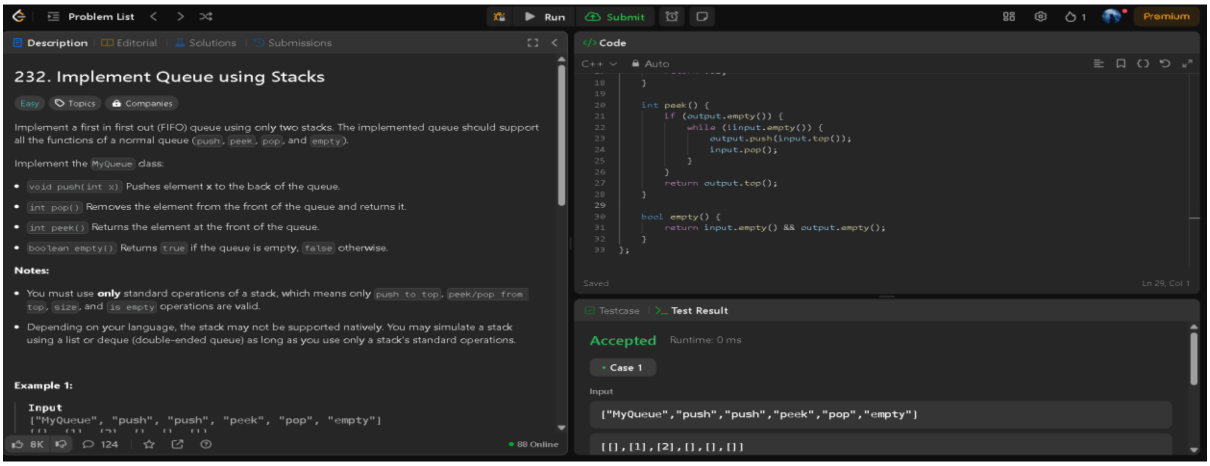


232. Implement Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push`, `peek`, `pop`, and `empty`).

Implement the `MyQueue` class:

- `void push(int x)` Pushes element x to the back of the queue.
- `int pop()` Removes the element from the front of the queue and returns it.
- `int peek()` Returns the element at the front of the queue.
- `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a stack, which means only `push to top`, `peek/pop from top`, `size`, and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

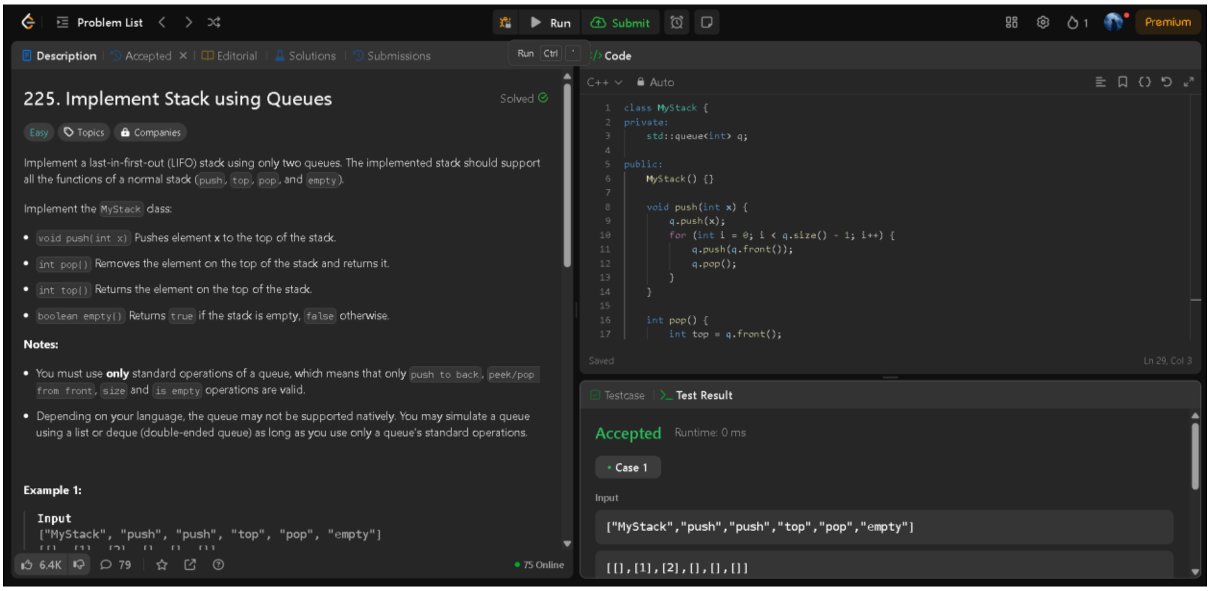
Example 1:

```
Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
```

```
    }

    int peek() {
        if (output.empty()) {
            while (!input.empty()) {
                output.push(input.top());
                input.pop();
            }
        }
        return output.top();
    }

    bool empty() {
        return input.empty() && output.empty();
    }
};
```

Accepted   Runtime: 0 ms

```
["MyQueue","push","push","peek","pop","empty"]
[[],[1],[2],[],[],[]]
```



225. Implement Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:

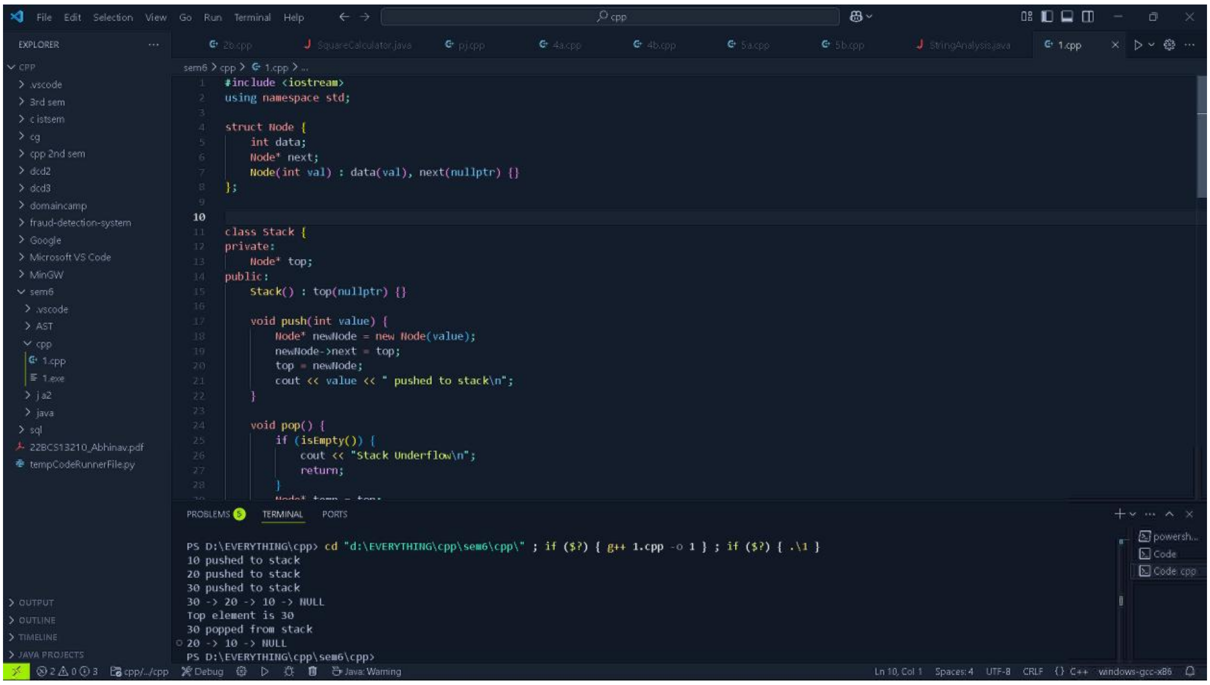
```
Input
["MyStack", "push", "push", "top", "pop", "empty"]
```

```
class MyStack {
private:
    std::queue<int> q;

public:
    MyStack() {}

    void push(int x) {
        q.push(x);
        for (int i = 0; i < q.size() - 1; i++) {
            q.push(q.front());
            q.pop();
        }
    }

    int pop() {
        int top = q.front();
```

Accepted   Runtime: 0 ms

```
["MyStack","push","push","top","pop","empty"]
[[],[1],[2],[],[],[]]
```

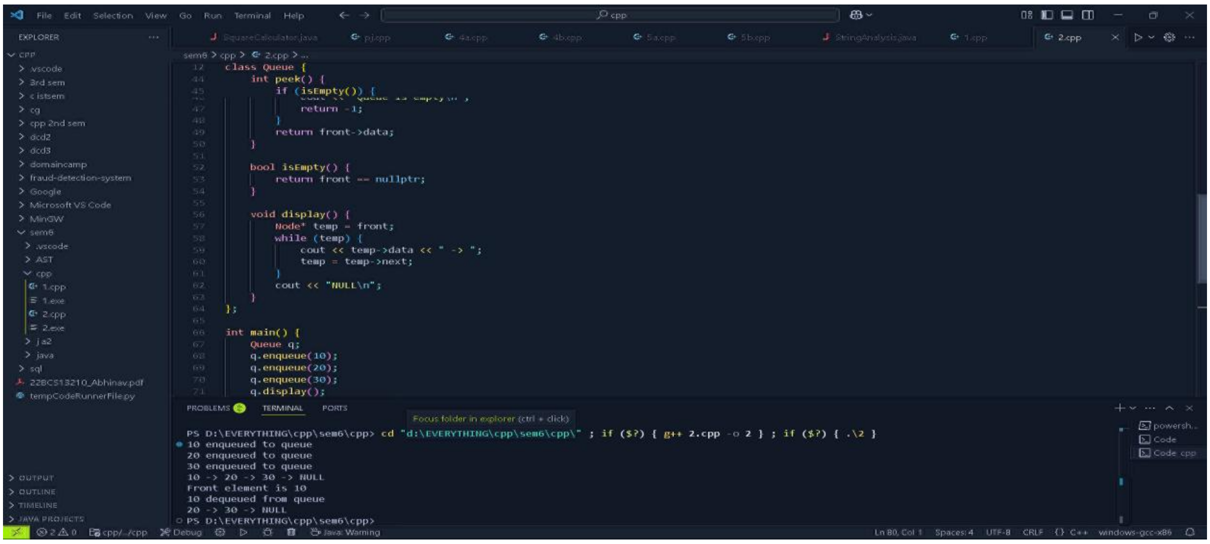
Implement Stack using Linked List.



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

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};


class Stack {
private:
    Node* top;
public:
    Stack() : top(nullptr) {}

    void push(int value) {
        Node* newNode = new Node(value);
        newNode->next = top;
        top = newNode;
        cout << value << " pushed to stack\n";
    }

    void pop() {
        if (isEmpty()) {
            cout << "Stack Underflow\n";
            return;
        }
        Node* temp = top;
```

```
PS D:\EVERYTHING\cpp> cd "d:\EVERYTHING\cpp\sem6\cpp\" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 -> 20 -> 10 -> NULL
Top element is 30
30 popped from stack
20 -> 10 -> NULL
PS D:\EVERYTHING\cpp\sem6\cpp>
```

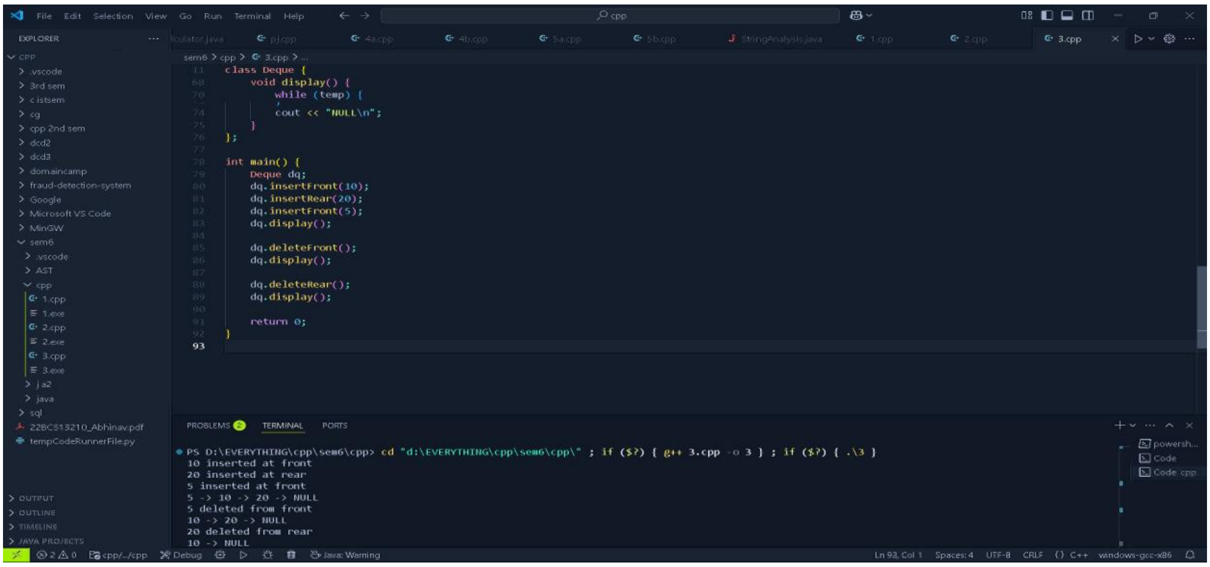
Queue using Linked List



```cpp
class Queue {
    int peek() {
        if (isEmpty()) {
            return -1;
        }
        return front->data;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    void display() {
        Node* temp = front;
        while (temp) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL\n";
    }
};

int main() {
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.display();
```

```
PS D:\EVERYTHING\cpp\sem6\cpp> cd "d:\EVERYTHING\cpp\sem6\cpp\" ; if ($?) { g++ 2.cpp -o 2 } ; if ($?) { .\2 }
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
10 -> 20 -> 30 -> NULL
Front element is 10
10 dequeued from queue
20 -> 30 -> NULL
PS D:\EVERYTHING\cpp\sem6\cpp>
```

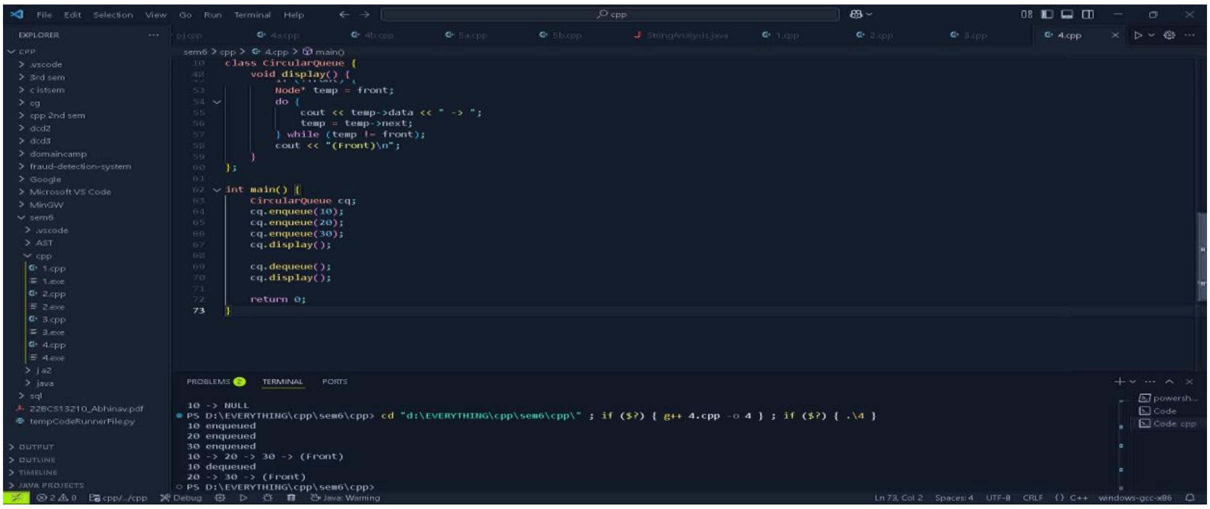
Deque using Doubly Linked List



```cpp
class Deque {
    void display() {
        while (temp) {
        cout << "NULL\n";
    }
};

int main() {
    Deque dq;
    dq.insertFront(10);
    dq.insertRear(20);
    dq.insertFront(5);
    dq.display();

    dq.deleteFront();
    dq.display();

    dq.deleteRear();
    dq.display();

    return 0;
}
```

```
PS D:\EVERYTHING\cpp\sem6\cpp> cd "d:\EVERYTHING\cpp\sem6\cpp\" ; if ($?) { g++ 3.cpp -o 3 } ; if ($?) { .\3 }
10 inserted at front
20 inserted at rear
5 inserted at front
5 -> 10 -> 20 -> NULL
5 deleted from front
10 -> 20 -> NULL
20 deleted from rear
10 -> NULL
```

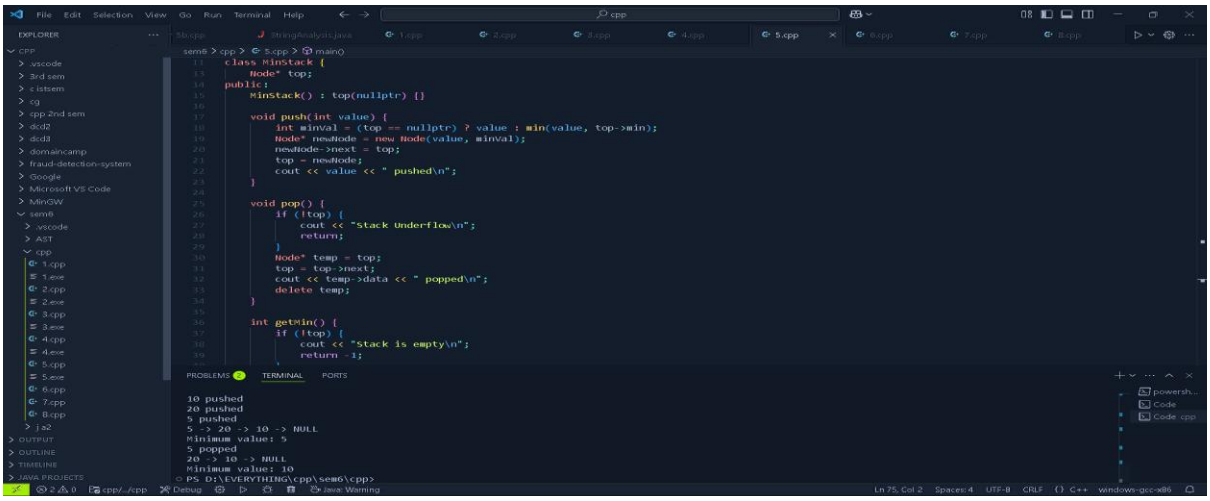
Circular Queue using Linked List



```cpp
class CircularQueue {
    void display() {
        Node* temp = front;
        do {
            cout << temp->data << " -> ";
            temp = temp->next;
        } while (temp != front);
        cout << "(Front)\n";
    }
};

int main() {
    CircularQueue cq;
    cq.enqueue(10);
    cq.enqueue(20);
    cq.enqueue(30);
    cq.display();

    cq.dequeue();
    cq.display();

    return 0;
}
```

```
10 -> NULL
PS D:\EVERYTHING\cpp\sem6\cpp> cd "d:\EVERYTHING\cpp\sem6\cpp\" ; if ($?) { g++ 4.cpp -o 4 } ; if ($?) { .\4 }
10 enqueued
20 enqueued
30 enqueued
10 -> 20 -> 30 -> (Front)
10 dequeued
20 -> 30 -> (Front)
PS D:\EVERYTHING\cpp\sem6\cpp>
```

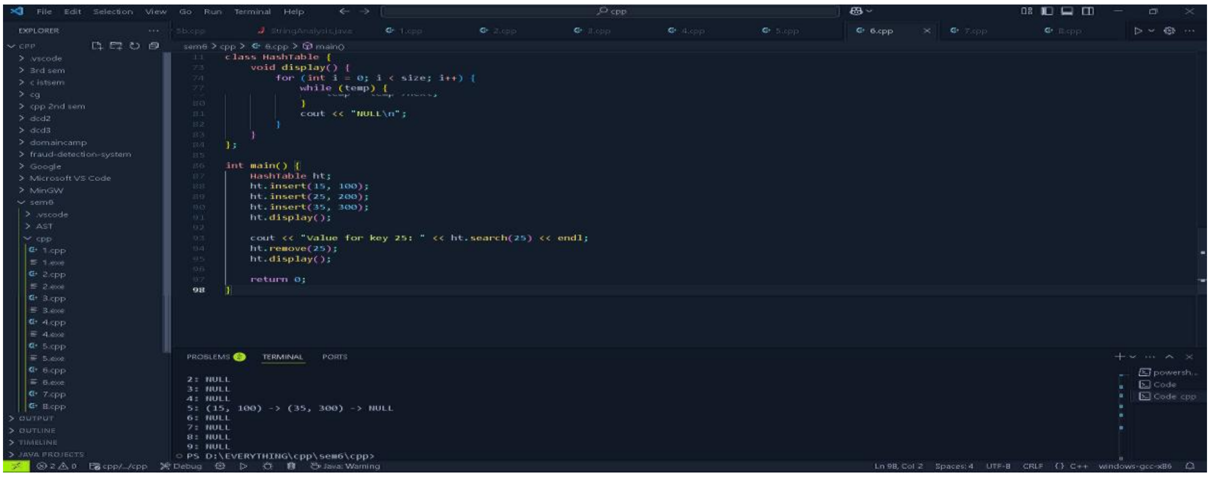
Min Stack using Linked List.



```
class MinStack {
    Node* top;
public:
    MinStack() : top(nullptr) {}

    void push(int value) {
        int minVal = (top == nullptr) ? value : min(value, top->min);
        Node* newNode = new Node(value, minVal);
        newNode->next = top;
        top = newNode;
        cout << value << " pushed\n";
    }

    void pop() {
        if (!top) {
            cout << "Stack Underflow\n";
            return;
        }
        Node* temp = top;
        top = top->next;
        cout << temp->data << " popped\n";
        delete temp;
    }

    int getMin() {
        if (!top) {
            cout << "Stack is empty\n";
            return -1;
```

```
10 pushed
20 pushed
5 pushed
5 -> 20 -> 10 -> NULL
Minimum value: 5
5 popped
20 -> 10 -> NULL
Minimum value: 10
PS D:\EVERYTHING\cpp\sem6\cpp>
```

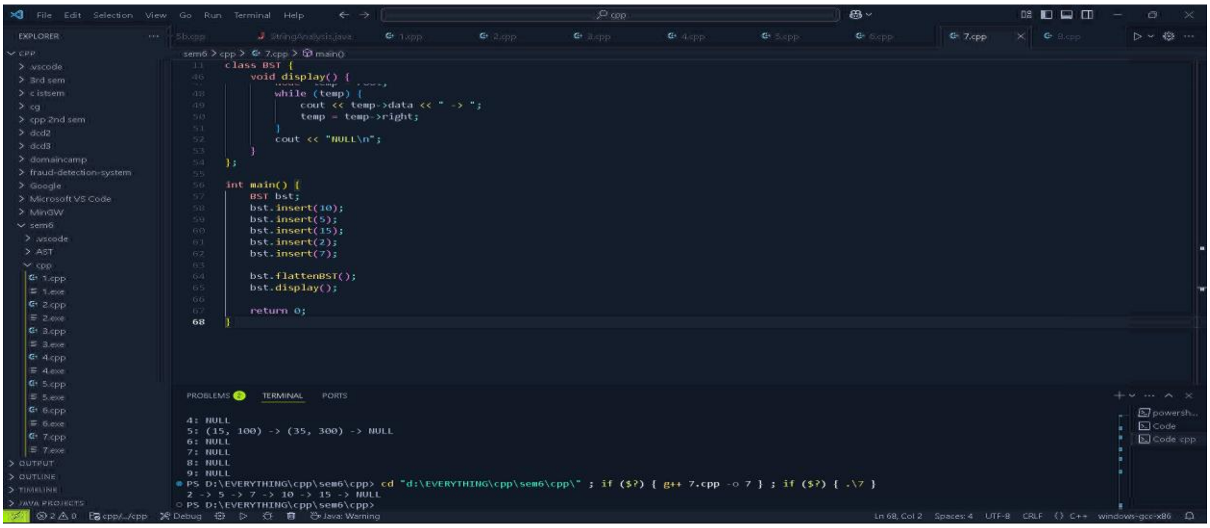
Hash Table using Linked List



```
class HashTable {
    void display() {
        for (int i = 0; i < size; i++) {
            while (temp) {
            }
            cout << "NULL\n";
        }
    }
};

int main() {
    HashTable ht;
    ht.insert(15, 100);
    ht.insert(25, 200);
    ht.insert(35, 300);
    ht.display();

    cout << "Value for key 25: " << ht.search(25) << endl;
    ht.remove(25);
    ht.display();

    return 0;
}
```

```
2: NULL
3: NULL
4: NULL
5: (15, 100) -> (35, 300) -> NULL
6: NULL
7: NULL
8: NULL
9: NULL
PS D:\EVERYTHING\cpp\sem6\cpp>
```

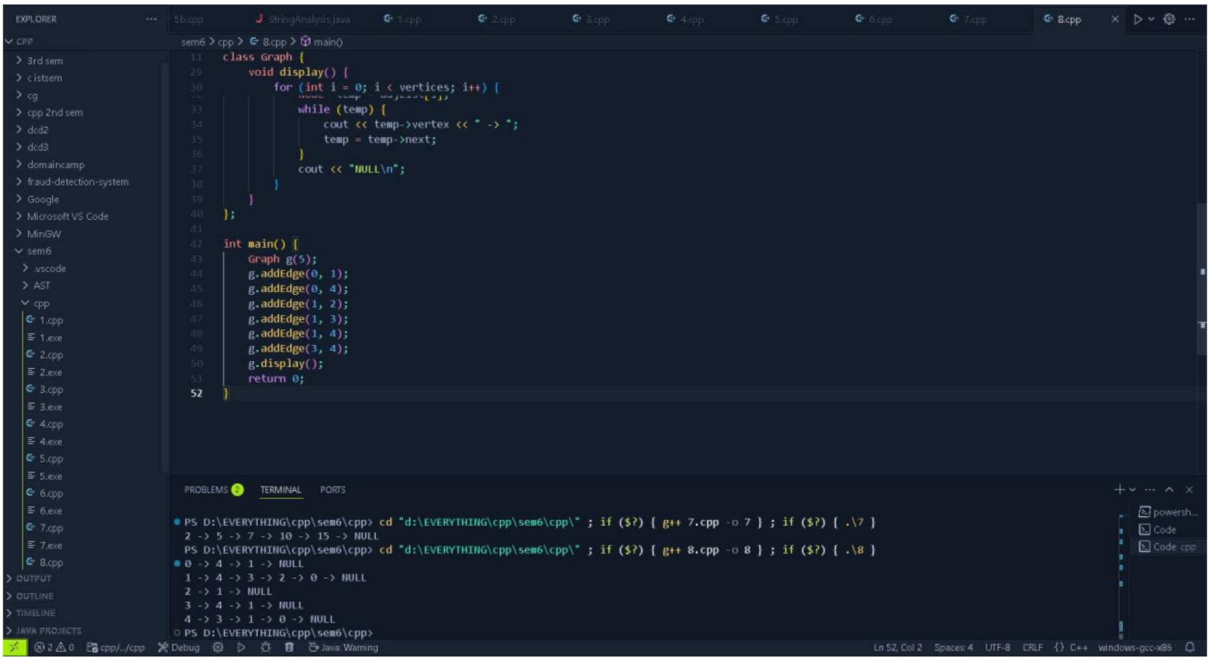
BST using Linked List



```
class BST {
    void display() {
        while (temp) {
            cout << temp->data << " -> ";
            temp = temp->right;
        }
        cout << "NULL\n";
    }
};

int main() {
    BST bst;
    bst.insert(10);
    bst.insert(5);
    bst.insert(15);
    bst.insert(2);
    bst.insert(7);

    bst.flattenBST();
    bst.display();

    return 0;
}
```

```
4: NULL
5: (15, 100) -> (35, 300) -> NULL
6: NULL
7: NULL
8: NULL
9: NULL
PS D:\EVERYTHING\cpp\sem6\cpp> cd "d:\EVERYTHING\cpp\sem6\cpp\" ; if ($?) { g++ 7.cpp -o 7 } ; if ($?) { .\7 }
2 -> 5 -> 7 -> 10 -> 15 -> NULL
PS D:\EVERYTHING\cpp\sem6\cpp>
```

Graph using Linked List



```
class Graph {
    void display() {
        for (int i = 0; i < vertices; i++) {
            while (temp) {
                cout << temp->vertex << " -> ";
                temp = temp->next;
            }
            cout << "NULL\n";
        }
    }
};

int main() {
    Graph g(5);
    g.addEdge(0, 1);
    g.addEdge(0, 4);
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(3, 4);
    g.display();
    return 0;
}
```

```
PS D:\EVERYTHING\cpp\sem6\cpp> cd "d:\EVERYTHING\cpp\sem6\cpp\" ; if ($?) { g++ 7.cpp -o 7 } ; if ($?) { .\7 }
2 -> 5 -> 7 -> 10 -> 15 -> NULL
PS D:\EVERYTHING\cpp\sem6\cpp> cd "d:\EVERYTHING\cpp\sem6\cpp\" ; if ($?) { g++ 8.cpp -o 8 } ; if ($?) { .\8 }
0 -> 4 -> 1 -> NULL
1 -> 4 -> 3 -> 2 -> 0 -> NULL
2 -> 1 -> NULL
3 -> 4 -> 1 -> NULL
4 -> 3 -> 1 -> 0 -> NULL
PS D:\EVERYTHING\cpp\sem6\cpp>
```