71 {
72     Node *newNode = new Node;
73     newNode->data = data;
74     newNode->next = nullptr;
75
76     if (head == nullptr)
77     {
78         head = newNode;
79         return;
80     }
81
82     Node *temp = head;
83     while (temp->next != nullptr)
84     {
85         temp = temp->next;
86     }
87
88     temp->next = newNode;
89 }
90
```

Deletion of a node

```

91 // now, the deletion of the chosen character
92 void DelNo(Node *&head, char data)
93 {
94     if (head == nullptr) {
95         cout << "This is an Empty list" << endl;
96         return;
97     }
98
99     if (head->data == data)
100     {
101         Node *temp = head;
102         head = head->next;
103         delete temp;
104         return;
105     }
106
107     Node *temp = head;
108     while (temp->next != nullptr)
109     {
110         if (temp->next->data == data)
111         {
112             Node *nodeToDelete = temp->next;
113             temp->next = temp->next->next;
114             delete nodeToDelete;
115             return;
116         }
117         temp = temp->next;
118     }
119
120     cout << "The node was not found." << endl;
121 }

```

Table 3-2. Code for the List Operations

a	Source Code	<pre> 115 // Task a final output is CPE101 116 cout << "Task a (Initial traversal of the list): "; 117 ListTraversal(head); </pre>
	Console	Task a (Initial traversal of the list): C P E 1 0 1
b	Source Code	<pre> 119 // Task b final output is GCPE101 120 InsAtHead(&head, 'G'); 121 cout << "Task b (Insert 'G' at the start): "; 122 ListTraversal(head); 123 </pre>
	Console	Task b (Insert 'G' at the start): G C P E 1 0 1
c	Source Code	<pre> 124 // Task c final output is GCPEE101 125 Node* temp = head; 126 while (temp != nullptr && temp->data != 'P') // Find node with 'P' 127 { 128 temp = temp->next; 129 } </pre>
	Console	Task c (Insert 'E' after 'P'): G C P E E 1 0 1
d	Source Code	<pre> 134 // Task d final output is GP EE101 135 deleno(&head, 'C'); 136 cout << "Task d (Delete node with 'C'): "; 137 ListTraversal(head); </pre>

	Console	Task d (Delete node with 'C'): G P E E 1 0 1
e	Source Code	<pre> 139 // Task e final ouput is GEE101 140 deleno(&head, 'P'); 141 cout << "Task e (Delete node with 'P'):"; 142 ListTraversal(head); 143 </pre>
	Console	Task e (Delete node with 'P'): G E E 1 0 1
f	Source Code	<pre> 144 // Task f final ouput is GEE101 145 cout << "Task f (Final list traversal): "; 146 ListTraversal(head); 147 </pre>
	Console	Task f (Final list traversal): G E E 1 0 1

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

Screenshots(s)	Analysis
<pre> 14 // The doubly Linked List 15 void ListTraversal(Node* n) { 16 { 17 cout << "Linked List: "; 18 while (n != nullptr) 19 { 20 cout << n->data; // Print the node data 21 if (n->next != nullptr) { 22 cout << " <-> "; // Print the symbol for the arrow back and forth 23 } 24 n = n->next; // Move it to the next 25 } 26 cout << endl; // Print the new Line output 27 } 28 </pre>	<p>In this function traversal, showing the data we can see the symbol that I used, the <-> which represents the bidirectional links of the codes.</p>
<pre> 29 // Function to insert a new node at the head 30 void InsAtHead(Node** head, char new_data) 31 { 32 Node* new_node = new Node(); // Create a node for the head to be inserted 33 new_node->data = new_data; // Set data 34 new_node->next = (*head); // Link new node to the current head 35 new_node->prev = nullptr; // Set prev (or previous) to nullptr 36 if ((*head) != nullptr) { 37 (*head)->prev = new_node; // updated prev 38 } 39 (*head) = new_node; // Move head to the new node 40 } 41 </pre>	<p>Here, I updated it with the prev and next, which are the previous of the node and the next of the node letting and making the list updated.</p>
<pre> 42 // Inserting a new node at any position in the doubly Linked list 43 void InsAtPos(Node* prev_node, char new_data) 44 { 45 if (prev_node == nullptr) 46 { 47 cout << "Previous node cannot be null." << endl; 48 return; 49 } 50 Node* new_node = new Node(); // Create a node for any postion 51 new_node->data = new_data; // Set data 52 new_node->next = prev_node->next; // Link new node to the next 53 new_node->prev = prev_node; // Link the new node to the prev 54 if (prev_node->next != nullptr) 55 { 56 prev_node->next->prev = new_node; // Update prev of the next node 57 } 58 prev_node->next = new_node; // Link it to the new node 59 } 60 </pre>	<p>In this part, I can manipulate the desired data that I can put inside the doubly linked lists at the specified position, still it links to the previous up to the next one.</p>

```

61 // Inserting a new node at the end of the doubly linked list
62 void InsAtEnd(Node** head, char new_data)
63 {
64     Node* new_node = new Node(); // Create a node at the end
65     new_node->data = new_data; // Set data
66     new_node->next = nullptr; // This is now the final node
67     new_node->prev = nullptr; // Set prev to nullptr since it's the end
68
69     // If empty, set the head to the new node (follow this condition for if)
70     if (*head == nullptr) {
71         *head = new_node;
72         return;
73     }
74
75     // Else, go back to the previous/last node (follow this condition for else)
76     Node* last = *head;
77     while (last->next != nullptr)
78     {
79         last = last->next;
80     }
81
82     last->next = new_node; // Link the last node to the new node
83     new_node->prev = last; // Updated node
84 }
85

```

With the use of the prev and next pointers to the existing and new node, it creates and inserts from the beginning to the end of the node until it reaches to the null

```

86 // Deleting a node from the doubly linked list
87 void deleno(Node** head, char key)
88 {
89     Node* temp = *head;
90     Node* prev = nullptr;
91
92     if (temp != nullptr && temp->data == key)
93     {
94         *head = temp->next; // Change the head to the next node
95         if (*head != nullptr) {
96             (*head)->prev = nullptr; // Update prev of the new head
97         }
98         delete temp; // delete the old head
99         return;
100     }
101
102     while (temp != nullptr && temp->data != key)
103     {
104         prev = temp;
105         temp = temp->next;
106     }
107
108     if (temp == nullptr)
109     {
110         cout << "Key not found." << endl;
111         return;
112     }
113
114     prev->next = temp->next;
115     if (temp->next != nullptr)
116     {
117         temp->next->prev = prev; // Update the previous to the new code
118     }
119     delete temp; // delete the temporary data in the Linked List
120 }
121

```

Lastly, I can now delete a specific node, which I can choose from the given with the data. With the help of the prev and next, I can delete a node maintaining the lists links from its previous and next nodes.

Table 3-4. Modified Operations for Doubly Linked Lists

7. Supplementary Activity

```

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

class Node {
public:
    string song;
    Node* next;
    Node(string s) : song(s), next(nullptr) {}
};

class Music_opm_list {
    Node* head = nullptr;
public:
    void AddSong(string s) {
        Node* new_node = new Node(s);
    }
}

```

```

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

void RemoveSong(string s) {
    if (!head) return;
    Node* temp = head, *prev = nullptr;
    if (head->song == s) {
        if (head->next == head) { delete head; head = nullptr; }
        else {
            while (temp->next != head) temp = temp->next;
            temp->next = head->next;
            delete head; head = temp->next;
        }
        return;
    }
    do {
        prev = temp; temp = temp->next;
    } while (temp != head && temp->song != s);
    if (temp != head) { prev->next = temp->next; delete temp; }
}

void dispsong() {
    if (!head) return;
    Node* temp = head;
    do {
        cout << "• " << temp->song << endl;
        temp = temp->next;
    } while (temp != head);
}

};

int main() {
    Music_opm_list playlist;

    // Create OPM song playlist
    cout << "Playlist: my Favorite OPM Songs\n";
    playlist.AddSong("Dahan Dahan");
    playlist.AddSong("Bawat Piyesa");
    playlist.AddSong("Sino");
    playlist.dispsong();

    // Add a song to the playlist
    cout << "\nAdding song: Pwede Ba\n";
}

```

```

playlist.AddSong("Pwede Ba");
playlist.dispsong();

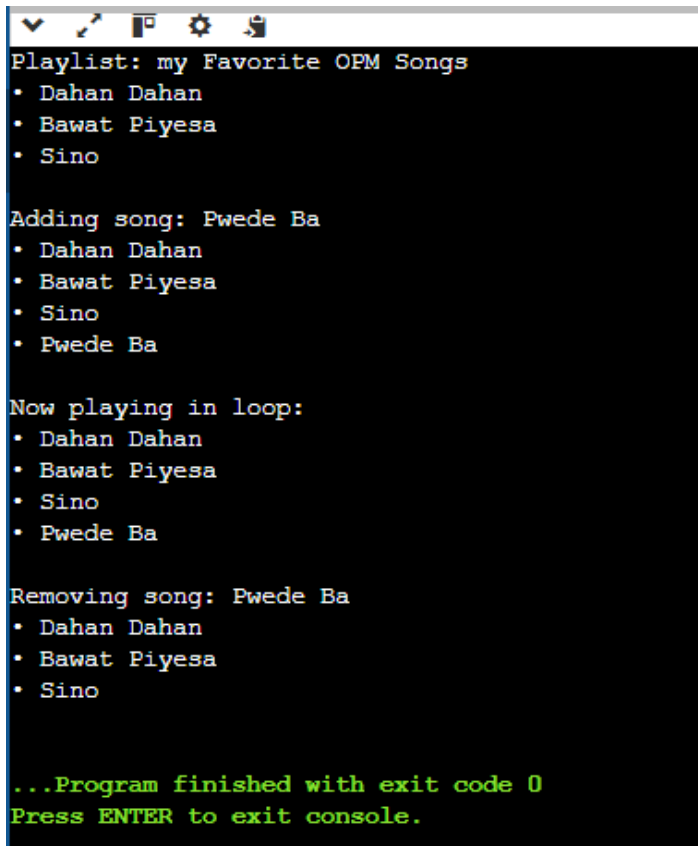
// Playing the song in loop
cout << "\nNow playing in loop: \n";
playlist.dispsong();

// Remove the song "Pwede Ba"
cout << "\nRemoving song: Pwede Ba\n";
playlist.RemoveSong("Pwede Ba");
playlist.dispsong();

return 0;
}

```

The output:



```

Playlist: my Favorite OPM Songs
• Dahan Dahan
• Bawat Piyesa
• Sino

Adding song: Pwede Ba
• Dahan Dahan
• Bawat Piyesa
• Sino
• Pwede Ba

Now playing in loop:
• Dahan Dahan
• Bawat Piyesa
• Sino
• Pwede Ba

Removing song: Pwede Ba
• Dahan Dahan
• Bawat Piyesa
• Sino

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

```

8. Conclusion

I have learned how to insert and delete data in nodes using linked lists, understanding the role of head and null. I also realized that singly linked lists and doubly linked lists share similarities, but differ in traversal part. Singly linked lists follow a linear path, while doubly linked lists allow backward and forward movement. In terms of efficiency, singly linked lists are better suited for simpler tasks, while doubly linked lists are ideal for more complex operations, it shows greater flexibility and requires more memory and run time. The supplementary activity demonstrated how to implement adding or deleting data in a playlist or list.

This activity was a bit challenging for me at first since it introduced new concepts and tasks I wasn't familiar with. However, through my own understanding and a little bit of research, I was able to grasp the use of singly and doubly linked lists, particularly how the prev and next pointers function. By working through the steps of adding and deleting songs in the code.

9. Assessment Rubric