Aryabhatta College (DU)

Practical_File_DS

Name	:	Atul_Arya
Roll_No.	:	CSC/22/11
University Roll No.	:	22059570009
Subject	:	Data Structure
Teacher	:	Sonal Linda
Submission Date	:	Dec 19, 2023

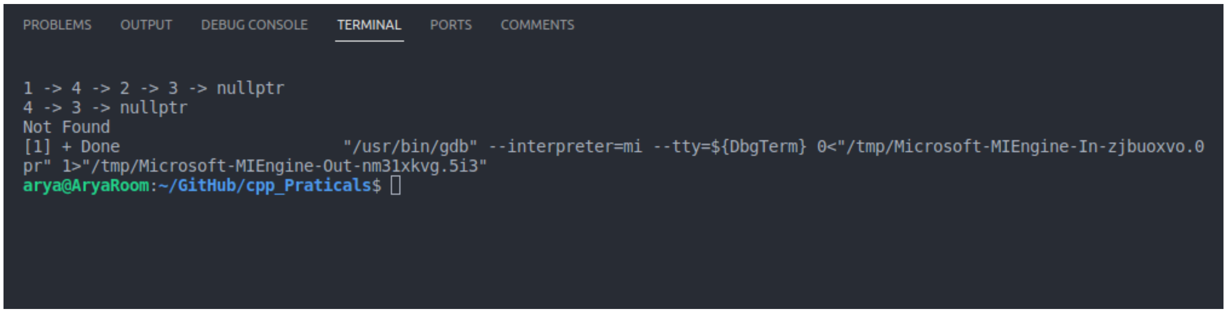
Atul_Arya
CSC/22/11

Index

Que-1.

```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

class SinglyLinkedList {
private:
    Node* head;

public:
    SinglyLinkedList() : head(nullptr) {}

    void insert_at_beginning(int data) {
        Node* new_node = new Node(data);
        new_node->next = head;
        head = new_node;
    }

    void insert_at_position(int data, int position) {
        if (position < 1) {
            cout << "Invalid position" << endl;
            return;
        }

        Node* new_node = new Node(data);
        if (position == 1) {
            new_node->next = head;
            head = new_node;
            return;
        }

        Node* current = head;
        int count = 1;
        while (current && count < position - 1) {
            current = current->next;
            count++;
        }

        if (!current) {
            cout << "Position out of range" << endl;
        } else {
            new_node->next = current->next;
            current->next = new_node;
        }
    }

    void remove_from_beginning() {
        if (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        } else {
            cout << "List is empty" << std::endl;
        }
    }

    void remove_from_position(int position) {
        if (position < 1 || !head) {
            cout << "Invalid position or empty list" << endl;
            return;
        }

        if (position == 1) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }

        Node* current = head;
        int count = 1;
        while (current && count < position - 1) {
            current = current->next;
            count++;
        }

        if (!current || !current->next) {
            cout << "Position out of range" << endl;
        } else {
            Node* temp = current->next;
            current->next = current->next->next;
            delete temp;
        }
    }

    Node* search(int target) {
        Node* current = head;
        while (current) {
            if (current->data == target) {
                return current;
            }
            current = current->next;
        }
        return nullptr;
    }

    void display() {
        Node* current = head;
        while (current) {
            cout << current->data << " -> ";
            current = current->next;
        }
        cout << "nullptr" << endl;
    }

    ~SinglyLinkedList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    SinglyLinkedList linked_list;

    int choice, data, position;

    do {
        cout << "\nMenu:\n";
        cout << "1. Insert at beginning\n";
        cout << "2. Insert at position\n";
        cout << "3. Remove from beginning\n";
        cout << "4. Remove from position\n";
        cout << "5. Search\n";
        cout << "6. Display\n";
        cout << "7. Exit\n";

        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter data to insert at the beginning: ";
                cin >> data;
                linked_list.insert_at_beginning(data);
                break;

            case 2:
                cout << "Enter data to insert: ";
                cin >> data;
                cout << "Enter position to insert at: ";
                cin >> position;
                linked_list.insert_at_position(data, position);
                break;

            case 3:
                linked_list.remove_from_beginning();
                break;

            case 4:
                cout << "Enter position to remove: ";
                cin >> position;
                linked_list.remove_from_position(position);
                break;

            case 5:
                cout << "Enter data to search: ";
                cin >> data;
                Node* search_result = linked_list.search(data);
                if (search_result) {
                    cout << "Found: " << search_result->data << endl;
                } else {
                    cout << "Not Found" << endl;
                }
                break;

            case 6:
                linked_list.display();
                break;

            case 7:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Try again.\n";
        }

      } while (choice != 7);

      return 0;
    }
```

Output:



```
1 -> 4 -> 2 -> 3 -> nullptr
4 -> 3 -> nullptr
Not Found
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-zjbuoxvo.0
pr" 1>"/tmp/Microsoft-MIEngine-Out-nm31xkvg.5i3"
arya@AryaRoom:~/GitHub/cpp_Praticals$ 
```

Que-2.

```
/*
Write a program to implement doubly linked list as an ADT that supports the following
op-erations:
```

```
i. Insert an element x at the beginning of the doubly linked list
ii. Insert an element x at the end of the doubly linked list
iii. Remove an element from the beginning of the doubly linked list
iv. Remove an element from the end of the doubly linked list
*/

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;
    Node(int value) : data(value), prev(nullptr), next(nullptr) {}
};

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

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    void insert_at_beginning(int data) {
        Node* new_node = new Node(data);
        if (!head) {
            head = new_node;
            tail = new_node;
        } else {
            new_node->next = head;
            head->prev = new_node;
            head = new_node;
        }
    }

    void insert_at_end(int data) {
        Node* new_node = new Node(data);
        if (!head) {
            head = new_node;
            tail = new_node;
        } else {
            new_node->prev = tail;
            tail->next = new_node;
            tail = new_node;
        }
    }

    void remove_from_beginning() {
        if (!head) {
            cout << "List is empty" << endl;
        } else {
            if (head == tail) {
                delete head;
                head = nullptr;
                tail = nullptr;
            } else {
                head = head->next;
                delete head->prev;
                head->prev = nullptr;
            }
        }
    }

    void remove_from_end() {
        if (!head) {
            cout << "List is empty" << endl;
        } else {
            if (head == tail) {
                delete tail;
                head = nullptr;
                tail = nullptr;
            } else {
                tail = tail->prev;
                delete tail->next;
                tail->next = nullptr;
            }
        }
    }

    void display() {
        if (!head) {
            cout << "List is empty" << endl;
        } else {
            Node* current = head;
            while (current) {
                cout << current->data << "->";
                current = current->next;
            }
            cout << "NULL" << endl;
        }
    }
};

int main() {
    DoublyLinkedList dll;

    int choice, data;

    do {
        cout << "\nMenu:\n";
        cout << "1. Insert at beginning\n";
        cout << "2. Insert at end\n";
        cout << "3. Remove from beginning\n";
        cout << "4. Remove from end\n";
        cout << "5. Display\n";
        cout << "6. Exit\n";

        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter data to insert at the beginning: ";
                cin >> data;
                dll.insert_at_beginning(data);
                break;

            case 2:
                cout << "Enter data to insert at the end: ";
                cin >> data;
                dll.insert_at_end(data);
                break;

            case 3:
                dll.remove_from_beginning();
                break;
            case 4:
                dll.remove_from_end();
                break;

            case 5:
                cout << "Doubly Linked List: ";
                dll.display();
                break;

            case 6:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Try again.\n";
        }

    } while (choice != 6);

    return 0;
}
```

Output:

```
arya@AryaRoomL:~/Desktop/PracticalFileDS/PracticalFile1$
./P2_CSC-22-11_DS

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
5. Display
6. Exit
Enter your choice: 1
Enter data to insert at the beginning: 4

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
```

```
5. Display
6. Exit
Enter your choice: 2
Enter data to insert at the end: 7

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
5. Display
6. Exit
Enter your choice: 5
Doubly Linked List: 4->7->NULL

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
5. Display
6. Exit
Enter your choice: 2
Enter data to insert at the end: 9

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
5. Display
6. Exit
Enter your choice: 3

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
5. Display
6. Exit
Enter your choice: 5
Doubly Linked List: 7->9->NULL

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
```

```
4. Remove from end
5. Display
6. Exit
Enter your choice: 4

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
5. Display
6. Exit
Enter your choice: 5
Doubly Linked List: 7->NULL

Menu:
1. Insert at beginning
2. Insert at end
3. Remove from beginning
4. Remove from end
5. Display
6. Exit
Enter your choice: 6
Exiting program.
```

Que-3.

```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

class CircularLinkedList {
private:
    Node* head;

public:
    CircularLinkedList() : head(nullptr) {}
```

```
void insert(int data) {
    Node* new_node = new Node(data);
    if (!head) {
        head = new_node;
        new_node->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = new_node;
        new_node->next = head;
    }
}

void remove(int data) {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }
    if (head->data == data) {
        if (head->next == head) {
            delete head;
            head = nullptr;
        } else {
            Node* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            temp->next = head->next;
            Node* temp2 = head;
            head = head->next;
            delete temp2;
        }
    } else {
        Node* current = head;
        Node* prev = nullptr;
        do {
            if (current->data == data) {
                prev->next = current->next;
                delete current;
                return;
            }
```

```
                prev = current;
                current = current->next;
            } while (current != head);
            cout << data << " not found in the list" << endl;
        }
    }

    Node* search(int data) {
        if (!head) {
            return nullptr;
        }
        Node* current = head;
        do {
            if (current->data == data) {
                cout << data << " found at address: " << current << endl;
                return current;
            }
            current = current->next;
        } while (current != head);
        cout << data << " not found!" << endl;
        return nullptr;
    }

    void display() {
        if (!head) {
            cout << "List is empty" << endl;
        } else {
            Node* temp = head;
            do {
                cout << temp->data << "->";
                temp = temp->next;
            } while (temp != head);
            cout << "[head]" << endl;
        }
    }
};

int main() {
    CircularLinkedList cll;

    int choice, data;

    do {
```

```
    cout << "\nMenu:\n";
    cout << "1. Insert\n";
    cout << "2. Remove\n";
    cout << "3. Search\n";
    cout << "4. Display\n";
    cout << "5. Exit\n";

    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1:
            cout << "Enter data to insert: ";
            cin >> data;
            cll.insert(data);
            break;

        case 2:
            cout << "Enter data to remove: ";
            cin >> data;
            cll.remove(data);
            break;

        case 3:
            cout << "Enter data to search: ";
            cin >> data;
            cll.search(data);
            break;

        case 4:
            cout << "Circular Linked List: ";
            cll.display();
            break;

        case 5:
            cout << "Exiting program.\n";
            break;

        default:
            cout << "Invalid choice. Try again.\n";
    }

} while (choice != 5);
```

```
    return 0;
}
```

Output:

```
arya@AryaRoomL:~/Desktop/PracticalFileDS/PracticalFile1$
./P3_CSC-22-11_DS

Menu:
1. Insert
2. Remove
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter data to insert: 3

Menu:
1. Insert
2. Remove
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter data to insert: 5

Menu:
1. Insert
2. Remove
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter data to insert: 8

Menu:
1. Insert
2. Remove
3. Search
4. Display
5. Exit
Enter your choice: 4
Circular Linked List: 3->5->8->[head]
```

```
Menu:
1. Insert
2. Remove
3. Search
4. Display
5. Exit
Enter your choice: 2
Enter data to remove: 4
4 not found in the list

Menu:
1. Insert
2. Remove
3. Search
4. Display
5. Exit
Enter your choice: 3
Enter data to search: 5
5 found at address: 0x555eb1c126f0

Menu:
1. Insert
2. Remove
3. Search
4. Display
5. Exit
Enter your choice: 5
Exiting program.
```

Que-4.

```
#include <iostream>
#include <stack>
#include <sstream>
using namespace std;

int performPostFix(char operator1, int operand1, int operand2) {
    switch (operator1) {
        case '+':
            return operand1 + operand2;
        case '-':
            return operand1 - operand2;
        case '*':
            return operand1 * operand2;
        case '/':
            if (operand2 != 0)
                return operand1 / operand2;
            else
                throw runtime_error("Division by 0");
        default:
            throw runtime_error("Invalid operator");
    }
}

int evaluatePostFix(string expression) {
    stack<int> stackPF;

    istringstream iss(expression);
    string term;

    while (iss >> term) {
        if (isdigit(term[0]) || (term[0] == '-' && isdigit(term[1]))) {
            stackPF.push(stoi(term));
        } else {
            int operand1 = stackPF.top();
            stackPF.pop();
            int operand2 = stackPF.top();
            stackPF.pop();
            int result = performPostFix(term[0], operand1, operand2);
            stackPF.push(result);
        }
    }
    if (stackPF.size() == 1)
        return stackPF.top();
    else
        throw runtime_error("Invalid expression");
}

int main() {
    string expression;

    int choice;
    do {
        cout << "\nMenu:\n";
        cout << "1. Enter a PostFix expression\n";
        cout << "2. Evaluate the last entered expression\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter a PostFix expression: ";
                cin.ignore(); // Clear the newline character from the buffer
                getline(cin, expression);
                break;

            case 2:
                if (expression.empty()) {
                    cout << "No expression entered yet. Please enter a PostFix expression
first.\n";
                } else {
                    cout << "PostFix expression: " << expression << endl;
                    try {
                        cout << "Result: " << evaluatePostFix(expression) << endl;
                    } catch (const exception& e) {
                        cerr << "Error: " << e.what() << endl;
                    }
                }
                break;

            case 3:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Try again.\n";
        }

    } while (choice != 3);

    return 0;
}
```

Output:

```
arya@AryaRoomL:~/Desktop/PracticalFileDS/PracticalFile2$
```

```
./P4_CSC-22-11_DS

Menu:
1. Enter a PostFix expression
2. Evaluate the last entered expression
3. Exit
Enter your choice: 1
Enter a PostFix expression: 3 4 + 5 *

Menu:
1. Enter a PostFix expression
2. Evaluate the last entered expression
3. Exit
Enter your choice: 2
PostFix expression: 3 4 + 5 *
Result: 35

Menu:
1. Enter a PostFix expression
2. Evaluate the last entered expression
3. Exit
Enter your choice: 3
Exiting program.
```

Que-5.

```
#include <iostream>
#include <stdexcept>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

class Queue {
private:
    Node* front;
    Node* rear;
    size_t count;

public:
```

```
Queue() : front(nullptr), rear(nullptr), count(0) {}

void enqueue(int data) {
    Node* newNode = new Node(data);

    if (isEmpty()) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    count++;
}

int dequeue() {
    if (isEmpty()) {
        throw out_of_range("Queue is Empty!");
    }

    int value = front->data;
    Node* temp = front;
    front = front->next;
    delete temp;
    count--;

    if (isEmpty()) {
        rear = front;
    }

    return value;
}

void display() {
    Node* temp = front;
    while (temp != nullptr) {
        cout << temp->data << "-";
        temp = temp->next;
    }
    cout << "nullptr" << endl;
}

bool isEmpty() {
    return count == 0;
    }

    int peek() {
        if (isEmpty()) {
            throw out_of_range("Queue is Empty!");
        }
        return front->data;
    }

    int size() {
        return count;
    }

};

int main() {
    Queue queue;

    int choice, data;

    do {
        cout << "\nMenu:\n";
        cout << "1. Enqueue\n";
        cout << "2. Dequeue\n";
        cout << "3. Peek\n";
        cout << "4. Display\n";
        cout << "5. Size\n";
        cout << "6. Exit\n";

        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter data to enqueue: ";
                cin >> data;
                queue.enqueue(data);
                break;

            case 2:
                try {
                    cout << "Dequeued element: " << queue.dequeue() << endl;
                } catch (const exception& e) {
                    cerr << "Error: " << e.what() << endl;
                }
                break;

            case 3:
                try {
                    cout << "Front element: " << queue.peek() << endl;
                } catch (const exception& e) {
                    cerr << "Error: " << e.what() << endl;
                }
                break;

            case 4:
                cout << "Queue: ";
                queue.display();
                break;

            case 5:
                cout << "Size of Queue: " << queue.size() << endl;
                break;

            case 6:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Try again.\n";
        }

    } while (choice != 6);

    return 0;
}
```

Output:

```
arya@AryaRoomL:~/Desktop/PracticalFileDS/PracticalFile2$
./P5_CSC-22-11_DS

Menu:
1. Enqueue
```

```
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 1
Enter data to enqueue: 4

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 1
Enter data to enqueue: 6

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 4
Queue: 4-6-nullptr

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 1
Enter data to enqueue: 7

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 4
Queue: 4-6-7-nullptr
```

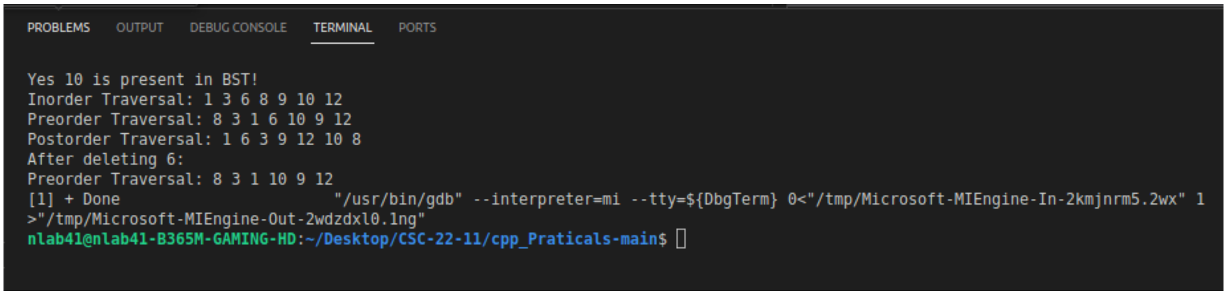
```
Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 2
Dequeued element: 4

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 4
Queue: 6-7-nullptr

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 3
Front element: 6

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 5
Size of Queue: 2

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Size
6. Exit
Enter your choice: 6
```

```
Exiting program.
```

Que-6.

```
#include <iostream>
using namespace std;

class TreeNode {
public:
    int key;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int value) : key(value), left(nullptr), right(nullptr) {}
};

class BinarySearchTree {
private:
    TreeNode* root;

    // ... (Other functions remain the same)

public:
    BinarySearchTree() : root(nullptr) {}

    void insert(int key) {
        root = insert(root, key);
    }

    void deleteNode(int key) {
        root = deleteNode(root, key);
    }

    bool search(int key) {
        return search(root, key);
    }

    void displayInorder() {
        cout << "Inorder Traversal: ";
        inorderTraversal(root);
        cout << endl;
    }
    void displayPreorder() {
        cout << "Preorder Traversal: ";
        preorderTraversal(root);
        cout << endl;
    }

    void displayPostorder() {
        cout << "Postorder Traversal: ";
        postorderTraversal(root);
        cout << endl;
    }
};

int main() {
    BinarySearchTree bst;
    int choice, key;

    do {
        cout << "\nMenu:\n";
        cout << "1. Insert\n";
        cout << "2. Delete\n";
        cout << "3. Search\n";
        cout << "4. Display Inorder\n";
        cout << "5. Display Preorder\n";
        cout << "6. Display Postorder\n";
        cout << "7. Exit\n";

        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter key to insert: ";
                cin >> key;
                bst.insert(key);
                break;

            case 2:
                cout << "Enter key to delete: ";
                cin >> key;
                bst.deleteNode(key);
                break;
            case 3:
                cout << "Enter key to search: ";
                cin >> key;
                if (bst.search(key))
                    cout << key << " is present in BST!" << endl;
                else
                    cout << key << " not found in BST!" << endl;
                break;

            case 4:
                bst.displayInorder();
                break;

            case 5:
                bst.displayPreorder();
                break;

            case 6:
                bst.displayPostorder();
                break;

            case 7:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Try again.\n";
        }

    } while (choice != 7);

    return 0;
}
```

Output:



```
Yes 10 is present in BST!
Inorder Traversal: 1 3 6 8 9 10 12
Preorder Traversal: 8 3 1 6 10 9 12
Postorder Traversal: 1 6 3 9 12 10 8
After deleting 6:
Preorder Traversal: 8 3 1 10 9 12
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-2kmjnrm5.2wx" 1
>"/tmp/Microsoft-MIEngine-Out-2wdzdxl0.1ng"
nlab41@nlab41-B365M-GAMING-HD:~/Desktop/CSC-22-11/cpp_Praticals-main$ 
```

Que-7.

```
#include <iostream>
#include <sstream>
#include <algorithm>
using namespace std;

class AVLNode {
public:
    int key;
    int height;
    AVLNode* left;
    AVLNode* right;

    AVLNode(int k) : key(k), height(1), left(nullptr), right(nullptr) {}
};

class AVLTree {
private:
    AVLNode* root;

    int getHeight(AVLNode* node) {
        if (node == nullptr)
            return 0;
        return node->height;
    }

    int getBalance(AVLNode* node) {
        if (node == nullptr)
            return 0;
        return getHeight(node->left) - getHeight(node->right);
    }
```

```
AVLNode* rotateRight(AVLNode* y) {
    AVLNode* x = y->left;
    AVLNode* T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;
    x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;

    return x;
}

AVLNode* rotateLeft(AVLNode* x) {
    AVLNode* y = x->right;
    AVLNode* T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;

    return y;
}

AVLNode* insertNode(AVLNode* node, int key) {
    if (node == nullptr)
        return new AVLNode(key);

    if (key < node->key)
        node->left = insertNode(node->left, key);
    else if (key > node->key)
        node->right = insertNode(node->right, key);
    else
        return node; // Duplicate keys are not allowed

    node->height = 1 + std::max(getHeight(node->left), getHeight(node->right));

    int balance = getBalance(node);

    // Left Heavy
    if (balance > 1) {
            if (key < node->left->key) {
                // Left-Left case
                return rotateRight(node);
            } else {
                // Left-Right case
                node->left = rotateLeft(node->left);
                return rotateRight(node);
            }
        }

        // Right Heavy
        if (balance < -1) {
            if (key > node->right->key) {
                // Right-Right case
                return rotateLeft(node);
            } else {
                // Right-Left case
                node->right = rotateRight(node->right);
                return rotateLeft(node);
            }
        }

        return node;
    }

    void inOrderTraversal(AVLNode* node) {
        if (node != nullptr) {
            inOrderTraversal(node->left);
            std::cout << node->key << " ";
            inOrderTraversal(node->right);
        }
    }

public:
    AVLTree() : root(nullptr) {}

    void insert(int key) {
        root = insertNode(root, key);
    }

    void inOrder() {
        inOrderTraversal(root);
        std::cout << std::endl;
    }
};

int main() {
    AVLTree avl;
    int choice, key;

    do {
        cout << "\nMenu:\n";
        cout << "1. Insert\n";
        cout << "2. In-order Traversal\n";
        cout << "3. Exit\n";

        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter key to insert: ";
                cin >> key;
                avl.insert(key);
                break;

            case 2:
                cout << "In-order Traversal: ";
                avl.inOrder();
                break;

            case 3:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Try again.\n";
        }

    } while (choice != 3);

    return 0;
}
```

Output:

```
arya@AryaRoomL:~/Desktop/PracticalFileDS/PracticalFile3$
./P7_CSC-22-11_DS

Menu:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter key to insert: 3

Menu:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter key to insert: 5

Menu:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter key to insert: 7

Menu:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 4
Invalid choice. Try again.

Menu:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 2
In-order Traversal: 3 5 7

Menu:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 3
Exiting program.
```
