PROGRAM NO: 7 DATE: 21-09-2023

**AIM**

To implement Singly Linked List with Insertion, Deletion and Display Operations.

**ALGORITHM**

1. START
2. FUNC CreateNode(item):
3. DECLARE node\* with malloc()
4. SET node.value = item, node.next = NULL
5. RETURN node. END FUNC
6. FUNC LinkedList(): // constructor
7. SET this.Head = NULL, this.\_length = 0
8. END FUNC
9. FUNC Insert\_Start(node): // Insertion at the beginning
10. SET node.next = this.Head, this.Head = node
11. INCREMENT this.\_length. END FUNC
12. FUNC Insert\_End(node): // Insertion at the end
13. SET lastNode.next = node
14. INCREMENT this.\_length. END FUNC
15. FUNC Insert\_Middle(node): // Insertion at the middle
16. SET node.next = prevNode.next, prevNode.next = node
17. INCREMENT this.\_length. END FUNC
18. FUNC Delete(index):
19. IF index == 0:
20. DECLARE temp = this.Head, deletedItem = this.Head.value
21. SET this.Head = this.Head.next
22. DECREMENT this.\_length, free(temp), RETURN deletedItem
23. END IF
24. DECLARE temp = this.Head.next, prevValue = this.Head
25. FOR (i=1 to i<index):
26. SET prevValue = temp, temp = temp.next
27. END FOR
28. DECLARE deletedItem = temp.value
29. SET prevValue.next = temp.next, free(temp)
30. DECREMENT this.\_length, RETURN deletedItem. END FUNC
31. FUNC Display(): // Singly Linked List Traversal
32. DECLARE node = this.Head, count = 0
33. Print “[“
34. WHILE (node != NULL):
35. Print node.value
36. SET node = node.next
37. IF count != this.\_length - 1:
38. Print “]”. END IF
39. INCREMENT count. END WHILE
40. Print “]”. END FUNC
41. STOP

**CODE**

#include <iostream>

#include <cstdlib>

using namespace std;

class LinkedListException {

    public:

    virtual string msg() = 0;

};

class LinkedListIndexOutOfBound: public LinkedListException {

    public:

    string msg() {

        return "Linked List Out of Bound";

    }

};

typedef struct Node {

    int value;

    Node\* next;

} Node;

class LinkedList {

    Node\* Head;

    int \_length;

    Node\* createNode(int item) {

        Node\* node = (Node\*) malloc(sizeof(Node));

        node->value = item;

        node->next = NULL;

        return node;

    }

    void Insert\_Start(Node\* node) {

        node->next = this->Head;

        this->Head = node;

        this->\_length++;

    }

    void Insert\_End(Node\* node, Node\* lastNode) {

        lastNode->next = node;

        this->\_length++;

    }

    void Insert\_Middle(Node\* node, Node\* prevNode) {

        node->next = prevNode->next;

        prevNode->next = node;

        this->\_length++;

    }

    public:

    LinkedList() {

        this->Head = NULL;

        this->\_length = 0;

    }

    int length() {

        return this->\_length;

    }

    void Insert(int index, int item) {

        Node\* node = this->createNode(item);

        if (index < 0 || (index > this->\_length)) throw LinkedListIndexOutOfBound();

        if (index == 0) {

            this->Insert\_Start(node);

            return;

        }

        int count = 1;

        Node\* temp = this->Head;

        while (temp->next != NULL) {

            if (count == index) {

                this->Insert\_Middle(node, temp);

                return;

            }

            temp = temp->next;

            count++;

        }

        if (this->\_length == index) {

            this->Insert\_End(node, temp);

        }

    }

    int Delete(int index) {

        if (index < 0 || (index >= this->\_length)) throw LinkedListIndexOutOfBound();

        if (index == 0) {

            // Deleting from the beginning

            Node\* temp = this->Head;

            int deletedItem = this->Head->value;

            this->Head = this->Head->next;

            this->\_length--;

            free(temp);

            return deletedItem;

        }

        // Accessing the node to delete

        Node\* temp = this->Head->next;

        Node\* prevValue = this->Head;

        for (int i=1; i<index; i++) {

            prevValue = temp;

            temp = temp->next;

        }

        // Deleting from the middle or end

        int deletedItem = temp->value;

        prevValue->next = temp->next;

        free(temp);

        this->\_length--;

        return deletedItem;

    }

    void Display() {

        Node\* node = this->Head;

        int count = 0;

        cout << "[";

        while (node != NULL) {

            cout << node->value;

            node = node->next;

            if (count != this->\_length - 1) {

                cout << ", ";

            }

            count++;

        }

        cout << "]";

    }

};

int main() {

    int choice, temp1, temp2;

    LinkedList List;

    while (1) {

        cout << "---- Linked List Data Structure ----" << endl;

        cout << "1. Insert" << endl;

        cout << "2. Delete" << endl;

        cout << "3. Display" << endl;

        cout << "4. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter item to insert : ";

            cin >> temp1;

            cout << "Enter index to insert to : ";

            cin >> temp2;

            try {

                List.Insert(temp2, temp1);

            } catch(LinkedListException &e) {

                cout << "ERROR: " << e.msg() << endl;

            }

        } else if (choice == 2) {

            cout << "Enter index to delete : ";

            cin >> temp1;

            try {

                cout << "Deleted Item: " << List.Delete(temp1);

            } catch(LinkedListException &e) {

                cout << "ERROR: " << e.msg() << endl;

            }

        } else if (choice == 3) {

            List.Display();

        } else if (choice == 4) {

            cout << "\n--------- Author ----------------" << endl;

            cout << "Ali Izzath Shazin" << endl;

            cout << "220071601028" << endl;

            cout << "B. Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option" << endl;

        }

        cout << endl;

    }

    return 0;

}

**OUTPUT**

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 1

Enter item to insert : 10

Enter index to insert to : 0

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 1

Enter item to insert : 20

Enter index to insert to : 1

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 1

Enter item to insert : 30

Enter index to insert to : 1

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 2

Enter index to delete : 0

Deleted Item: 10

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 3

[30, 20]

---- Linked List Data Structure ----

1. Insert

2. Delete

3. Display

4. Exit

Enter Choice: 4

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE