PROGRAM NO: 8 DATE: 28-09-2023

**AIM**

To implement Doubly 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, node.prev = NULL
5. RETURN node. END FUNC
6. FUNC DoublyLinkedList(): // constructor
7. SET this.Head = NULL, this.End = 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. IF node.next == NULL: SET this.End = node
12. ELSE: SET node.next.prev = node, ENDIF
13. INCREMENT this.\_length. END FUNC
14. FUNC Insert\_End(node): // Insertion at the end
15. SET node.prev = this.End, this.End.next = node, this.End = node
16. INCREMENT this.\_length. END FUNC
17. FUNC Insert\_Middle(node, prevNode): // Insertion at the middle
18. SET node.next = prevNode.next, node.prev = prevNode, node.next.prev = node, prevNode.next = node
19. INCREMENT this.\_length. END FUNC
20. FUNC Delete\_Start():
21. DECLARE temp = this.Head, deletedItem = this.Head.value
22. SET this.Head = this.Head.next
23. IF this.Head == NULL: SET this.End = NULL
24. ELSE: SET this.Head.prev = NULL, ENDIF
25. DECREMENT this.\_length, free(temp)
26. RETURN deletedItem, END FUNC
27. FUNC Delete\_End():
28. DECLARE temp = this.Head, deletedItem = this.Head.value
29. SET this.End = this.End.prev
30. IF this.End == NULL: SET this.Head = NULL
31. ELSE: SET this.End.next= NULL, ENDIF
32. DECREMENT this.\_length, free(temp)
33. RETURN deletedItem, END FUNC
34. FUNC Delete\_Middle(index, nodeToDelete)
35. DECLARE deletedItem = nodeToDelete.value
36. SET nodeToDelete.prev.next = nodeToDelete.next, nodeToDelete.next.prev = nodeToDelete.prev
37. DECREMENT this.\_length, free(temp)
38. RETURN deletedItem, END FUNC
39. FUNC Display():
40. DECLARE node = this.Head, count = 0
41. Print “[“
42. WHILE (node != NULL):
43. Print node.value
44. SET node = node.next
45. IF count != this.\_length - 1:
46. Print “]”. END IF
47. INCREMENT count. END WHILE
48. Print “]”. END FUNC
49. FUNC DisplayReverse():
50. DECLARE node = this.End, count = 0
51. Print “[“
52. WHILE (node != NULL):
53. Print node.value
54. SET node = node.prev
55. IF count != this.\_length - 1:
56. Print “]”. END IF
57. INCREMENT count. END WHILE
58. Print “]”. END FUNC
59. STOP

**CODE**

#include <iostream>

#include <cstdlib>

using namespace std;

class DoublyLinkedListException {

    public:

    virtual string msg() = 0;

};

class DoublyLinkedListIndexOutOfBound: public DoublyLinkedListException {

    public:

    string msg() {

        return "Linked List Out of Bound";

    }

};

typedef struct Node {

    int value;

    Node\* next;

    Node\* prev;

} Node;

class DoublyLinkedList {

    Node\* Head;

    Node\* End;

    int \_length;

    Node\* createNode(int item) {

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

        node->value = item;

        node->next = NULL;

        node->prev = NULL;

        return node;

    }

    void Insert\_Start(Node\* node) {

        node->next = this->Head;

        this->Head = node;

        if (node->next == NULL) {

            // If first item

            this->End = node;

        } else {

            // If not first item

            node->next->prev = node;

        }

        this->\_length++;

    }

    void Insert\_End(Node\* node) {

        node->prev = this->End;

        this->End->next = node;

        this->End = node;

        this->\_length++;

    }

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

        node->next = prevNode->next;

        node->prev = prevNode;

        node->next->prev = node;

        prevNode->next = node;

        this->\_length++;

    }

    int Delete\_Start() {

        Node\* temp = this->Head;

        int deletedItem = this->Head->value;

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

        if (this->Head == NULL) {

            // If all items deleted

            this->End = NULL;

        } else {

            // If more items are left

            this->Head->prev = NULL;

        }

        this->\_length--;

        free(temp);

        return deletedItem;

    }

    int Delete\_End() {

        Node\* temp = this->End;

        int deletedItem = this->End->value;

        this->End = this->End->prev;

        if (this->End == NULL) {

            // If all items deleted

            this->Head = NULL;

        } else {

            // If more items are left

            this->End->next = NULL;

        }

        this->\_length--;

        free(temp);

        return deletedItem;

    }

    int Delete\_Middle(int index, Node\* nodeToDelete) {

        int deletedItem = nodeToDelete->value;

        nodeToDelete->prev->next = nodeToDelete->next;

        nodeToDelete->next->prev = nodeToDelete->prev;

        this->\_length--;

        free(nodeToDelete);

        return deletedItem;

    }

    public:

    DoublyLinkedList() {

        this->Head = NULL;

        this->\_length = 0;

    }

    int length() {

        return this->\_length;

    }

    Node\* Get\_Node(int index, int instance) {

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

        Node\* temp;

        int count = 0;

        if (index <= ((this->\_length - 1) / 2)) {

            temp = this->Head;

            for (int i=0; i<index; i++) {

                count++;

                temp = temp->next;

            }

            if (instance == 1) cout << "Number of nodes traversed from left: " << count << endl;

        } else {

            temp = this->End;

            for (int i=this->\_length - 1; i>index; i--) {

                count++;

                temp = temp->prev;

            }

            if (instance == 1) cout << "Number of nodes traversed from right: " << count << endl;

        }

        return temp;

    }

    void Insert(int index, int item) {

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

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

        if (index == 0) {

            this->Insert\_Start(node);

            return;

        }

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

            this->Insert\_End(node);

            return;

        }

        Node\* temp = this->Get\_Node(index, 0);

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

        return;

    }

    int Delete(int index) {

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

        if (index == 0) {

            return this->Delete\_Start();

        }

        if (index == this->\_length - 1) {

            return this->Delete\_End();

        }

        Node\* temp = this->Get\_Node(index, 0);

        return this->Delete\_Middle(index, temp);

    }

    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 << "]";

    }

    void DisplayReverse() {

        Node\* node = this->End;

        int count = 0;

        cout << "[";

        while (node != NULL) {

            cout << node->value;

            node = node->prev;

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

                cout << ", ";

            }

            count++;

        }

        cout << "]";

    }

};

int main() {

    int choice, temp1, temp2;

    DoublyLinkedList List;

    while (1) {

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

        cout << "1. Insert" << endl;

        cout << "2. Delete" << endl;

        cout << "3. Display" << endl;

        cout << "4. Display Reverse" << endl;

        cout << "5. Access A Node" << endl;

        cout << "6. 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(DoublyLinkedListException &e) {

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

            }

        } else if (choice == 2) {

            cout << "Enter index to delete : ";

            cin >> temp1;

            try {

                int deletedValue = List.Delete(temp1);

                cout << "Deleted Item: " << deletedValue;

            } catch(DoublyLinkedListException &e) {

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

            }

        } else if (choice == 3) {

            List.Display();

        } else if (choice == 4) {

            List.DisplayReverse();

        } else if (choice == 5) {

            cout << "Enter index to access : ";

            cin >> temp1;

            try {

                int accessedValue = List.Get\_Node(temp1, 1)->value;

                cout << "Accessed Node: " << accessedValue;

            } catch(DoublyLinkedListException &e) {

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

            }

        } else if (choice == 6) {

            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**

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 1

Enter item to insert : 10

Enter index to insert to : 0

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 1

Enter item to insert : 20

Enter index to insert to : 1

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 1

Enter item to insert : 30

Enter index to insert to : 1

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 3

[10, 30, 20]

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 4

[20, 30, 10]

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 5

Enter index to access : 0

Number of nodes traversed from left: 0

Accessed Node: 10

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 5

Enter index to access : 2

Number of nodes traversed from right: 0

Accessed Node: 20

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 2

Enter index to delete : 1

Deleted Item: 30

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 3

[10, 20]

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

1. Insert

2. Delete

3. Display

4. Display Reverse

5. Access A Node

6. Exit

Enter Choice: 6

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

Ali Izzath Shazin

220071601028

B. Tech CSE A