Assignment-4(A)

UCS540 (Data Structures and Algorithms)

Submitted by, Harpartap Singh, 102104119

Group- 3EE3

1. **Write a program to implement a single Link List (with function insertion at the first node, insertion at the last node, insertion and deletion at the middle node, and traversing of the link list after each node insertion).**

Code:

```
#include <iostream>
struct Node {
   int data;
   Node* next;
};
// Function to insert a node at the beginning of the linked list
Node* insertAtBeginning(Node* head, int data) {
   Node* newNode = new Node;
   newNode->data = data;
   newNode->next = head;
   return newNode;
}
// Function to insert a node at the end of the linked list
Node* insertAtEnd(Node* head, int data) {
   Node* newNode = new Node;
   newNode->data = data;
   newNode->next = nullptr;
   if (head == nullptr) {
      return newNode;
   }
```

```
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}
// Function to insert a node at the middle of the linked list
Node* insertAtMiddle(Node* head, int position, int data) {
    if (position == 0) {
        return insertAtBeginning(head, data);
    }
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    Node* temp = head;
    for (int i = 0; i < position - 1 && temp != nullptr; ++i) {
        temp = temp->next;
    }
    if (temp == nullptr) {
        std::cout << "Position out of range" << std::endl;
        return head;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    return head;
}
// Function to delete a node at the middle of the linked list
Node* deleteAtMiddle(Node* head, int position) {
    if (position == 0 && head != nullptr) {
```

```
        Node* temp = head;
        head = head->next;
        delete temp;
        return head;
    }
    Node* temp = head;
    for (int i = 0; temp != nullptr && i < position - 1; ++i) {
        temp = temp->next;
    }
    if (temp == nullptr || temp->next == nullptr) {
        std::cout << "Position out of range" << std::endl;
        return head;
    }
    Node* toDelete = temp->next;
    temp->next = toDelete->next;
   delete toDelete;
   return head;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        std::cout << temp->data << " -> ";
       temp = temp->next;
    }
    std::cout << "NULL" << std::endl;
}
int main() {
    Node* head = nullptr;
    head = insertAtBeginning(head, 5);
```

```
    printList(head);
    head = insertAtBeginning(head, 10);
    printList(head);
    head = insertAtEnd(head, 15);
    printList(head);
    head = insertAtMiddle(head, 1, 20);
    printList(head);
    head = deleteAtMiddle(head, 2);
    printList(head);
    return 0;
}
```

Output:

```
Output
/tmp/FDw3gGZtoS.o
5 -> NULL
10 -> 5 -> NULL
10 -> 5 -> 15 -> NULL
10 -> 20 -> 5 -> 15 -> NULL
10 -> 20 -> 15 -> NULL
```

---

2. **Write a program to implement single Link List (with function deletion at first node, deletion at last node, and deletion at middle node and traversing of link list after each node deletion).**

Code:

```
#include <iostream>
struct Node {
    int data;
    Node* next;
```

```
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}
// Function to insert a node at the beginning of the linked list
Node* insertAtBeginning(Node* head, int data) {
    Node* newNode = createNode(data);
    newNode->next = head;
    return newNode;
}
// Function to insert a node at the end of the linked list
Node* insertAtEnd(Node* head, int data) {
    Node* newNode = createNode(data);
    if (head == nullptr) {
        return newNode;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}
// Function to delete a node at the beginning of the linked list
Node* deleteAtBeginning(Node* head) {
    if (head == nullptr) {
        std::cout << "List is empty" << std::endl;
```

```
        return nullptr;
    }
    Node* temp = head;
    head = head->next;
    delete temp;
    return head;
}
// Function to delete a node at the end of the linked list
Node* deleteAtEnd(Node* head) {
    if (head == nullptr) {
        std::cout << "List is empty" << std::endl;
        return nullptr;
    }
    if (head->next == nullptr) {
        delete head;
        return nullptr;
    }
    Node* temp = head;
    while (temp->next->next != nullptr) {
        temp = temp->next;
    }
    delete temp->next;
    temp->next = nullptr;
    return head;
}
// Function to delete a node at the middle of the linked list
Node* deleteAtMiddle(Node* head, int position) {
    if (position == 0) {
        return deleteAtBeginning(head);
    }
    Node* temp = head;
```

```
    Node* prev = nullptr;
    for (int i = 0; temp != nullptr && i < position; ++i) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == nullptr) {
        std::cout << "Position out of range" << std::endl;
        return head;
    }
    prev->next = temp->next;
    delete temp;
    return head;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        std::cout << temp->data << " -> ";
        temp = temp->next;
    }
    std::cout << "NULL" << std::endl;
}
int main() {
    Node* head = nullptr;
    head = insertAtBeginning(head, 5);
    printList(head);
    head = insertAtBeginning(head, 10);
    printList(head);
    head = insertAtEnd(head, 15);
    printList(head);
    head = deleteAtBeginning(head);
```

```
    printList(head);
    head = deleteAtEnd(head);
    printList(head);
    head = insertAtEnd(head, 20);
    printList(head);
    head = insertAtEnd(head, 25);
    printList(head);
    head = deleteAtMiddle(head, 1);
    printList(head);
    return 0;
}
```

Output: