# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BESE 13AB**

# Lab 3: Implementation of Linked List with its Operations

**Date: 6th October, 2023**

**Time: 10:00am – 12:50 pm & 2:30pm – 5:00pm**

**Name:** Aimen Munawar

**Class:** BESE-13-A

**CMS ID:** 415867

# Lab Engineer: Anum Asif

# Lab 3: Implementation of Linked List with its Operations

**Introduction**

Students have learned the fundamental concepts of linked lists in the lectures. This lab will introduce students with the practical implementation of a linked list and different operations that can be performed on a linked list.

**Objectives**

Objective of this lab is to get familiar with singly linked list and implement them in C++.

**Tools/Software Requirement**

Visual Studio C++

**Helping Material**

Lecture slides. Text book.

**Description**

**Singly Linked List**

A Linked List is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of two parts i.e. data part and a reference part (also known as, a link) to the next node in the sequence. This structure allows efficient insertion or removal of elements from any position in the sequence.

## Singly-linked-list.svg

## The basic operation consist of

* ***Creating*** the list.
* ***Initialize*** pointers to NULL.
* ***Inserting*** nodes at beginning, last and from a specific location.
* ***Deletion*** of nodes from beginning, last and from a specific location.
* ***Traversing*** the list.
* ***Destroying*** the list.

**Lab Tasks**

Write a C++ program that can.

1. Create a simple linked list using functions to insert nodes at the head.

|  |
| --- |
| Solution |
| Task 1 Code:  #include<iostream>  using namespace std;  class Node {  public:  int value; // Integer value stored in the node  Node\* next; // Pointer to the next node in the list  Node(int data = 0, Node\* nextNode = nullptr) : value(data), next(nextNode) {}  };  class List {  public:  Node\* head; // Pointer to the first node in the list  List(); // Constructor to initialize the linked list  ~List(); // Destructor to free memory  void insertAtHead(int newValue); // Insert a node at the beginning of the list  };  List::List() {  head = nullptr; // Initialize the head pointer to null, indicating an empty list  }  List::~List() {  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }  }  void List::insertAtHead(int newValue) {  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // Set the new node's next pointer to the current head node  newNode->next = head;  // Update the head pointer to point to the new node  head = newNode;  }  int main() {  List linkedList; // Create a linked list object  cout << "\nInsert any value to Create a node\n" << endl;  int inputValue;  cin >> inputValue;  linkedList.insertAtHead(inputValue); // Insert at head  return 0;  }  Task 1 Output Screenshot: |

1. Make a function that can insert another node at 3rd location.

|  |
| --- |
| Solution |
| Task 2 Code:  #include <iostream>  using namespace std;  class Node {  public:  int value; // Integer value stored in the node  Node\* next; // Pointer to the next node in the list  Node(int data = 0, Node\* nextNode = nullptr) : value(data), next(nextNode) {}  };  class List {  public:  Node\* head; // Pointer to the first node in the list  List(); // Constructor to initialize the linked list  ~List(); // Destructor to free memory  void insertAtHead(int newValue); // Insert a node at the beginning of the list  void insertAtLocation(int location, int newValue); // Insert a node at a specific location in the list  };  List::List() {  head = nullptr; // Initialize the head pointer to null, indicating an empty list  }  List::~List() {  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }  }  void List::insertAtHead(int newValue) {  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // Set the new node's next pointer to the current head node  newNode->next = head;  // Update the head pointer to point to the new node  head = newNode;  }  void List::insertAtLocation(int location, int newValue) {  if (location < 1) {  cout << "Not a valid location for insertion." << endl;  return;  }  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // If the list is empty and the location is 1, set the new node as the head  if (head == nullptr) {  if (location == 1) {  head = newNode;  }  else {  cout << "Not a valid location for insertion." << endl;  }  return;  }  // If the location is 1, call insertAtHead function  if (location == 1) {  insertAtHead(newValue);  return;  }  // Traverse the list to the specified location  Node\* temp = head;  for (int i = 1; i < location - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  cout << "Not a valid location for insertion." << endl;  return;  }  }  // Insert the new node at the specified location  newNode->next = temp->next;  temp->next = newNode;  }  int main() {  List linkedList; // Create a linked list object  cout << "\nInsert any value to Create a node\n" << endl;  int inputValue;  cin >> inputValue;  linkedList.insertAtHead(inputValue); // Insert at head  int location, value;  cout << "\nEnter the location and the value to be inserted\n"<< endl;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location  cout << "\nEnter the location and the value to be inserted\n"<< endl;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location  return 0;  }  Task 2 Output Screenshot: |

1. Make a function that can display the lists made in 1 and 2.

|  |
| --- |
| Solution |
| Task 3 Code:  #include <iostream>  using namespace std;  class Node {  public:  int value; // Integer value stored in the node  Node\* next; // Pointer to the next node in the list  Node(int data = 0, Node\* nextNode = nullptr) : value(data), next(nextNode) {}  };  class List {  public:  Node\* head; // Pointer to the first node in the list  List(); // Constructor to initialize the linked list  ~List(); // Destructor to free memory  void insertAtHead(int newValue); // Insert a node at the beginning of the list  void insertAtLocation(int location, int newValue); // Insert a node at a specific location in the list  void displayList(); // Display all elements in the list  };  List::List() {  head = nullptr; // Initialize the head pointer to null, indicating an empty list  }  List::~List() {  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }  }  void List::insertAtHead(int newValue) {  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // Set the new node's next pointer to the current head node  newNode->next = head;  // Update the head pointer to point to the new node  head = newNode;  }  void List::insertAtLocation(int location, int newValue) {  if (location < 1) {  cout << "Not a valid location for insertion." << endl;  return;  }  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // If the list is empty and the location is 1, set the new node as the head  if (head == nullptr) {  if (location == 1) {  head = newNode;  }  else {  cout << "Not a valid location for insertion." << endl;  }  return;  }  // If the location is 1, call insertAtHead function  if (location == 1) {  insertAtHead(newValue);  return;  }  // Traverse the list to the specified location  Node\* temp = head;  for (int i = 1; i < location - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  cout << "Not a valid location for insertion." << endl;  return;  }  }  // Insert the new node at the specified location  newNode->next = temp->next;  temp->next = newNode;  }  void List::displayList() {  Node\* temp = head;  if (head == nullptr) {  cout << "List is currently empty." << endl;  return;  }  while (temp != nullptr) {  cout << temp->value << " "; // Print node value  temp = temp->next; // Move to the next node  }  cout << endl;  }  int main() {  //Task 1  List linkedList; // Create a linked list object  cout << "\nInsert any value to Create a node\n" << endl;  int inputValue;  cin >> inputValue;  linkedList.insertAtHead(inputValue); // Insert at head    //Task 2  cout << "\nEnter the location and the value to be inserted\n"<< endl;  int location, value;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location  cout << "\nEnter the location and the value to be inserted\n"<< endl;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location    //Task 3  cout << "\nDisplay Linked list entries\n" << endl;  linkedList.displayList(); // Display list  return 0;  }  Task 3 Output Screenshot: |

1. Write a function that can delete node from the linked list selected by the user. Display it as well.

|  |
| --- |
| Solution |
| Task 4 Code:  #include <iostream>  using namespace std;  class Node {  public:  int value; // Integer value stored in the node  Node\* next; // Pointer to the next node in the list  Node(int data = 0, Node\* nextNode = nullptr) : value(data), next(nextNode) {}  };  class List {  public:  Node\* head; // Pointer to the first node in the list  List(); // Constructor to initialize the linked list  ~List(); // Destructor to free memory  void insertAtHead(int newValue); // Insert a node at the beginning of the list  void insertAtLocation(int location, int newValue); // Insert a node at a specific location in the list  void displayList(); // Display all elements in the list  void Delete(int delValue); // Delete a node with a specific value  };  List::List() {  head = nullptr; // Initialize the head pointer to null, indicating an empty list  }  List::~List() {  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }  }  void List::insertAtHead(int newValue) {  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // Set the new node's next pointer to the current head node  newNode->next = head;  // Update the head pointer to point to the new node  head = newNode;  }  void List::insertAtLocation(int location, int newValue) {  if (location < 1) {  cout << "Not a valid location for insertion." << endl;  return;  }  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // If the list is empty and the location is 1, set the new node as the head  if (head == nullptr) {  if (location == 1) {  head = newNode;  }  else {  cout << "Not a valid location for insertion." << endl;  }  return;  }  // If the location is 1, call insertAtHead function  if (location == 1) {  insertAtHead(newValue);  return;  }  // Traverse the list to the specified location  Node\* temp = head;  for (int i = 1; i < location - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  cout << "Not a valid location for insertion." << endl;  return;  }  }  // Insert the new node at the specified location  newNode->next = temp->next;  temp->next = newNode;  }  void List::Delete(int delValue) {  if (head == nullptr) {  return; // If the list is empty, nothing to delete  }  if (delValue == 1) {  // If the node to be deleted is the head, update head pointer  Node\* temp = head;  head = head->next;  delete temp;  return;  }  Node\* temp = head;  for (int i = 1; i < delValue - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  return; // Location exceeds the list length  }  }  if (temp->next != nullptr) {  Node\* toDelete = temp->next;  temp->next = temp->next->next;  delete toDelete;  }  }  void List::displayList() {  Node\* temp = head;  if (head == nullptr) {  cout << "List is currently empty." << endl;  return;  }  while (temp != nullptr) {  cout << temp->value << " "; // Print node value  temp = temp->next; // Move to the next node  }  cout << endl;  }  int main() {  //Task 1  List linkedList; // Create a linked list object  cout << "\nInsert any value to Create a node\n" << endl;  int inputValue;  cin >> inputValue;  linkedList.insertAtHead(inputValue); // Insert at head    //Task 2  cout << "\nEnter the location and the value to be inserted\n"<< endl;  int location, value;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location  cout << "\nEnter the location and the value to be inserted\n"<< endl;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location    //Task 3  cout << "\nDisplay Linked list entries\n" << endl;  linkedList.displayList(); // Display list    //Task 4  cout << "Delete from entry location" << endl;  int deleteLocation;  cin >> deleteLocation;  linkedList.Delete(deleteLocation); // Delete node      return 0;  }  Task 4 Output Screenshot: |

1. Write a function that can count the number of nodes present in list.

|  |
| --- |
| Solution |
| Task 5 Code:  #include <iostream>  using namespace std;  class Node {  public:  int value; // Integer value stored in the node  Node\* next; // Pointer to the next node in the list  Node(int data = 0, Node\* nextNode = nullptr) : value(data), next(nextNode) {}  };  class List {  public:  Node\* head; // Pointer to the first node in the list  List(); // Constructor to initialize the linked list  ~List(); // Destructor to free memory  void insertAtHead(int newValue); // Insert a node at the beginning of the list  void insertAtLocation(int location, int newValue); // Insert a node at a specific location in the list  void displayList(); // Display all elements in the list  void Delete(int delValue); // Delete a node with a specific value  void countList(); // Count the total number of elements in the list  };  List::List() {  head = nullptr; // Initialize the head pointer to null, indicating an empty list  }  List::~List() {  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }  }  void List::insertAtHead(int newValue) {  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // Set the new node's next pointer to the current head node  newNode->next = head;  // Update the head pointer to point to the new node  head = newNode;  }  void List::insertAtLocation(int location, int newValue) {  if (location < 1) {  cout << "Not a valid location for insertion." << endl;  return;  }  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // If the list is empty and the location is 1, set the new node as the head  if (head == nullptr) {  if (location == 1) {  head = newNode;  }  else {  cout << "Not a valid location for insertion." << endl;  }  return;  }  // If the location is 1, call insertAtHead function  if (location == 1) {  insertAtHead(newValue);  return;  }  // Traverse the list to the specified location  Node\* temp = head;  for (int i = 1; i < location - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  cout << "Not a valid location for insertion." << endl;  return;  }  }  // Insert the new node at the specified location  newNode->next = temp->next;  temp->next = newNode;  }  void List::Delete(int delValue) {  if (head == nullptr) {  return; // If the list is empty, nothing to delete  }  if (delValue == 1) {  // If the node to be deleted is the head, update head pointer  Node\* temp = head;  head = head->next;  delete temp;  return;  }  Node\* temp = head;  for (int i = 1; i < delValue - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  return; // Location exceeds the list length  }  }  if (temp->next != nullptr) {  Node\* toDelete = temp->next;  temp->next = temp->next->next;  delete toDelete;  }  }  void List::displayList() {  Node\* temp = head;  if (head == nullptr) {  cout << "List is currently empty." << endl;  return;  }  while (temp != nullptr) {  cout << temp->value << " "; // Print node value  temp = temp->next; // Move to the next node  }  cout << endl;  }  void List::countList() {  Node\* temp = head;  int count = 0;  while (temp != nullptr) {  count++; // Increment the count for each node  temp = temp->next; // Move to the next node  }  cout << "Size of linked list is " << count << "." << endl;  }  int main() {  //Task 1  List linkedList; // Create a linked list object  cout << "\nInsert any value to Create a node\n" << endl;  int inputValue;  cin >> inputValue;  linkedList.insertAtHead(inputValue); // Insert at head    //Task 2  cout << "\nEnter the location and the value to be inserted\n"<< endl;  int location, value;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location  cout << "\nEnter the location and the value to be inserted\n"<< endl;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location    //Task 3  cout << "\nDisplay Linked list entries\n" << endl;  linkedList.displayList(); // Display list    //Task 4  cout << "Delete from entry location" << endl;  int deleteLocation;  cin >> deleteLocation;  linkedList.Delete(deleteLocation); // Delete node    //Task 5  cout << "\nDisplay Number of Entries\n" << endl;  linkedList.countList(); // Count list entries  return 0;  }  Task 5 Output Screenshot: |

1. Create menu in main function to give call to all of the above functions depending upon user’s input.

|  |
| --- |
| Solution |
| Task 6 Code:  #include <iostream>  using namespace std;  class Node {  public:  int value; // Integer value stored in the node  Node\* next; // Pointer to the next node in the list  Node(int data = 0, Node\* nextNode = nullptr) : value(data), next(nextNode) {}  };  class List {  public:  Node\* head; // Pointer to the first node in the list  List(); // Constructor to initialize the linked list  ~List(); // Destructor to free memory  void insertAtHead(int newValue); // Insert a node at the beginning of the list  void insertAtLocation(int location, int newValue); // Insert a node at a specific location in the list  void Delete(int delValue); // Delete a node with a specific value  void displayList(); // Display all elements in the list  void countList(); // Count the total number of elements in the list  };  List::List() {  head = nullptr; // Initialize the head pointer to null, indicating an empty list  }  List::~List() {  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }  }  void List::insertAtHead(int newValue) {  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // Set the new node's next pointer to the current head node  newNode->next = head;  // Update the head pointer to point to the new node  head = newNode;  }  void List::insertAtLocation(int location, int newValue) {  if (location < 1) {  cout << "Not a valid location for insertion." << endl;  return;  }  // Create a new node with the given value  Node\* newNode = new Node(newValue);  // If the list is empty and the location is 1, set the new node as the head  if (head == nullptr) {  if (location == 1) {  head = newNode;  }  else {  cout << "Not a valid location for insertion." << endl;  }  return;  }  // If the location is 1, call insertAtHead function  if (location == 1) {  insertAtHead(newValue);  return;  }  // Traverse the list to the specified location  Node\* temp = head;  for (int i = 1; i < location - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  cout << "Not a valid location for insertion." << endl;  return;  }  }  // Insert the new node at the specified location  newNode->next = temp->next;  temp->next = newNode;  }  void List::Delete(int delValue) {  if (head == nullptr) {  return; // If the list is empty, nothing to delete  }  if (delValue == 1) {  // If the node to be deleted is the head, update head pointer  Node\* temp = head;  head = head->next;  delete temp;  return;  }  Node\* temp = head;  for (int i = 1; i < delValue - 1; i++) {  if (temp->next != nullptr) {  temp = temp->next;  }  else {  return; // Location exceeds the list length  }  }  if (temp->next != nullptr) {  Node\* toDelete = temp->next;  temp->next = temp->next->next;  delete toDelete;  }  }  void List::displayList() {  Node\* temp = head;  if (head == nullptr) {  cout << "List is currently empty." << endl;  return;  }  while (temp != nullptr) {  cout << temp->value << " "; // Print node value  temp = temp->next; // Move to the next node  }  cout << endl;  }  void List::countList() {  Node\* temp = head;  int count = 0;  while (temp != nullptr) {  count++; // Increment the count for each node  temp = temp->next; // Move to the next node  }  cout << "Size of linked list is " << count << "." << endl;  }  int main() {  List linkedList; // Create a linked list object  while (true) { // Infinite loop for user interaction  char userChoice = NULL; // User's choice of operation  cout << " Select from one of the following\n A.Insert at HEAD \n B.Display Linked list entries \n C.Insert at location \n D.Delete from location\n E.Display number of list entries\n" << endl;  cin >> userChoice;  switch (userChoice) {  case 'A':  cout << "\nInsert any value to Create a node\n" << endl;  int inputValue;  cin >> inputValue;  linkedList.insertAtHead(inputValue); // Insert at head  break;  case 'B':  cout << "\nDisplay Linked list entries\n" << endl;  linkedList.displayList(); // Display list  break;  case 'C':  cout << "\nEnter the location and the value to be inserted\n" << endl;  int location, value;  cin >> location >> value;  linkedList.insertAtLocation(location, value); // Insert at location  break;  case 'D':  cout << "Delete from entry location" << endl;  int deleteLocation;  cin >> deleteLocation;  linkedList.Delete(deleteLocation); // Delete node  break;  case 'E':  cout << "\nDisplay Number of Entries\n" << endl;  linkedList.countList(); // Count list entries  break;  default:  cout << "Invalid choice" << endl; // Invalid input  }  }  return 0;  }  Task 6 Output Screenshot: |

**Important Note:** Please note that you have to develop your solution in C++ (OOP) i.e. using classes and objects. Solution written in a procedural style will not be accepted.

**Hint:** First you will create the relevant classes, and the functions will belong to the List class.

//class of node

class Node {

public:

int value;

node \*next;

};

Required functions for list class are:

void insertAtHead(int new\_value);

void insertAtLocation(int location,int new\_value)

void delete(int del\_value)

void displayList()

void countList()

**Solution**

|  |
| --- |
| Solution |
| Task 1 Code:  Task 1 Output Screenshot: |

### Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. This lab grading policy is as follows: Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.