| **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:** 09/27/24 |
| **Section:**  CPE21s4 | **Date Submitted:** 09/27/24 |
| **Name(s):** Zolina, Anna Marie | **Instructor:** Mrs. Maria Rizette Sayo |
| **6. Output** | |
| | **Screenshot** |  | | --- | --- | | **Discussion** |  |   Table 3-1. Output of Initial/Simple Implementation   | **Operation** | **Screenshot** | | --- | --- | | Traversal |  | | Insertion at head |  | | Insertion at any part of the list |  | | Insertion at the end |  | | Deletion of a node |  |   Table 3-2. Code for the List Operations   |  | **Source Code** |  | | --- | --- | --- | | **Console** |  | |  | **Source Code** |  | | **Console** |  | |  | **Source Code** |  | | **Console** |  | |  | **Source Code** |  | | **Console** |  | |  | **Source Code** |  | | **Console** |  | |  | **Source Code** |  | | **Console** |  |   Table 3-3. Code and Analysis for Singly Linked Lists   | **Screenshot(s)** | **Analysis** | | --- | --- | |  |  | |  |  | |  |  | |  |  | |  |  |   Table 3-4. Modified Operations for Doubly Linked Lists | |
| **7. Supplementary Activity** | |
|  | |
| **8. Conclusion** | |
|  | |
| **9. Assessment Rubric** | |
|  | |

CODE

#include <iostream>

using namespace std;

class Node {

public:

char data;

Node\* next;

};

// Function to traverse and print the linked list

void traverseList(Node\* head) {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data;

temp = temp->next;

}

cout << endl;

}

// Insert node at the head

void insertAtHead(Node\*& head, char data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = head;

head = newNode;

}

// Insert node after a given previous node

void insertAfter(Node\* prevNode, char data) {

if (prevNode == nullptr) {

cout << "Previous node cannot be null." << endl;

return;

}

Node\* newNode = new Node();

newNode->data = data;

newNode->next = prevNode->next;

prevNode->next = newNode;

}

// Insert node at the end

void insertAtEnd(Node\*& head, char data) {

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;

}

// Delete a node by value

void deleteNode(Node\*& head, char key) {

Node\* temp = head;

Node\* prev = nullptr;

if (temp != nullptr && temp->data == key) {

head = temp->next; // Change head

delete temp; // Free old head

return;

}

while (temp != nullptr && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == nullptr) return; // Key not found

prev->next = temp->next;

delete temp; // Free memory

}

int main() {

Node\* head = nullptr;

// Create initial list: "101"

insertAtEnd(head, '1');

insertAtEnd(head, '0');

insertAtEnd(head, '1');

// Insert 'E', 'P', 'C' at the head ("CPE101")

insertAtHead(head, 'E');

insertAtHead(head, 'P');

insertAtHead(head, 'C');

cout << "Initial list: ";

traverseList(head);

// Task (b): Insert 'G' at the start of the list ("GCPE101")

insertAtHead(head, 'G');

cout << "After inserting 'G' at the start: ";

traverseList(head);

// Task (c): Insert 'E' after 'P' ("GCPEE101")

Node\* temp = head;

while (temp != nullptr && temp->data != 'P') {

temp = temp->next;

}

insertAfter(temp, 'E');

cout << "After inserting 'E' after 'P': ";

traverseList(head);

// Task (d): Delete 'C' ("GPEE101")

deleteNode(head, 'C');

cout << "After deleting 'C': ";

traverseList(head);

// Task (e): Delete 'P' ("GEE101")

deleteNode(head, 'P');

cout << "After deleting 'P': ";

traverseList(head);

// Task (f): Final list: "GEE101"

cout << "Final list: ";

traverseList(head);

return 0;

}