Activity No. <n>	
<Replace with Title>	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 09/27/2024**
Section: CPE21S1	**Date Submitted: 09/29/2024**
Name(s): Bautista, Jhon Hendricks T.	**Instructor: Mrs. Maria Rizette Sayo**

6. Output

Screenshot

main.cpp | Run | Output | Clea

```cpp
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;
    return 0;
}
```

```
/tmp/ZsoOEm8Ne9.o


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

Discussion

The linked list was properly initialized and there was no error in creating it. The problem is there is no function or method of code to print the linked list.

Table 3-1. Output of Initial/Simple Implementation

Operation	Screenshot

Traversal	

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

Insertion at head	

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

}
```

Insertion at any part of the list	

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

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

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

    if (temp == nullptr) {
        cout << "Position out of bounds. Node not inserted." << endl;
        delete newNode;
        return;
    }

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

Insertion at the end	

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

Table 3-2. Code for the List Operations

a	**Source Code**	

```
cout << "Original List: ";
traversal(head);
```

	Console	

```
Original List: C P E 0 1 0
```

b	**Source Code**	

```
insertAtHead('G', head);
cout << "After Insert at Head: ";
traversal(head);
```

	Console	

```
After Insert at Head: G C P E 0 1 0
```

c	**Source Code**	

```
insertAtPosition('E', 3, head);
cout << "After Insert at Position 3: ";
traversal(head);
```

	Console	`After Insert at Position 3: G C P E E 0 1 0`

d	**Source Code**	`deleteNode('C', head);`
		`cout << "After Deleting: ";`
		`traversal(head);`

	Console	`After Deleting: G P E E 0 1 0`

e	**Source Code**	`deleteNode('P', head);`
		`cout << "After Deleting: ";`
		`traversal(head);`

	Console	`After Deleting: G E E 0 1 0`

f	**Source Code**	`deleteNode('P', head);`
		`cout << "After Deleting: ";`
		`traversal(head);`

	Console	`After Deleting: G E E 0 1 0`

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

Screenshot(s)	Analysis

`void traversal(Node* head) {`
`  while (head != nullptr) {`
`  cout << head->value << " ";`
`  head = head->next;`
`  }`
`  cout << endl;`
`}`
	In this code, it shows the function needed to traverse a double link list. we can see that it is similar to how a single link list is traversed by using the head node.

```cpp
void insertAtHead(int value, Node*& head) {
    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = nullptr;

    if (head != nullptr) {
        head->prev = newNode;
    }
    head = newNode;
}
```

In this picture, it is a function used to insert a node and value at the start of the double linked list. Now this is different from a single link list. The head node and previous node need to be tracked so that we are inserted at the head.

```cpp
void insertAtEnd(int value, Node*& head) {
    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        newNode->prev = nullptr;
        head = newNode;
        return;
    }

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

In this code we can see a function created to insert a node and value at the end of the list. In a double linked list we need to keep track of the next and previous nodes because they are connected. we used a temporary node to find the end of the list and then inserted a node once found by the while condition.

```cpp
void insertAtAnyPosition(int value, Node*& head, int position) {
    Node* newNode = new Node();
    newNode->value = value;

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

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

    newNode->next = current->next;
    newNode->prev = current;

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

    current->next = newNode;
}
```

The picture shows the function for inserting at any position in a double linked list. There is some similarities in a single linked list but here we need to keep track and update accordingly the previous and current nodes to properly find the right position and insert a new node in the list.

```cpp
void deleteNode(int value, Node*& head) {
    Node* current = head;

    while (current != nullptr) {
        if (current->value == value) {
            if (current == head) {
                head = current->next;
                if (head != nullptr) {
                    head->prev = nullptr;
                }
            } else {
                current->prev->next = current->next;
                if (current->next != nullptr) {
                    current->next->prev = current->prev;
                }
            }
            delete current;
            return;
        }
        current = current->next;
    }
    cout << "Node with value " << value << " not found" << endl;
}
```

For the deletion of a node in a double link list we need to traverse the link list and find the selected node. Then we see if statements used to check and then apply the right operations to delete and relink the nodes after a deletion of a node in the list. we need to update it properly so as to not cause a segmentation fault.

Table 3-4. Modified Operations for Doubly Linked Lists

7. Supplementary Activity

```cpp
};


void traversal(Node* head) {
    int song_num = 1;
    cout << "Your songs:" << endl;
    while (head != nullptr) {
        cout << song_num << " " << head->song << endl;
        song_num++;
        head = head->next;
    }
};


void addsong(string song, Node*& head){
    Node* newNode = new Node();
    newNode -> song = song;
    newNode -> next = head;
    head = newNode;
    cout<<"song added SUCCESSFULY"<<endl;
}

void deletesong(string song, Node*& head){
    Node* current = head;
    Node* prev = nullptr;

    while(current != nullptr){
        if(current -> song == song ){
            if(current == head){
                head = head -> next;
            }
            else{
                prev -> next = current -> next;
            }
            delete current;
            current = prev ? prev -> next : head;
        }
        else{
            prev = current;
            current = current -> next;
        }
    }
```

```
Enter your song: faded

=============MUSIC PLAYER==========
[1] view playlist
[2] add song to playlist
[3] delete a song from playlist
[4] End Program
Enter your choice: 3

DELETE SONG
Your songs:
1 faded
Select song to delete: faded
=======DELETING======
PLAYLIST UPDATED SUCCESSFULY

=============MUSIC PLAYER==========
[1] view playlist
[2] add song to playlist
[3] delete a song from playlist
[4] End Program
Enter your choice: 1

Your songs:

=============MUSIC PLAYER==========
[1] view playlist
[2] add song to playlist
[3] delete a song from playlist
[4] End Program
Enter your choice: 4
=====STREAMING ENDED=====

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

8. Conclusion

After this activity, I was able to learn about C++ syntax and how to create a linked list. Linked List is a kind of a linear data structure. In this activity, I was able to understand how a linked list is structured. I learned that linked lists are dynamic and they are created by usage of pointers. The activity started first by stating the initialization of the list and then explained the different operations. I have practiced basic operations on linked lists, such as traversing the list to display the songs, adding a new song to the head of the list, and deleting a song. This experience taught me how to manage pointers and understand the underlying structure of the list.

9. Assessment Rubric