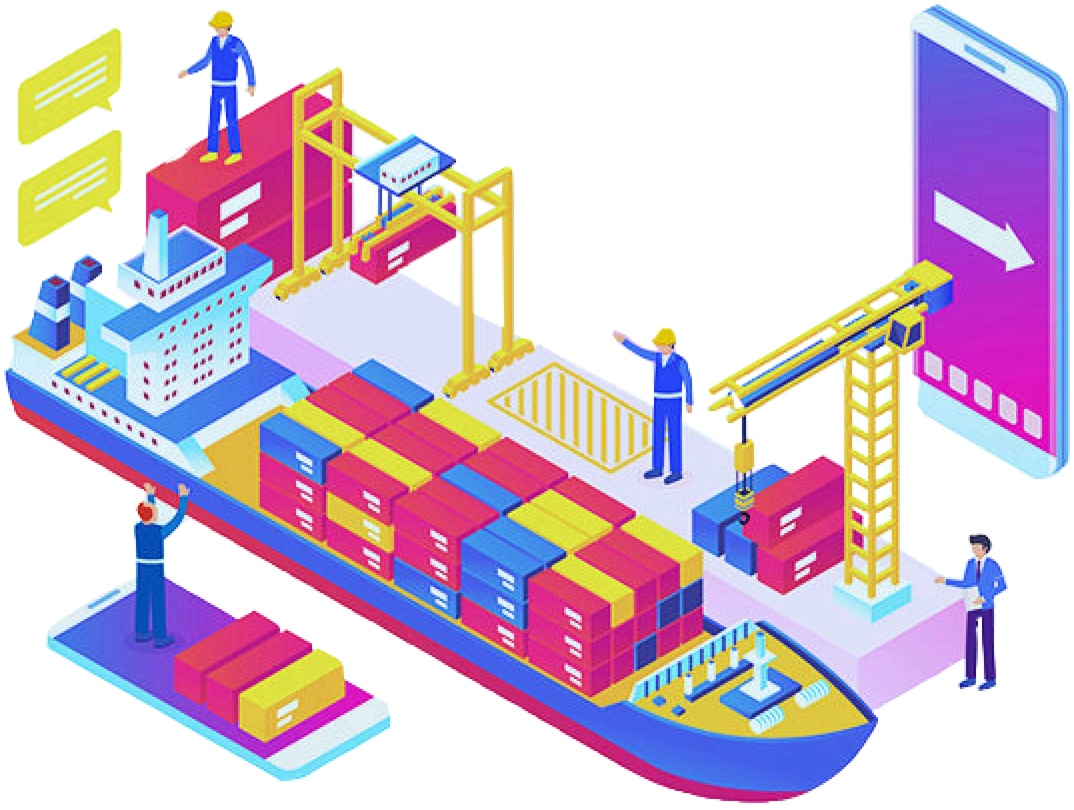
Object Oriented Programming



Assignment Title: Object-Oriented Shop Container Management System

Section: E

Fatima Ishtiaq, 23i-0696

Table of content

1. Introduction

The **Shop Management System** is a C++ project designed to simulate the management of a shop's inventory. The system allows for efficient management of shop containers (items), with functionalities to add, update, delete, and search for containers. The system uses a **linked list** data structure to manage these containers, which provides efficient insertion, deletion, and traversal operations.

This project demonstrates core concepts in data structures and algorithms while offering an interactive way to manage inventory through basic console input/output operations.

2. Project Overview

The **Shop Management System** provides the following key features:

- **Add Containers**: Add new containers to the shop with a name and a number.
- **Update Container Name**: Update the name of a container based on its unique number.
- **Delete Containers**: Remove a container from the shop based on its number.
- **Remove Duplicates**: Automatically remove duplicate containers based on the name and number.
- **Sort Containers**: Sort containers by their associated number.
- **Find Containers**: Search for containers by their number or within a given range.
- **Print Shop Inventory**: Display the current list of containers in the shop.
- **Copy Constructor**: Create a copy of the shop inventory with no references to the original shop.

The program interacts with users through a text-based interface that allows adding, updating, deleting, sorting, and searching containers. It also includes unit tests using the **Google Test framework** to validate the correctness of the operations.

3. Key Components

The main components of the **Shop Management System** include:

Shop Class

The core class of the system, `Shop`, manages the containers using a **linked list**. This class supports the following methods:

- **add_Container**: Adds a new container to the list.
- **update_name_at_containerNumber**: Updates a container’s name by its unique number.
- **delete_Chain**: Deletes a container from the list.

- **Sort_Chain**: Sorts the containers by their associated number.
- **remove_Duplicate**: Removes duplicate containers from the list.
- **findContainer**: Finds a container by its number.
- **findContainer2**: Finds containers within a given number range.
- **print_Shop**: Prints the list of all containers currently in the shop.

Node Structure

Each container is represented by a **node** in the linked list, which holds information about the container's name, number, and pointers to the next and previous nodes.

Unit Tests

The project includes unit tests to verify the correctness of the implemented methods using the **Google Test framework**. The tests cover functionalities such as adding, deleting, updating containers, and removing duplicates.

4. Program Flow

1. **Initialization**: The program starts by creating a new Shop object, initializing an empty linked list.
2. **User Interaction**: The user can choose from a list of operations like adding, deleting, and updating containers.
3. **Container Management**: The program processes the user's inputs and manipulates the containers accordingly, updating the linked list structure.
4. **Display Inventory**: After any operation, the updated inventory of the containers is displayed.
5. **Unit Tests**: Before or after program execution, unit tests can be run to ensure the correctness of the implemented features.

The program runs in a loop, allowing users to continue performing operations until they decide to exit the application.

5. Code Structure

The project is structured as follows:

- **ShopContainerContents.h**: Defines the structure of the `Shop` class and declares all its methods.
- **ShopContainer.cpp**: Implements the methods declared in `Shop.h`, including the logic for managing the containers in the linked list.

- **test.cpp**: Contains unit tests written using the Google Test framework to verify that the Shop class's methods work as expected.

6. Enhancements and Suggestions

While the current version of the **Shop Management System** provides basic functionalities for managing inventory, several enhancements can be made in future versions:

- **Graphical User Interface (GUI)**: A GUI can be developed to provide a more interactive and user-friendly interface.
- **Persistence**: Implement file handling to save and load the inventory data, so users do not lose data when the program exits.
- **Search Optimization**: Use more efficient search algorithms (such as binary search on sorted data) to improve the search functionality.
- **Error Handling**: Add more robust error handling for invalid inputs and edge cases (e.g., when trying to update or delete a non-existing container).
- **Multi-Threading**: Implement multi-threading for handling large inventories to improve performance.

7. Conclusion

The **Shop Management System** project provides a simple yet effective way to manage the inventory of a shop using a **linked list** data structure. The system supports essential operations like adding, deleting, and updating containers, as well as sorting and finding containers by number. The included unit tests ensure the correctness of the implemented features.

While the current implementation is functional, there are several potential improvements that could enhance the system's usability, performance, and scalability. With additional features like a GUI, data persistence, and search optimizations, the Shop Management System can evolve into a more robust and user-friendly tool for managing inventory.