Riphah International University Islamabad

Lab # 6

Subject: DLD

Submitted to: Mam Sidra

Submitted by: Fatima Nazir (45317)

LAB TASK 1

```
class DoublyLinkedList {
private:
    Node* head;

public:
    DoublyLinkedList() {
        head = nullptr;
    }

    // Insert a new node at the end
    void insertNode(int value) {
        Node* newNode = new Node(value);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }

    // Delete the first node
    void deleteFirstNode() {
```

```
    if (!head) {
        cout << "List is empty!" << endl;
        return;
    }

    Node* temp = head;
    head = head->next;
    if (head) head->prev = nullptr;
    delete temp;
}

// Delete a node after a given value
void deleteAfterValue(int value) {
    Node* temp = head;
    while (temp && temp->data != value) {
        temp = temp->next;
    }

    if (!temp || !temp->next) {
        cout << "No node found after " << value << endl;
        return;
    }

    Node* toDelete = temp->next;
    temp->next = toDelete->next;
    if (toDelete->next) toDelete->next->prev = temp;
```

```
        delete toDelete;
    }

    // Display the list
    void display() {
        if (!head) {
            cout << "List is empty!" << endl;
            return;
        }

        Node* temp = head;
        cout << "NULL <==> ";
        while (temp) {
            cout << temp->data << " <==> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};

// Main Function
int main() {
    DoublyLinkedList list;

    // Insert some nodes
```

```
    }
};

// Main Function
int main() {
    DoublyLinkedList list;

    // Insert some nodes
    list.insertNode(1);
    list.insertNode(45);
    list.insertNode(60);
    list.insertNode(12);

    cout << "Original List:\n";
    list.display();

    cout << "\nDeleting first node:\n";
    list.deleteFirstNode();
    list.display();

    cout << "\nDeleting node after 45:\n";
    list.deleteAfterValue(45);
    list.display();

    return 0;
```

```
Original List:
NULL <==> 1 <==> 45 <==> 60 <==> 12 <==> NULL

Deleting first node:
NULL <==> 45 <==> 60 <==> 12 <==> NULL

Deleting node after 45:
NULL <==> 45 <==> 12 <==> NULL


=== Code Execution Successful ===
```

LAB TASK 2

```cpp
struct Node {
    int score;
    Node* next;
    Node* prev;

    Node(int s) {
        score = s;
        next = nullptr;
        prev = nullptr;
    }
};

// Doubly Linked List Class
class GolfScores {
private:
    Node* head;

public:
    GolfScores() {
        head = nullptr;
    }

    // Insert player score in sorted order
    void insertScore(int score) {
        Node* newNode = new Node(score);
```

```
    if (!head || head->score >= score) {
        newNode->next = head;
        if (head) head->prev = newNode;
        head = newNode;
        return;
    }

    Node* temp = head;
    while (temp->next && temp->next->score < score) {
        temp = temp->next;
    }

    newNode->next = temp->next;
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
    newNode->prev = temp;
}

// Delete player by score
void deleteScore(int score) {
    if (!head) {
        cout << "List is empty!\n";
        return;
    }
```

```
    Node* temp = head;
    while (temp && temp->score != score) {
        temp = temp->next;
    }

    if (!temp) {
        cout << "Player with score " << score << " not
            found!\n";
        return;
    }

    if (temp->prev) temp->prev->next = temp->next;
    if (temp->next) temp->next->prev = temp->prev;
    if (temp == head) head = temp->next;
    delete temp;


/ Display all players
oid display() {
    if (!head) {
        cout << "List is empty!\n";
        return;
    }

    Node* temp = head;
    cout << "Players (sorted by score): NULL <==> ";
```

```
    while (temp) {
        cout << temp->score << " <==> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}

// Display lowest score
void displayLowest() {
    if (!head) {
        cout << "List is empty!\n";
        return;
    }
    cout << "Lowest score: " << head->score << endl;
}

// Display players with the same score
void displaySameScore(int score) {
    Node* temp = head;
    bool found = false;
    while (temp) {
        if (temp->score == score) {
            cout << score << " ";
            found = true;
        }
        temp = temp->next;
```

```
    if (!found) {
        cout << "No players with score " << score;
    }
    cout << endl;
}

// Display backward from a given score
void displayBackwardFrom(int score) {
    Node* temp = head;
    while (temp && temp->score != score) {
        temp = temp->next;
    }

    if (!temp) {
        cout << "Score not found!\n";
        return;
    }

    cout << "Backward from " << score << ": NULL <==> ";
    while (temp) {
        cout << temp->score << " <==> ";
        temp = temp->prev;
    }
    cout << "NULL\n";
}
```

```cpp
int main() {
    GolfScores list;

    // Insert scores
    list.insertScore(72);
    list.insertScore(68);
    list.insertScore(75);
    list.insertScore(70);
    list.insertScore(68);

    cout << "\nOriginal List:\n";
    list.display();

    cout << "\nDeleting score 70:\n";
    list.deleteScore(70);
    list.display();

    cout << "\nLowest score:\n";
    list.displayLowest();

    cout << "\nPlayers with score 68:\n";
    list.displaySameScore(68);

    cout << "\nDisplay backward from 75:\n";
    list.displayBackwardFrom(75);
```

```
Original List:
Players (sorted by score): NULL <==> 68 <==> 68 <==> 70 <==> 72 <==> 75 <==
    NULL

Deleting score 70:
Players (sorted by score): NULL <==> 68 <==> 68 <==> 72 <==> 75 <==> NULL

Lowest score:
Lowest score: 68

Players with score 68:
68 68

Display backward from 75:
Backward from 75: NULL <==> 75 <==> 72 <==> 68 <==> 68 <==> NULL


=== Code Execution Successful ===
```