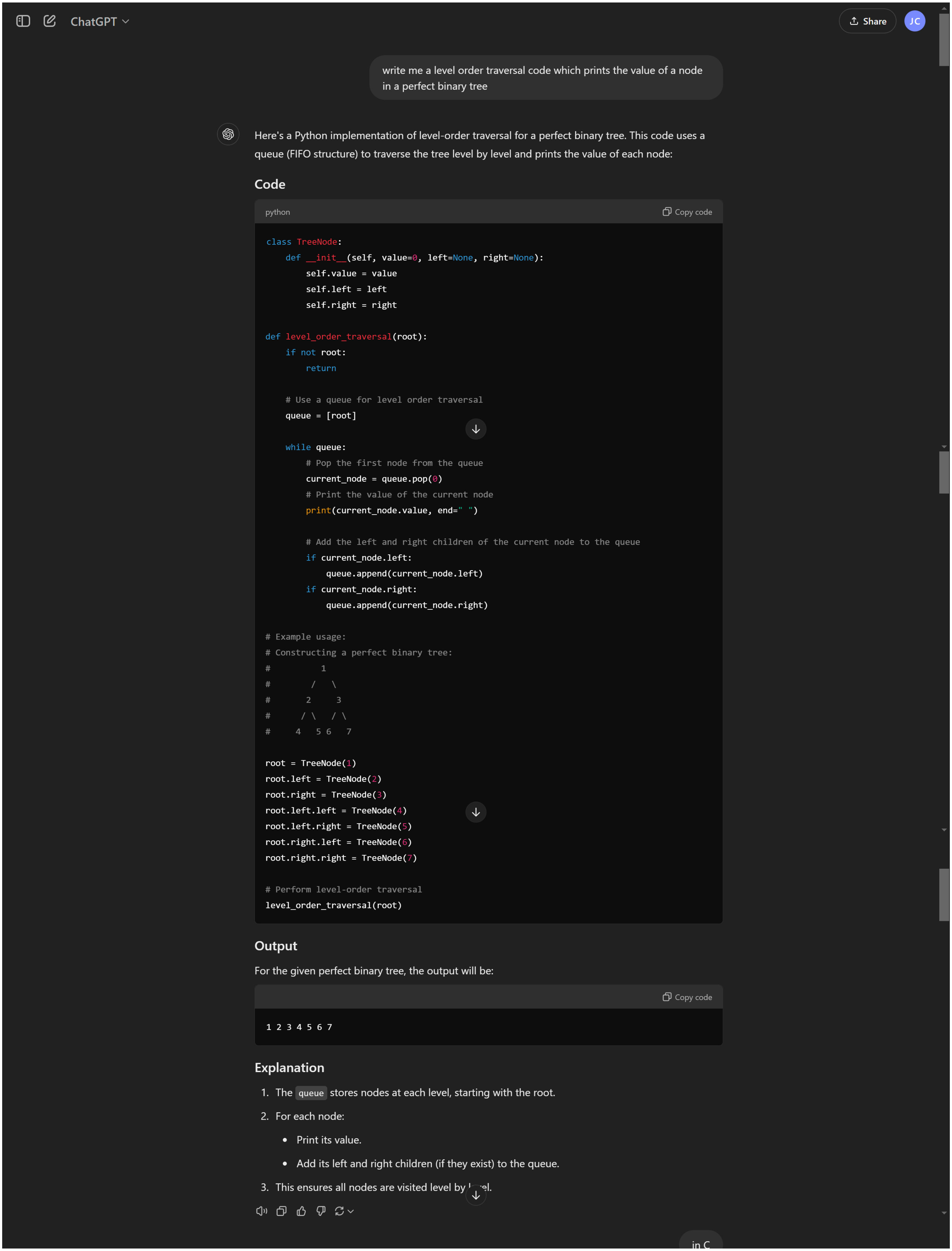


write me a level order traversal code which prints the value of a node in a perfect binary tree

Here's a Python implementation of level-order traversal for a perfect binary tree. This code uses a queue (FIFO structure) to traverse the tree level by level and prints the value of each node:

Code

```python
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

def level_order_traversal(root):
    if not root:
        return

    # Use a queue for level order traversal
    queue = [root]

    while queue:
        # Pop the first node from the queue
        current_node = queue.pop(0)
        # Print the value of the current node
        print(current_node.value, end=" ")

        # Add the left and right children of the current node to the queue
        if current_node.left:
            queue.append(current_node.left)
        if current_node.right:
            queue.append(current_node.right)

# Example usage:
# Constructing a perfect binary tree:
#          1
#        /   \
#       2     3
#      / \   / \
#     4   5 6   7

root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)

# Perform level-order traversal
level_order_traversal(root)
```

Output

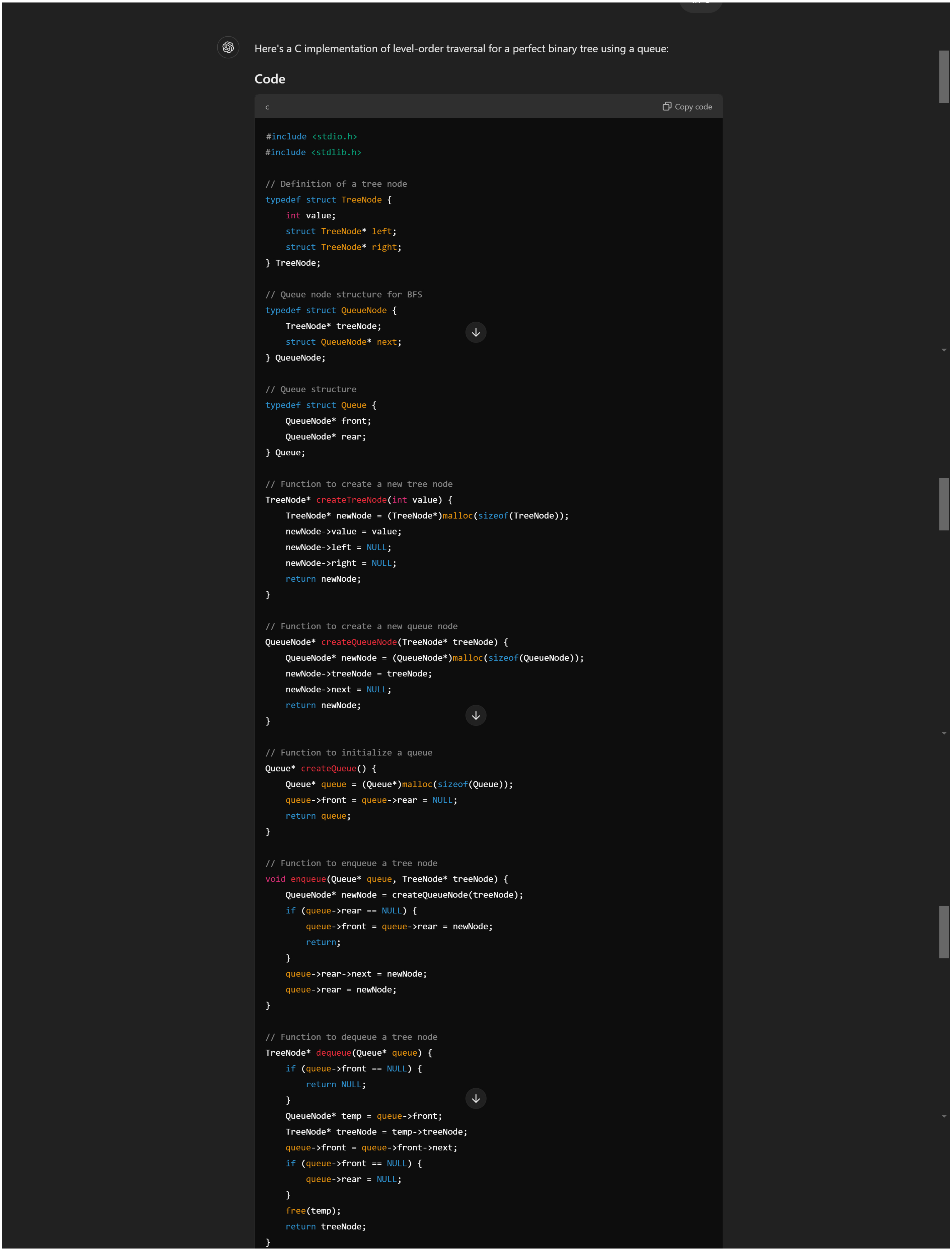
For the given perfect binary tree, the output will be:

```
1 2 3 4 5 6 7
```

Explanation

1. The `queue` stores nodes at each level, starting with the root.
2. For each node:
   - Print its value.
   - Add its left and right children (if they exist) to the queue.
3. This ensures all nodes are visited level by level.

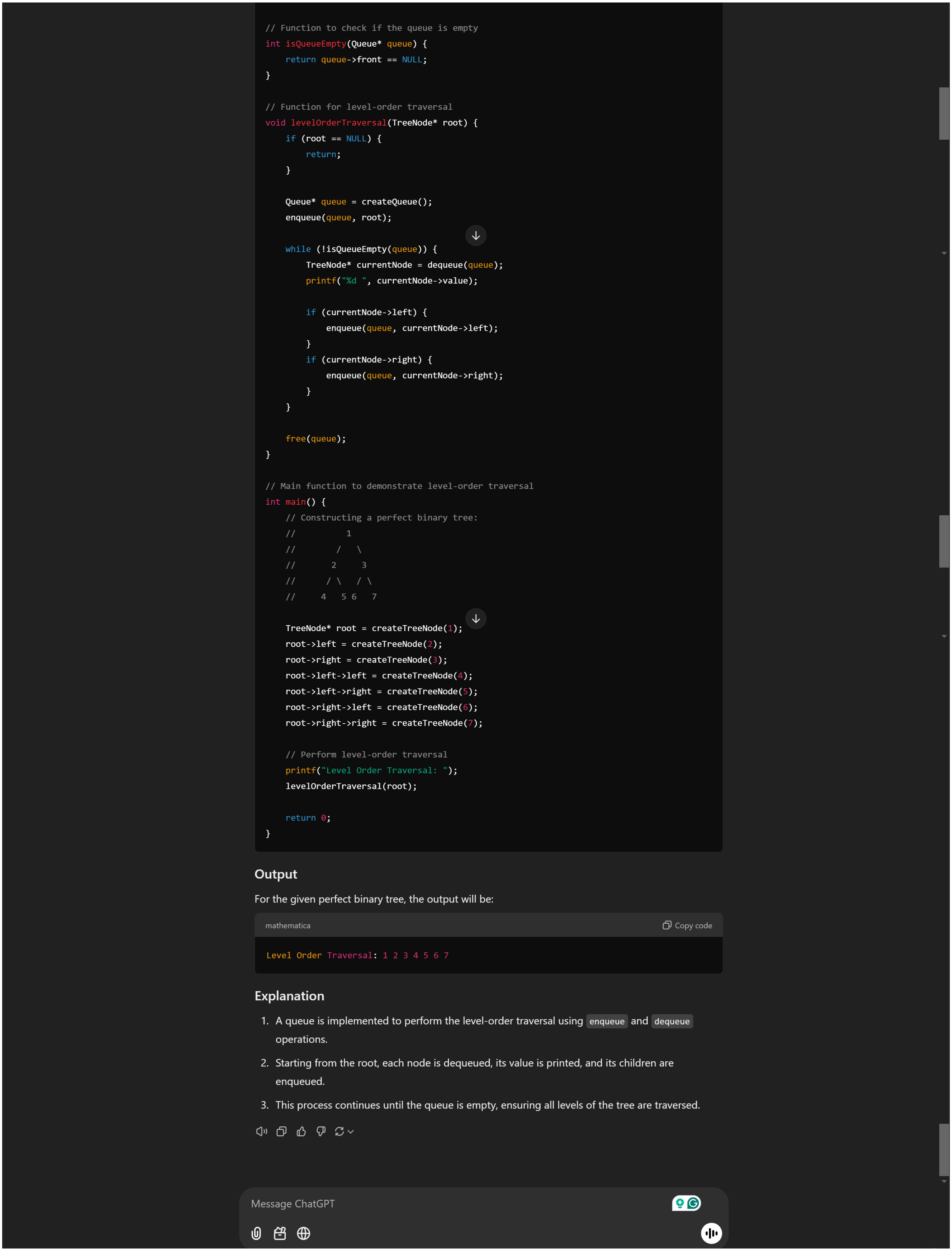
in C



Here's a C implementation of level-order traversal for a perfect binary tree using a queue:

Code

```c
#include <stdio.h>
#include <stdlib.h>

// Definition of a tree node
typedef struct TreeNode {
    int value;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;

// Queue node structure for BFS
typedef struct QueueNode {
    TreeNode* treeNode;
    struct QueueNode* next;
} QueueNode;

// Queue structure
typedef struct Queue {
    QueueNode* front;
    QueueNode* rear;
} Queue;

// Function to create a new tree node
TreeNode* createTreeNode(int value) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->value = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to create a new queue node
QueueNode* createQueueNode(TreeNode* treeNode) {
    QueueNode* newNode = (QueueNode*)malloc(sizeof(QueueNode));
    newNode->treeNode = treeNode;
    newNode->next = NULL;
    return newNode;
}

// Function to initialize a queue
Queue* createQueue() {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to enqueue a tree node
void enqueue(Queue* queue, TreeNode* treeNode) {
    QueueNode* newNode = createQueueNode(treeNode);
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
        return;
    }
    queue->rear->next = newNode;
    queue->rear = newNode;
}

// Function to dequeue a tree node
TreeNode* dequeue(Queue* queue) {
    if (queue->front == NULL) {
        return NULL;
    }
    QueueNode* temp = queue->front;
    TreeNode* treeNode = temp->treeNode;
    queue->front = queue->front->next;
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    free(temp);
    return treeNode;
}
```



```
// Function to check if the queue is empty
int isQueueEmpty(Queue* queue) {
    return queue->front == NULL;
}

// Function for level-order traversal
void levelOrderTraversal(TreeNode* root) {
    if (root == NULL) {
        return;
    }

    Queue* queue = createQueue();
    enqueue(queue, root);

    while (!isQueueEmpty(queue)) {
        TreeNode* currentNode = dequeue(queue);
        printf("%d ", currentNode->value);

        if (currentNode->left) {
            enqueue(queue, currentNode->left);
        }
        if (currentNode->right) {
            enqueue(queue, currentNode->right);
        }
    }

    free(queue);
}

// Main function to demonstrate level-order traversal
int main() {
    // Constructing a perfect binary tree:
    //          1
    //        /   \
    //       2     3
    //      / \   / \
    //     4   5 6   7

    TreeNode* root = createTreeNode(1);
    root->left = createTreeNode(2);
    root->right = createTreeNode(3);
    root->left->left = createTreeNode(4);
    root->left->right = createTreeNode(5);
    root->right->left = createTreeNode(6);
    root->right->right = createTreeNode(7);

    // Perform level-order traversal
    printf("Level Order Traversal: ");
    levelOrderTraversal(root);

    return 0;
}
```

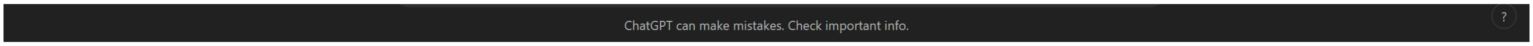
Output

For the given perfect binary tree, the output will be:

```
Level Order Traversal: 1 2 3 4 5 6 7
```

Explanation

1. A queue is implemented to perform the level-order traversal using `enqueue` and `dequeue` operations.
2. Starting from the root, each node is dequeued, its value is printed, and its children are enqueued.
3. This process continues until the queue is empty, ensuring all levels of the tree are traversed.


ChatGPT can make mistakes. Check important info.