Week4 - Lab Practice : Queue

Group 6 :

- Chea Kuyhong (IDTB110172)
- Chhuong Sophakvatey (IDTB110187)
- Ean Mana (IDTB110367)
- Ly Sokunnita (IDTB110369)
- Nget Sokunkanha (IDTB110054)

- Overview :

This project implements a cafeteria ordering queue using a Linked List with `Node` objects. New orders are added at the rear (tail) and served from the front (head). This enforces FIFO (first-in, first-out) so the first order added is the first served.

- Add orders
- Serve the next order
- Display all waiting orders
- Exit.

We chose to use Linked List (head/tail) for this project because it fits a real cafeteria queue because customers continuously join and leave. The number of orders is unpredictable, so a dynamic structure is needed. Linked lists allow fast insertions at the end (enqueue) and quick removals from the front (dequeue) without shifting elements or defining a fixed size. Each operation is O(1). This makes it more efficient than an array-based queue, which may require compaction or resizing.

Diagram :

Rear

↓

Front -> [Order 1] -> [Order 2] -> [Order 3] -> null

Front points to the oldest order (to be served next). Rear points to the newest order (just added).

- Methods :

Class : Node

- Node(int id, string name, double price, int amount, int d, int m, int y) – Creates a new order node with item details, date, and total price.
- void display_Date() – Prints the order date in DD/MM/YY format.

Class : Queue

- void Enqueue(...) : Add a new order at the rear of the queue. (O(1))
- void Dequeue() : Removes the order at the front; prints a message if empty. (O(1))
- void display_Queue() : Displays all orders in FIFO order. (O(n))

Data Members :

- Node* Front – Points to the first order (head).
- Node* Rear – Points to the last order (tail).
- int length – Tracks number of active orders.

Behavior:

- On empty: Dequeue() and display_Queue() print "The queue is empty!".
- On full: Not applicable (linked list expands dynamically).

Invariants:

Front == nullptr < => Queue empty; Rear->next == nullptr; length equals total nodes.

- Edge Cases Handled

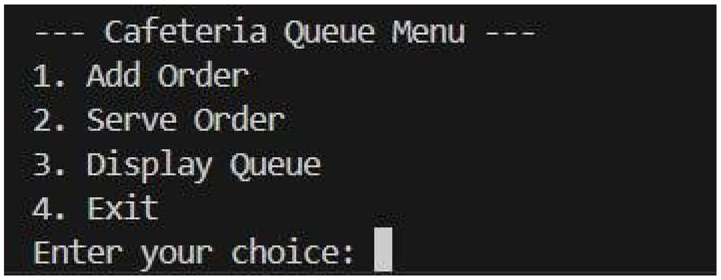
Case	**Description**	**Location**
Empty Dequeue	Checks if Front == nullptr before deleting	Queue::Dequeue()
Empty Display	Checks if Front == nullptr and prints message	Queue::display_Queue()
Removing Last Node	If Front == Rear, both set to nullptr	Queue::Dequeue()
Memory Safety	Nodes are new on enqueue and delete on dequeue	queue.hpp
User Input	Uses cin and getline, minimal validation	main.cpp

- Complexity and Space Analysis :
    - Enqueue : O(1) — adds one node at the end.
    - Dequeue : O(1) — removes one node at front.
    - Display Queue : O(n) — traverses all nodes.
    - Space Cost : O(n) — one node per active order.
    - Compaction : Not required (linked list variant grows and shrinks dynamically).

This efficiency makes linked list queues ideal for continuously changing data sizes, like customer lines.

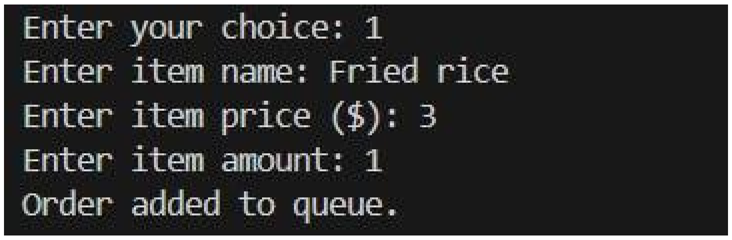
- Output :

Menu :



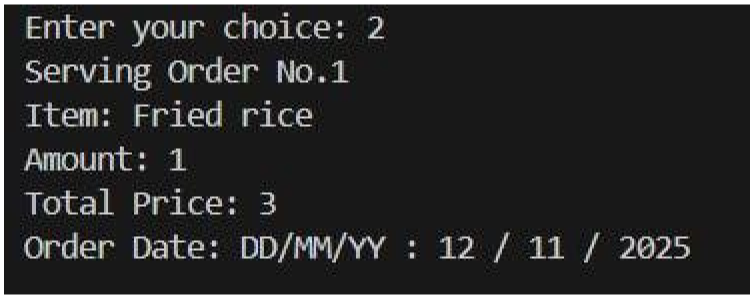
```
--- Cafeteria Queue Menu ---
1. Add Order
2. Serve Order
3. Display Queue
4. Exit
Enter your choice: 
```

Choice 1 :



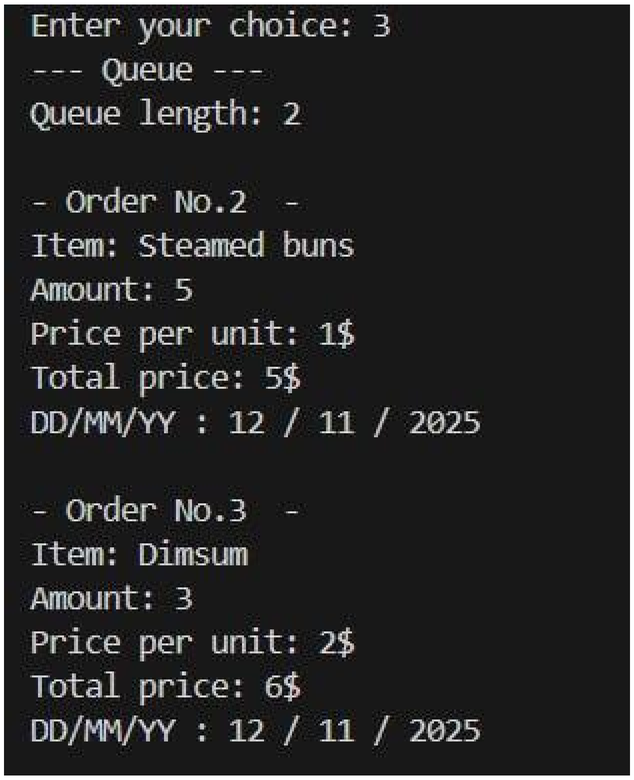
```
Enter your choice: 1
Enter item name: Fried rice
Enter item price ($): 3
Enter item amount: 1
Order added to queue.
```

Choice 2 :



```
Enter your choice: 2
Serving Order No.1
Item: Fried rice
Amount: 1
Total Price: 3
Order Date: DD/MM/YY : 12 / 11 / 2025
```

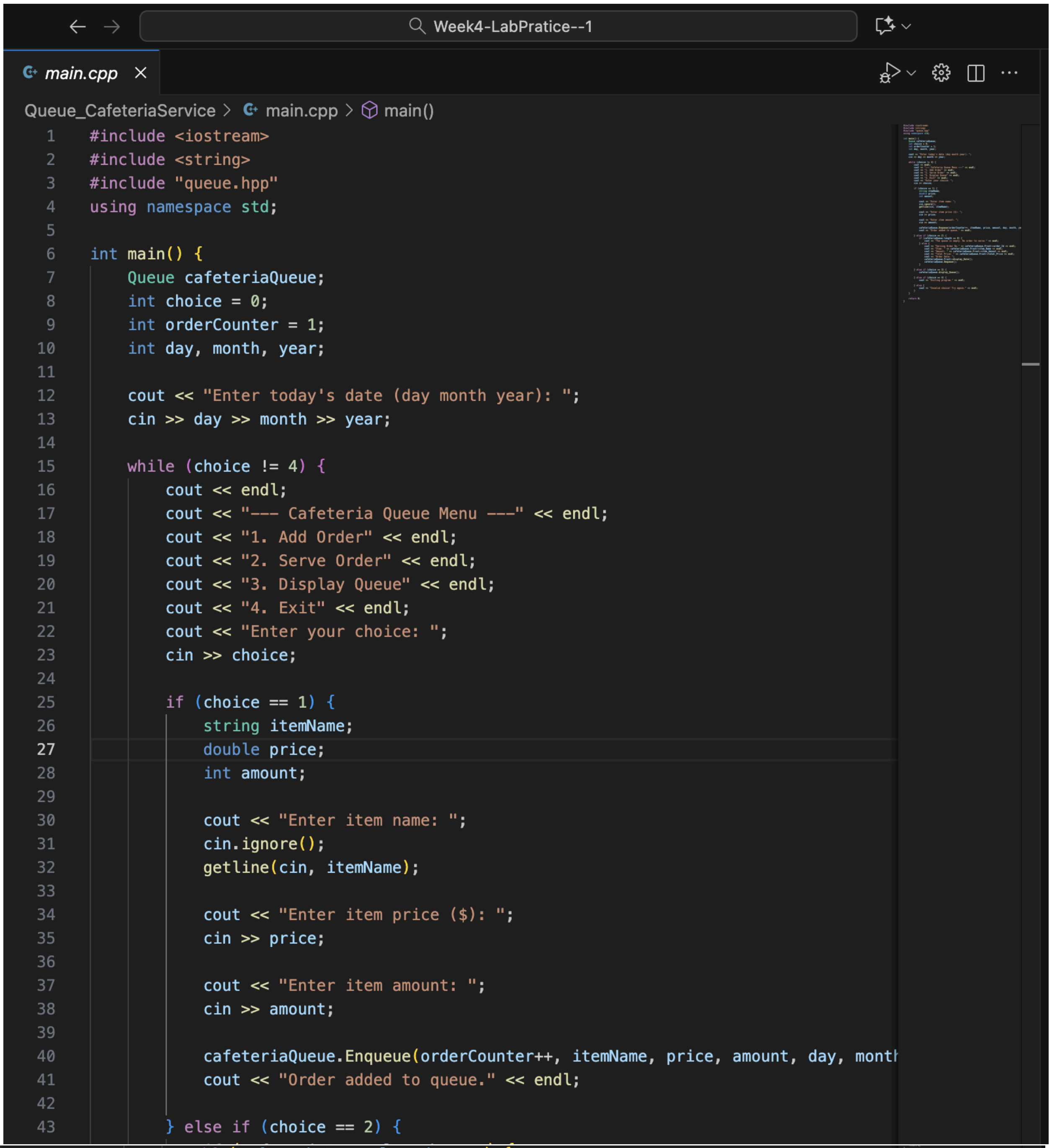
Choice 3 :



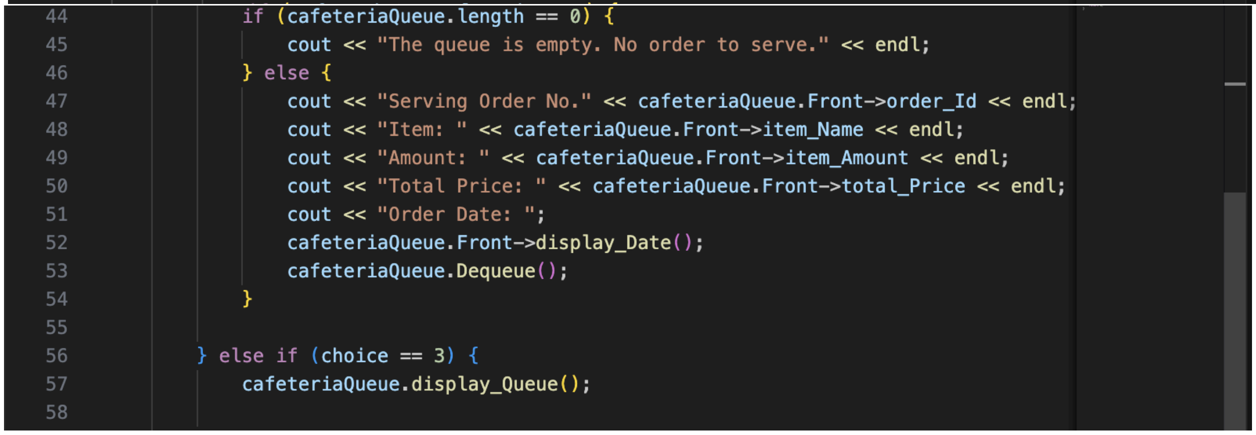
```
Enter your choice: 3
--- Queue ---
Queue length: 2

- Order No.2  -
Item: Steamed buns
Amount: 5
Price per unit: 1$
Total price: 5$
DD/MM/YY : 12 / 11 / 2025

- Order No.3  -
Item: Dimsum
Amount: 3
Price per unit: 2$
Total price: 6$
DD/MM/YY : 12 / 11 / 2025
```

- Code

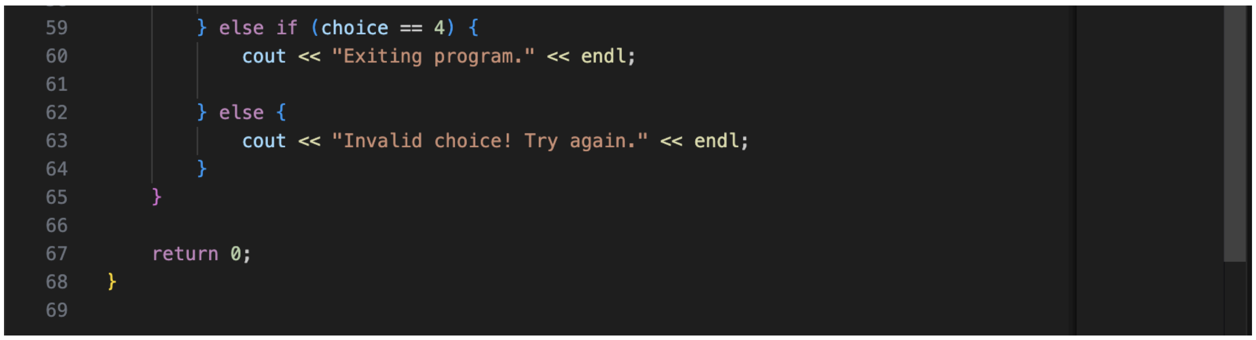
main.cpp



```cpp
#include <iostream>
#include <string>
#include "queue.hpp"
using namespace std;

int main() {
    Queue cafeteriaQueue;
    int choice = 0;
    int orderCounter = 1;
    int day, month, year;

    cout << "Enter today's date (day month year): ";
    cin >> day >> month >> year;

    while (choice != 4) {
        cout << endl;
        cout << "--- Cafeteria Queue Menu ---" << endl;
        cout << "1. Add Order" << endl;
        cout << "2. Serve Order" << endl;
        cout << "3. Display Queue" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1) {
            string itemName;
            double price;
            int amount;

            cout << "Enter item name: ";
            cin.ignore();
            getline(cin, itemName);

            cout << "Enter item price ($): ";
            cin >> price;

            cout << "Enter item amount: ";
            cin >> amount;

            cafeteriaQueue.Enqueue(orderCounter++, itemName, price, amount, day, mont
            cout << "Order added to queue." << endl;

        } else if (choice == 2) {
```

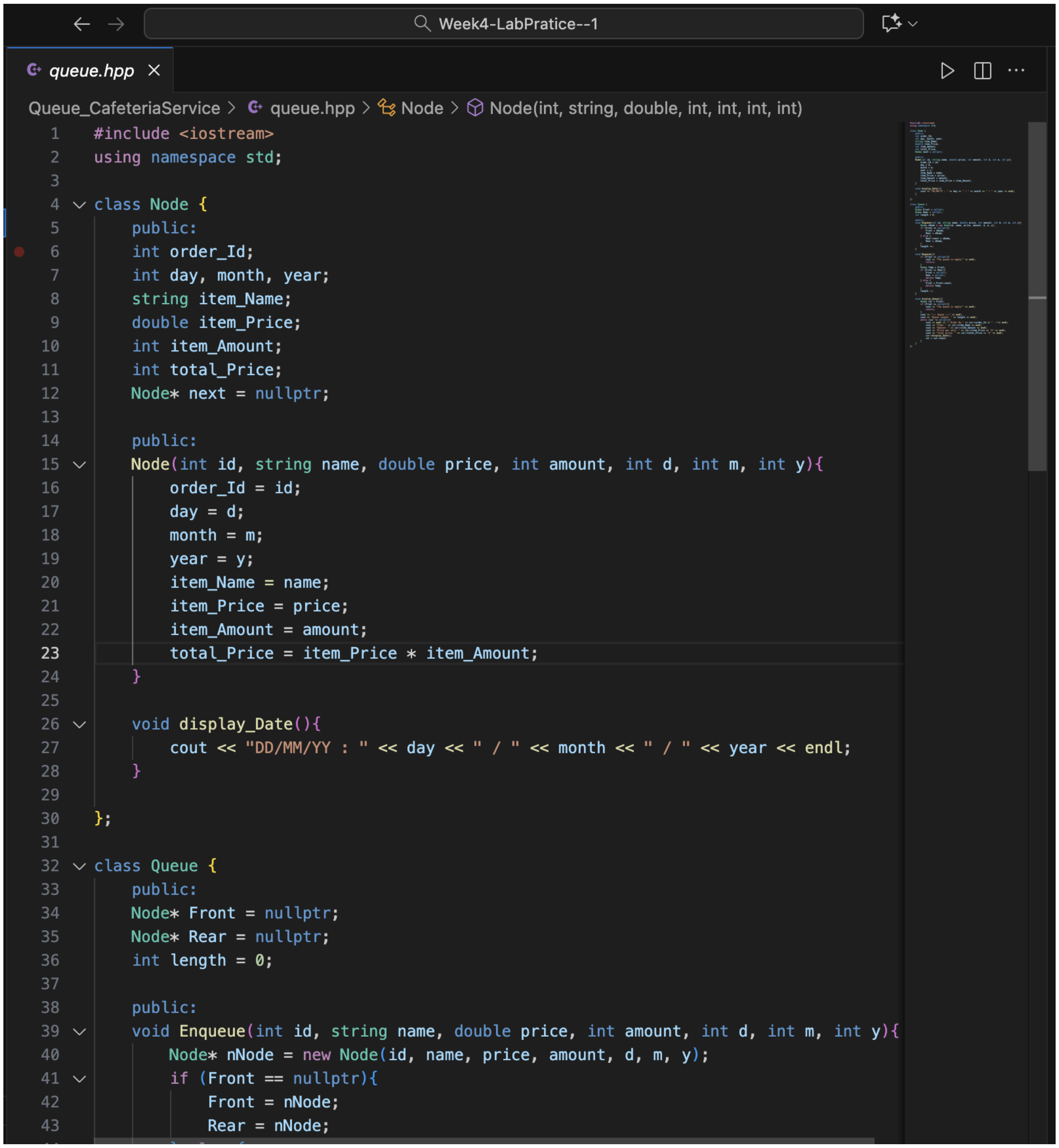


```cpp
            if (cafeteriaQueue.length == 0) {
                cout << "The queue is empty. No order to serve." << endl;
            } else {
                cout << "Serving Order No." << cafeteriaQueue.Front->order_Id << endl;
                cout << "Item: " << cafeteriaQueue.Front->item_Name << endl;
                cout << "Amount: " << cafeteriaQueue.Front->item_Amount << endl;
                cout << "Total Price: " << cafeteriaQueue.Front->total_Price << endl;
                cout << "Order Date: ";
                cafeteriaQueue.Front->display_Date();
                cafeteriaQueue.Dequeue();
            }

        } else if (choice == 3) {
            cafeteriaQueue.display_Queue();

```

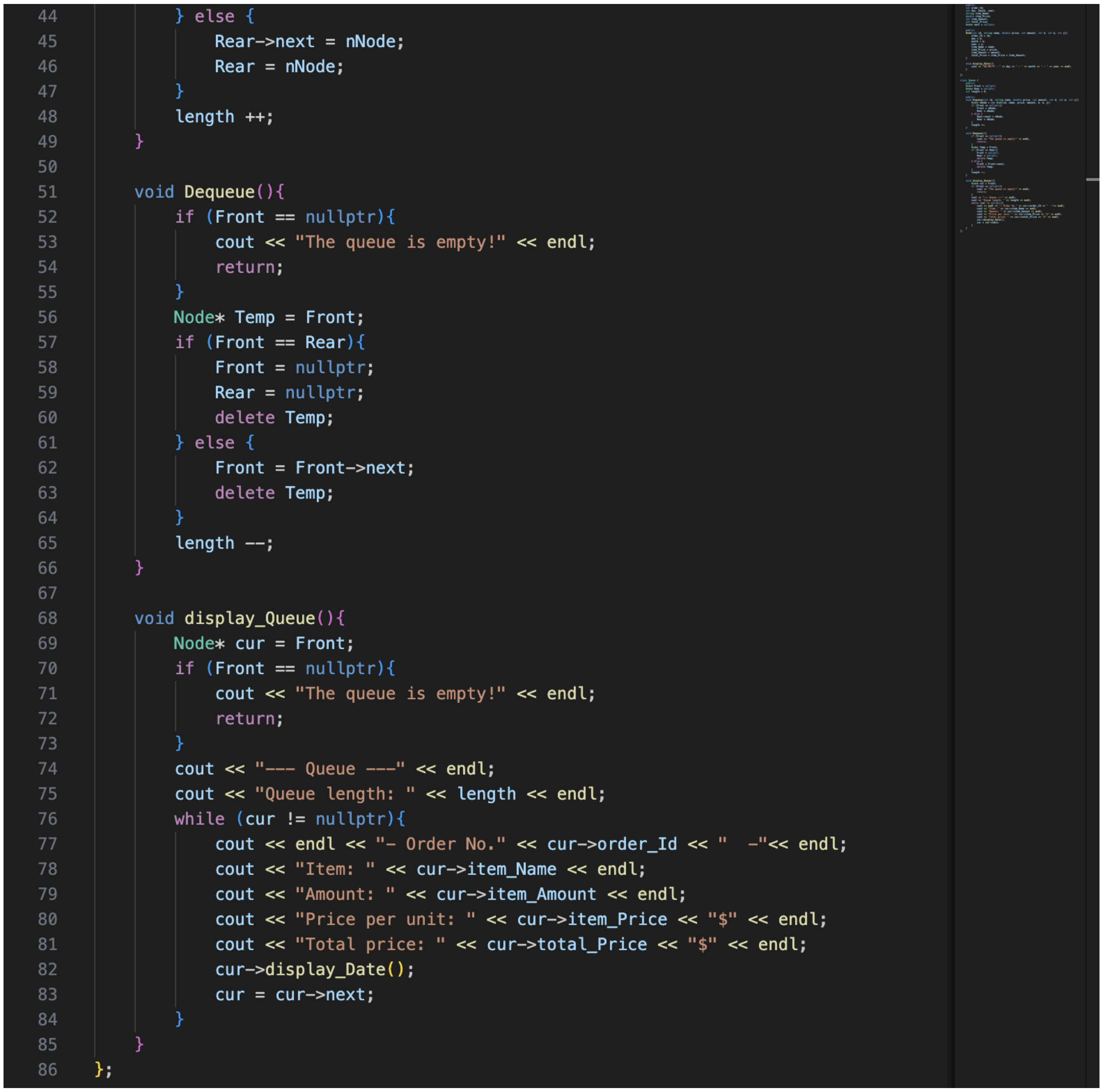


```cpp
            } else if (choice == 4) {
                cout << "Exiting program." << endl;

            } else {
                cout << "Invalid choice! Try again." << endl;
            }
        }

        return 0;
    }

```

queue.hpp


Week4-LabPratice--1
queue.hpp
Queue_CafeteriaService > queue.hpp > Node > Node(int, string, double, int, int, int, int)

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

class Node {
    public:
    int order_Id;
    int day, month, year;
    string item_Name;
    double item_Price;
    int item_Amount;
    int total_Price;
    Node* next = nullptr;

    public:
    Node(int id, string name, double price, int amount, int d, int m, int y){
        order_Id = id;
        day = d;
        month = m;
        year = y;
        item_Name = name;
        item_Price = price;
        item_Amount = amount;
        total_Price = item_Price * item_Amount;
    }

    void display_Date(){
        cout << "DD/MM/YY : " << day << " / " << month << " / " << year << endl;
    }

};

class Queue {
    public:
    Node* Front = nullptr;
    Node* Rear = nullptr;
    int length = 0;

    public:
    void Enqueue(int id, string name, double price, int amount, int d, int m, int y){
        Node* nNode = new Node(id, name, price, amount, d, m, y);
        if (Front == nullptr){
            Front = nNode;
            Rear = nNode;
```


```cpp
        } else {
            Rear->next = nNode;
            Rear = nNode;
        }
        length ++;
    }

    void Dequeue(){
        if (Front == nullptr){
            cout << "The queue is empty!" << endl;
            return;
        }
        Node* Temp = Front;
        if (Front == Rear){
            Front = nullptr;
            Rear = nullptr;
            delete Temp;
        } else {
            Front = Front->next;
            delete Temp;
        }
        length --;
    }

    void display_Queue(){
        Node* cur = Front;
        if (Front == nullptr){
            cout << "The queue is empty!" << endl;
            return;
        }
        cout << "--- Queue ---" << endl;
        cout << "Queue length: " << length << endl;
        while (cur != nullptr){
            cout << endl << "- Order No." << cur->order_Id << "  -"<< endl;
            cout << "Item: " << cur->item_Name << endl;
            cout << "Amount: " << cur->item_Amount << endl;
            cout << "Price per unit: " << cur->item_Price << "$" << endl;
            cout << "Total price: " << cur->total_Price << "$" << endl;
            cur->display_Date();
            cur = cur->next;
        }
    }
};
```