Executive Summary

This report provides a detailed analysis of the Furniture Village Ordering system designed to Furniture Village. This is a menu-driven C++ program system developed to automate and streamline the ordering process of the store. The system addresses the inefficiencies in manual operations by automating key tasks such as order processing, inventory management, and bill generation. This automated solution significantly helps to reduce the employee workloads and enhance customer satisfaction.

This system is developed primarily for customer and administrators, customers are given the features to add items to their order, view the available list of furniture's and search specific furniture items and place orders. While the administrators are provided with additional features of managing inventory by adding new items to the inventory, tracking stock levels by searching them.

The system includes key functionalities such as user authentication for both the customers and admin to login securely, order processing where customers can search items with the category and add them to their order, bill generation where order bills are saved with item details, and inventory management where admin can add new items while the system also updates inventory quantities once orders are placed.

The system's requirements, design, and implementation are then documented in the SRS, that provides detailed analysis of the functionalities and the features available. The system's design such as flowcharts and pseudo codes of key functions are also included to better understand how the system was developed. Finally, the System was subjected to unit testing, user acceptance testing and manual test cases were made for each function to ensure the system functions as expected.

Table of Contents

Table of Figures

Table of Tables

Introduction

The Furniture Village is a lifestyle specialty store, currently carries out its operations manually leading to significant inefficiencies. To address these drawbacks, the Furniture Village Ordering System was developed to automate key tasks such as order processing, inventory management and bill generation reducing workloads of employees and improving customer satisfaction.

This menu-driven C++ program was primarily designed to customers for adding, browsing and placing orders while it's also for administrators, to add new furniture items. The system also includes features of searching furniture items with category and generating order bills with details of items and costs. This was built following structured programming styles, and modularizations. Key data structures such as arrays and control structures to store and manage data efficiently ensuring maintainability, usability and reliability. File handlings were implemented to generate and save order bills.

This report provides a includes a detailed SRS document for System developed, where the system's objectives, features, requirements and design are included. Flow charts and pseudo codes were used to show the logical flow of the system's main functionalities. The system is ensured that it works well by conducting user acceptance testing and making test cases for each function.

Task 01

System Requirement Specification

For Furniture Village Ordering System

1. Introduction

1.1 Purpose

Purpose of this document is to outline the features and functions of the FURNITURE VILLAGE Furniture Ordering System. This system is a C++ program built to automate key functions related to the ordering process of Furniture Village aiming to improve the customer satisfaction and reduce employee-workloads. This system achieves these goals by allowing customers to add furniture items to their order, place orders, view the list of available furniture's and search them. It reduces the employee workloads by calculating the totals of orders and updating the inventory after order's are placed automatedly with minimal human-intervention. Additionally, it provides admin the capability to add new furniture items to the inventory and track inventory stocks.

1.2 Intended Audience

This system is mainly intended for use of customers and store owner/employees (Admin) in order to automate the manual processes and reduce workloads. The system allows;

- **Customers** to browse the furniture's, and place orders.
- **Administrators** to view the available furniture's, search specific furniture items and add new furniture items to the inventory.
- **Employees** to assist customers with placing and processing orders.
- **Developers** to maintain and improve the system.
- **Tester** to test whether each function works as expected and solve any bugs or issues with the system.

1.3 Scope

This menu-driven system is designed to automate the key functions related to the ordering process of furniture's. This includes the features of login with valid credentials, adding items to order, browse furniture items available with prices and quantities, and place orders generating order bills. This system will improve the efficiency of the store operations by automating the tasks like order processing, bill generation enhancing the employee productivity and customer satisfaction. The system also allows the admin to add new furniture items to the inventory making the inventory management convenient. While the system has certain limitations such as not having a data base for inventory and users which could be an issue for future growth. It fulfills the core functions that are required to carry out the ordering process efficiently.

1.4 Definitions, Acronyms, and Abbreviations

CLI:	Command Line Interface
ID:	Identifier
Admin:	Administrator
FOS:	Furniture ordering system
IDE:	Integrated development environment
I/O:	Input and output

2. Overall Description

2.1 Product Perspective

The Furniture Village Ordering system is a standalone C++ console application designed to replace the manual ordering processes with an automation solution. This system integrates with the store operations and addresses the inefficiencies and errors by streamlining the key tasks like order processing, billing, furniture browsing and inventory control. This automated solution will reduce the workloads of employees improving their productivity while it will also enhance the customer satisfaction.

2.2 Product Functions

The system includes several key functions to ease the ordering process. These include:

User Authentication – Users must log in with valid username and password to access the system.

Add Furniture – Customer can browse the available furniture items in the inventory and add them to their order by inputting the furniture ID and the desired quantity.

Add New Furniture – Admin can add new furniture items to the inventory by entering valid item ID, Name, Category, Price and Quantity

List Furniture – Users can view the list of all the available furniture items including their ID, Name, Category, Price and Quantity.

Search Furniture – Users can search specific furniture items with their category such as Bedroom, Dining, Office, and Sofa.

Place Order – Customers can place orders, this will calculate the total amount then update the inventory and generate an order bill with an order id and the items purchased.

Help – Users can use the “Help” option to get guidance on how to use the system without any issue.

Exit – Users will be allowed to exit the system carefully.

2.3 User Class

User Class	Description and Characteristics
Customer	Customer will be the primary user. They are given the privilege to select any item they need and add those to their order and purchase them. They can search items, and view available items.
Admin	The administrator is responsible to track and manage the items available. The system provides the admin the capability to view the available furniture’s, search specific items and special authority to add new items to the inventory.

Table 1 User class

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Login Authentication

- System authenticates users with the predefined username and password.
- It checks the validity of entered credentials ("shanuka"/"shanuka123" for regular users, "admin"/"admin123" for administrators).

3.1.2 Display Menu

- System displays the Main Menu with options to select such as Add Furniture, List Furnitures, Search Furniture, Place order, Help and Exit.
- System processes the user choice and navigates to the corresponding function.

3.1.3 Add Furniture

- System allows customer to add furniture item to their order from the inventory.
- System prompts the user to enter item ID, and quantity then verify them to add the item or displays an error message.

3.1.4 List of Available Furnitures

- System displays the list of all the available furniture's.
- System iterates through the inventory array and displays ID, Name, Category, Price, Quantity of each item.

3.1.5 Search Furniture

- System allows to search for furniture's by category.
- System iterates through the inventory array and displays the items matching the entered category.

3.1.6 Place Order

- System processes the current order, and generates a order bill after updating the inventory.
- System calculates the total and updates the inventory, then generates the order bill with the (order ID, item details, total amount) and saves as a text file.

3.1.7 Help

- System displays the Help section with instructions on how to use the system.

3.1.8 Exit

- System terminates the program allowing the user to safely exit the program.

3.2 Non-Functional Requirements

3.2.1 Performance

- System should respond to user inputs quickly within 2-3 seconds for the key functions.
- System generates the orders bills and load main catalogs under 2-3 seconds.

3.2.2 Usability

- System should be user-friendly, CLI should be easy to navigate without confusions.
- System provides helpful prompts to guide the users to perform the tasks accurately through its menu-driven interface.

3.2.3 Security

- System should securely store sensitive data such as passwords and usernames.
- FOS System uses hardcoded credentials which could pose a security threat.
- System denies unauthorized accesses validating the usernames and password.

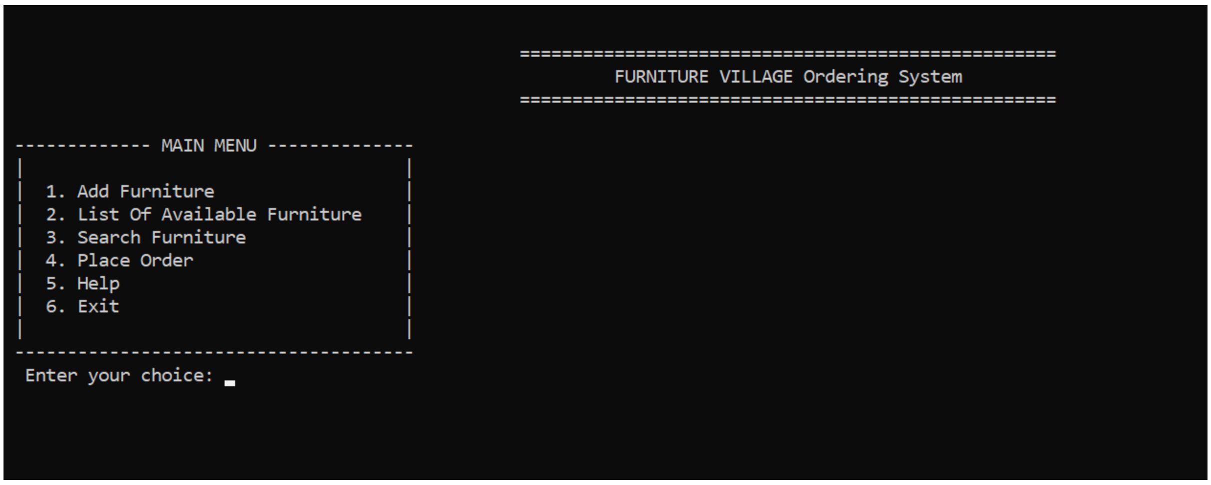
3.2.4 Reliability

- System should not get stuck in infinite loops and crash when wrong inputs are entered.
- System handles the invalid inputs of the user correctly proving them with error messages to guide them without crashing the program.

4. External Interface Requirements

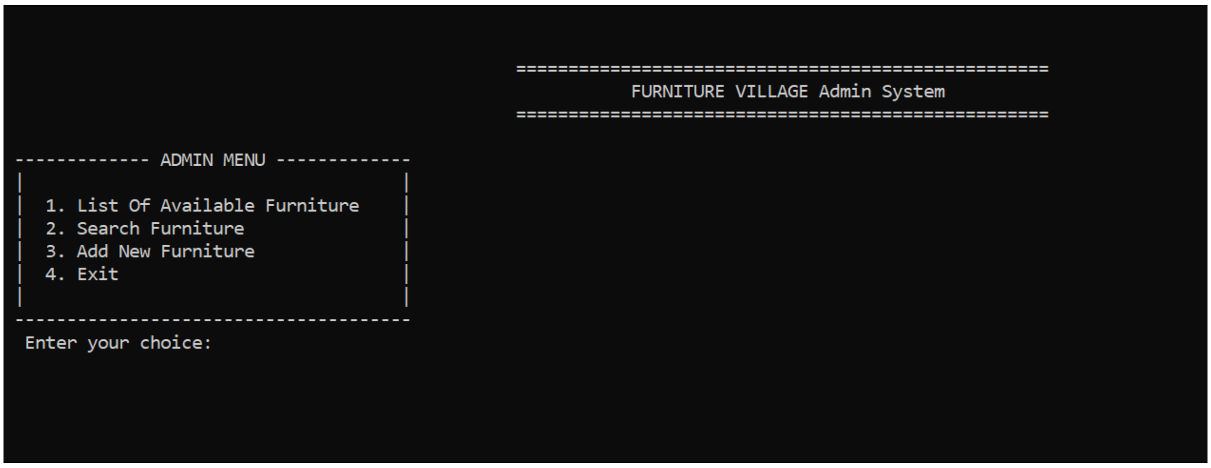
4.1 User Interfaces

This C++ program uses CLI where all the interactions are text-based through the console.



```
                                              ===================================================
                                                        FURNITURE VILLAGE Ordering System
                                              ===================================================

------------ MAIN MENU -------------
|                                    |
|  1. Add Furniture                  |
|  2. List Of Available Furniture    |
|  3. Search Furniture               |
|  4. Place Order                    |
|  5. Help                           |
|  6. Exit                           |
|                                    |
--------------------------------------
 Enter your choice: _
```

Figure 1 Customer user interface



```
                                              ===================================================
                                                          FURNITURE VILLAGE Admin System
                                              ===================================================

------------ ADMIN MENU ------------
|                                    |
|  1. List Of Available Furniture    |
|  2. Search Furniture               |
|  3. Add New Furniture              |
|  4. Exit                           |
|                                    |
--------------------------------------
 Enter your choice:
```

Figure 2 Admin user interface

4.2 Software Interfaces

Various software tools and libraries were used to develop the System to ensure its efficiency, and functionality.

- Dev C++ software was used to code, compile and test the system which is an IDE for C++
- Standard C++ libraries like iostream for I/O operations, fstream for file handling, string for string manipulations, iomanip for formatting outputs, cctype for character handling and limits for numeric limits and input validation were used.

4.3 Hardware interfaces

Minimum Requirements

	Specifications
Processor	Intel Core i5 or equivalent
RAM	8GB
OS	Windows 10 or later

Table 2 Hardware requirements

5 System Design

5.1 Flow Charts & Pseudo codes

5.1.1 Add Furniture

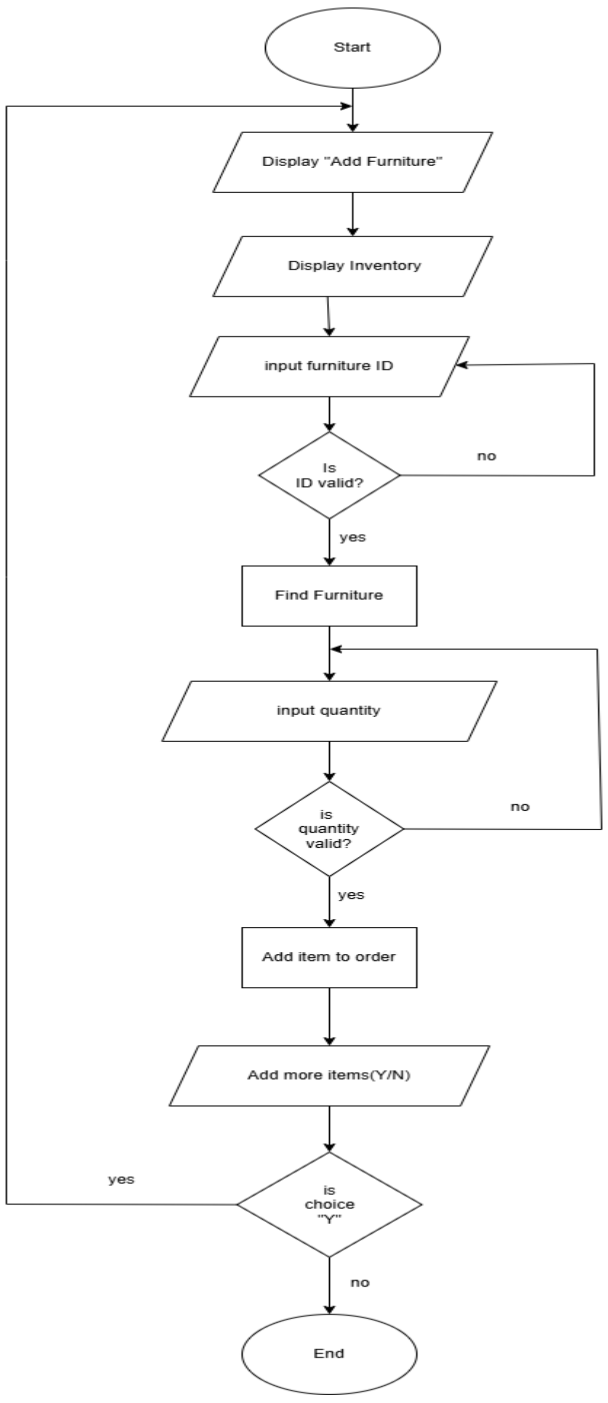


Figure 3 Add furniture Flow chart

```
BEGIN
  DO
    DISPLAY "ADD FURNITURE"
    DISPLAY inventory list

    REPEAT
      GET furnitureId from user input
      IF furnitureId is invalid
        DISPLAY error message
      END IF
    UNTIL (validId = true)

    FIND selectedFurniture based on furnitureId
    IF selectedFurniture is not found
      DISPLAY error message and CONTINUE
    END IF

    REPEAT
      GET quantity from user input
      IF quantity is invalid or exceeds available stock
        DISPLAY error message
      END IF
    UNTIL (validQuantity = true)

    CREATE new OrderItem with selectedFurniture and quantity
    ADD new item to current order

    DISPLAY "Item added to your order"

    REPEAT
      GET add more items (Y/N) from user input
      IF input is invalid
        DISPLAY error message
      END IF
    UNTIL (validInput = true)
  WHILE (addMore == 'Y')
END
```

Figure 4 Add Furniture Pseudo code

5.1.2 List Available Furniture

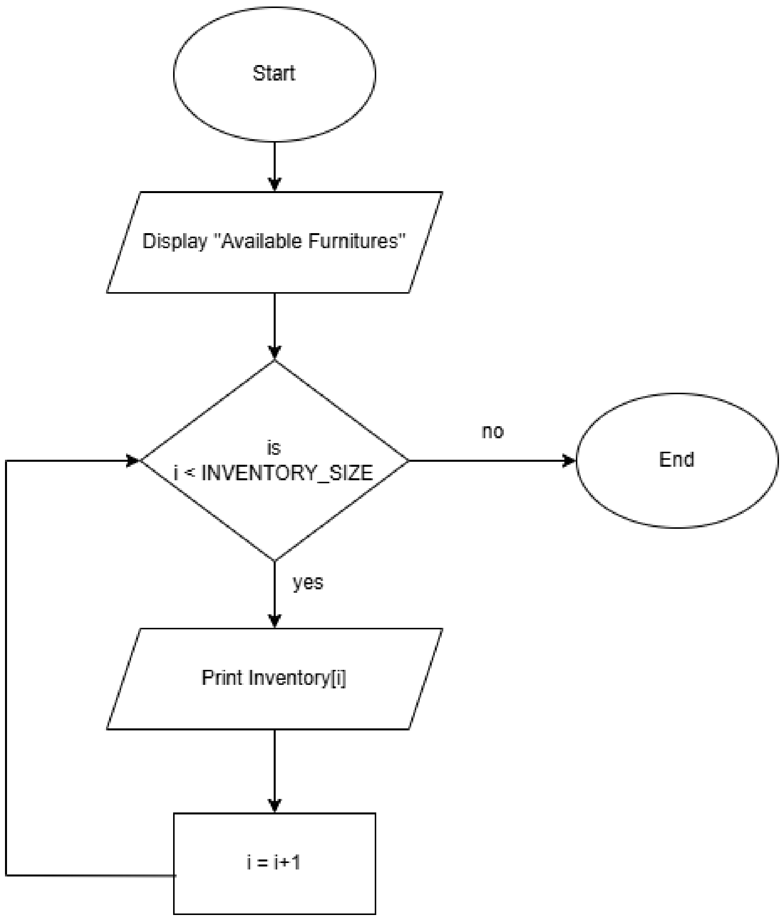
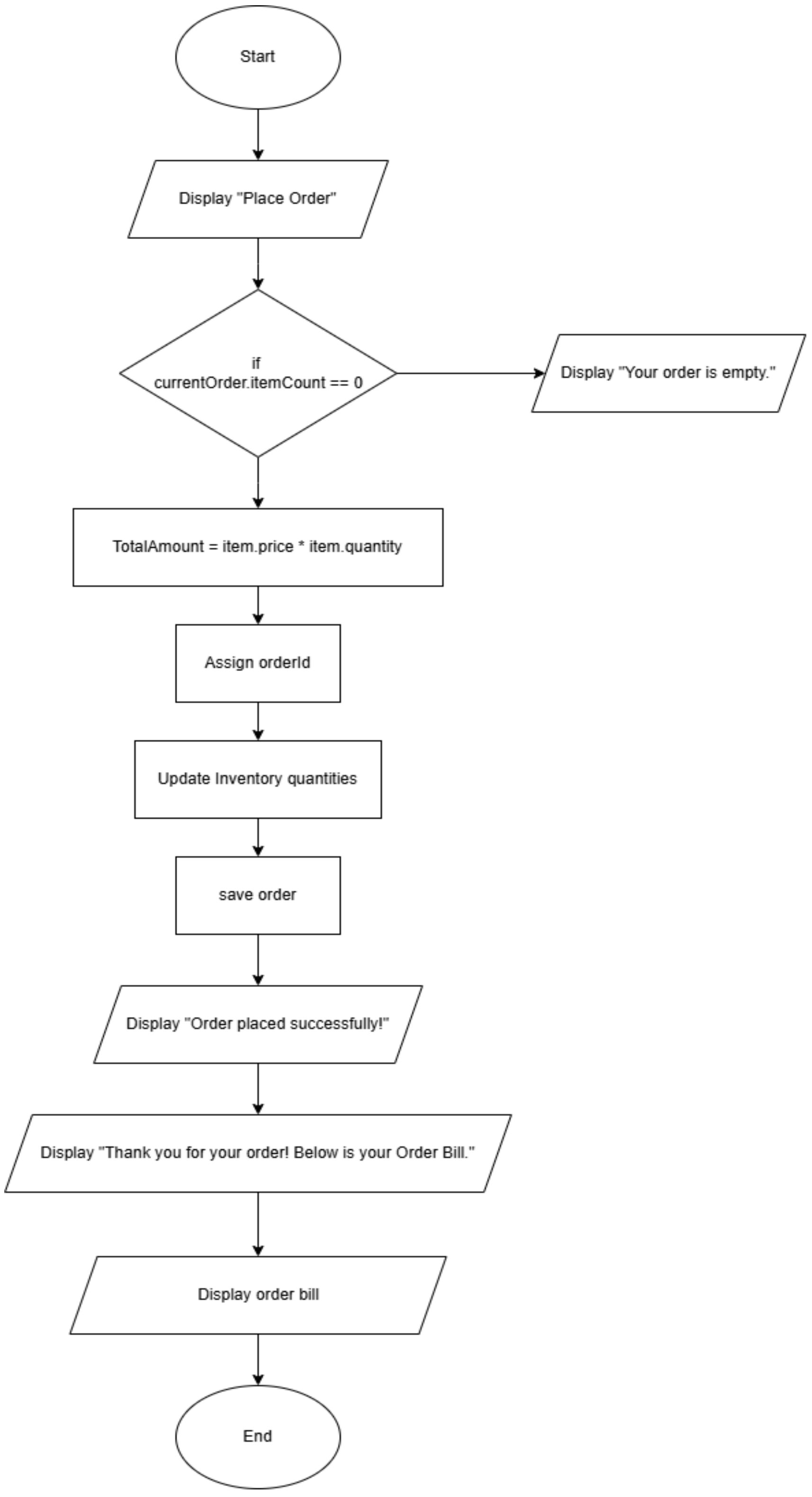


Figure 5 List Available Furniture Flow Chart

```
BEGIN

  DISPLAY "AVAILABLE FURNITURES"
  DISPLAY Column Headers: ID, Name, Category, Price, Quantity

  FOR i = 0 TO INVENTORY_SIZE - 1 DO
    PRINT inventory[i].id, inventory[i].name, inventory[i].category, inventory[i].price, inventory[i].quantity
  END FOR

END
```

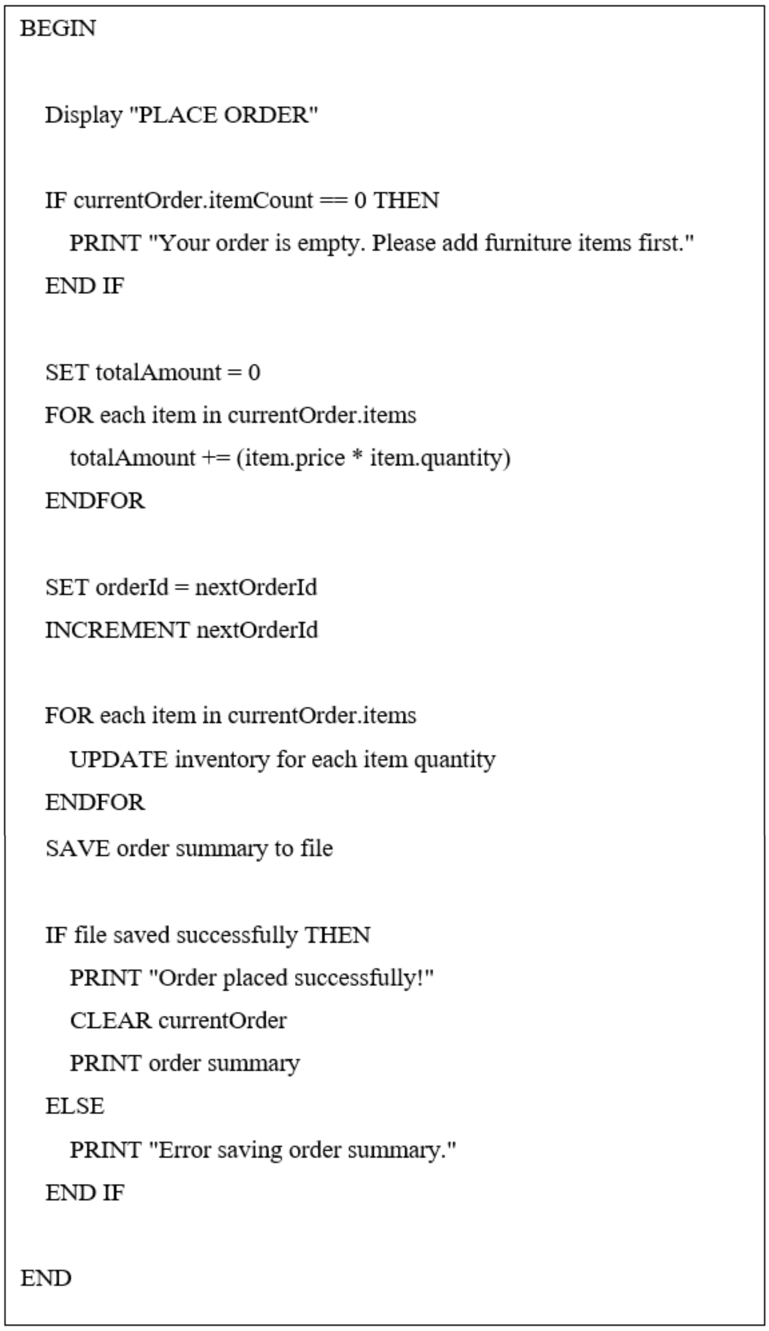
Figure 6 List Available Furniture Pseudo code

5.1.3 Place Order



Start
Display "Place Order"
if currentOrder.itemCount == 0
Display "Your order is empty."
TotalAmount = item.price * item.quantity
Assign orderId
Update Inventory quantities
save order
Display "Order placed successfully!"
Display "Thank you for your order! Below is your Order Bill."
Display order bill
End

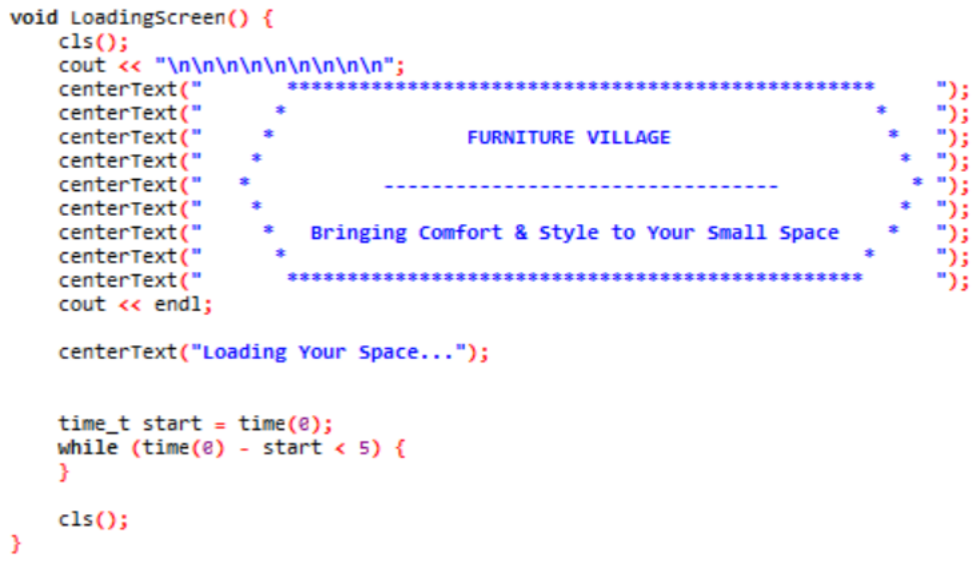
Figure 7 Place Order Flow Chart



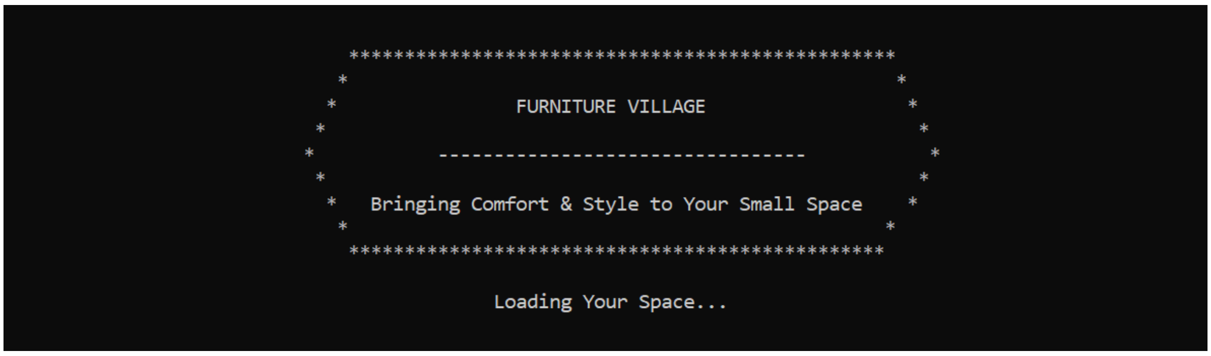
```
BEGIN

  Display "PLACE ORDER"

  IF currentOrder.itemCount == 0 THEN
    PRINT "Your order is empty. Please add furniture items first."
  END IF

  SET totalAmount = 0
  FOR each item in currentOrder.items
    totalAmount += (item.price * item.quantity)
  ENDFOR

  SET orderId = nextOrderId
  INCREMENT nextOrderId

  FOR each item in currentOrder.items
    UPDATE inventory for each item quantity
  ENDFOR
  SAVE order summary to file

  IF file saved successfully THEN
    PRINT "Order placed successfully!"
    CLEAR currentOrder
    PRINT order summary
  ELSE
    PRINT "Error saving order summary."
  END IF

END
```

Figure 8 Place Order Pseudo code

- **Loading Screen**



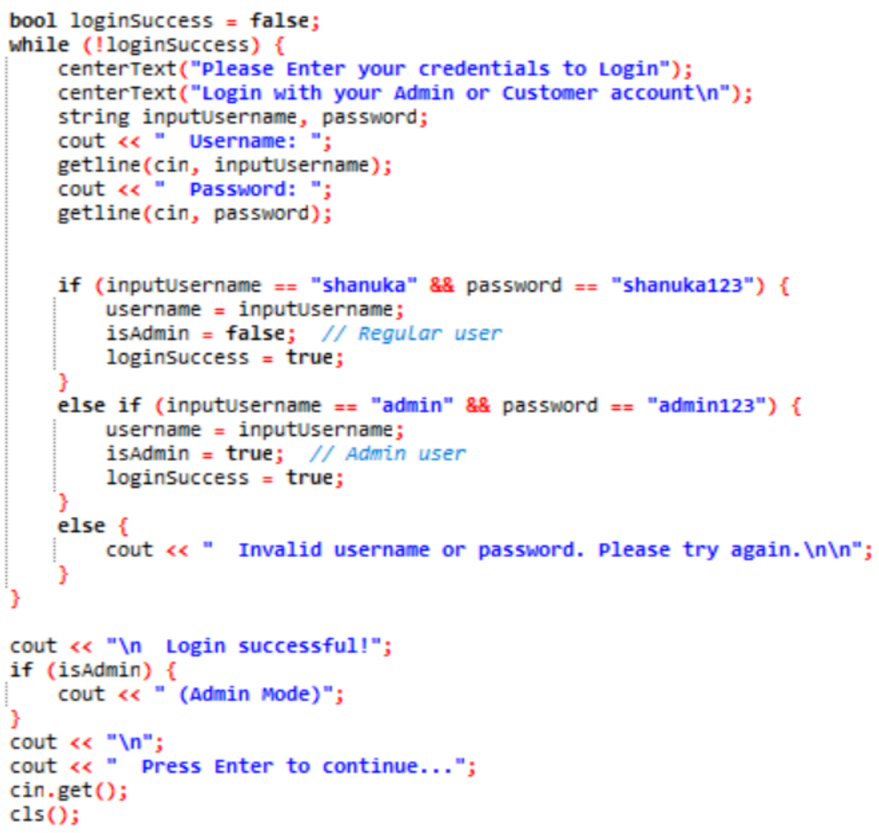
```
void LoadingScreen() {
    cls();
    cout << "\n\n\n\n\n\n\n\n\n";
    centerText("        **************************************************     ");
    centerText("       *                                                  *    ");
    centerText("      *                 FURNITURE VILLAGE                  *   ");
    centerText("     *                                                      *  ");
    centerText("    *         ---------------------------------              * ");
    centerText("     *                                                      *  ");
    centerText("      *   Bringing Comfort & Style to Your Small Space     *   ");
    centerText("       *                                                  *    ");
    centerText("        *************************************************      ");
    cout << endl;

    centerText("Loading Your Space...");


    time_t start = time(0);
    while (time(0) - start < 5) {
    }

    cls();
}
```



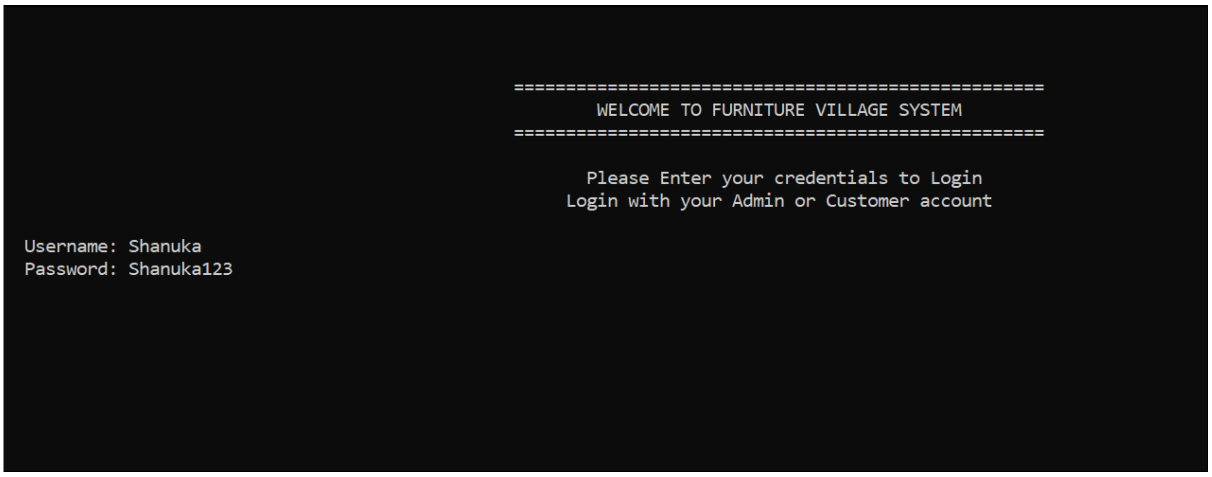
```
        **************************************************
       *                                                  *
      *                 FURNITURE VILLAGE                  *
     *                                                      *
    *         ---------------------------------              *
     *                                                      *
      *   Bringing Comfort & Style to Your Small Space     *
       *                                                  *
        *************************************************

                    Loading Your Space...
```

Figure 9 Loading Screen

- **Login Page**



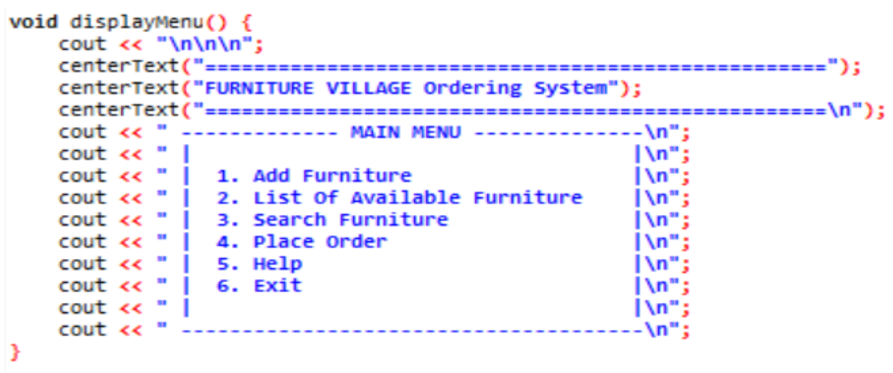
```
bool loginSuccess = false;
while (!loginSuccess) {
    centerText("Please Enter your credentials to Login");
    centerText("Login with your Admin or Customer account\n");
    string inputUsername, password;
    cout << "  Username: ";
    getline(cin, inputUsername);
    cout << "  Password: ";
    getline(cin, password);


    if (inputUsername == "shanuka" && password == "shanuka123") {
        username = inputUsername;
        isAdmin = false;  // Regular user
        loginSuccess = true;
    }
    else if (inputUsername == "admin" && password == "admin123") {
        username = inputUsername;
        isAdmin = true;  // Admin user
        loginSuccess = true;
    }
    else {
        cout << "  Invalid username or password. Please try again.\n\n";
    }
}

cout << "\n  Login successful!";
if (isAdmin) {
    cout << " (Admin Mode)";
}
cout << "\n";
cout << "  Press Enter to continue...";
cin.get();
cls();
```



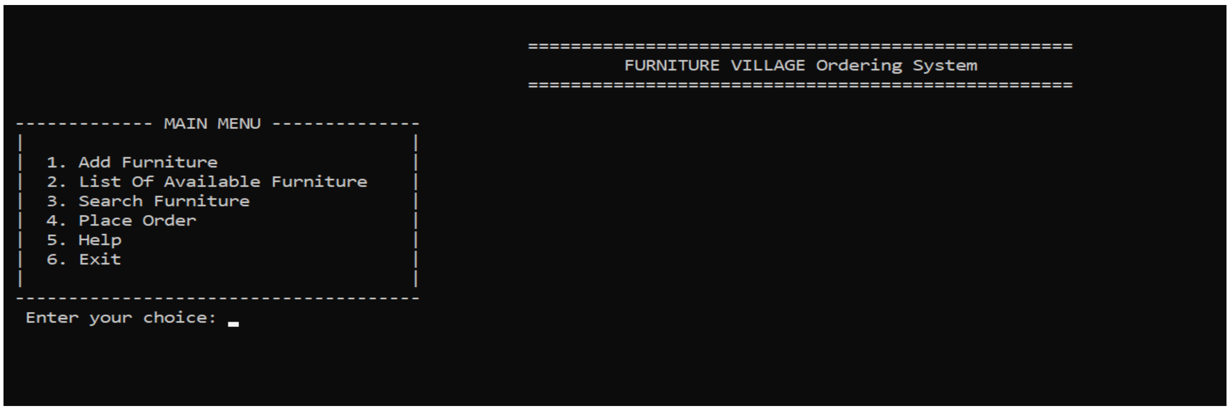
```
                                                 ==================================================
                                                          WELCOME TO FURNITURE VILLAGE SYSTEM
                                                 ==================================================

                                                          Please Enter your credentials to Login
                                                      Login with your Admin or Customer account

Username: Shanuka
Password: Shanuka123
```

Figure 10 Login Page

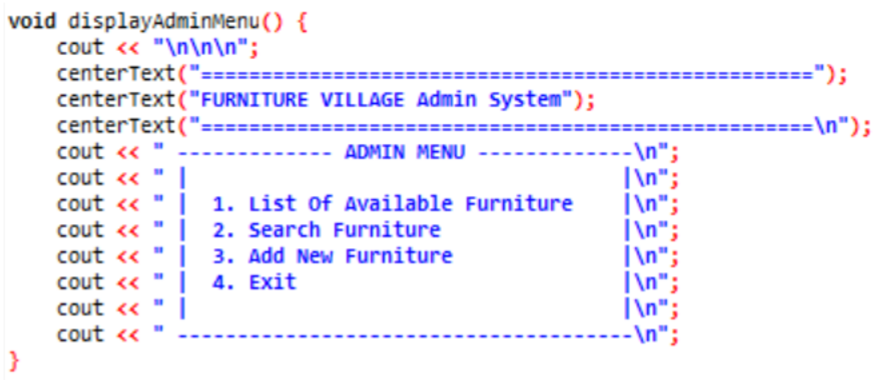
- **Main Menu & Admin Menu**



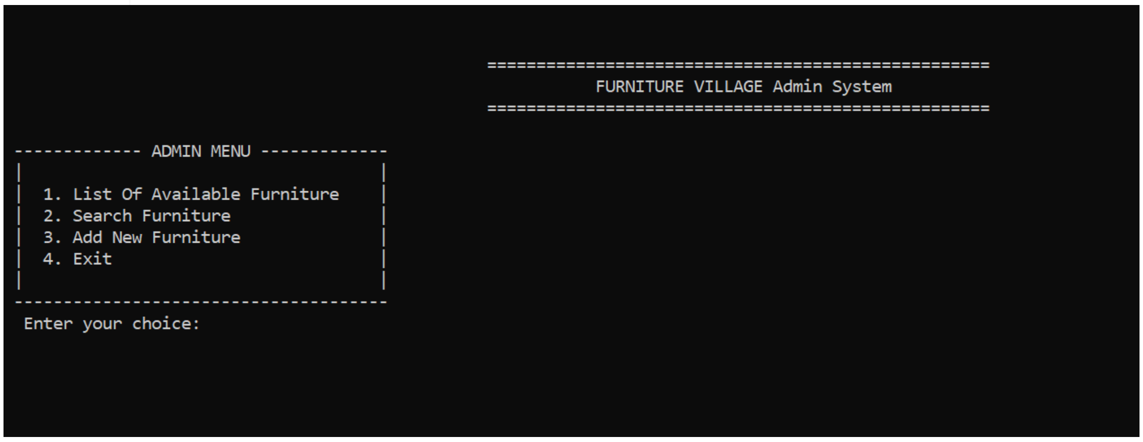
```
void displayMenu() {
    cout << "\n\n\n";
    centerText("====================================================");
    centerText("FURNITURE VILLAGE Ordering System");
    centerText("====================================================\n");
    cout << " ------------- MAIN MENU --------------\n";
    cout << " |                                     |\n";
    cout << " |   1. Add Furniture                  |\n";
    cout << " |   2. List Of Available Furniture    |\n";
    cout << " |   3. Search Furniture               |\n";
    cout << " |   4. Place Order                    |\n";
    cout << " |   5. Help                           |\n";
    cout << " |   6. Exit                           |\n";
    cout << " |                                     |\n";
    cout << " --------------------------------------\n";
}
```



```
                                                ====================================================
                                                         FURNITURE VILLAGE Ordering System
                                                ====================================================

 ------------- MAIN MENU --------------
 |                                     |
 |   1. Add Furniture                  |
 |   2. List Of Available Furniture    |
 |   3. Search Furniture               |
 |   4. Place Order                    |
 |   5. Help                           |
 |   6. Exit                           |
 |                                     |
 --------------------------------------
  Enter your choice: _
```



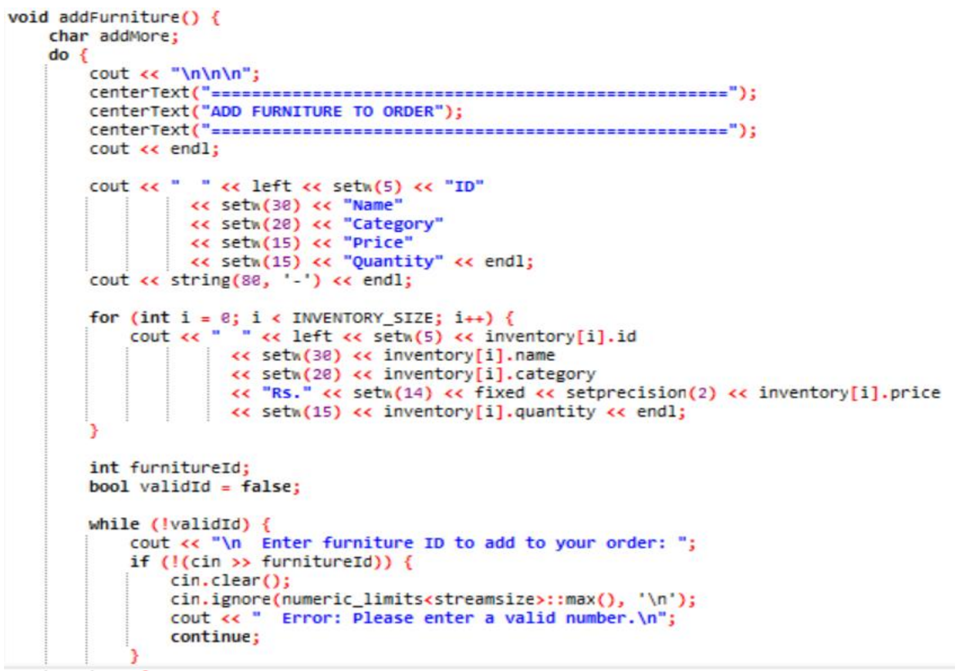
```
void displayAdminMenu() {
    cout << "\n\n\n";
    centerText("====================================================");
    centerText("FURNITURE VILLAGE Admin System");
    centerText("====================================================\n");
    cout << " ------------- ADMIN MENU -------------\n";
    cout << " |                                     |\n";
    cout << " |   1. List Of Available Furniture    |\n";
    cout << " |   2. Search Furniture               |\n";
    cout << " |   3. Add New Furniture              |\n";
    cout << " |   4. Exit                           |\n";
    cout << " |                                     |\n";
    cout << " --------------------------------------\n";
}
```



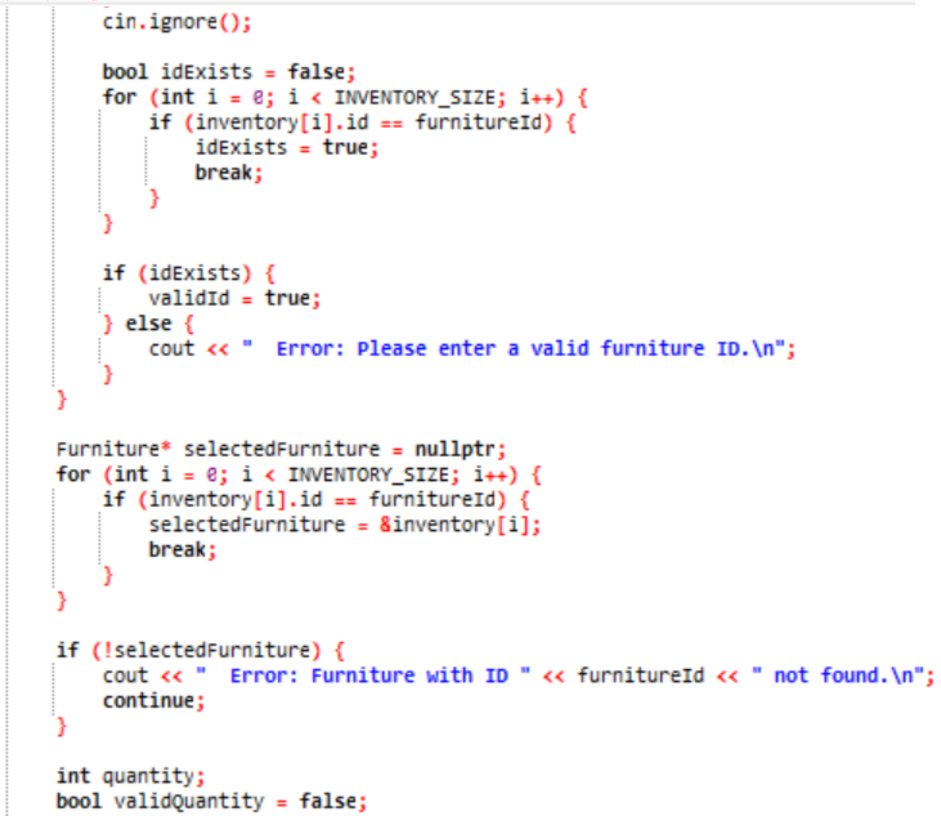
```
                                               ====================================================
                                                          FURNITURE VILLAGE Admin System
                                               ====================================================

 ------------- ADMIN MENU -------------
 |                                     |
 |   1. List Of Available Furniture    |
 |   2. Search Furniture               |
 |   3. Add New Furniture              |
 |   4. Exit                           |
 |                                     |
 --------------------------------------
  Enter your choice:
```

Figure 11 Main Menu & Admin Menu

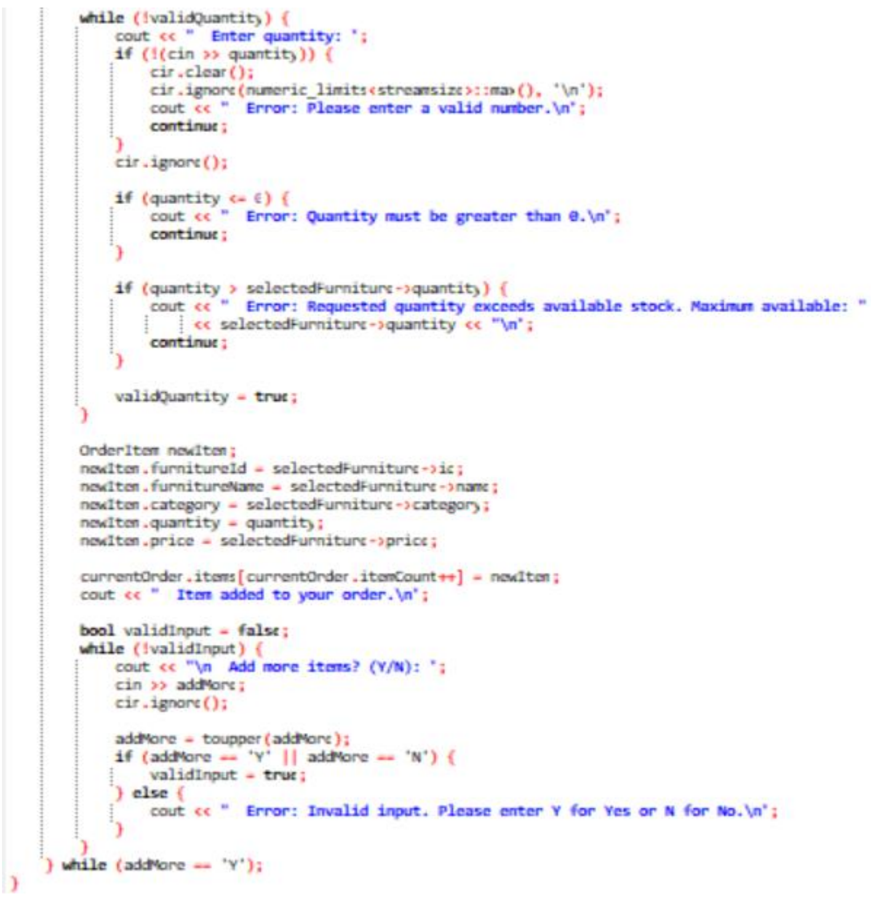
- **Add Furniture**



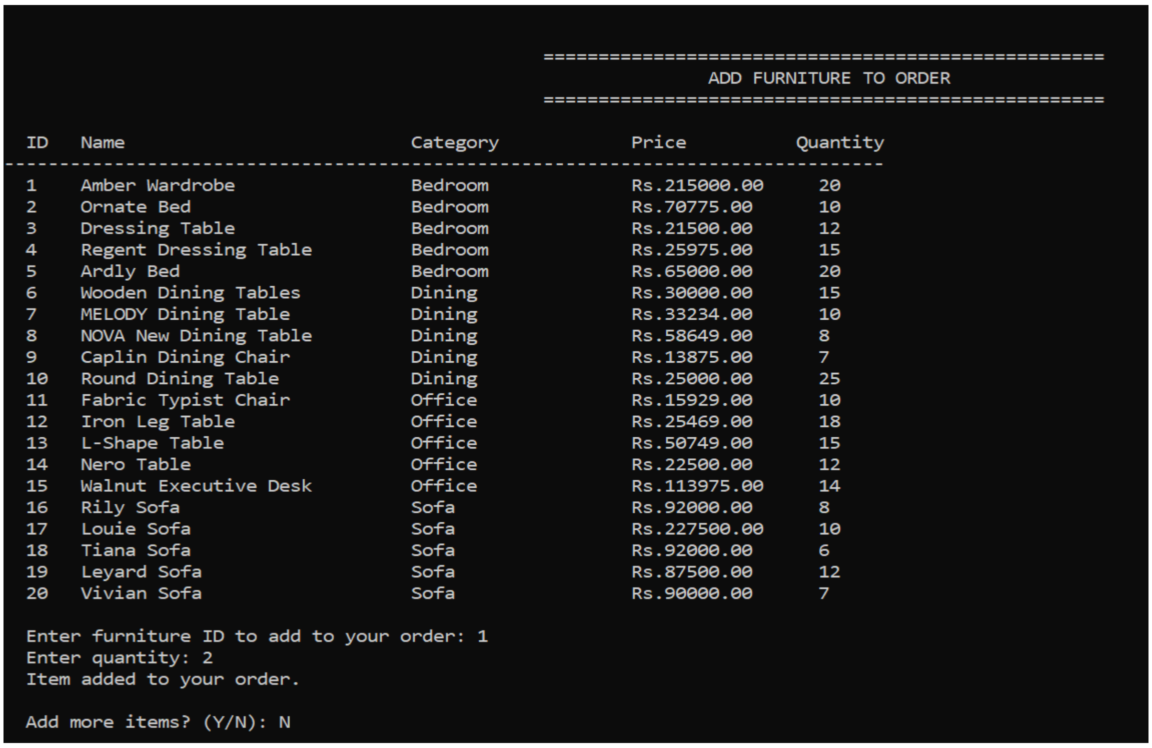
```
void addFurniture() {
    char addMore;
    do {
        cout << "\n\n\n";
        centerText("===================================================");
        centerText("ADD FURNITURE TO ORDER");
        centerText("===================================================");
        cout << endl;

        cout << "  " << left << setw(5) << "ID"
                  << setw(30) << "Name"
                  << setw(20) << "Category"
                  << setw(15) << "Price"
                  << setw(15) << "Quantity" << endl;
        cout << string(80, '-') << endl;

        for (int i = 0; i < INVENTORY_SIZE; i++) {
            cout << "  " << left << setw(5) << inventory[i].id
                      << setw(30) << inventory[i].name
                      << setw(20) << inventory[i].category
                      << "Rs." << setw(14) << fixed << setprecision(2) << inventory[i].price
                      << setw(15) << inventory[i].quantity << endl;
        }

        int furnitureId;
        bool validId = false;

        while (!validId) {
            cout << "\n  Enter furniture ID to add to your order: ";
            if (!(cin >> furnitureId)) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << "  Error: Please enter a valid number.\n";
                continue;
            }
```



```
            cin.ignore();

            bool idExists = false;
            for (int i = 0; i < INVENTORY_SIZE; i++) {
                if (inventory[i].id == furnitureId) {
                    idExists = true;
                    break;
                }
            }

            if (idExists) {
                validId = true;
            } else {
                cout << "  Error: Please enter a valid furniture ID.\n";
            }
        }

        Furniture* selectedFurniture = nullptr;
        for (int i = 0; i < INVENTORY_SIZE; i++) {
            if (inventory[i].id == furnitureId) {
                selectedFurniture = &inventory[i];
                break;
            }
        }

        if (!selectedFurniture) {
            cout << "  Error: Furniture with ID " << furnitureId << " not found.\n";
            continue;
        }

        int quantity;
        bool validQuantity = false;
```



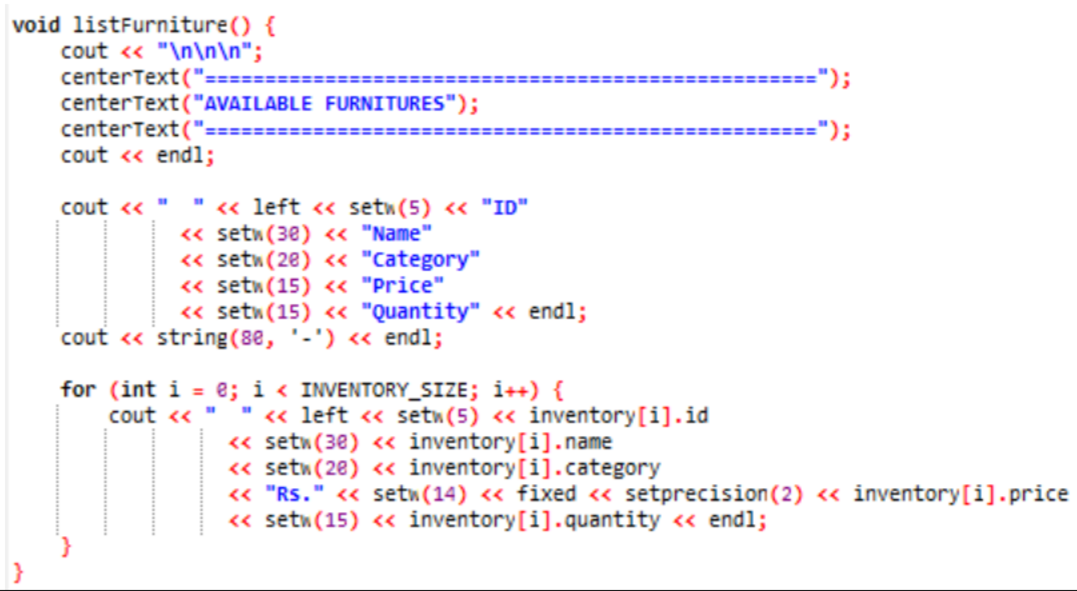
```
        while (!validQuantity) {
            cout << "  Enter quantity: ";
            if (!(cin >> quantity)) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << "  Error: Please enter a valid number.\n";
                continue;
            }
            cin.ignore();

            if (quantity <= 0) {
                cout << "  Error: Quantity must be greater than 0.\n";
                continue;
            }

            if (quantity > selectedFurniture->quantity) {
                cout << "  Error: Requested quantity exceeds available stock. Maximum available: "
                     << selectedFurniture->quantity << "\n";
                continue;
            }

            validQuantity = true;
        }

        OrderItem newItem;
        newItem.furnitureId = selectedFurniture->id;
        newItem.furnitureName = selectedFurniture->name;
        newItem.category = selectedFurniture->category;
        newItem.quantity = quantity;
        newItem.price = selectedFurniture->price;

        currentOrder.items[currentOrder.itemCount++] = newItem;
        cout << "  Item added to your order.\n";

        bool validInput = false;
        while (!validInput) {
            cout << "\n  Add more items? (Y/N): ";
            cin >> addMore;
            cin.ignore();

            addMore = toupper(addMore);
            if (addMore == 'Y' || addMore == 'N') {
                validInput = true;
            } else {
                cout << "  Error: Invalid input. Please enter Y for Yes or N for No.\n";
            }
        }
    } while (addMore == 'Y');
}
```



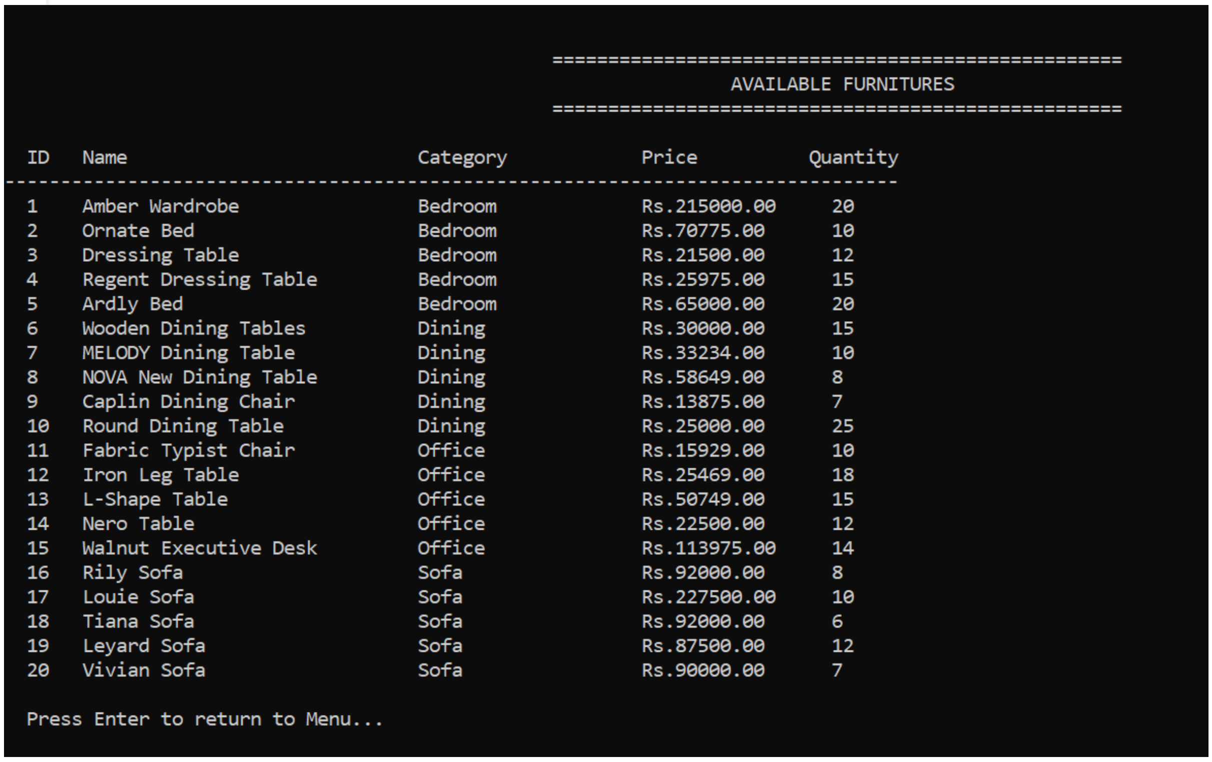
```
                                             ====================================================
                                                              ADD FURNITURE TO ORDER
                                             ====================================================

  ID   Name                          Category            Price          Quantity
------------------------------------------------------------------------------------------
  1    Amber Wardrobe                Bedroom             Rs.215000.00     20
  2    Ornate Bed                    Bedroom             Rs.70775.00      10
  3    Dressing Table                Bedroom             Rs.21500.00      12
  4    Regent Dressing Table         Bedroom             Rs.25975.00      15
  5    Ardly Bed                     Bedroom             Rs.65000.00      20
  6    Wooden Dining Tables          Dining              Rs.30000.00      15
  7    MELODY Dining Table           Dining              Rs.33234.00      10
  8    NOVA New Dining Table         Dining              Rs.58649.00      8
  9    Caplin Dining Chair           Dining              Rs.13875.00      7
  10   Round Dining Table            Dining              Rs.25000.00      25
  11   Fabric Typist Chair           Office              Rs.15929.00      10
  12   Iron Leg Table                Office              Rs.25469.00      18
  13   L-Shape Table                 Office              Rs.50749.00      15
  14   Nero Table                    Office              Rs.22500.00      12
  15   Walnut Executive Desk         Office              Rs.113975.00     14
  16   Rily Sofa                     Sofa                Rs.92000.00      8
  17   Louie Sofa                    Sofa                Rs.227500.00     10
  18   Tiana Sofa                    Sofa                Rs.92000.00      6
  19   Leyard Sofa                   Sofa                Rs.87500.00      12
  20   Vivian Sofa                   Sofa                Rs.90000.00      7

  Enter furniture ID to add to your order: 1
  Enter quantity: 2
  Item added to your order.

  Add more items? (Y/N): N
```

Figure 12 Add Furniture

- **List of Available Furnitures**



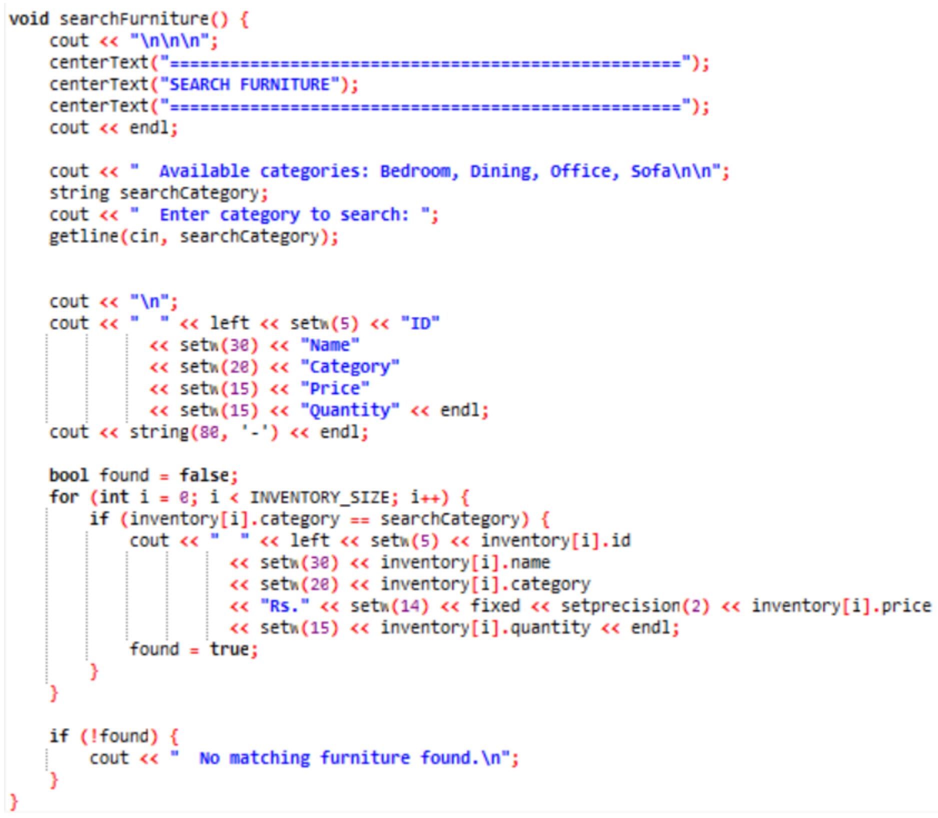
```cpp
void listFurniture() {
    cout << "\n\n\n";
    centerText("===================================================");
    centerText("AVAILABLE FURNITURES");
    centerText("===================================================");
    cout << endl;

    cout << "  " << left << setw(5) << "ID"
              << setw(30) << "Name"
              << setw(20) << "Category"
              << setw(15) << "Price"
              << setw(15) << "Quantity" << endl;
    cout << string(80, '-') << endl;

    for (int i = 0; i < INVENTORY_SIZE; i++) {
        cout << "  " << left << setw(5) << inventory[i].id
                  << setw(30) << inventory[i].name
                  << setw(20) << inventory[i].category
                  << "Rs." << setw(14) << fixed << setprecision(2) << inventory[i].price
                  << setw(15) << inventory[i].quantity << endl;
    }
}
```



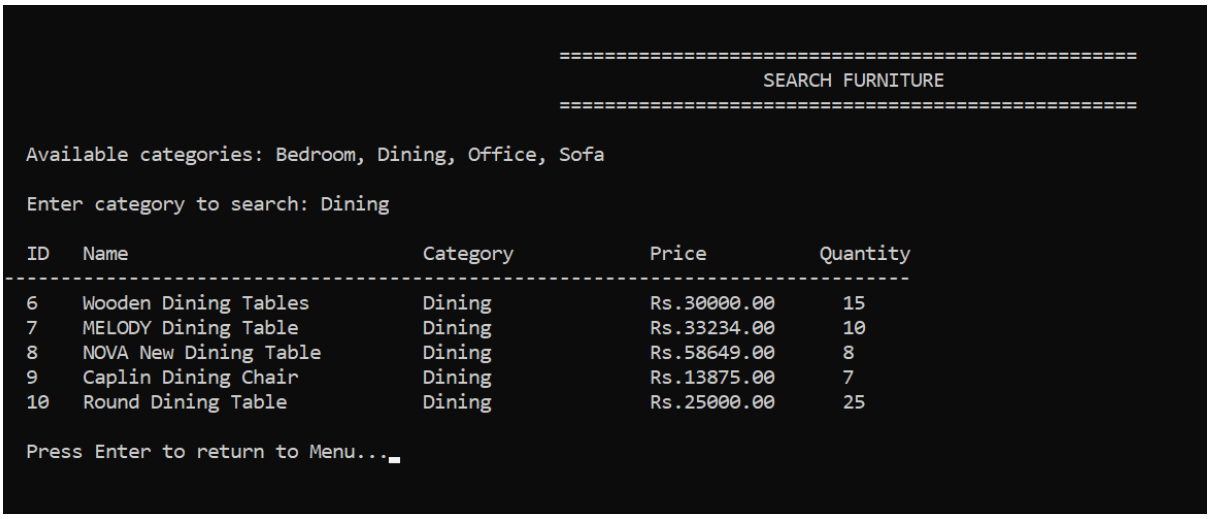
```
                                                 ===================================================
                                                                AVAILABLE FURNITURES
                                                 ===================================================

  ID   Name                           Category            Price          Quantity
--------------------------------------------------------------------------------
  1    Amber Wardrobe                 Bedroom             Rs.215000.00     20
  2    Ornate Bed                     Bedroom             Rs.70775.00      10
  3    Dressing Table                 Bedroom             Rs.21500.00      12
  4    Regent Dressing Table          Bedroom             Rs.25975.00      15
  5    Ardly Bed                      Bedroom             Rs.65000.00      20
  6    Wooden Dining Tables           Dining              Rs.30000.00      15
  7    MELODY Dining Table            Dining              Rs.33234.00      10
  8    NOVA New Dining Table          Dining              Rs.58649.00      8
  9    Caplin Dining Chair            Dining              Rs.13875.00      7
  10   Round Dining Table             Dining              Rs.25000.00      25
  11   Fabric Typist Chair            Office              Rs.15929.00      10
  12   Iron Leg Table                 Office              Rs.25469.00      18
  13   L-Shape Table                  Office              Rs.50749.00      15
  14   Nero Table                     Office              Rs.22500.00      12
  15   Walnut Executive Desk          Office              Rs.113975.00     14
  16   Rily Sofa                      Sofa                Rs.92000.00      8
  17   Louie Sofa                     Sofa                Rs.227500.00     10
  18   Tiana Sofa                     Sofa                Rs.92000.00      6
  19   Leyard Sofa                    Sofa                Rs.87500.00      12
  20   Vivian Sofa                    Sofa                Rs.90000.00      7

  Press Enter to return to Menu...
```

Figure 13 List of Available Furnitures

- **Search Furniture**



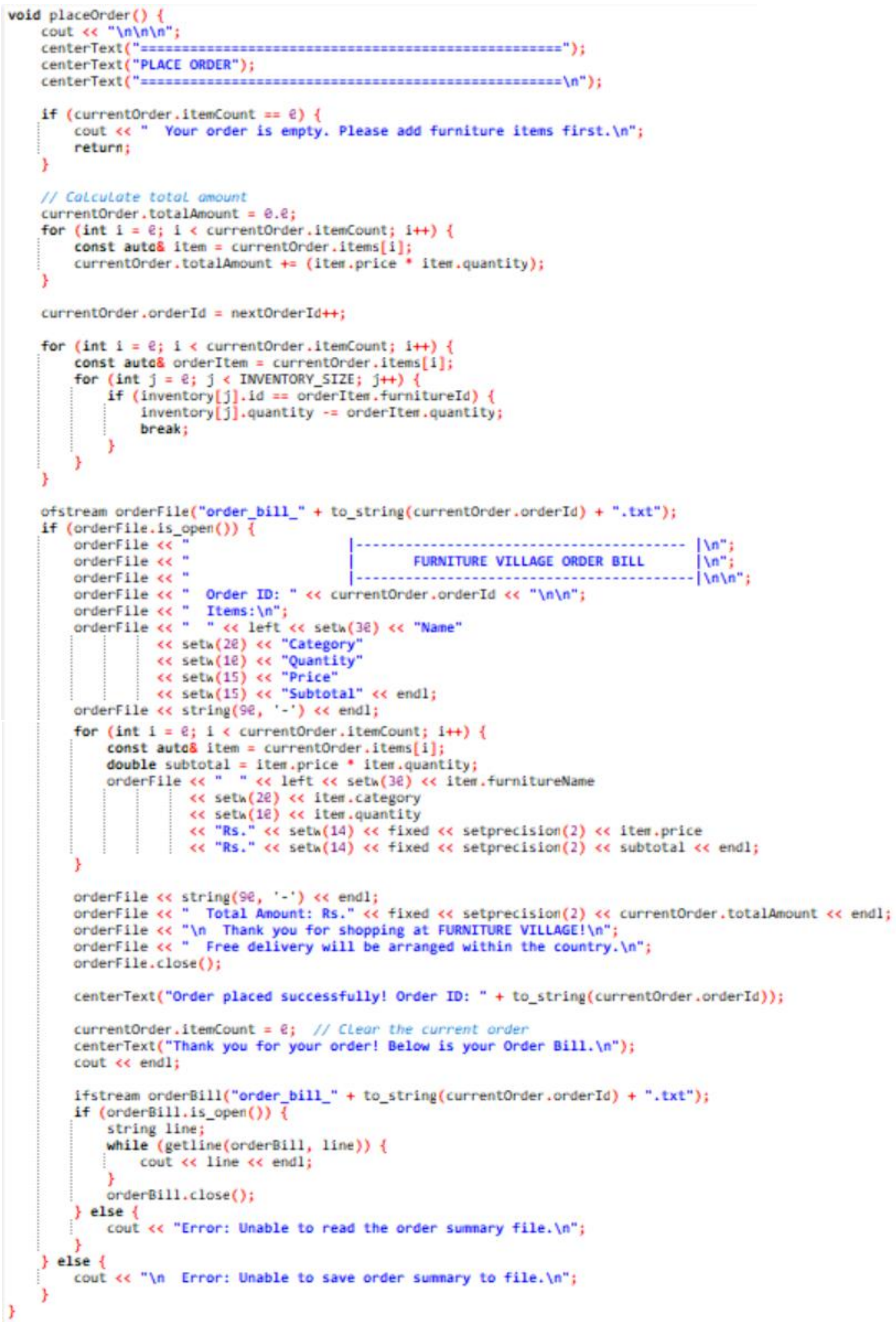
```
void searchFurniture() {
    cout << "\n\n\n";
    centerText("==================================================");
    centerText("SEARCH FURNITURE");
    centerText("==================================================");
    cout << endl;

    cout << "  Available categories: Bedroom, Dining, Office, Sofa\n\n";
    string searchCategory;
    cout << "  Enter category to search: ";
    getline(cin, searchCategory);


    cout << "\n";
    cout << "  " << left << setw(5) << "ID"
          << setw(30) << "Name"
          << setw(20) << "Category"
          << setw(15) << "Price"
          << setw(15) << "Quantity" << endl;
    cout << string(80, '-') << endl;

    bool found = false;
    for (int i = 0; i < INVENTORY_SIZE; i++) {
        if (inventory[i].category == searchCategory) {
            cout << "  " << left << setw(5) << inventory[i].id
                  << setw(30) << inventory[i].name
                  << setw(20) << inventory[i].category
                  << "Rs." << setw(14) << fixed << setprecision(2) << inventory[i].price
                  << setw(15) << inventory[i].quantity << endl;
            found = true;
        }
    }

    if (!found) {
        cout << "  No matching furniture found.\n";
    }
}
```



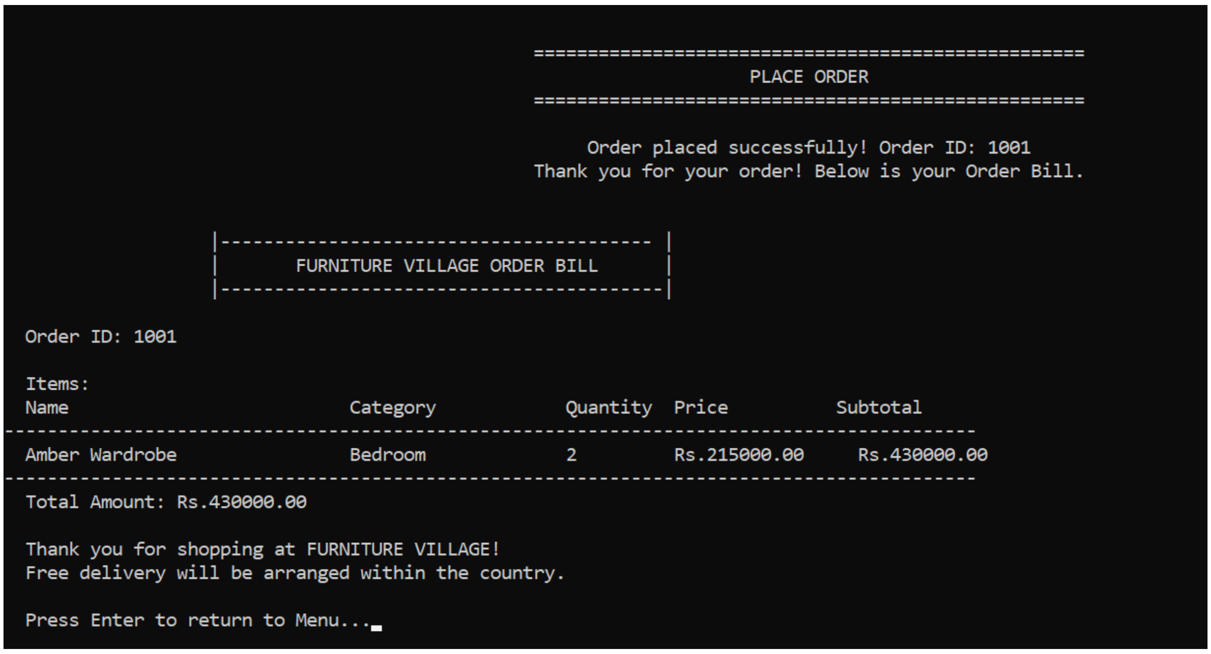
```
                                        ==================================================
                                                        SEARCH FURNITURE
                                        ==================================================

  Available categories: Bedroom, Dining, Office, Sofa

  Enter category to search: Dining

  ID   Name                          Category            Price          Quantity
--------------------------------------------------------------------------------
  6    Wooden Dining Tables          Dining              Rs.30000.00      15
  7    MELODY Dining Table           Dining              Rs.33234.00      10
  8    NOVA New Dining Table         Dining              Rs.58649.00      8
  9    Caplin Dining Chair           Dining              Rs.13875.00      7
  10   Round Dining Table            Dining              Rs.25000.00      25

  Press Enter to return to Menu...
```

Figure 14 Search Furniture

- **Place Order**



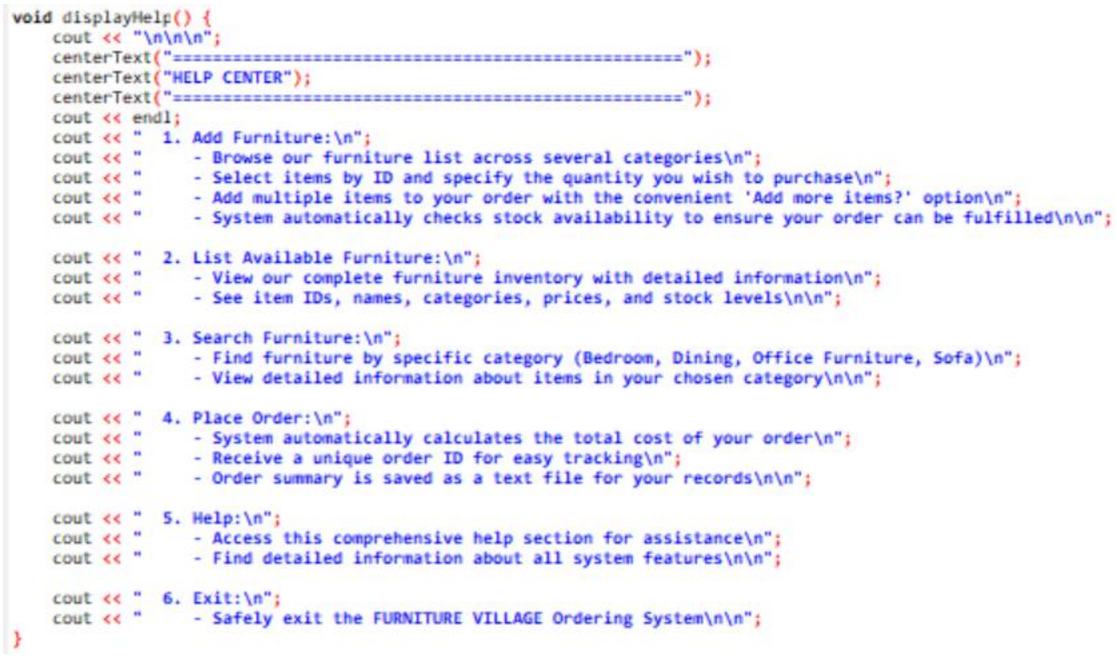
```cpp
void placeOrder() {
    cout << "\n\n\n";
    centerText("==================================================");
    centerText("PLACE ORDER");
    centerText("==================================================\n");

    if (currentOrder.itemCount == 0) {
        cout << "  Your order is empty. Please add furniture items first.\n";
        return;
    }

    // Calculate total amount
    currentOrder.totalAmount = 0.0;
    for (int i = 0; i < currentOrder.itemCount; i++) {
        const auto& item = currentOrder.items[i];
        currentOrder.totalAmount += (item.price * item.quantity);
    }

    currentOrder.orderId = nextOrderId++;

    for (int i = 0; i < currentOrder.itemCount; i++) {
        const auto& orderItem = currentOrder.items[i];
        for (int j = 0; j < INVENTORY_SIZE; j++) {
            if (inventory[j].id == orderItem.furnitureId) {
                inventory[j].quantity -= orderItem.quantity;
                break;
            }
        }
    }

    ofstream orderFile("order_bill_" + to_string(currentOrder.orderId) + ".txt");
    if (orderFile.is_open()) {
        orderFile << "                         |---------------------------------------- |\n";
        orderFile << "                         |        FURNITURE VILLAGE ORDER BILL      |\n";
        orderFile << "                         |-----------------------------------------|\n\n";
        orderFile << "  Order ID: " << currentOrder.orderId << "\n\n";
        orderFile << "  Items:\n";
        orderFile << "  " << left << setw(30) << "Name"
                  << setw(20) << "Category"
                  << setw(10) << "Quantity"
                  << setw(15) << "Price"
                  << setw(15) << "Subtotal" << endl;
        orderFile << string(90, '-') << endl;
        for (int i = 0; i < currentOrder.itemCount; i++) {
            const auto& item = currentOrder.items[i];
            double subtotal = item.price * item.quantity;
            orderFile << "  " << left << setw(30) << item.furnitureName
                      << setw(20) << item.category
                      << setw(10) << item.quantity
                      << "Rs." << setw(14) << fixed << setprecision(2) << item.price
                      << "Rs." << setw(14) << fixed << setprecision(2) << subtotal << endl;
        }

        orderFile << string(90, '-') << endl;
        orderFile << "  Total Amount: Rs." << fixed << setprecision(2) << currentOrder.totalAmount << endl;
        orderFile << "\n  Thank you for shopping at FURNITURE VILLAGE!\n";
        orderFile << "  Free delivery will be arranged within the country.\n";
        orderFile.close();

        centerText("Order placed successfully! Order ID: " + to_string(currentOrder.orderId));

        currentOrder.itemCount = 0;  // Clear the current order
        centerText("Thank you for your order! Below is your Order Bill.\n");
        cout << endl;

        ifstream orderBill("order_bill_" + to_string(currentOrder.orderId) + ".txt");
        if (orderBill.is_open()) {
            string line;
            while (getline(orderBill, line)) {
                cout << line << endl;
            }
            orderBill.close();
        } else {
            cout << "Error: Unable to read the order summary file.\n";
        }
    } else {
        cout << "\n  Error: Unable to save order summary to file.\n";
    }
}
```



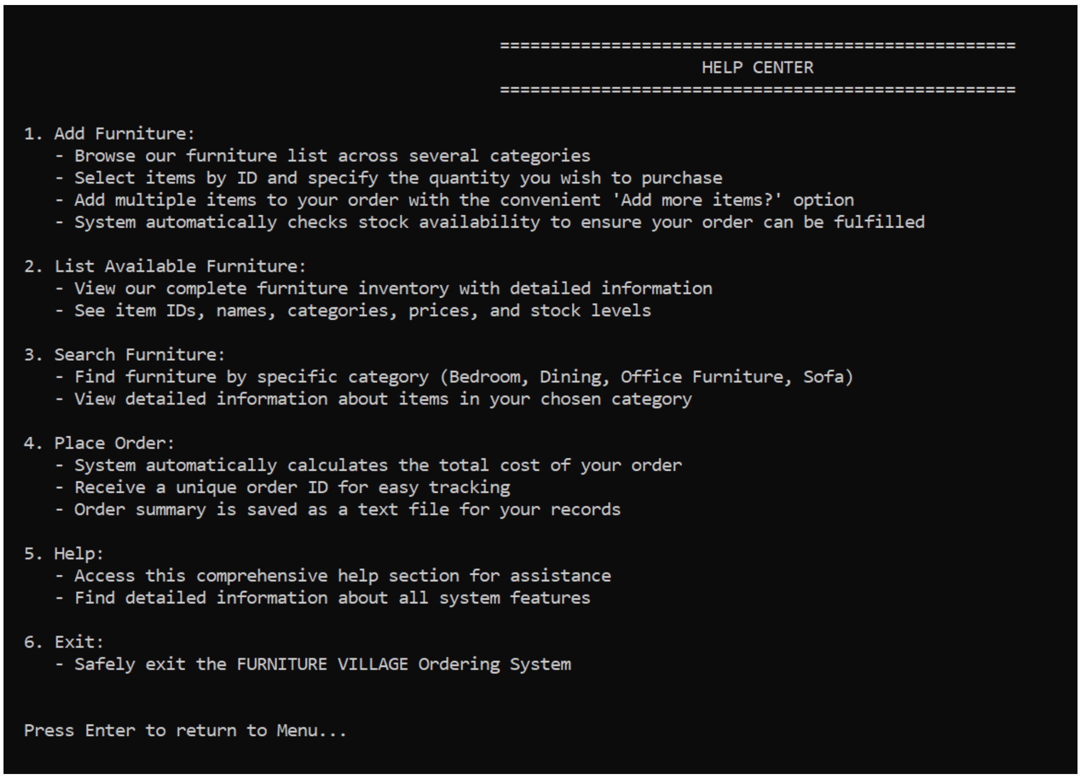
```
                                                   ===================================================
                                                                       PLACE ORDER
                                                   ===================================================

                                                        Order placed successfully! Order ID: 1001
                                                   Thank you for your order! Below is your Order Bill.


                      |----------------------------------------|
                      |       FURNITURE VILLAGE ORDER BILL       |
                      |----------------------------------------|

  Order ID: 1001

  Items:
  Name                          Category            Quantity  Price           Subtotal
-------------------------------------------------------------------------------------------
  Amber Wardrobe                Bedroom             2         Rs.215000.00      Rs.430000.00
-------------------------------------------------------------------------------------------
  Total Amount: Rs.430000.00

  Thank you for shopping at FURNITURE VILLAGE!
  Free delivery will be arranged within the country.

  Press Enter to return to Menu...
```

Figure 15 Place Order

- **Help**



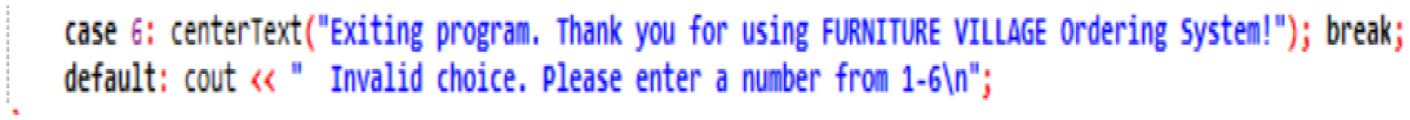
```cpp
void displayHelp() {
    cout << "\n\n\n";
    centerText("====================================================");
    centerText("HELP CENTER");
    centerText("====================================================");
    cout << endl;
    cout << "  1. Add Furniture:\n";
    cout << "     - Browse our furniture list across several categories\n";
    cout << "     - Select items by ID and specify the quantity you wish to purchase\n";
    cout << "     - Add multiple items to your order with the convenient 'Add more items?' option\n";
    cout << "     - System automatically checks stock availability to ensure your order can be fulfilled\n\n";

    cout << "  2. List Available Furniture:\n";
    cout << "     - View our complete furniture inventory with detailed information\n";
    cout << "     - See item IDs, names, categories, prices, and stock levels\n\n";

    cout << "  3. Search Furniture:\n";
    cout << "     - Find furniture by specific category (Bedroom, Dining, Office Furniture, Sofa)\n";
    cout << "     - View detailed information about items in your chosen category\n\n";

    cout << "  4. Place Order:\n";
    cout << "     - System automatically calculates the total cost of your order\n";
    cout << "     - Receive a unique order ID for easy tracking\n";
    cout << "     - Order summary is saved as a text file for your records\n\n";

    cout << "  5. Help:\n";
    cout << "     - Access this comprehensive help section for assistance\n";
    cout << "     - Find detailed information about all system features\n\n";

    cout << "  6. Exit:\n";
    cout << "     - Safely exit the FURNITURE VILLAGE Ordering System\n\n";
}
```



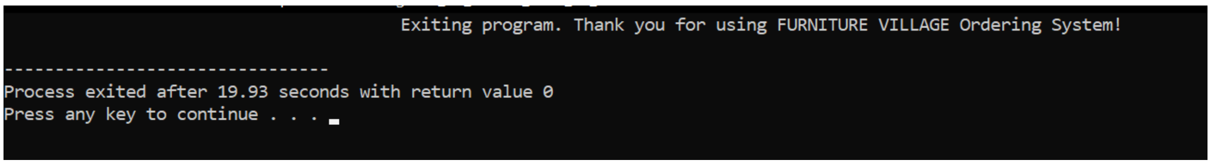
```
                                                ================================================
                                                                   HELP CENTER
                                                ================================================

1. Add Furniture:
   - Browse our furniture list across several categories
   - Select items by ID and specify the quantity you wish to purchase
   - Add multiple items to your order with the convenient 'Add more items?' option
   - System automatically checks stock availability to ensure your order can be fulfilled

2. List Available Furniture:
   - View our complete furniture inventory with detailed information
   - See item IDs, names, categories, prices, and stock levels

3. Search Furniture:
   - Find furniture by specific category (Bedroom, Dining, Office Furniture, Sofa)
   - View detailed information about items in your chosen category

4. Place Order:
   - System automatically calculates the total cost of your order
   - Receive a unique order ID for easy tracking
   - Order summary is saved as a text file for your records

5. Help:
   - Access this comprehensive help section for assistance
   - Find detailed information about all system features

6. Exit:
   - Safely exit the FURNITURE VILLAGE Ordering System


Press Enter to return to Menu...
```

Figure 16 Help

- **Exit**



```
case 6: centerText("Exiting program. Thank you for using FURNITURE VILLAGE Ordering System!"); break;
default: cout << "  Invalid choice. Please enter a number from 1-6\n";
```



```
                                       Exiting program. Thank you for using FURNITURE VILLAGE Ordering System!

--------------------------------
Process exited after 19.93 seconds with return value 0
Press any key to continue . . .
```

Figure 17 Exit

Task 02

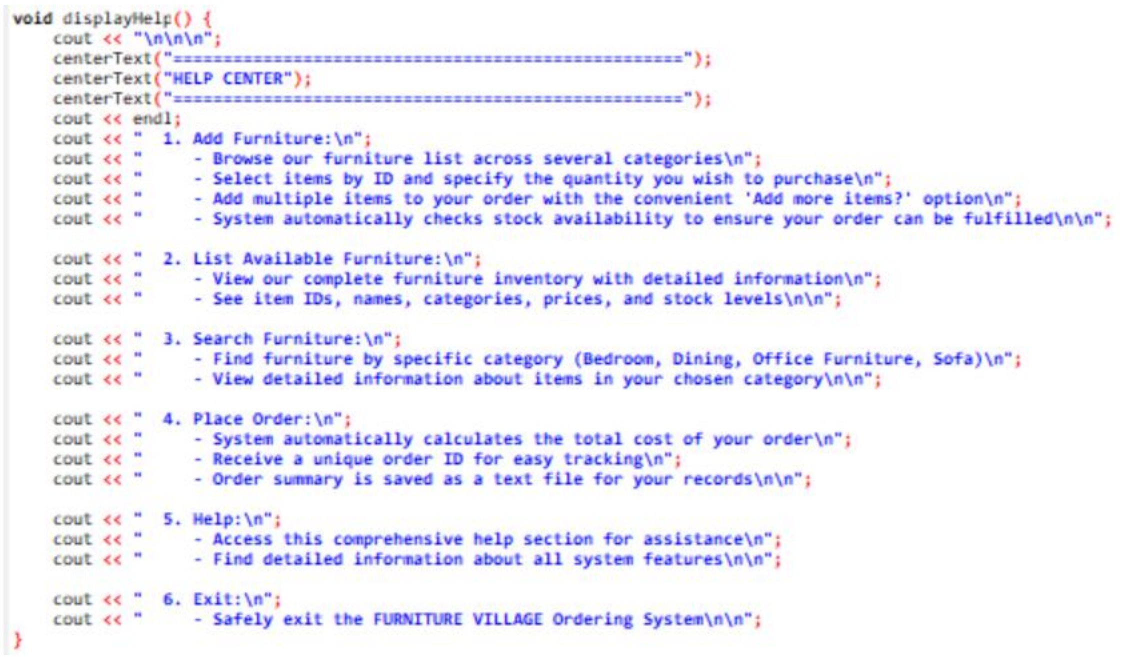
1. Structured Programming Style

Structured programming is an approach or programming paradigm of writing codes in a well-organized manner that ensures clarity, maintainability, reliability. This style discourages the use of unrestricted control flow and emphasizes on making the program with control flow structures breaking into smaller manageable functions (Ole-Johan, et al., 1972).

Structured programming uses three core control structures. They are:

1.1 Sequence Structures

Sequence structure follows a linear flow in which each step in an algorithm is executed in order. This executes each instruction exactly once before moving to the next and it doesn't skip or jump any algorithm step (Rama & Carol, 2009).



```
void displayHelp() {
    cout << "\n\n\n";
    centerText("====================================================");
    centerText("HELP CENTER");
    centerText("====================================================");
    cout << endl;
    cout << "  1. Add Furniture:\n";
    cout << "     - Browse our furniture list across several categories\n";
    cout << "     - Select items by ID and specify the quantity you wish to purchase\n";
    cout << "     - Add multiple items to your order with the convenient 'Add more items?' option\n";
    cout << "     - System automatically checks stock availability to ensure your order can be fulfilled\n\n";

    cout << "  2. List Available Furniture:\n";
    cout << "     - View our complete furniture inventory with detailed information\n";
    cout << "     - See item IDs, names, categories, prices, and stock levels\n\n";

    cout << "  3. Search Furniture:\n";
    cout << "     - Find furniture by specific category (Bedroom, Dining, Office Furniture, Sofa)\n";
    cout << "     - View detailed information about items in your chosen category\n\n";

    cout << "  4. Place Order:\n";
    cout << "     - System automatically calculates the total cost of your order\n";
    cout << "     - Receive a unique order ID for easy tracking\n";
    cout << "     - Order summary is saved as a text file for your records\n\n";

    cout << "  5. Help:\n";
    cout << "     - Access this comprehensive help section for assistance\n";
    cout << "     - Find detailed information about all system features\n\n";

    cout << "  6. Exit:\n";
    cout << "     - Safely exit the FURNITURE VILLAGE Ordering System\n\n";
}
```

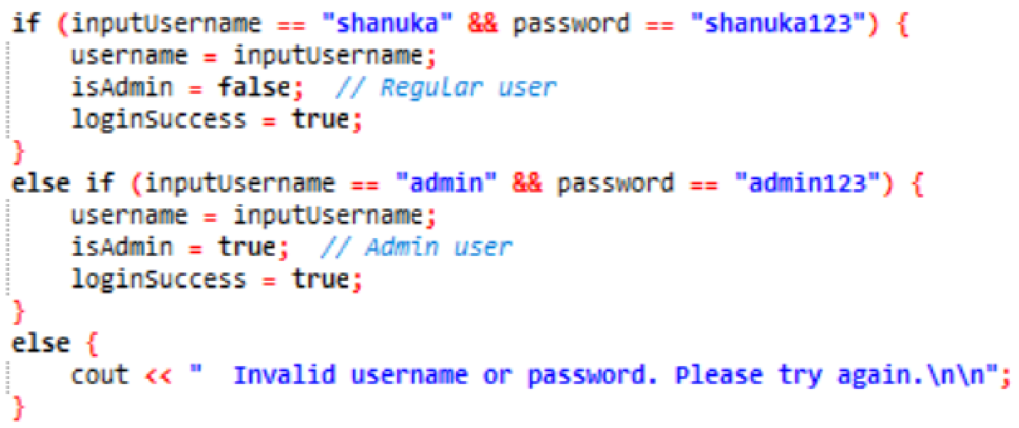
Figure 18 Sequence Structure

1.2 Selection Structure

Selection structure directs the flow of execution based on specific conditions. It checks a condition and depending on the result the statements or set of statements that should be executed are determined. Selection statements like “if”, “if-else” and “switch” are used to make decisions and execute various code paths (Hanly & Koffman, 2007).

1.2.1 If else Statement

If-else statement checks a condition and decides which code to execute based on the condition true or false. If the condition is true, the first statement runs, if not the second statement runs. These statements can be single, block of multiple or null commands (Dixit, 2005). If else statements are used here to check login credentials.

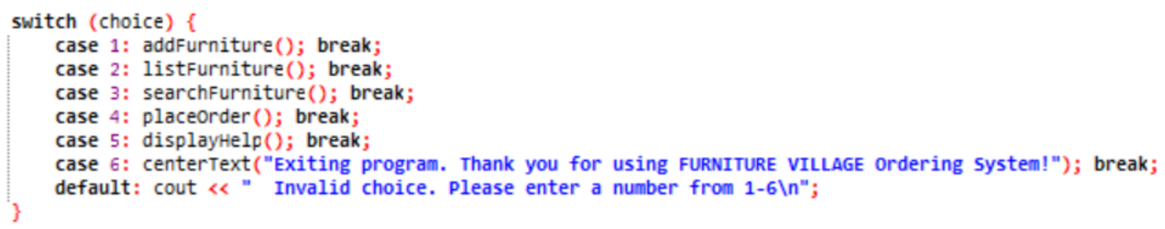


```
if (inputUsername == "shanuka" && password == "shanuka123") {
    username = inputUsername;
    isAdmin = false;  // Regular user
    loginSuccess = true;
}
else if (inputUsername == "admin" && password == "admin123") {
    username = inputUsername;
    isAdmin = true;  // Admin user
    loginSuccess = true;
}
else {
    cout << "  Invalid username or password. Please try again.\n\n";
}
```

Figure 19 If-else Statement

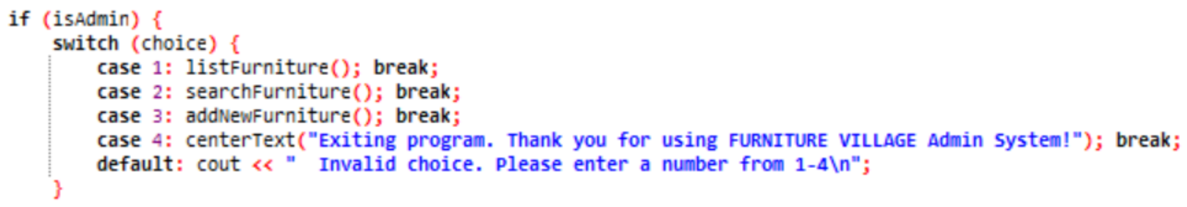
1.2.2 Switch Statement

Switch statement is used when there are multiple decisions that depend on same variable. It evaluates a control expression and directs execution to matching case based on its value. The expression must be a type of int or char, it is ideal choice rather than if-else or if-else-if ladder and commonly used in menu-driven programs (Dixit, 2005). These are used for the menus, value of `choice` is evaluated and executes different code based on its value.



```
switch (choice) {
    case 1: addFurniture(); break;
    case 2: listFurniture(); break;
    case 3: searchFurniture(); break;
    case 4: placeOrder(); break;
    case 5: displayHelp(); break;
    case 6: centerText("Exiting program. Thank you for using FURNITURE VILLAGE Ordering System!"); break;
    default: cout << "  Invalid choice. Please enter a number from 1-6\n";
}
```

Figure 20 Switch for Main Menu



```
if (isAdmin) {
    switch (choice) {
        case 1: listFurniture(); break;
        case 2: searchFurniture(); break;
        case 3: addNewFurniture(); break;
        case 4: centerText("Exiting program. Thank you for using FURNITURE VILLAGE Admin System!"); break;
        default: cout << "  Invalid choice. Please enter a number from 1-4\n";
    }
```

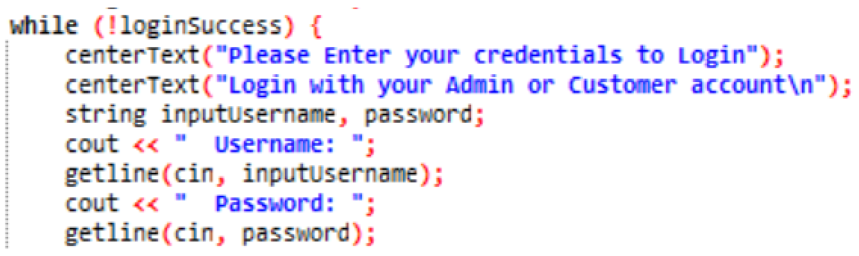
Figure 21 Switch for Admin Menu

1.3 Repetition Structure

Repetition statements also know as iteration or Looping is the process of repeatedly executing a set of statements until a given condition is met. These places the instructions inside a loop and can run multiple times (Rama & Carol, 2009).

1.3.1 While-Loop

While loop is a conditional loop that executes a set of statements repeatedly as long as a specified condition remains true. Before each iteration the condition is checked so that it won't run all the conditions that are set false initially (Rama & Carol, 2009). Here the loop repeatedly asks for the login credentials until it is valid.

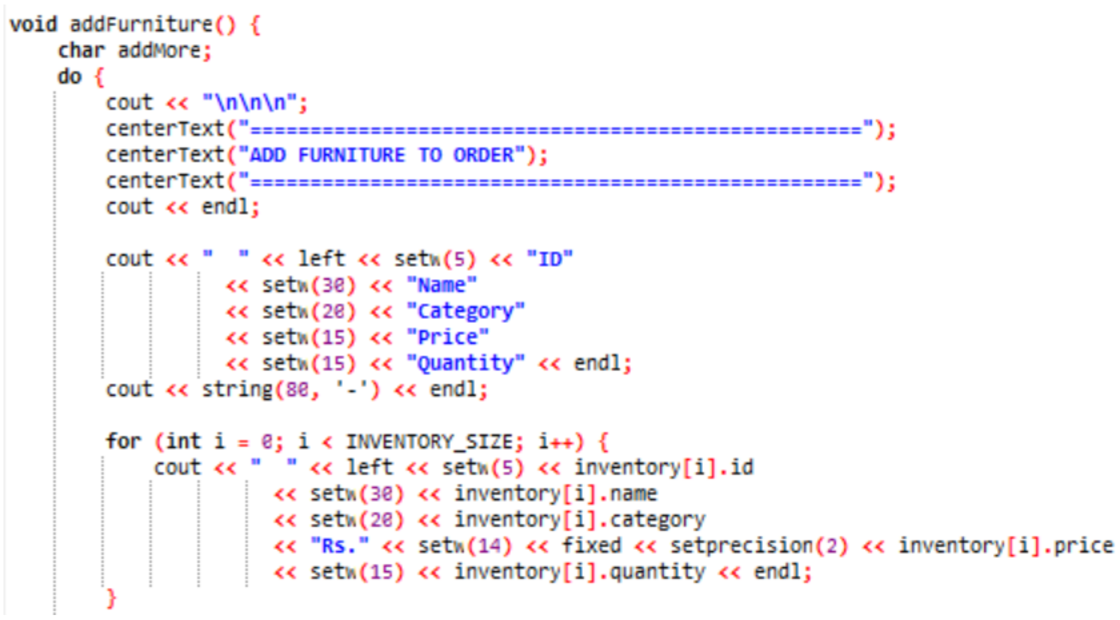


```
while (!loginSuccess) {
    centerText("Please Enter your credentials to Login");
    centerText("Login with your Admin or Customer account\n");
    string inputUsername, password;
    cout << "  Username: ";
    getline(cin, inputUsername);
    cout << "  Password: ";
    getline(cin, password);
```

Figure 22 While loop for login

1.3.2 Do-While Loop

Do-While Loop is a conditional loop that tests the condition at the end of each iteration so that the loop at least runs once before checking to continue (Rama & Carol, 2009). Here loop runs when the user chooses to add more items.

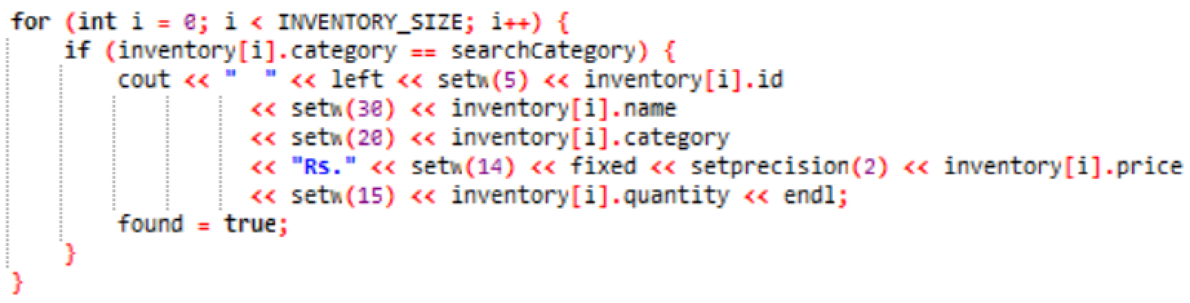


```
void addFurniture() {
    char addMore;
    do {
        cout << "\n\n\n";
        centerText("==================================================");
        centerText("ADD FURNITURE TO ORDER");
        centerText("==================================================");
        cout << endl;

        cout << "  " << left << setw(5) << "ID"
             << setw(30) << "Name"
             << setw(20) << "Category"
             << setw(15) << "Price"
             << setw(15) << "Quantity" << endl;
        cout << string(80, '-') << endl;

        for (int i = 0; i < INVENTORY_SIZE; i++) {
            cout << "  " << left << setw(5) << inventory[i].id
                 << setw(30) << inventory[i].name
                 << setw(20) << inventory[i].category
                 << "Rs." << setw(14) << fixed << setprecision(2) << inventory[i].price
                 << setw(15) << inventory[i].quantity << endl;
        }
```

Figure 23 Do-While Loop for Add Furniture

1.3.3 For Loop

For Loop can repeat a block of code a specific number of times. This consists 3-main sections of initialization, condition and manipulation. This loop runs until the condition remains true and it makes counting iterations efficient (Hanly & Koffman, 2007). Here the loop prints details of each item by iterating through the inventory array.



```
for (int i = 0; i < INVENTORY_SIZE; i++) {
    if (inventory[i].category == searchCategory) {
        cout << "  " << left << setw(5) << inventory[i].id
                << setw(30) << inventory[i].name
                << setw(20) << inventory[i].category
                << "Rs." << setw(14) << fixed << setprecision(2) << inventory[i].price
                << setw(15) << inventory[i].quantity << endl;
        found = true;
    }
}
```

Figure 24 For Loop for Inventory

2. Modularization

Modularization is a technique of dividing complex programs/ problems into manageable, smaller, organizable modules or units. This approach "divide and conquer" allows us to divide a large application into manageable tasks to resolve various challenges faced simplifying the error-handling, debugging processes. Trough this Modular structured approach in building programs and problem solving by separating various functionality into self-contained units or modules the code's organization, readability, and maintainability in programming is improved (Collopy, 2000).

For the Furniture ordering system, we made for Furniture Village the C++ code was organized to ensure readability, reusability and maintainability by using functions to specific tasks like displayMenu(), displayAdminMenu(), addFurniture() and structs to group data together.

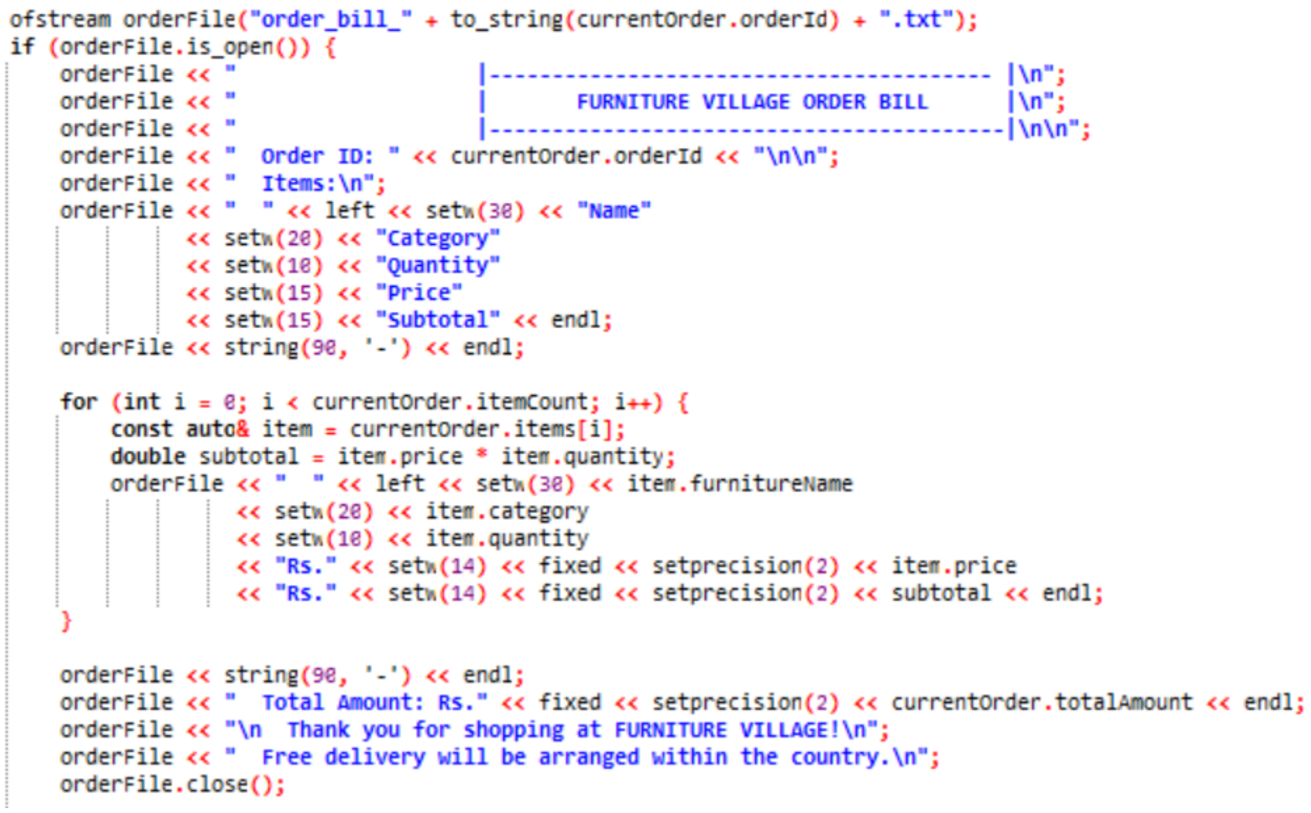
3. Appropriate Storage & Backup Requirements

3.1 File Handling

File handling is the process of storing data in external files to read and write from those. This allows the program to handle large amount of data that is not in the program’s memory. In C++, ofstream is for reading and ifstream is for writing to files.

3.1.1 Writing to a File (ofstream)

This creates an output file stream named “oderFile” for the Order Bill with the formatted text.

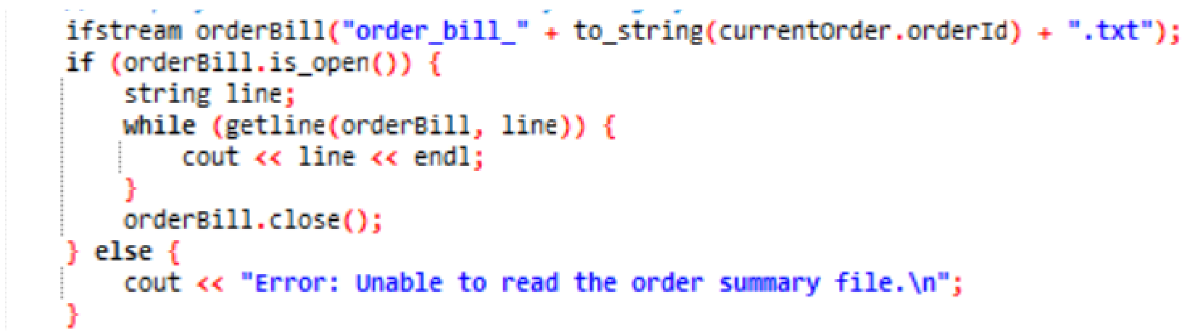


```cpp
ofstream orderFile("order_bill_" + to_string(currentOrder.orderId) + ".txt");
if (orderFile.is_open()) {
    orderFile << "                    |---------------------------------------- |\n";
    orderFile << "                    |       FURNITURE VILLAGE ORDER BILL      |\n";
    orderFile << "                    |-----------------------------------------|\n\n";
    orderFile << "  Order ID: " << currentOrder.orderId << "\n\n";
    orderFile << "  Items:\n";
    orderFile << "  " << left << setw(30) << "Name"
              << setw(20) << "Category"
              << setw(10) << "Quantity"
              << setw(15) << "Price"
              << setw(15) << "Subtotal" << endl;
    orderFile << string(90, '-') << endl;

    for (int i = 0; i < currentOrder.itemCount; i++) {
        const auto& item = currentOrder.items[i];
        double subtotal = item.price * item.quantity;
        orderFile << "  " << left << setw(30) << item.furnitureName
                  << setw(20) << item.category
                  << setw(10) << item.quantity
                  << "Rs." << setw(14) << fixed << setprecision(2) << item.price
                  << "Rs." << setw(14) << fixed << setprecision(2) << subtotal << endl;
    }

    orderFile << string(90, '-') << endl;
    orderFile << "  Total Amount: Rs." << fixed << setprecision(2) << currentOrder.totalAmount << endl;
    orderFile << "\n  Thank you for shopping at FURNITURE VILLAGE!\n";
    orderFile << "  Free delivery will be arranged within the country.\n";
    orderFile.close();
```

Figure 25 Ofstream for Saving Order Bill

3.1.2 Reading from a File (ifstream)

This creates an input file stream named “orderBill” and reads the file line by line displaying it on the console.

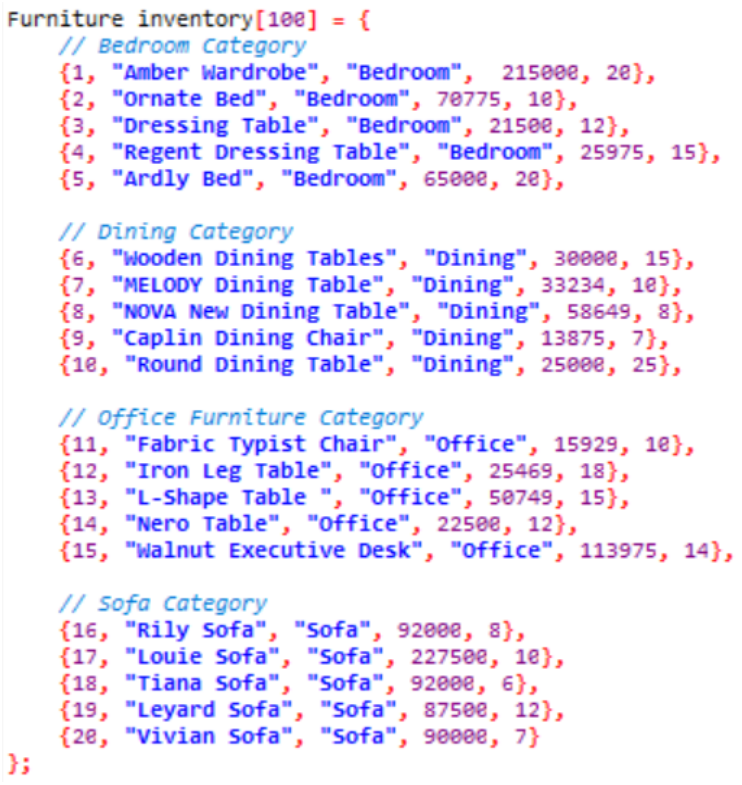


```cpp
ifstream orderBill("order_bill_" + to_string(currentOrder.orderId) + ".txt");
if (orderBill.is_open()) {
    string line;
    while (getline(orderBill, line)) {
        cout << line << endl;
    }
    orderBill.close();
} else {
    cout << "Error: Unable to read the order summary file.\n";
}
```

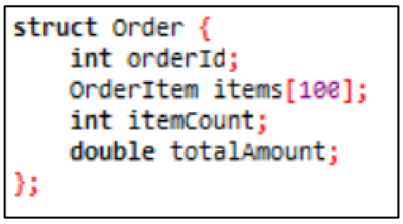
Figure 26 Ifstream for Reading Order Bill

3.2 Arrays

An array is a method to organize a collection of items that have same type into a single data structure. Here an array of structures named “inventory” is declared to store up to 100 furniture items.



```
Furniture inventory[100] = {
    // Bedroom Category
    {1, "Amber Wardrobe", "Bedroom",  215000, 20},
    {2, "Ornate Bed", "Bedroom", 70775, 10},
    {3, "Dressing Table", "Bedroom", 21500, 12},
    {4, "Regent Dressing Table", "Bedroom", 25975, 15},
    {5, "Ardly Bed", "Bedroom", 65000, 20},

    // Dining Category
    {6, "Wooden Dining Tables", "Dining", 30000, 15},
    {7, "MELODY Dining Table", "Dining", 33234, 10},
    {8, "NOVA New Dining Table", "Dining", 58649, 8},
    {9, "Caplin Dining Chair", "Dining", 13875, 7},
    {10, "Round Dining Table", "Dining", 25000, 25},

    // Office Furniture Category
    {11, "Fabric Typist Chair", "Office", 15929, 10},
    {12, "Iron Leg Table", "Office", 25469, 18},
    {13, "L-Shape Table ", "Office", 50749, 15},
    {14, "Nero Table", "Office", 22500, 12},
    {15, "Walnut Executive Desk", "Office", 113975, 14},

    // Sofa Category
    {16, "Rily Sofa", "Sofa", 92000, 8},
    {17, "Louie Sofa", "Sofa", 227500, 10},
    {18, "Tiana Sofa", "Sofa", 92000, 6},
    {19, "Leyard Sofa", "Sofa", 87500, 12},
    {20, "Vivian Sofa", "Sofa", 90000, 7}
};
```

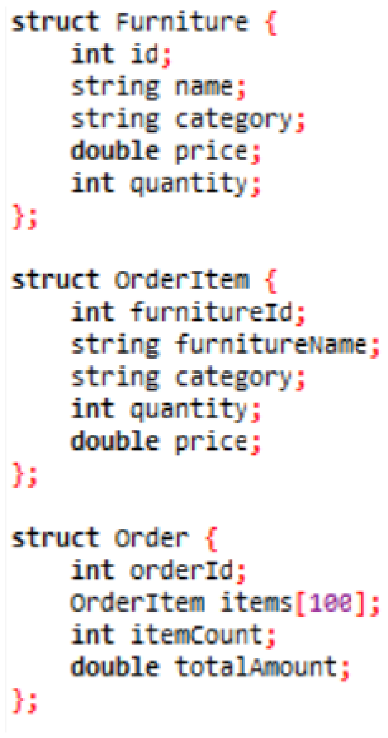


```
struct Order {
    int orderId;
    OrderItem items[100];
    int itemCount;
    double totalAmount;
};
```

Figure 27 Array for Furniture items

3.3 Structs (Records)

Structures are helpful in organizing complex data efficiently. These provides a way to group different types of data into a single unit making it easier to handle related data, and ensures reliability. Here several structs are used for Furniture to group related properties like (ID, name, category, price and quantity), and Order to store (order Id, list of items, and total cost).



```
struct Furniture {
    int id;
    string name;
    string category;
    double price;
    int quantity;
};

struct OrderItem {
    int furnitureId;
    string furnitureName;
    string category;
    int quantity;
    double price;
};

struct Order {
    int orderId;
    OrderItem items[100];
    int itemCount;
    double totalAmount;
};
```

Figure 28 Structures (Records)

Task 03

1. Test Plan

1.1 Objective

Objective of this test plan is to verify the functionality, reliability and usability of Furniture Village Ordering system ensuring that the system will operate without errors in order to improve user experience.

1.2 Scope

Test plan covers the functionalities of user authentication, furniture management such as adding furniture’s to order, adding new items, searching specific items, and viewing list of items, finally, order management where the placing order is validated.

1.3 Test Environment

- Os: Windows 11
- Complier: Dev C++ complier
- IDE: Dev C++

1.4 Test Tools

- Manual testing will be conducted to evaluate the system’s functionality and usability.
- Feedback form will be used to collect user ideas in User Acceptance Testing (UAT).

2. Test Cases & Results

2.1 User Authentication

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC001	Verify Admin User-Login	-Run program -Enter valid credentials for admin	Username: admin Password: admin123	Admin successfully logged-in, display admin menu	Pass
TC002	Verify regular User-login	-Run program -Enter valid credentials for regular customer	Username: shanuka Password: shanuka123	User successfully logged-in, display Main-Menu	Pass

Table 3 User Authentication Test Case

2.2 Add Furniture

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC003	Add items to order	-Select “Add Furniture” option. -Enter valid ID and quantity	Valid Furniture ID and quantity	Item added to order	Pass

Table 4 Add Furniture Test Case

2.3 List of Available Furnitures

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC004	Display available furniture list	-Select “List of Furnitures” option.	N/A	List of all the furniture items displayed	Pass

Table 5 List of Available Furnitures Test Case

2.4 Search Furniture

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC005	Search furniture by category	-Select “Search Furniture” option. -Enter valid category name	Valid category:	Display all items matching with the entered category	Pass

Table 6 Search Furniture Test Case

2.5 Place Order

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC006	Place order with items added	-Select “Place order” option.	Items should be in order	Generates order-bill. Saves as text-file and displays bill	Pass

Table 7 Place Order Test case

2.6 Help

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC007	Display help section	-Select “Help” option.	N/A	Display detailed help information’s	Pass

Table 8 Help Test Case

2.7 Exit

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC008	Terminate the program	- Select “Exit” option.	N/A	Display Thank you message, exits program	Pass

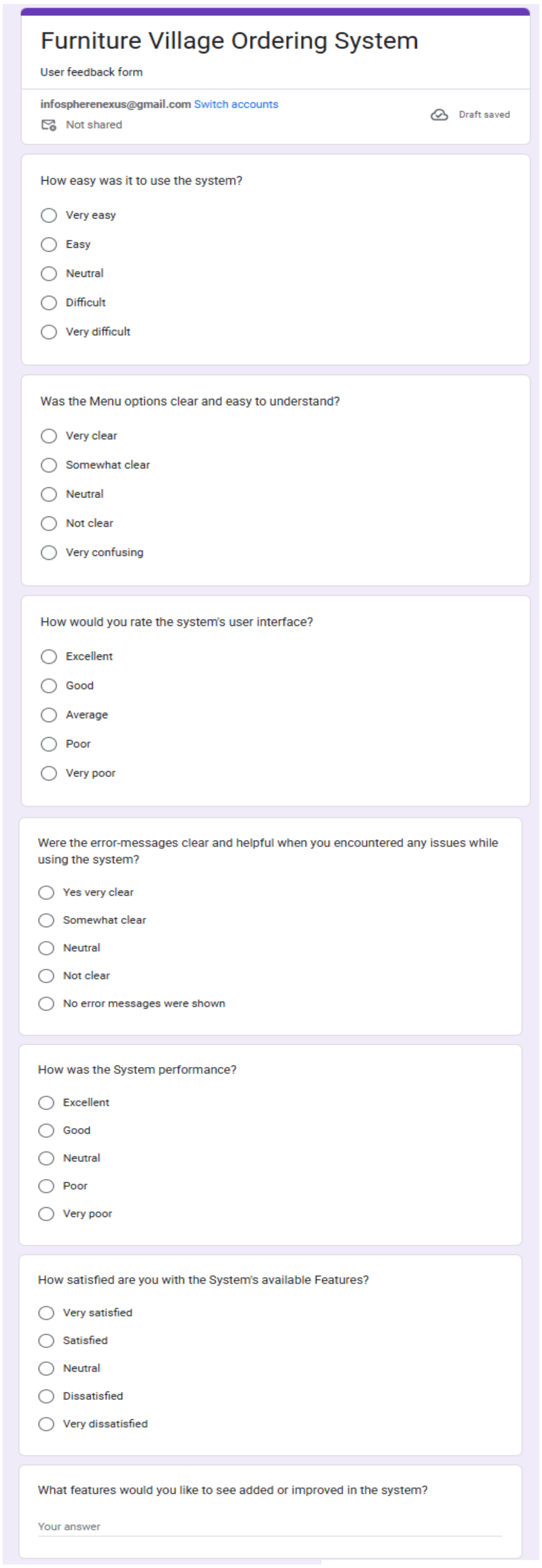
Table 9 Exit Test Case

2.8 Add New Furniture (Only Admin)

Test Case ID	Test Scenario	Steps	Test Data	Expected Result	Status
TC009	Adding new item to inventory by admin	-Select “Add New Furniture” option from AdminMenu.	Valid id, name, category, price and quantity	Display New furniture Item added successfully.	Pass

Table 10 Add New Furniture Test Case

3. Feedback Form



Furniture Village Ordering System

User feedback form

infospherenexus@gmail.com Switch accounts

Not shared

Draft saved

How easy was it to use the system?

- Very easy
- Easy
- Neutral
- Difficult
- Very difficult

Was the Menu options clear and easy to understand?

- Very clear
- Somewhat clear
- Neutral
- Not clear
- Very confusing

How would you rate the system's user interface?

- Excellent
- Good
- Average
- Poor
- Very poor

Were the error-messages clear and helpful when you encountered any issues while using the system?

- Yes very clear
- Somewhat clear
- Neutral
- Not clear
- No error messages were shown

How was the System performance?

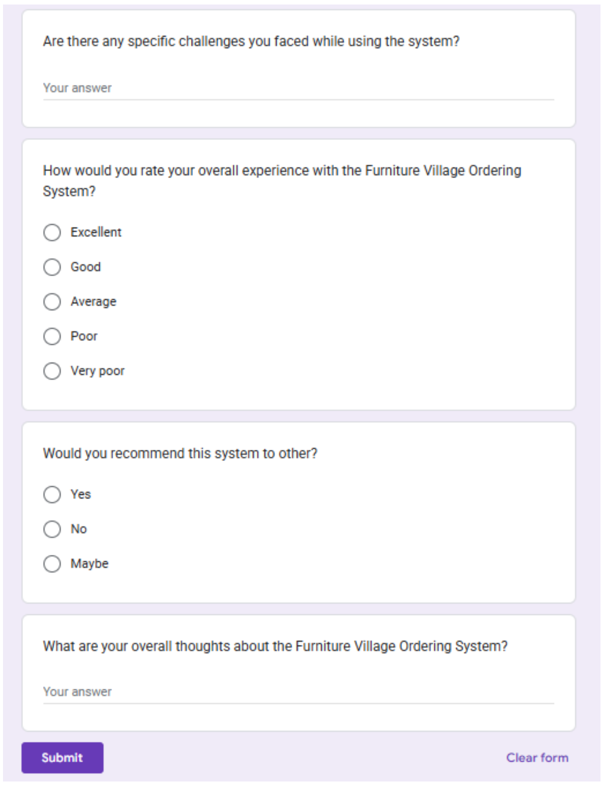
- Excellent
- Good
- Neutral
- Poor
- Very poor

How satisfied are you with the System's available Features?

- Very satisfied
- Satisfied
- Neutral
- Dissatisfied
- Very dissatisfied

What features would you like to see added or improved in the system?

Your answer



Are there any specific challenges you faced while using the system?

Your answer

How would you rate your overall experience with the Furniture Village Ordering System?

- Excellent
- Good
- Average
- Poor
- Very poor

Would you recommend this system to other?

- Yes
- No
- Maybe

What are your overall thoughts about the Furniture Village Ordering System?

Your answer

Submit

Clear form

Figure 29 Feedback Form

Conclusion

In conclusion, the Furniture Village Ordering System we programmed for Furniture Village effectively automates the key ordering processes, like inventory management, order processing, and bill generation addressing the manual operations and more importantly improving customer satisfaction. The system successfully streamlines store operations eliminating the manual errors and reducing employee workloads. However, the system has several limitations such as lack of proper database for user data and inventory management which can be an issue for scalability and security.

Overall, the system meets the core requirements such as Add Furnitures to order, List of Available Furnitures, Search furniture’s, and Place Orders along with adding furniture items to inventory. This system can be further developed with additional features and improvements like database integration, graphical user interface, and more dynamic inventory management options allowing to manage Furniture Villages operations more efficiently and effectively while they grow.

References

Collopy, 2000. *Introduction To Cobol: A Guide To Modular Structured Programming.* s.l.:Pearson Education.

Dixit, J., 2005. *Fundamentals of Computer Programming and Information Technology.* s.l.:Laxmi Publications Pvt Limited.

Hanly, J. R. & Koffman, E. B., 2007. *Problem Solving and Program Design in C.* s.l.:Pearson Addison Wesley.

Dahl, O.j., Dijkstra, E.W., & Hoare, C.A.R., 1972. *Structured Programming..* s.l.:New York: Academic Press..

Rama, R. & Carol, Z., 2009. *C Programming for Scientists and Engineers with Applications.* s.l.:Jones & Bartlett Learning.

Appendix

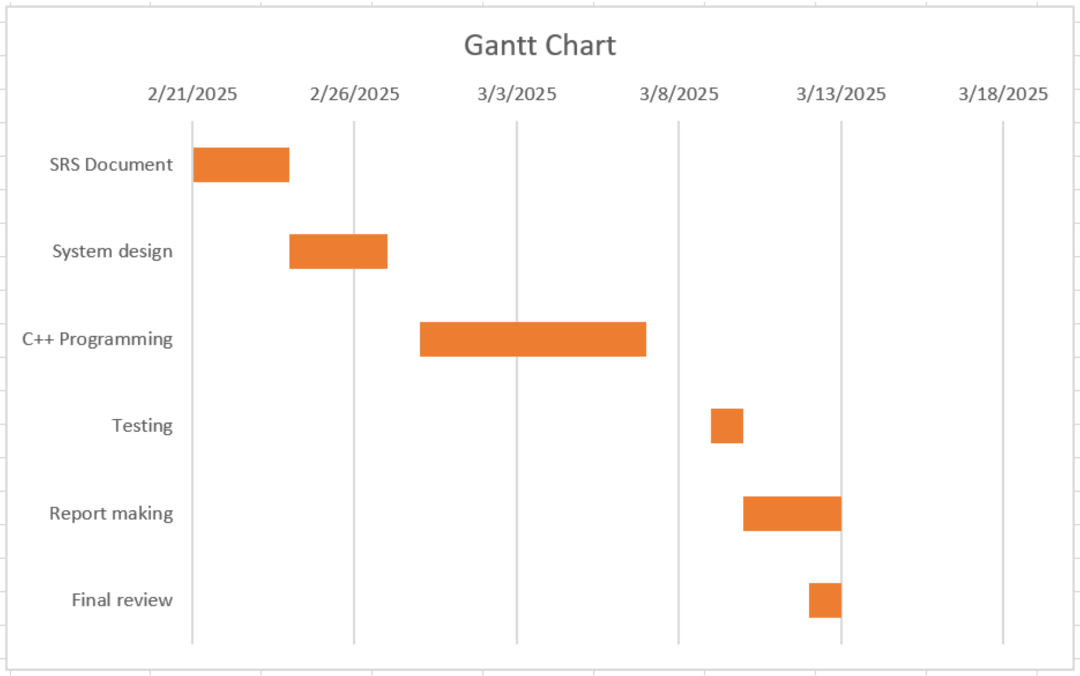


Figure 30 Appendix-Gantt Chart