ACTIVITY NO. 3

LINKED LISTS	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:**
Section: CPE21S4	**Date Submitted:**
Name: Cruz, Axl Waltz E.	**Instructor:** Engr. Jimlord Quejado

1. Objective(s)

- Implement the list ADT using singly and doubly linked lists
- Define operations based on list ADT from the module discussion

2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:

a. Construct C++ code for a singly and doubly linked list in C++
b. Solve given problems utilizing linked lists in C++

3. Discussion

PART A: What is a linked list?

Linked Lists are a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

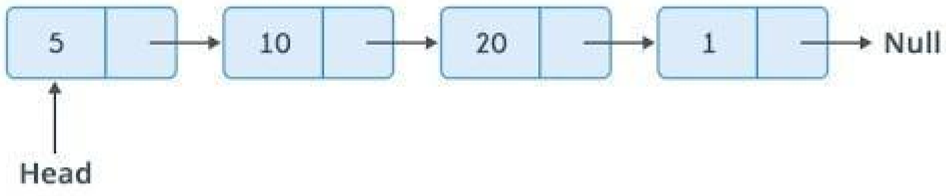


Image Source: Hackearth.com

Why Linked List?

- Arrays are useful but have the following limitations:
- Fixed size.
- Allocated memory remains to be the upper limit.
- Expensive operations (such as insertion) which requires movement of all existing elements and creation of room for new elements.

Advantages over Arrays

- Dynamic size
- Ease of insertion/deletion

Drawbacks

- Random access is not allowed. We have to access elements sequentially starting from the first node
- Extra memory space for a pointer is required with each element of the list.
- Not cache friendly. No locality reference.

Representation:

- A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.
- Each node in a list consists of at least two parts:
    - Data
    - Pointer (Or Reference) to the next node

PART B: Doubly Linked Lists

Doubly Linked Lists are traversed in either direction. It is a linked list in which every node has a next pointer and a backpointer.

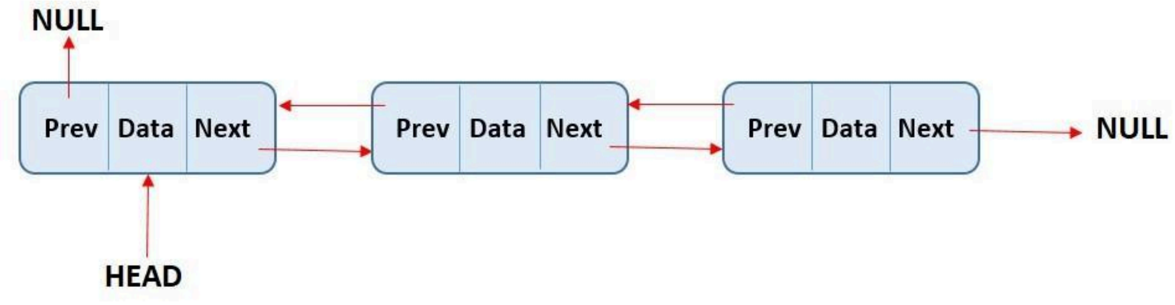


Image Source: AlphaCodingSkills

Every node contains address of next node (except the last node). Every node contains address of previous node (except the first node).

PART C: Common Operations on

Linked Lists Typical operations:

- Initialize the list
- Destroy the list
- Determine if list empty
- Search list for a given item
- Insert an item
- Delete an item, and so on

4. Materials and Equipment

Personal Computer with C++ IDE
Recommended IDE:

- CLion (must use TIP email to download)
- DevC++ (use the embarcadero fork or configure to C++17)

5. Procedure

ILO A: Construct C++ code for a singly and doubly linked list in C++

A.1. Singly Linked List

To start, we will do a simple implementation of a linked list. We must keep in mind the visual representation of the individual nodes that will make up our linked list:

Data	Next
28	
Contains the the data of the of node	Contains address the next node

Every node in a singly linked list will have 2 compartments, the data and the pointer to the next. The data contains the element of a single type that we want it to contain. The link will point to the next node in our list. In C++, the node is defined as:

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

We will implement a list to represent the string “CPE010” that looks like the figure below:

C → P → E → 0 → 1 → 0 → NULL

Implementation will follow the given steps:

1. Create the node pointers and initialize as NULL
2. Create new instances of the node class and allocate them in the heap
3. Define every node in the list
4. Point the last node

to null Simple

implementation:

```
#include<iostream>
#include<utility>

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

    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;
    }
```

Although we have created the linked list, this is an implementation that is useless and meant only to show what we want to happen for the given output. **Run your code and screenshot the output, then briefly discuss how the output came to be and how could it be improved (table 3-1)?**

Imagine if we had to make multiple items in the list in the hundreds or thousands! This would be too tedious, ineffective, and inefficient. So, we must implement certain methods. These methods/operations associated with the linked lists are:

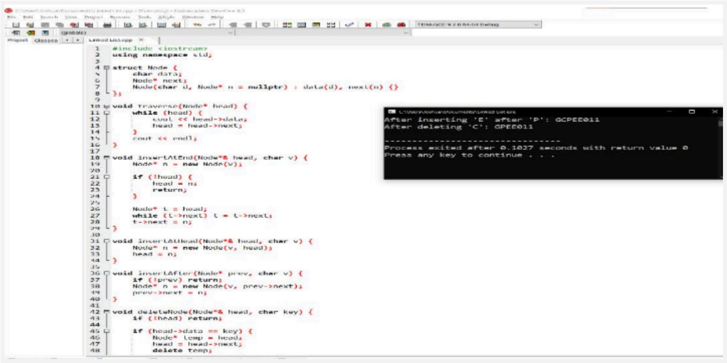
- Traversal
- Insertion at head
- Insertion at any part of the list
- Insertion at the end
- Deletion of a

node Linked List

Traversal

```
    Algorithm: ListTraversal (parameter: pointer to node n)
```

```
WHILE n IS NOT EQUAL TO null
      PRINT data OF n
      GO TO NEXT NODE n := next
      ENDWHILE
```



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

```
    PRINT next line
    END
```

Sometimes we have a linked list, and we need to insert a node somewhere other than at the end of the list. We will look at a couple of different ways to insert a node into an existing list.

To insert a node at the head:

1. Allocate memory for the new node
2. Put our data into the new node
3. Set Next of the new node to point to the previous Head
4. Reset Head to point to the new node

To insert a node at any location between the head and tail:

1. Check if it is the head node (previous node is null)
2. If null, print "Previous node cannot be null."
3. Allocate a new node
4. Store data in the new node
5. Point new node to the node previous node was pointing to
6. Point previous node to the new node

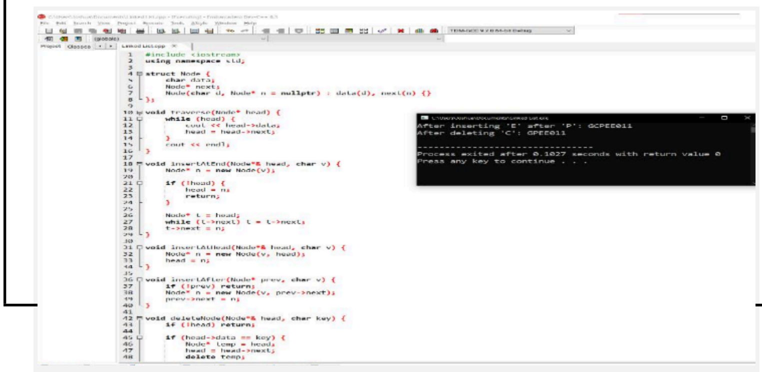
To insert a node at the end:

1. Allocate new node
2. Dereference to the head node
3. Store data in new node
4. Point next of new node to NULL
5. Traverse the list until next of the node is null
6. Point the next of the current node to the new node

To delete a node from linked list:

1. Find previous node of the node to be deleted.
2. Change the next of previous node.
3. Free memory for the node to be deleted.

Create code for each of the pseudocode given for all list operations above. Provide screenshots in table 3-2.

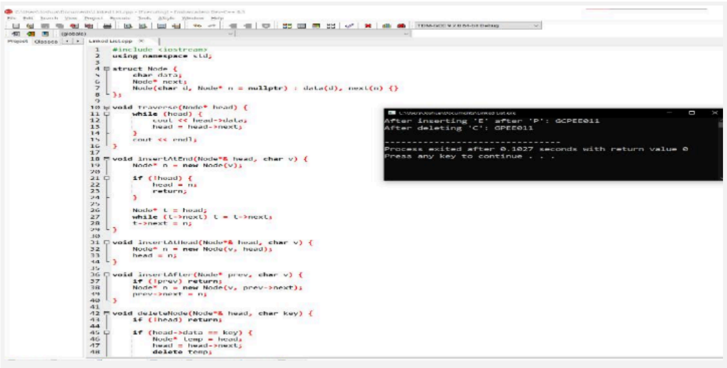


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

In your driver function, show the use of each list operation and show the output in table 3-3 found in section 6 with a descriptive caption for each. The tasks you have to perform are as follows:

a. Traverse the list by passing the head of the created list into the function
b. Insert the element ‘G’ at the start of the list to replace the current node. Output should now show “GCPE101”
c. Insert an element “E” with the previous node element being “P”. Output should now show “GCPEE101”.
d. Delete the node containing the element C.



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

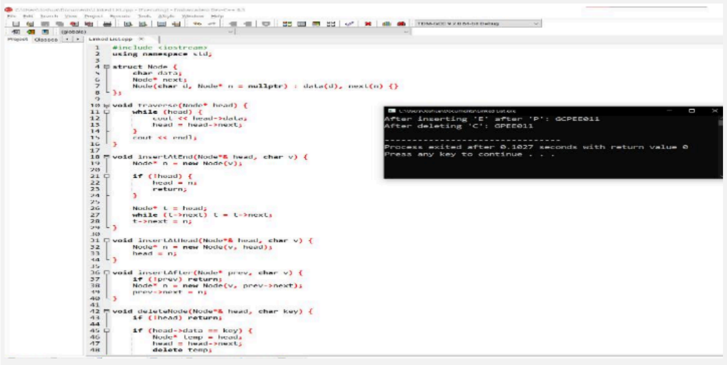
e. Delete the node containing the element P.
f. Show the elements in the list. Output should be "GEE101".

A.2. Doubly Linked List

The singly linked list allows for direct access from a list node only to the next node in the list. A doubly linked list allows convenient access from a list node to the next node and also to the preceding node on the list.

The doubly linked list node accomplishes this in the obvious way by storing two pointers: one to the node following it (as in the singly linked list), and a second pointer to the node preceding it.

Prev	Data	Next
←	28	→
Contains the address of the previous node	Contains the data of the node	Contains the addres



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

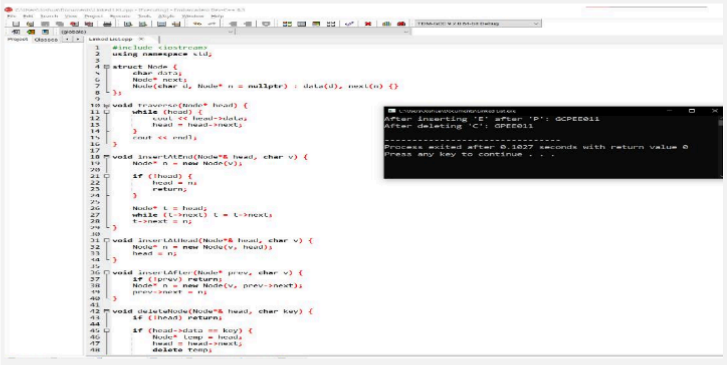
softh	enext	node

This means that in addition to our implementation of the singly linked list, we'll have addition compartment.

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

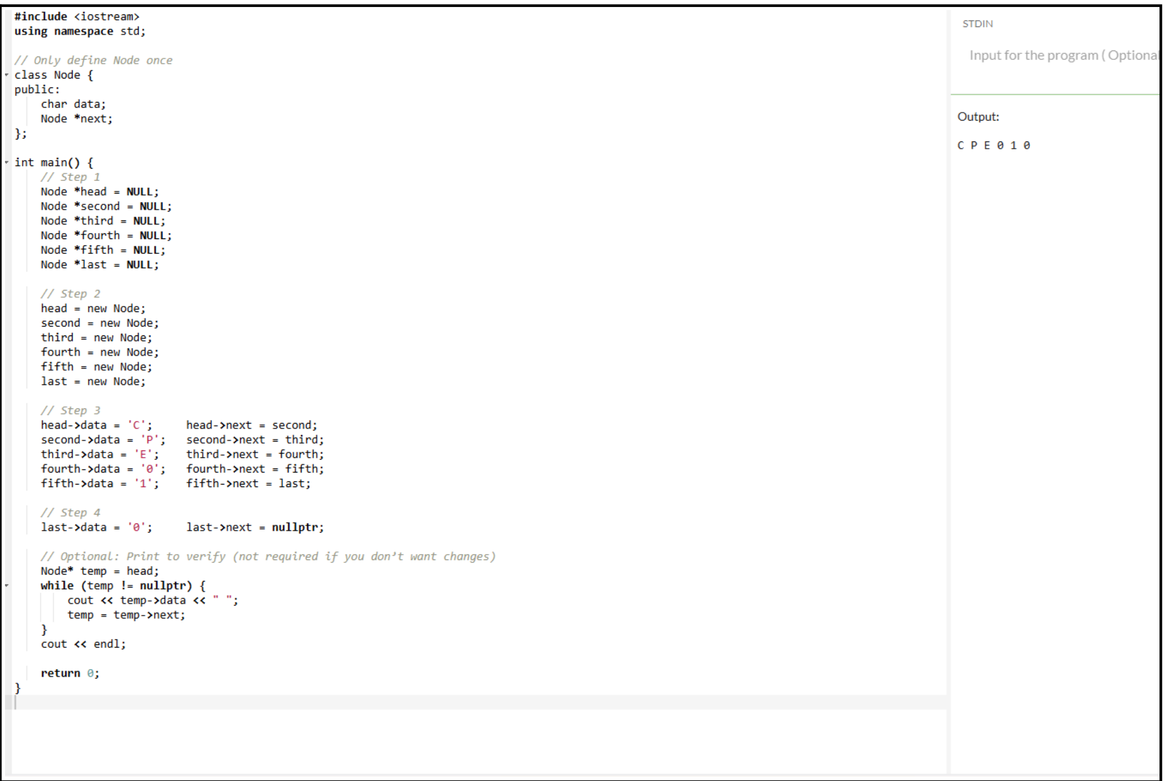
Modify the given operations used in the singly linked lists to work on the new construct of a doubly linked list. Provide a screenshot of your code and analysis in table 3-4.

6. Output



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

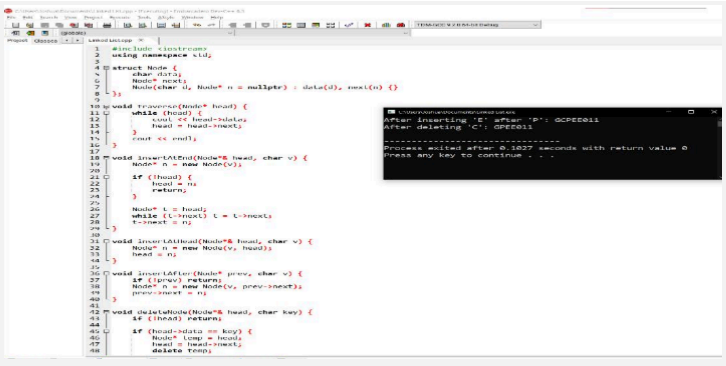
Screenshot	
Discussion	This list is properly constructed, each node is correctly linked and the data forms the string when read sequentially, to improve the code encapsulation with a link list could do and memory management which is a destructor to allocate memory and never freeing it.

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

// Only define Node once
class Node {
public:
    char data;
    Node *next;
};

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;

    // Optional: Print to verify (not required if you don’t want changes)
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;

    return 0;
}
```

```
C P E 0 1 0
```

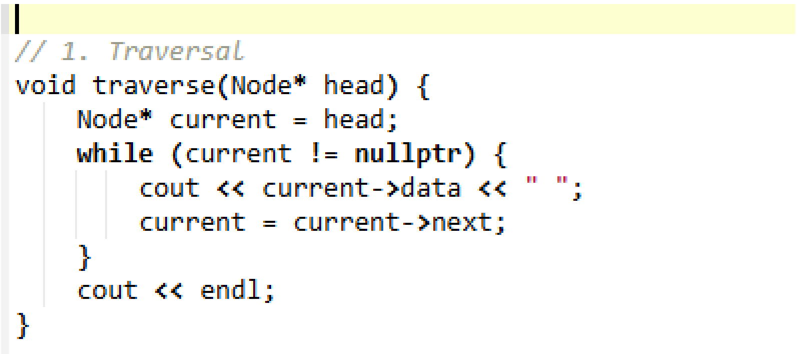
Table 3-1. Output of Initial/Simple Implementation

Operation	**Screenshot**

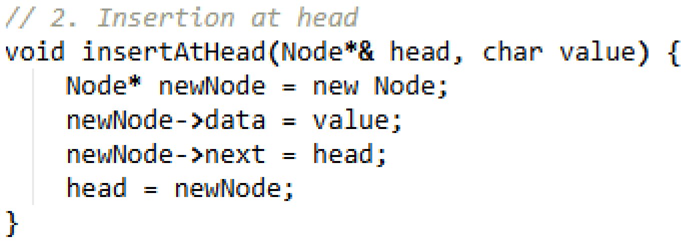


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

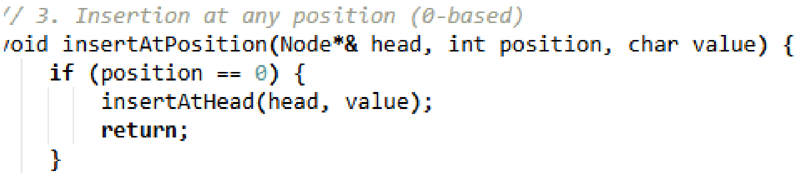
```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

<table>
<tr>
<td>Traversal</td>
<td>


```
// 1. Traversal
void traverse(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}
```

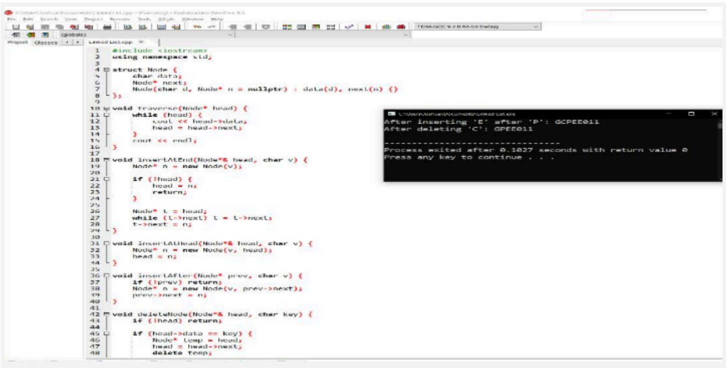
</td>
</tr>
<tr>
<td>Insertion at head</td>
<td>


```
// 2. Insertion at head
void insertAtHead(Node*& head, char value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}
```

</td>
</tr>
<tr>
<td>Insertion at any part of the list</td>
<td>


```
'/ 3. Insertion at any position (0-based)
/oid insertAtPosition(Node*& head, int position, char value) {
    if (position == 0) {
        insertAtHead(head, value);
        return;
    }
```

</td>
</tr>
</table>

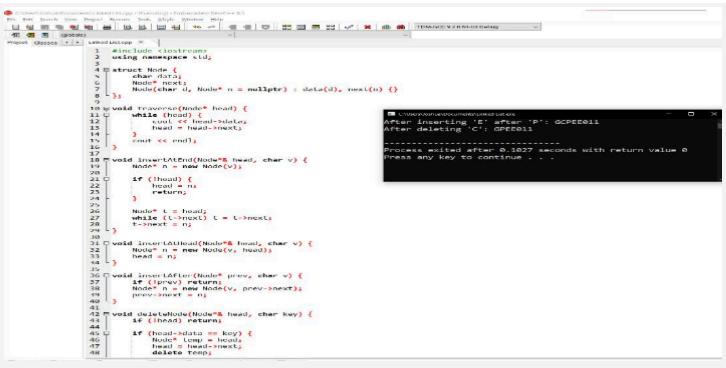


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

Insertion at the end

```
// 4. Insertion at end
void insertAtEnd(Node*& head, char value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

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

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

    temp->next = newNode;
}
```

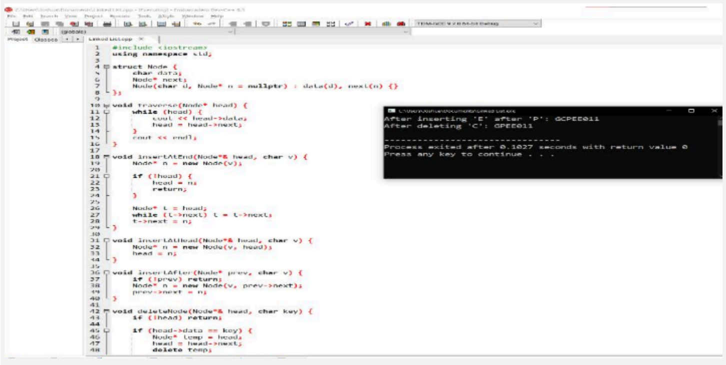


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

Deletion of a node	`// 5. Deletion of a node by value`
	`void deleteNode(Node*& head, char value) {`
	`    if (head == nullptr) return;`
	
	`    if (head->data == value) {`
	`        Node* temp = head;`
	`        head = head->next;`
	`        delete temp;`
	`        return;`
	`    }`
	
	`    Node* prev = head;`
	`    Node* curr = head->next;`
	
	`    while (curr != nullptr && curr->data != value) {`
	`        prev = curr;`
	`        curr = curr->next;`
	`    }`
	
	`    if (curr == nullptr) {`
	`        cout << "Value not found." << endl;`
	`        return;`
	`    }`
	
	`    prev->next = curr->next;`
	`    delete curr;`
	`}`

Table 3-2. Code for the List Operations

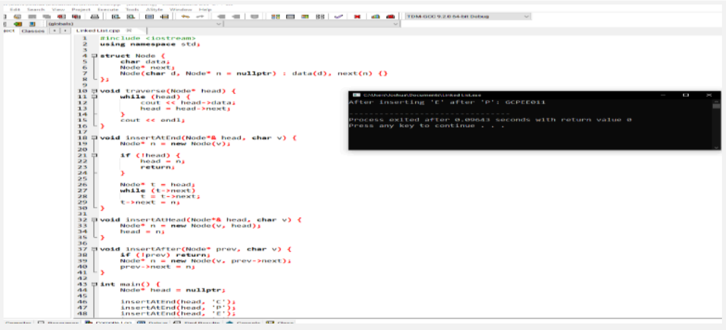


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

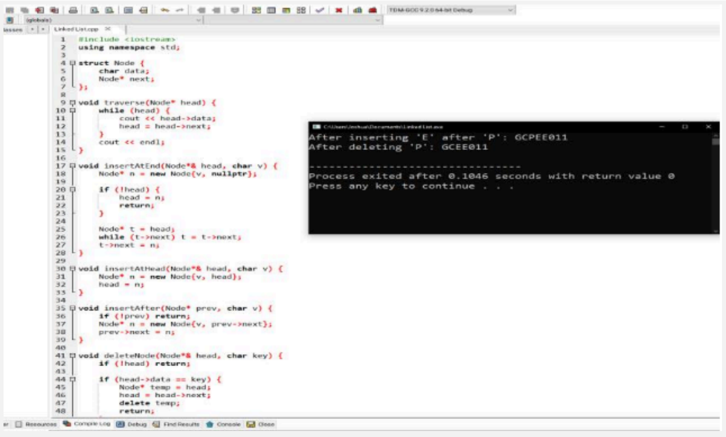
Creates a new node with its next pointer referencing the current head, then updates the head to this new node.

Analysis: It locates the node “P,” creates a new node, a



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next)
        t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

int main() {
    Node* head = nullptr;

    insertAtEnd(head, 'C');
    insertAtEnd(head, 'P');
    insertAtEnd(head, 'E');
```

```
After inserting 'E' after 'P': GCPEE011

--------------------------------
Process exited after 0.09643 seconds with return value 0
Press any key to continue . . .
```

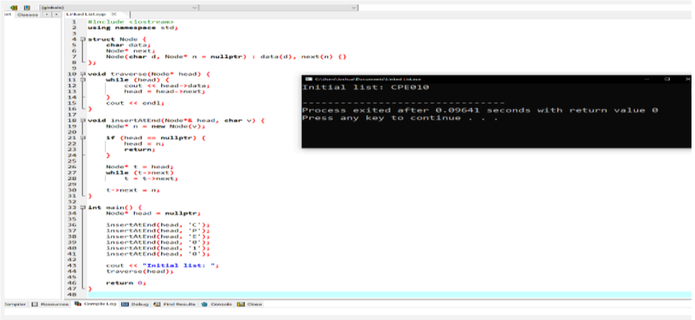


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node{v, nullptr};

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node{v, head};
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node{v, prev->next};
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

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

```
After inserting 'E' after 'P': GCPEE011
After deleting 'P': GCEE011

--------------------------------
Process exited after 0.1046 seconds with return value 0
Press any key to continue . . .
```

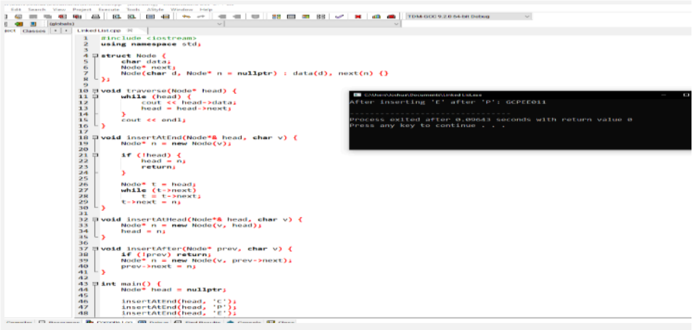
Table 3-3. Code and Analysis for Singly Linked L

The function moves through the list by advancir



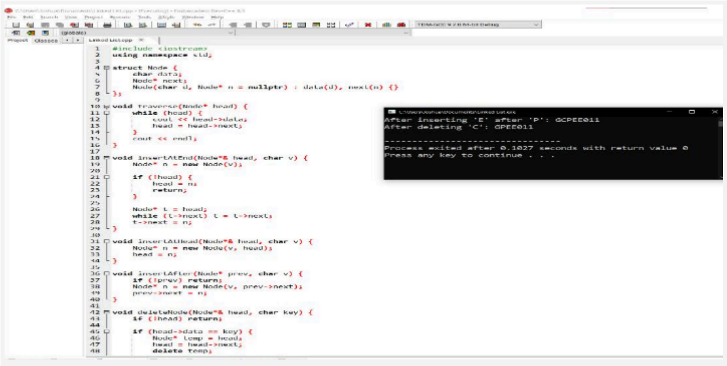
```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

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

    Node* t = head;
    while (t->next)
        t = t->next;

    t->next = n;
}

int main() {
    Node* head = nullptr;

    insertAtEnd(head, 'C');
    insertAtEnd(head, 'P');
    insertAtEnd(head, 'E');
    insertAtEnd(head, '0');
    insertAtEnd(head, '1');
    insertAtEnd(head, '0');

    cout << "Initial list: ";
    traverse(head);

    return 0;
}
```

```
Initial list: CPE010

--------------------------------
Process exited after 0.09641 seconds with return value 0
Press any key to continue . . .
```



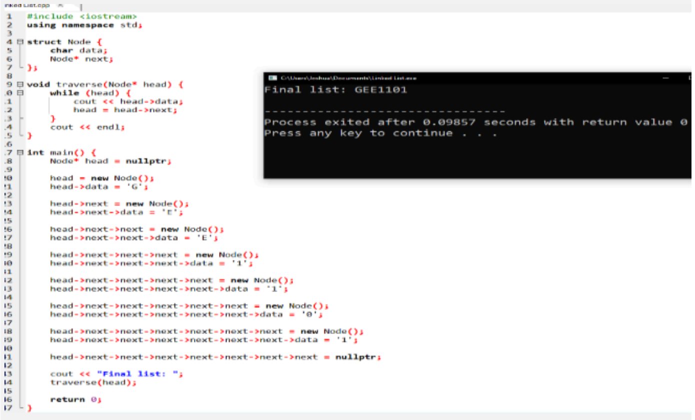
```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next)
        t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

int main() {
    Node* head = nullptr;

    insertAtEnd(head, 'C');
    insertAtEnd(head, 'P');
    insertAtEnd(head, 'E');
```

```
After inserting 'E' after 'P': GCPEE011

--------------------------------
Process exited after 0.09643 seconds with return value 0
Press any key to continue . . .
```



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

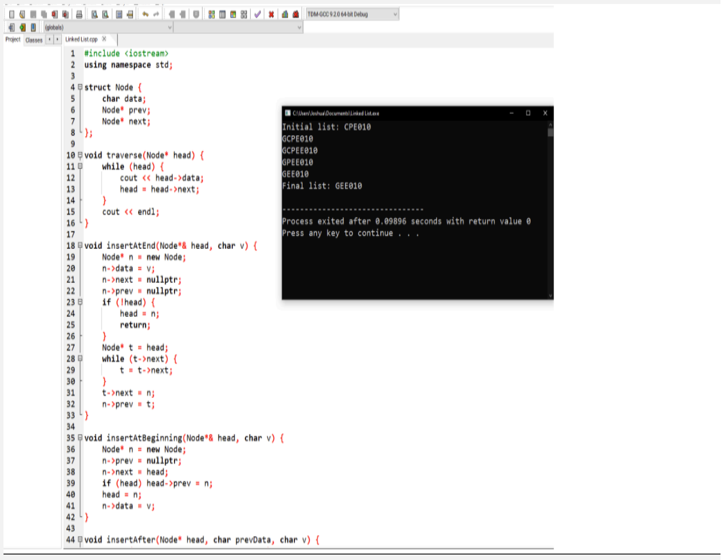
```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```



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

struct Node {
    char data;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;

    head = new Node();
    head->data = 'G';

    head->next = new Node();
    head->next->data = 'E';

    head->next->next = new Node();
    head->next->next->data = 'E';

    head->next->next->next = new Node();
    head->next->next->next->data = '1';

    head->next->next->next->next = new Node();
    head->next->next->next->next->data = '1';

    head->next->next->next->next->next = new Node();
    head->next->next->next->next->next->data = '0';

    head->next->next->next->next->next->next = new Node();
    head->next->next->next->next->next->next->data = '1';

    head->next->next->next->next->next->next->next = nullptr;

    cout << "Final list: ";
    traverse(head);

    return 0;
}
```

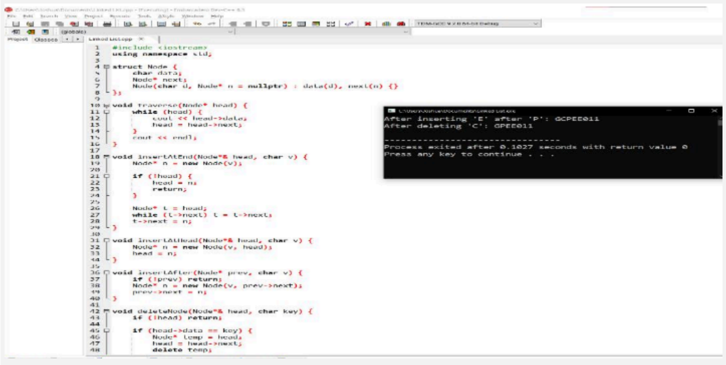
```
Final list: GEE1101

--------------------------------
Process exited after 0.09857 seconds with return value 0
Press any key to continue . . .
```

Table 3-4. Modified Operations for Doubly Linke



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

struct Node {
    char data;
    Node* prev;
    Node* next;
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node;
    n->data = v;
    n->next = nullptr;
    n->prev = nullptr;
    if (!head) {
        head = n;
        return;
    }
    Node* t = head;
    while (t->next) {
        t = t->next;
    }
    t->next = n;
    n->prev = t;
}

void insertAtBeginning(Node*& head, char v) {
    Node* n = new Node;
    n->prev = nullptr;
    n->next = head;
    if (head) head->prev = n;
    head = n;
    n->data = v;
}

void insertAfter(Node* head, char prevData, char v) {
```

```
Initial list: CPE010
GCPE010
GCPEE010
GPEE010
GEE010
Final list: GEE010

--------------------------------
Process exited after 0.09896 seconds with return value 0
Press any key to continue . . .
```



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```


7. Supplementary Activity

ILO B: Solve given problems utilizing linked lists in C++

Problem Title: Implementing a Song Playlist using Linked List
Source: Packt Publishing

Problem Description:
In this activity, we'll look at some applications for which a singly linked list is not enough or not convenient. We will build a tweaked version that fits the application. We often encounter cases where we have to customize default implementations, such as when looping songs in a music player or in games where multiple players take a turn one by one in a circle.

These applications have one common property – we traverse the elements of the sequence in a circular fashion. Thus, the node after the last node will be the first node while traversing the list. This is called a circular linked list.

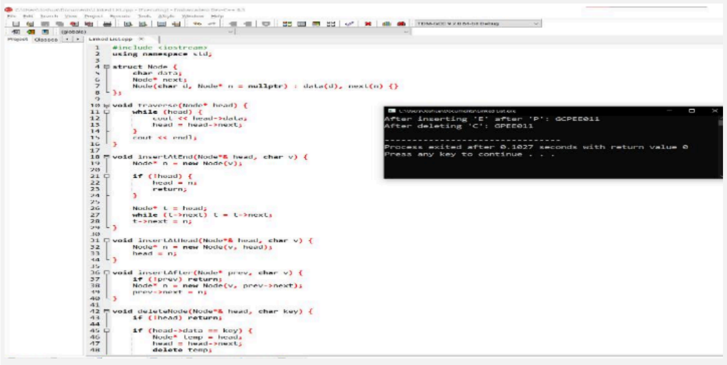
We'll take the use case of a music player. It should have following functions supported:

- Create a playlist using multiple songs.
- Add songs to the playlist.
- Remove a song from the playlist.
- Play songs in a loop (for this activity, we will print all the songs once).

Here are the steps to solve the problem:

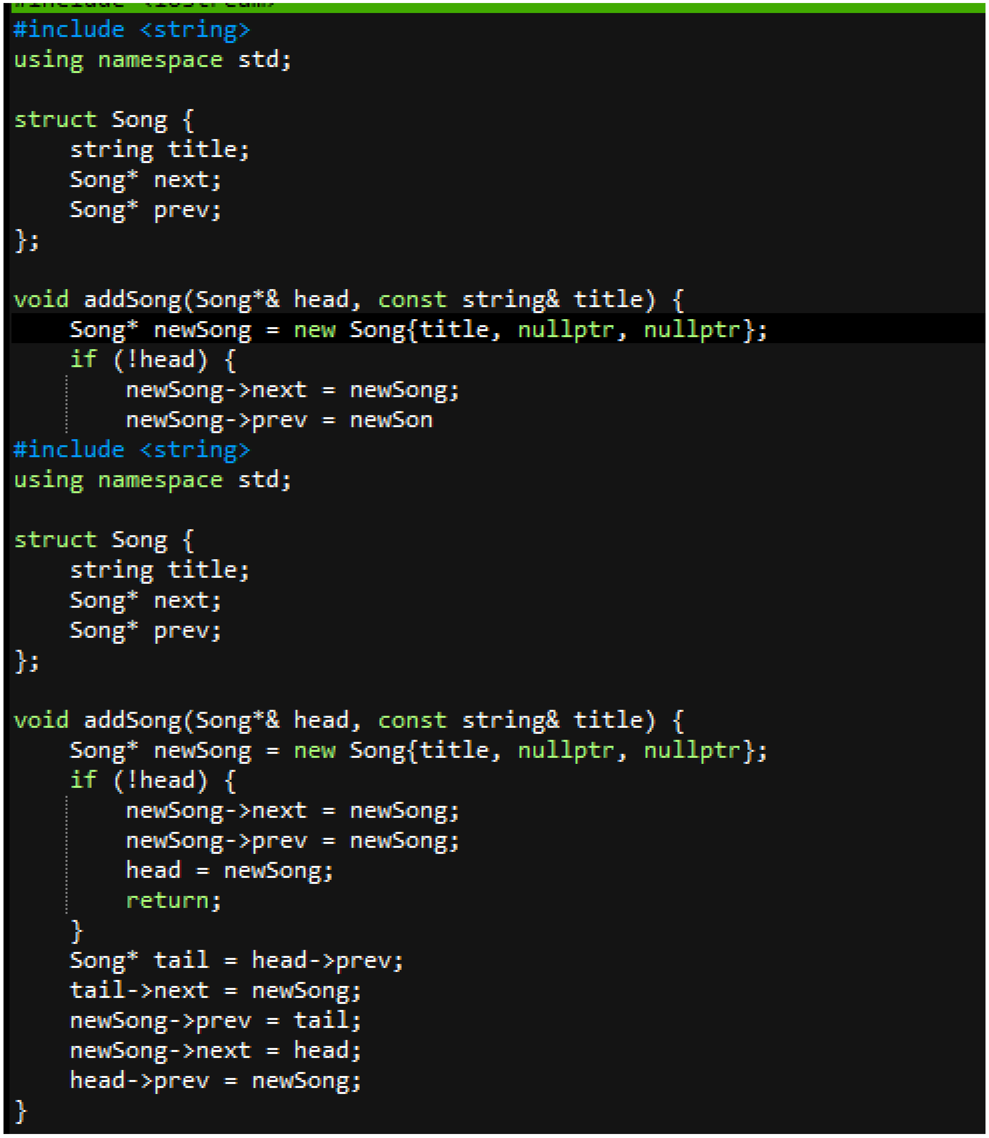
- Design the basic structure that supports circular data representation.
- After that, implement the insert and delete functions in the structure.
- Implement a function for traversing the playlist.

The driver function should allow for common operations on a playlist such as: next, previous, play all songs, insert and remove.

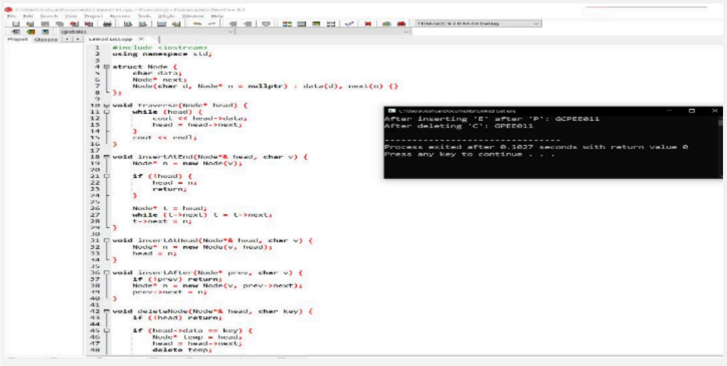


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```

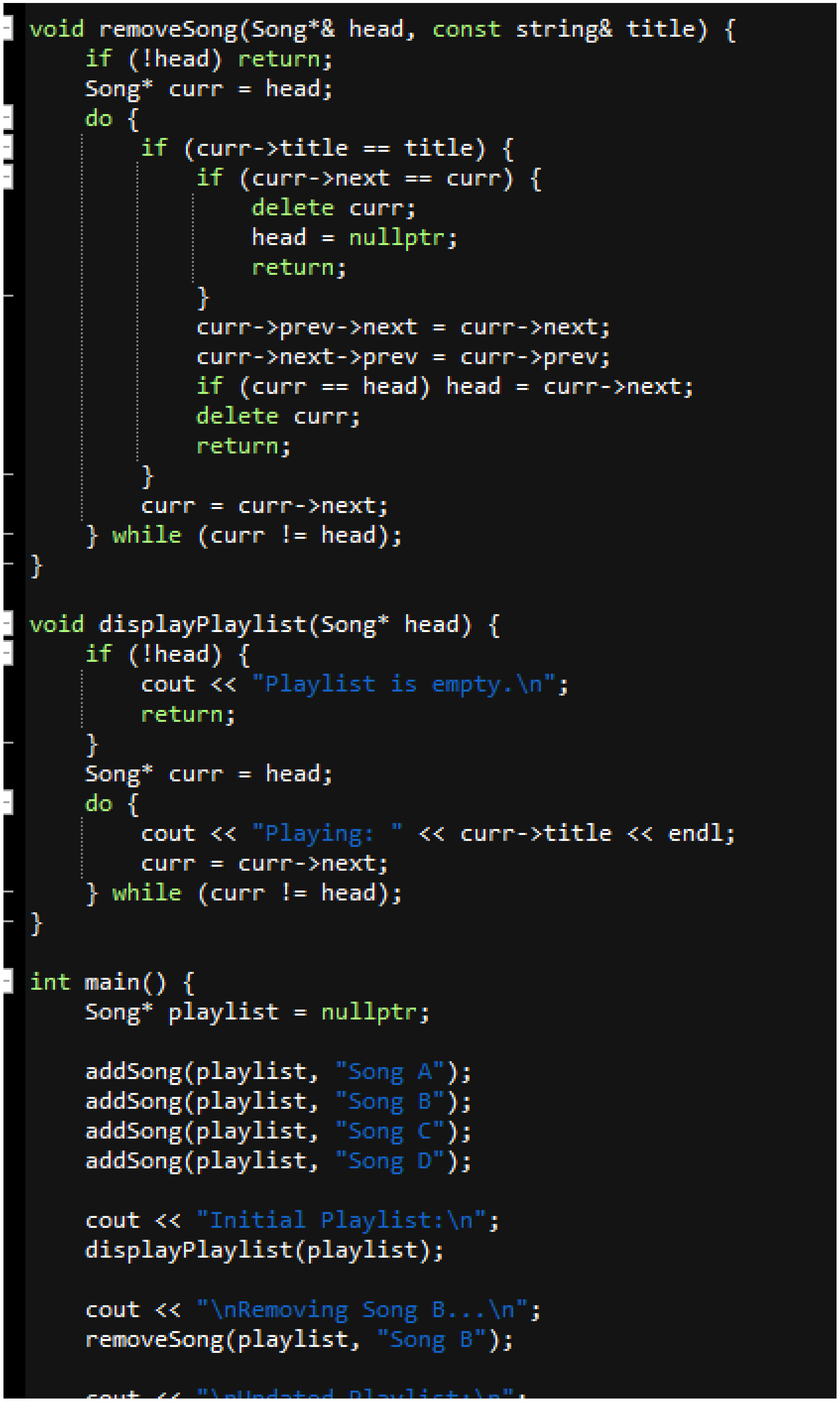


```
#include <iostream>
#include <string>
using namespace std;

struct Song {
    string title;
    Song* next;
    Song* prev;
};

void addSong(Song*& head, const string& title) {
    Song* newSong = new Song{title, nullptr, nullptr};
    if (!head) {
        newSong->next = newSong;
        newSong->prev = newSon
#include <string>
using namespace std;

struct Song {
    string title;
    Song* next;
    Song* prev;
};

void addSong(Song*& head, const string& title) {
    Song* newSong = new Song{title, nullptr, nullptr};
    if (!head) {
        newSong->next = newSong;
        newSong->prev = newSong;
        head = newSong;
        return;
    }
    Song* tail = head->prev;
    tail->next = newSong;
    newSong->prev = tail;
    newSong->next = head;
    head->prev = newSong;
}
```

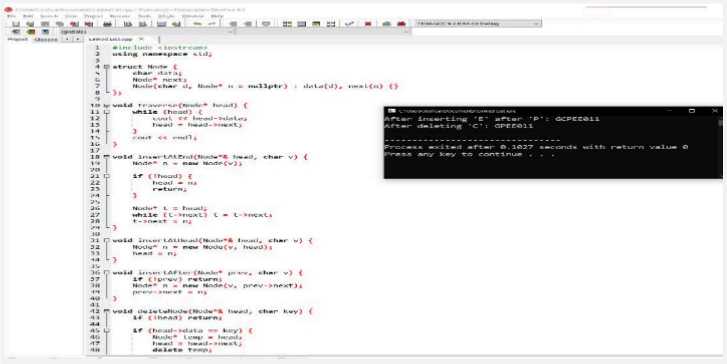


```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```



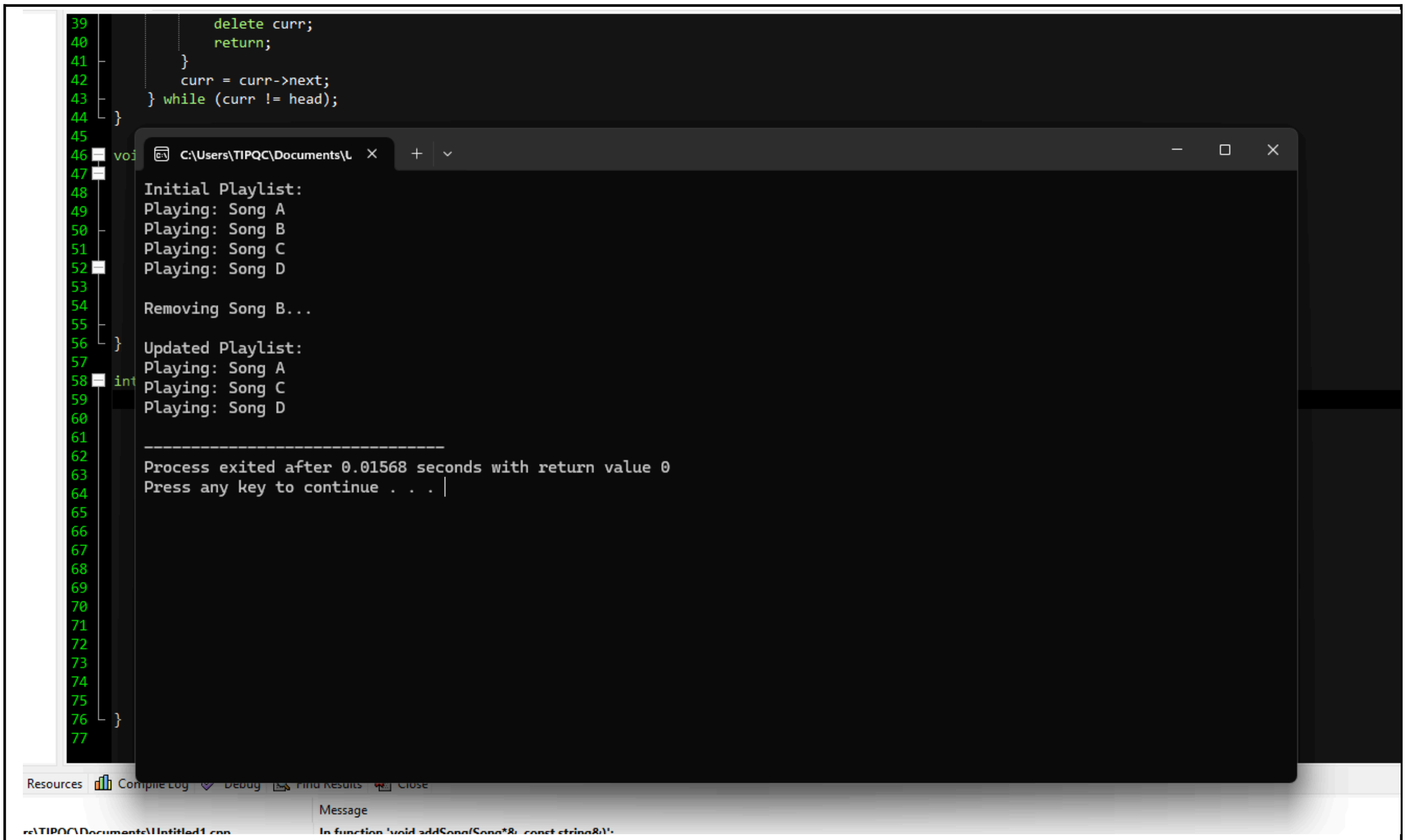
```cpp
void removeSong(Song*& head, const string& title) {
    if (!head) return;
    Song* curr = head;
    do {
        if (curr->title == title) {
            if (curr->next == curr) {
                delete curr;
                head = nullptr;
                return;
            }
            curr->prev->next = curr->next;
            curr->next->prev = curr->prev;
            if (curr == head) head = curr->next;
            delete curr;
            return;
        }
        curr = curr->next;
    } while (curr != head);
}

void displayPlaylist(Song* head) {
    if (!head) {
        cout << "Playlist is empty.\n";
        return;
    }
    Song* curr = head;
    do {
        cout << "Playing: " << curr->title << endl;
        curr = curr->next;
    } while (curr != head);
}

int main() {
    Song* playlist = nullptr;

    addSong(playlist, "Song A");
    addSong(playlist, "Song B");
    addSong(playlist, "Song C");
    addSong(playlist, "Song D");

    cout << "Initial Playlist:\n";
    displayPlaylist(playlist);

    cout << "\nRemoving Song B...\n";
    removeSong(playlist, "Song B");

    cout << "\nUpdated Playlist:\n";
```



File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes
Linked List.cpp
TDM-GCC 9.2.0 64-bit Debug

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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```



```
            delete curr;
            return;
        }
        curr = curr->next;
    } while (curr != head);
}

voi
int
}
```

```
Initial Playlist:
Playing: Song A
Playing: Song B
Playing: Song C
Playing: Song D

Removing Song B...

Updated Playlist:
Playing: Song A
Playing: Song C
Playing: Song D

--------------------------------
Process exited after 0.01568 seconds with return value 0
Press any key to continue . . .
```

```
rs\TIPQC\Documents\Untitled1.cpp     In function 'void addSong(Song*&, const string&)':
```

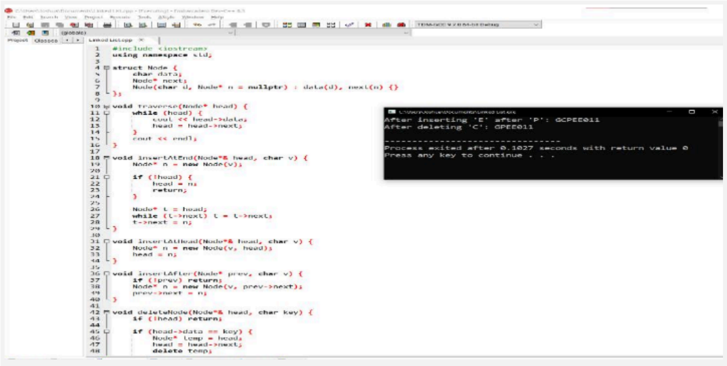
8. Conclusion

I learned about the link list on how nodes are made up in data structure where it stores data and a pointer to the next node allowing dynamic memory usage and easy insertion or deletion. Unlike arrays they didn't store elements in contiguous memory like link list do.

Provide the following:

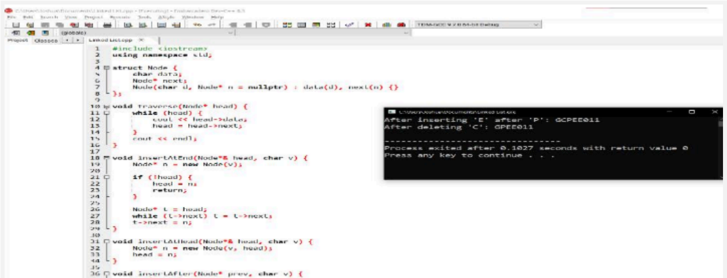
- Summary of lessons learned
- Analysis of the procedure
- Analysis of the supplementary activity
- Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

9. Assessment Rubric



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```



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

struct Node {
    char data;
    Node* next;
    Node(char d, Node* n = nullptr) : data(d), next(n) {}
};

void traverse(Node* head) {
    while (head) {
        cout << head->data;
        head = head->next;
    }
    cout << endl;
}

void insertAtEnd(Node*& head, char v) {
    Node* n = new Node(v);

    if (!head) {
        head = n;
        return;
    }

    Node* t = head;
    while (t->next) t = t->next;
    t->next = n;
}

void insertAtHead(Node*& head, char v) {
    Node* n = new Node(v, head);
    head = n;
}

void insertAfter(Node* prev, char v) {
    if (!prev) return;
    Node* n = new Node(v, prev->next);
    prev->next = n;
}

void deleteNode(Node*& head, char key) {
    if (!head) return;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
```

```
After inserting 'E' after 'P': GCPEE011
After deleting 'C': GPEE011

--------------------------------
Process exited after 0.1027 seconds with return value 0
Press any key to continue . . .
```