Don Eleazar T. Fernandez
CPE21S4, CPE 010
Hands-on Activity 3.1 Linked Lists (Code)

Table 3 - 1:

```
#include<iostream>
#include<utility>
class Node{
public:
char data;
Node *next;
};
int main(){
//step 1
Node *head = NULL;
20;
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;
//step 3
head->data = 'C';
head->next = second;
second->data = 'P';
second->next = third;
third->data = 'E';
third->next = fourth;
fourth->data = '0';
fourth->next = fifth;
fifth->data = '1';
fifth->next = last;
//step 4
last->data = '0';
last->next = nullptr;
}
```

Table 3 - 2:

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

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

void listTraversal(Node* n) {
   while (n != nullptr) {
      cout << n -> data << " ";
      n = n -> next;
   }
   cout << endl;
}

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

void inAtAny(Node* prevNode, char newData) {
   if (prevNode == nullptr) {
      cout << "Previous node cannot be null." << endl;
      return;
   }
   Node* newNode = new Node();
   newNode -> data = newData;
   newNode -> next = prevNode -> next;
   prevNode -> next = newNode;
}

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

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

void delNode(Node*& head, char 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;
}

int main() {
    // Step 1
    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;
    // Step 3
    head->data = 'C';
    head->next = second;
```

```
    second->data = 'P';
    second->next = third;
    third->data = 'E';
    third->next = fourth;
    fourth->data = '0';
    fourth->next = fifth;
    fifth->data = '1';
    fifth->next = last;
    // Step 4
    last->data = '0';
    last->next = nullptr;

    inAtHead(head, '1');
    inAtAny(head -> next, '2');
    inAtEnd(head, '3');
    delNode(head, 'C');
    listTraversal(head);

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

    return 0;
}
```

Table 3 - 3:

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

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

void listTraversal(Node* n) {
    while (n != nullptr) {
        cout << n -> data << " ";
        n = n -> next;
    }
```

```
    cout << endl;
}

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

void inAtAny(Node* prevNode, char newData) {
    if (prevNode == nullptr) {
        cout << "Previous node cannot be null." << endl;
        return;
    }
    Node* newNode = new Node();
    newNode -> data = newData;
    newNode -> next = prevNode -> next;
    prevNode -> next = newNode;
}

void inAtEnd(Node* & head, char newData) {
    Node* newNode = new Node();
    newNode -> data = newData;
    newNode -> next = nullptr;
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* last = head;
    while (last -> next != nullptr) {
        last = last -> next;
    }
    last -> next = newNode;
}

void delNode(Node*& head, char 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;
}

int main() {
    // Step 1
    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;
    // Step 3
    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;

    listTraversal(head);
    cout << endl << "Final Output: " << endl;
    inAtAny(head -> next, 'E');
    inAtHead(head, 'G');
    delNode(head, 'C');
```

```
    delNode(head, 'P');

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

    return 0;
}
```

Supplementary Activty:

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

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

void listTraversal(Node* head) {
    if (head == nullptr) {
        cout << "Playlist is empty!" << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp -> songName << " -> ";
        temp = temp -> next;
    } while (temp != head);
    cout << "(back to start)" << endl;
}

void inAtHead(Node* &head, string newSong) {
    Node* newNode = new Node();
    newNode -> songName = newSong;
    if (head == nullptr) {
        newNode -> next = newNode;
        head = newNode;
    } else {
        Node* temp = head;
```

```
        while (temp -> next != head) {
            temp = temp -> next;
        }
        newNode -> next = head;
        temp -> next = newNode;
        head = newNode;
    }
}

void inAtEnd(Node* &head, string newSong) {
    Node* newNode = new Node();
    newNode -> songName = newSong;
    if (head == nullptr) {
        newNode -> next = newNode;
        head = newNode;
    } else {
        Node* temp = head;
        while (temp -> next != head) {
            temp = temp -> next;
        }
        temp -> next = newNode;
        newNode -> next = head;
    }
}

void delNode(Node*& head, string songName) {
    if (head == nullptr) return;

    Node *temp = head, *prev = nullptr;

    if (temp != nullptr && temp -> songName == songName) {
        if (temp -> next == head) {
            delete temp;
            head = nullptr;
            return;
        }

        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 -> songName != songName);

    if (temp == head) return;

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

int main() {
    Node *head = nullptr;
    inAtEnd(head, "Song A");
    inAtEnd(head, "Song B");
    inAtEnd(head, "Song C");
    inAtEnd(head, "Song D");
    cout << "Initial Playlist: " << endl;
    listTraversal(head);

    inAtHead(head, "Song X");
    cout << "\nAfter adding 'Song X' at the head: " << endl;
    listTraversal(head);

    delNode(head, "Song B");
    cout << "\nAfter deleting 'Song B': " << endl;
    listTraversal(head);

    inAtEnd(head, "Song Y");
    cout << "\nAfter adding 'Song Y' at the end: " << endl;
    listTraversal(head);

    return 0;
}
```