Bite & Byte

Hablado, Zergio Achillis E.
Larracas, Jerry Matthew L.
Limpio, Ramelissa Melith V.
Meija, Princess Iana B.
Orzal, Chloe Nickhaela C.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

Many fast food stores face problems with their ordering process. Customers often wait in line for a very long time and get the wrong, missing orders and delays when there are too many customers at once because the ordering process is still done manually, which can be tiring and confusing for both staff and customers. According to Pangilinan (2025), issues in fast food service responsiveness, shows that slow and inefficient order handling affects an average customer level satisfaction of 3.6 out of 5. The categories to be rated include food, value, cleanliness, service, and location. Among the categories, the service got the lowest rating of 4.01 out of 5 from 129 respondents. Even if the introduction of the self ordering kiosks model, long lines still form during rush hours, providing that the main problem in the fast food ordering system continues to exist.

Most fast-food systems do not include features like menu updates, a proper stocking system, separate access for managers and cashiers, estimated waiting time, and coupon or discount options. Without these features, it becomes difficult to maintain smooth service and efficient management

To fix these issues, The project focuses on creating a fast food ordering system that helps customers to order faster and more organized. The system includes menu display, a stocking system to track available items, and separate access for the manager and cashier to manage tasks better. The system will also show the estimated waiting time for each customer's order and allows the use of coupons or discounts to make it more convenient.

The Project

The project aims to solve the difficulty in managing multiple orders, incorrect orders, and long waiting times. For an organized system to handle different orders, it will use loops and arrays. A menu with the string variable, so the customers can see what they are ordering, an if-else statement that estimates the time when the order is finished, and a looping structured inventory system that shows both the customer and the cashier that a product is out of stock. It will also include a conditional coupon and discount system that also uses the if else statement, where it would be at the added total in the order, and lastly a manager and cashier admin with another if else statement to know who's using the code. The manager has the permission to set the stock of how many is available. Overall, the code breaks it into smaller and easier to make it more understandable. It also helps in identifying errors more quickly, allowing easier fixing and debugging when each part of the code is manageable in small and bite sized pieces.

Objectives

The goal of this project is to develop a C++ based fast food ordering system that automates the ordering process to minimize manual errors.

Develop a C++ program that can:

1. Display a menu and ordering interface that allows the user to order Burger Steak, Coca-Cola and Chicken Nuggets
2. Store and display an order summary showing item details, item quantity and total bill of the customer.
3. Implement a stocking system that tracks product availability and updates after each transaction.
4. Create an employee access system, where the cashier handles orders, and the manager can change the available stocks and turn off the system.
5. Integrate a coupon and discount functionalities that apply special discounts such as 20% for PWD or senior citizen and 10% for coupons to the total bill.
6. Calculate and display an estimated waiting time based on the number of items ordered and the ongoing orders before.

Flowchart of the System

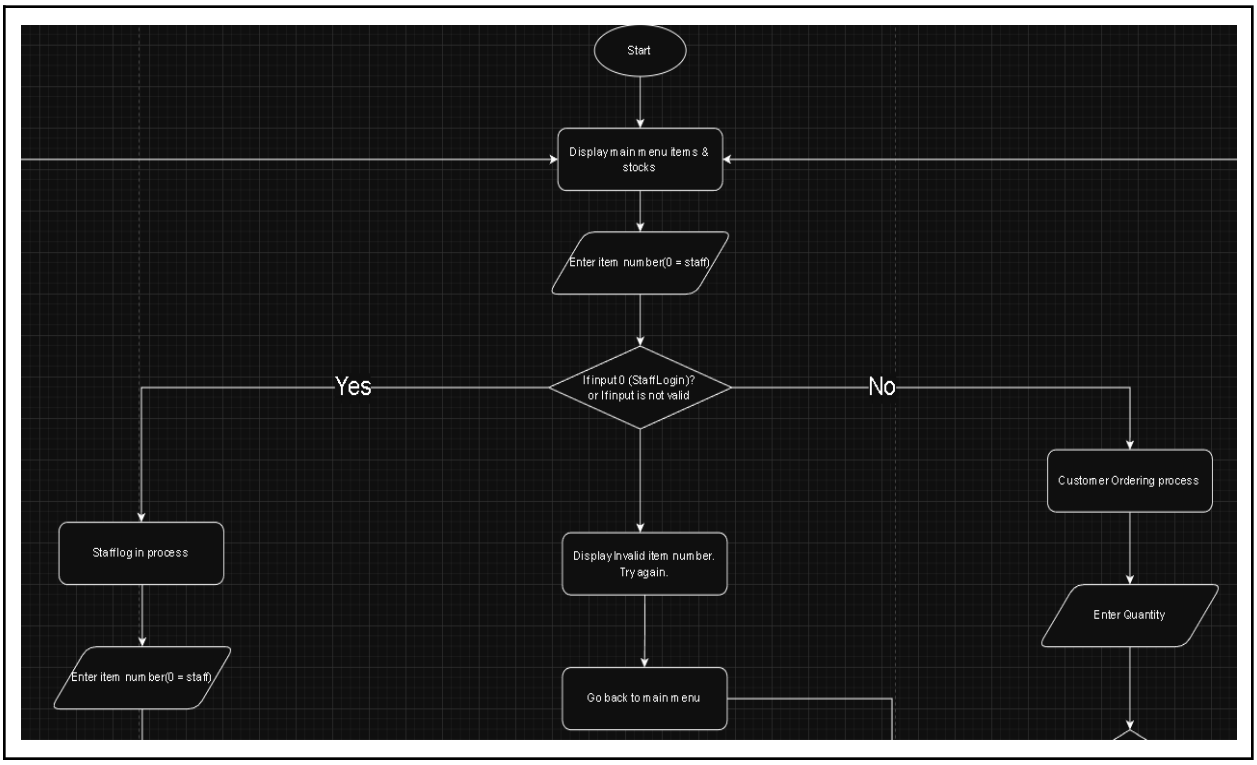


Figure 1: Flow chart of Fast Food Ordering System

Figure 1 illustrates the start of the system. The system will display main menu items and available stocks. The system then asks the user to enter an item number. If the input is invalid, then the system will go back to the main menu and if the input is 0, the system will go to the staff log in process. But if the user inputs a valid number, then it will undergo a customer ordering process.

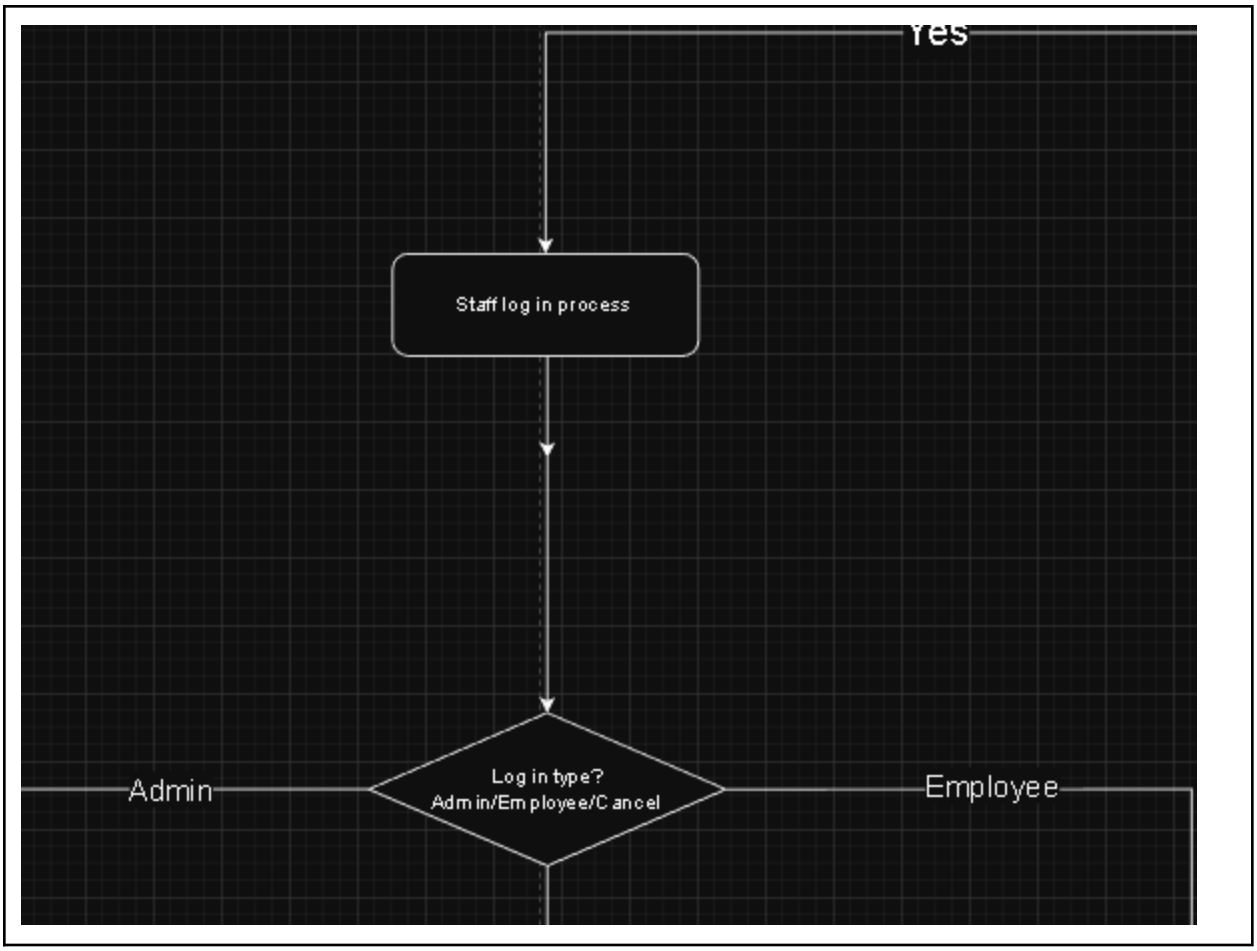


Figure 2: Flowchart part of Staff log in process

Figure 2 shows that if the input is 0, then the system will proceed to the staff log in process. Then the system will then ask the user what type of login if it is admin or employee or cancel the log in process.

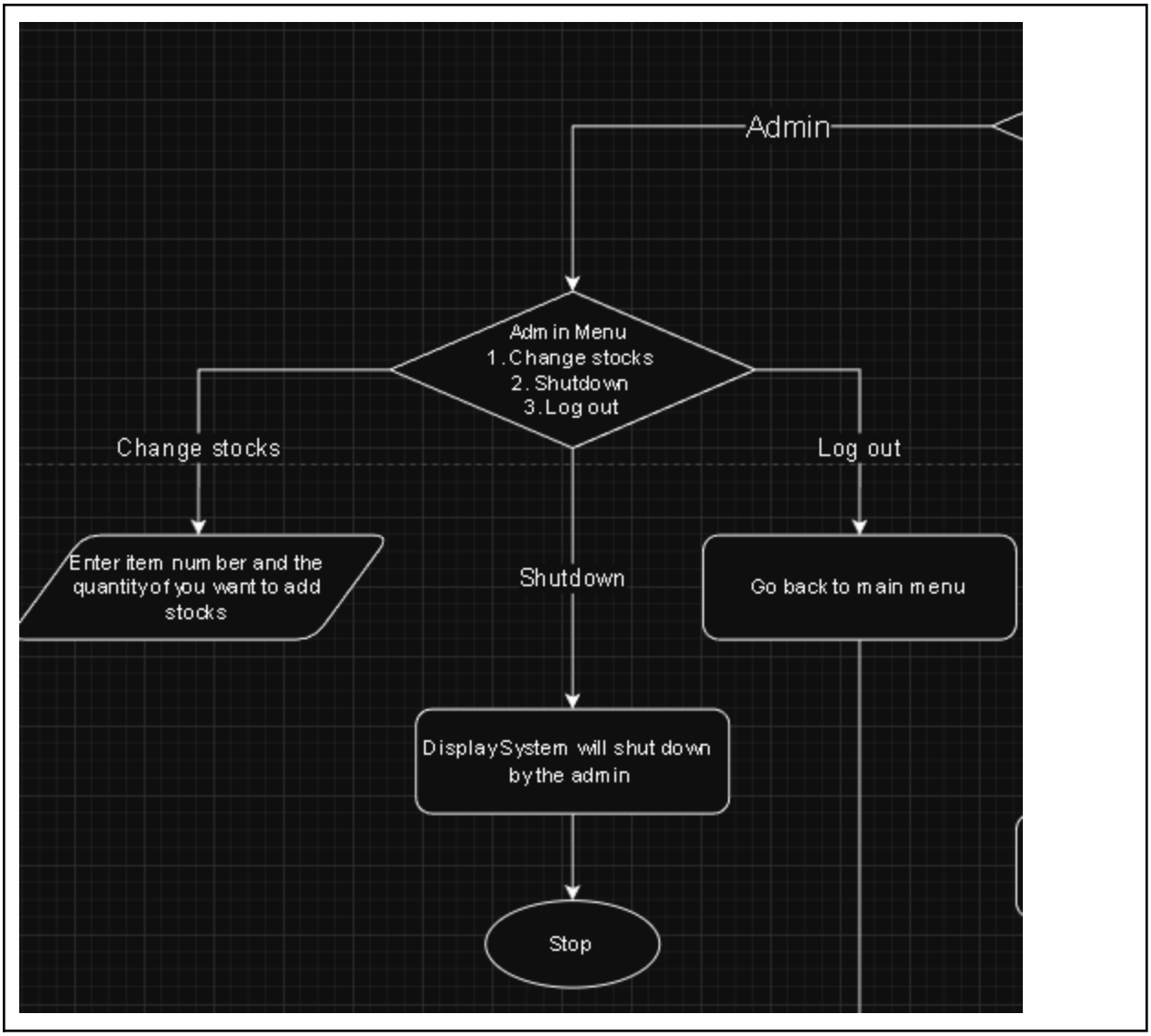


Figure 3: Flow chart part of admin log in process and its functions

Figure 3 shows that if the user is logged in as an admin, the system will display an admin menu and has a function to change stocks, shutdown and log out as an admin.

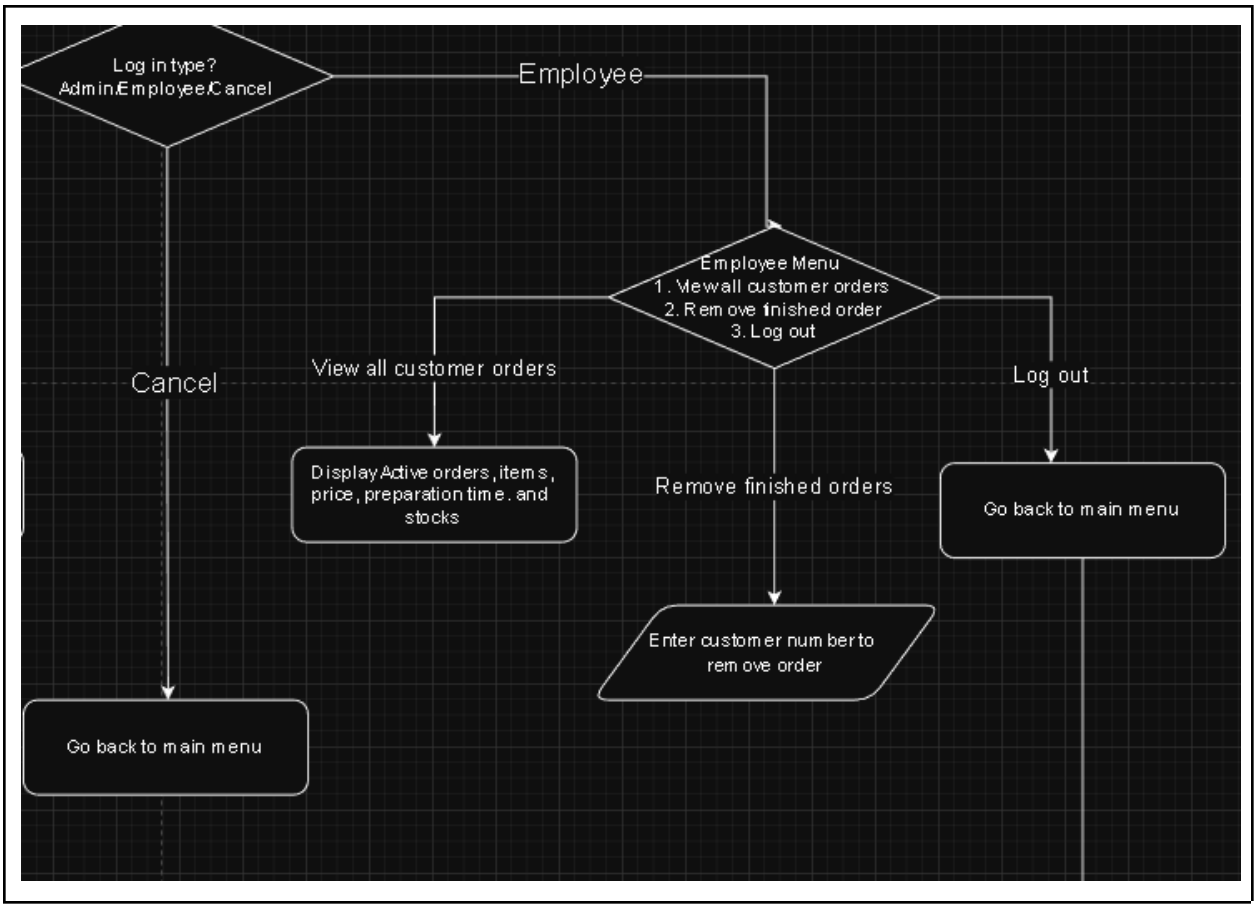


Figure 4: Flow chart part of employee log in process and its functions.

Figure 4 represents that if the user is logged in as an employee, the system will display an employee menu that has a function to view all customer orders, remove finished orders, and log out as an employee.

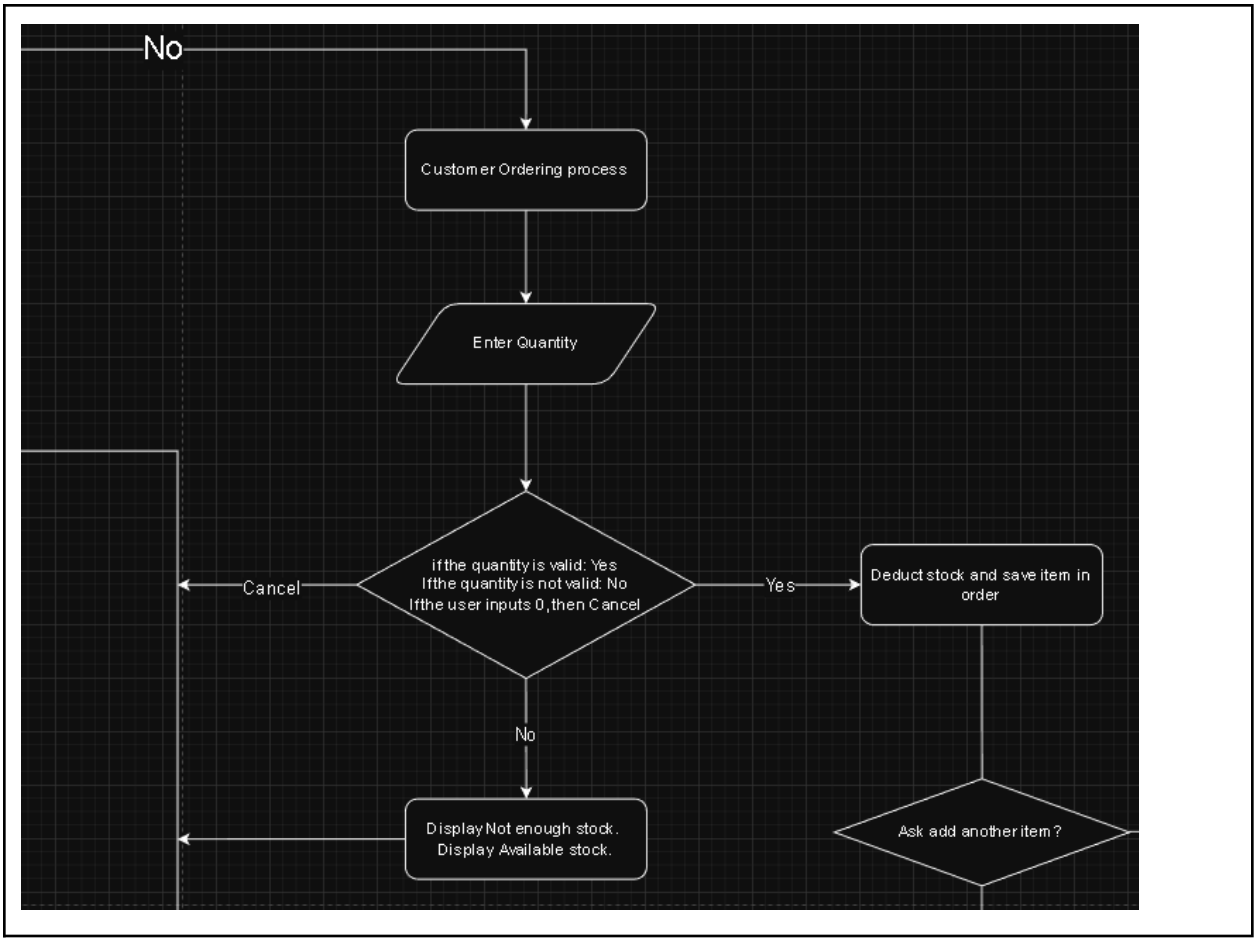


Figure 5: Flow chart part of customer ordering process

Figure 5 shows if the input is valid, then the system will proceed to the customer ordering process. The system will then ask the user to enter a quantity. If the quantity is valid, the system will deduct the stock of the selected item and save the item in the order list. If the quantity is not valid, the system will display not enough stock and shows the only available stock. And if the user inputs 0, the system will return to the main menu.

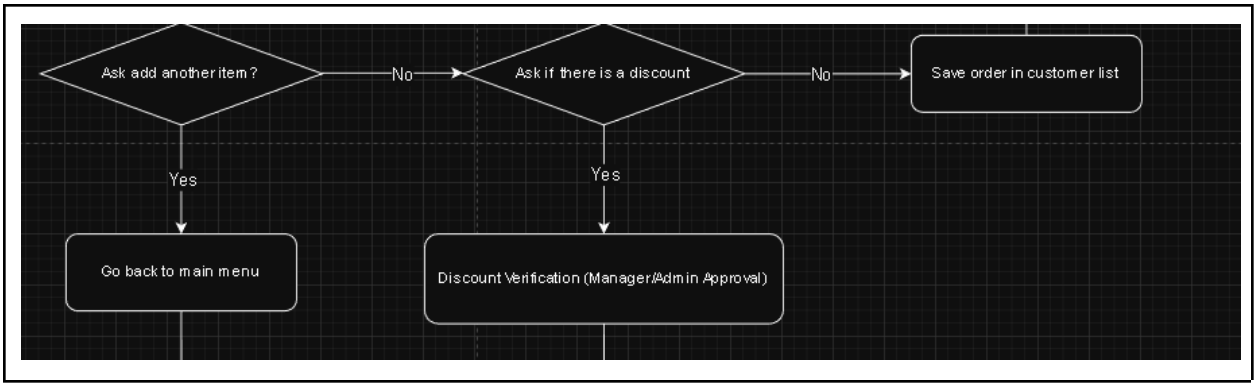


Figure 6: Flow chart part of customer ordering process and discount verification.

Figure 6 shows that if the item successfully deducted and saved in the order list, the system will then ask the user to add another item or not. If yes, then the system will return to the main menu and if no, the system will then ask again if there is a discount. If not, the system will save the order in the customer list with the complete receipt, including prices and estimated time. And if yes, the system will then proceed to the discount verification process.

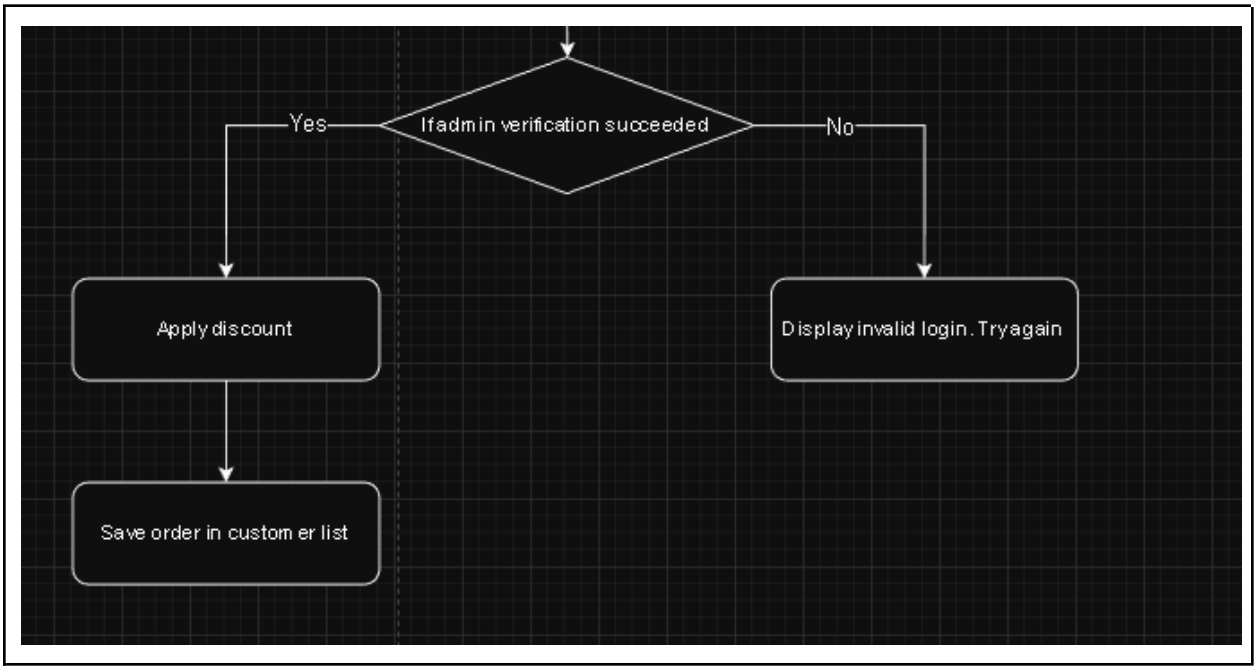


Figure 7: Flow chart part of discount verification process

Figure 7 shows the discount verification process where the admin or manager only has access to approve the discount.

Pseudocode

This part will show the pseudocode of the system.

```
START

Create menu items, prices, preparation time, and stocks
Create empty list of customer orders
Set customer count to 0
Set system status to running

REPEAT while system is running:
      Create a new order for a customer
      Set discount to 0% (No discount yet)

DO the following while ordering:

    DISPLAY menu items, prices, and remaining stocks
          Ask user to select an item
          Ask user for quantity
                IF quantity is 0:
                     Cancel and return to menu
                IF quantity is more than stocks:
                      Display error and ask again
                END IF

     Deduct stock from item
     Add item, quantity, price, and preparation time to customer order

    Ask if the customer want to add another item
        IF yes → repeat ordering
        IF no → proceed
        END IF

    IF user enters 0:
       Go to staff login

    IF admin shuts down system
       stop process
   OTHERWISE → go back to menu
   END IF

    IF item is not valid:
       Display error and ask again
    END IF

END DO

IF system was shut down → STOP process
```

```
Ask if the customer has discount
      IF yes:
          Ask discount type (Senior/PWD = 20%, Coupon = 10%)
          Ask manager/admin to login for approval
                IF login success:
                    Apply discount
                IF login failed:
                    Do not apply discount
      END IF

Save order to customer list
Increase customer count
Display all active orders

END REPEAT

WHEN admin shuts down:
     DISPLAY "System stopped by admin"

END
```

Figure 8: Pseudo code of Fast Food Ordering System

Figure 2 presents the pseudo code of the Fast Food Ordering System. It shows the whole process of the system including displaying the menu and inventory program that allows staff to select items, enter quantities, and check stock availability before confirming orders. It deducts stock, records order details, and allows multiple items per transaction while validating inputs. The system also includes discounts that require admin or manager approval. The system will continue to process until the admin shuts it down.

```
STAFF LOGIN PROCESS

Ask if login is Admin or Employee
IF Admin:
   Ask username & password
IF correct → open Admin menu
   ELSE Go back to the main menu and display error
END IF

IF Employee:
   Ask username & password
   IF correct → open Employee menu
   ELSE Go back to the main menu and display error
END IF
```

Figure 9: Pseudo code of Fast Food Ordering System

Figure 3 presents the pseudocode of the staff login process of the system. It shows the process of staff login where the user is asked to log in as admin or employee and ask its username and password to login. If the user inputs invalid credentials, then the system will display an error message and the system will go back to the main menu.

Data Dictionary

This part will show the data dictionary of the system consisting of all the data types, size of the data type, data name, and description.

A data dictionary is where each data type of the system is being written down using a table that consists of the data name, size, data type and description of its process. The Data name is the name of the variable in the system. The size of the data type is how much memory in the variable used in the system, measured in bytes. The Data type is the kind of data that the variable stores. The description of the data type explains what the variable is used for in the system.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
1. OrderItem	~32-40 bytes	struct	Stores details of one ordered item (product, qty, price, prepTime).
2. CustomerOrder	~340 bytes - 420 bytes	struct	Stores multiple order items, item count, and discount for one customer.
3. customers[50]	~17kb - 21kb	CustomerOrder array	Stores up to 50 customer orders.
4. customerCount	4 bytes	int	Counts stored customer orders.
5. product[3]	~72 - 96 bytes	String array	Menu item names (each string approx 24-32 bytes, so it is ~72-96 bytes).
6, price[3]	12 bytes	Int array	Price per menu item.
7. avg_time[3]	12 bytes	Int array	Preparation time per item.
8. programRunning	1 byte	bool	System ON/OFF status.
9. itemCount	4 bytes	int	Counts items ordered by one customer.
10 . Discount	4 bytes	float	Discount multiplier
11. prepTime	4 bytes	int	Total preparation time.
12. qty	4 bytes	int	Quantity ordered.
13. choice	4 bytes	int	Menu input for admin/employee.
14. index	4 bytes	int	Selected customer number for deletion.

15. username	24 - 32 bytes	string	Login username input.
16. password	24 - 32 bytes	string	Login password input.
17. retry	1 byte	char	Retry function if login fails
18. sel	4 bytes	int	Login type selection if Admin/Employee.
19. item_number	4 bytes	int	Selected product number.
20. quantity	4 bytes	int	Quantity input of selected item.
21. addMore	24 - 32 bytes	string	Ask the user yes/no for adding another item.
22. hasDiscount	24 - 32 bytes	string	Ask the user if there is a discount.
23. discType	4 bytes	int	Discount type chosen (PWD/Coupons).
24. u	24 - 32 bytes	string	input's username
25. p	24 - 32 bytes	string	input's password
26. linePrice	4 bytes	float	Stores total price per item after discount.
27. total	4 bytes	float	Total bill of customer order.
28. totalTime	4 bytes	int	Total preparation time of all items in order.
29. newStock	4 bytes	int	Stores updated stock value entered by admin.
30. stocks[3]	12 bytes	Int array	Current stock per item.

Code

This part will show the code of the system.

```cpp
#include <iostream>
#include <string>
//Includes <iomanip> for aligned menu formatting
#include <iomanip>
using namespace std;

//structure for order items
struct OrderItem {
    string product;
    int qty;
    int price;
    int prepTime;
};

//structure for the customer's items
struct CustomerOrder {
    OrderItem items[10];
    int itemCount;
    float discount;
};

// Maximum number of customer orders the system can store
CustomerOrder customers[50];
int customerCount = 0;

//Menu items, their prices, average preparation time and stocks counts
string product[] = {"Burger Steak", "Coca-cola", "Chicken Nuggets"};
int price[] = {90, 25, 70};
int avg_time[] = {2, 1, 2};
int stocks[] = {5, 5, 5};

//Keeps the program running until it is manually shut down
bool programRunning = true;

```

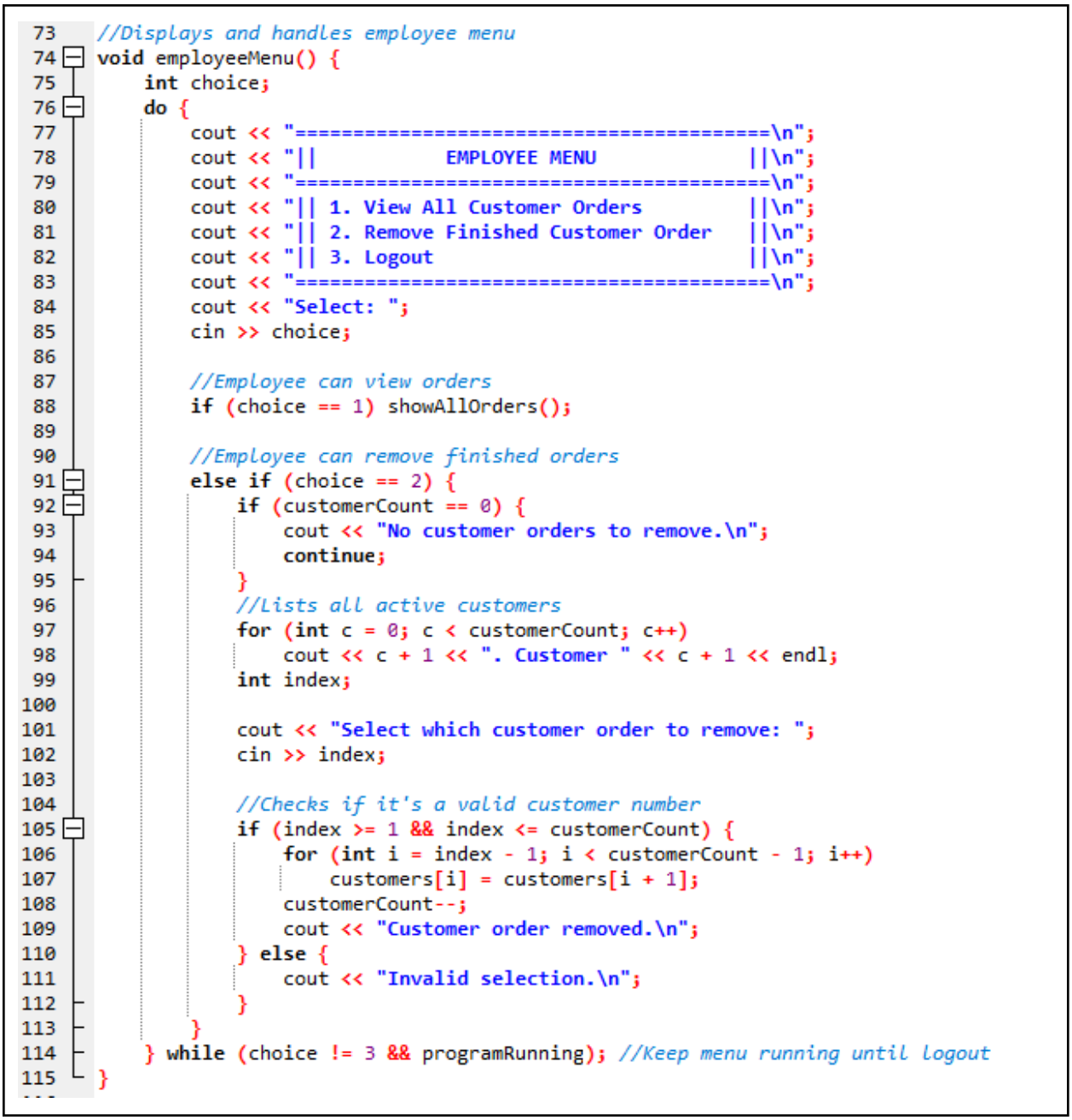
Figure 10. Initial Setups

This figure represents the initial setup of the program, including the menu items, stock counts and data structures for customer orders. Ensuring that the system is ready to accept customer inputs and tracks all transactions.

```
35  //Displays all active customer orders
36  void showAllOrders() {
37      if (customerCount == 0) {
38      cout << "\n=======================\n";
39      cout << "||   No Active Orders  ||\n";
40      cout << "=======================\n";
41      return;
42      }
43
44      cout << "\n==============================\n";
45      cout << "||       ACTIVE ORDERS       ||\n";
46      cout << "==============================\n";
47
48      //Loops through all customers
49      for (int c = 0; c < customerCount; c++) {
50      float total = 0;
51      int totalTime = 0;
52
53      cout << "Customer " << c + 1 << ":\n";
54
55      //Loops through each item of the customer
56      for (int i = 0; i < customers[c].itemCount; i++) {
57          float linePrice = customers[c].items[i].price * customers[c].discount;
58          cout << " "  << customers[c].items[i].product
59               << " x" << customers[c].items[i].qty
60               << " = " << linePrice << " \n";
61
62          total += linePrice;
63          totalTime += customers[c].items[i].prepTime;
64      }
65      //Displays total and estimated preparation time
66      cout << "----------------------------\n";
67      cout << "   Total: " << total << "              \n";
68      cout << "   Estimated Time: " << totalTime << " min   \n";
69      cout << "==============================\n";
70      }
71  }
72
```

Figure 11. Show all orders

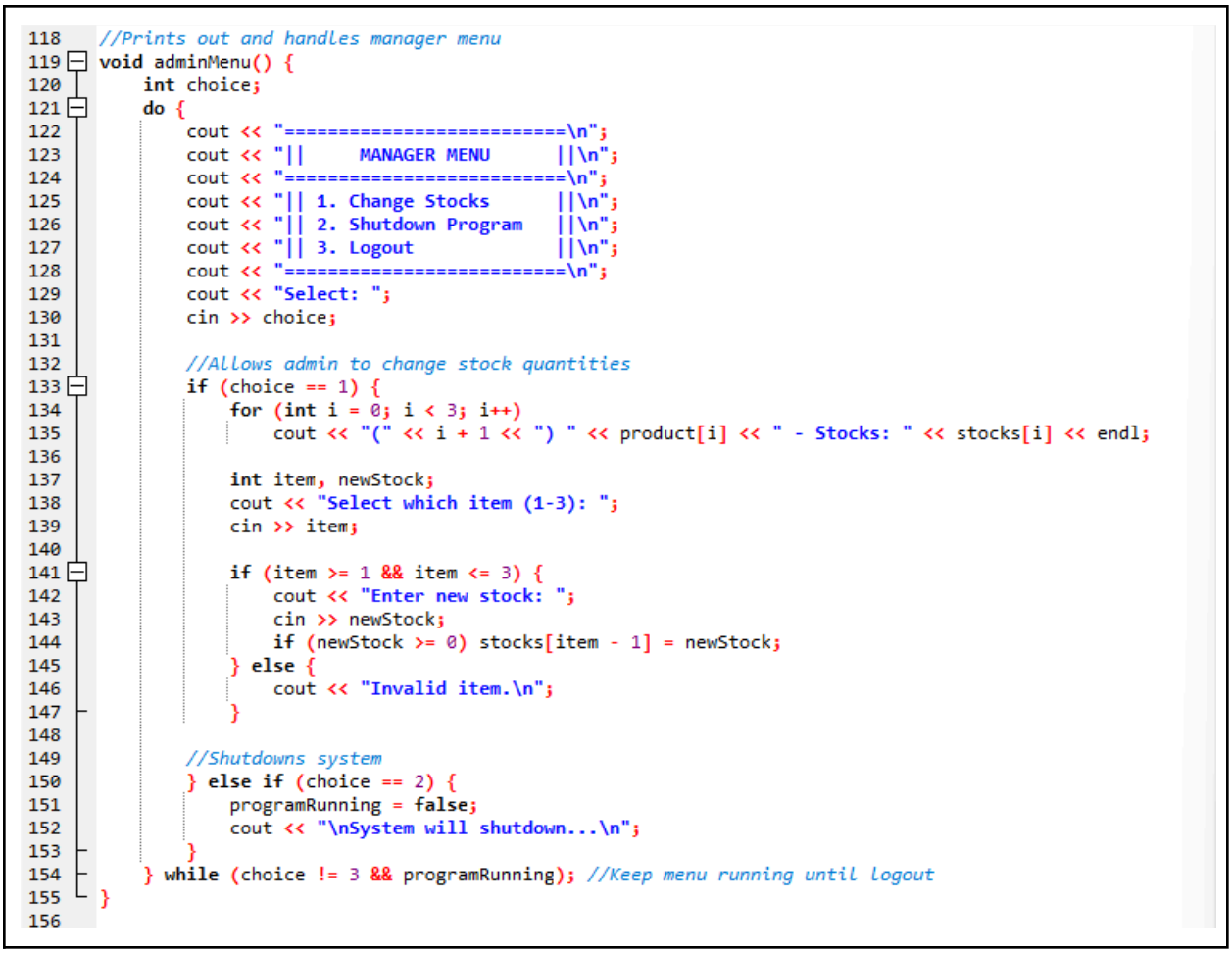
This figure demonstrates the display of all active customer orders in the system. Showing each customer, their items, quantity, total price and estimated preparation time.



```
73  //Displays and handles employee menu
74  void employeeMenu() {
75      int choice;
76      do {
77          cout << "========================================\n";
78          cout << "||           EMPLOYEE MENU             ||\n";
79          cout << "========================================\n";
80          cout << "|| 1. View All Customer Orders         ||\n";
81          cout << "|| 2. Remove Finished Customer Order   ||\n";
82          cout << "|| 3. Logout                           ||\n";
83          cout << "========================================\n";
84          cout << "Select: ";
85          cin >> choice;
86
87          //Employee can view orders
88          if (choice == 1) showAllOrders();
89
90          //Employee can remove finished orders
91          else if (choice == 2) {
92              if (customerCount == 0) {
93                  cout << "No customer orders to remove.\n";
94                  continue;
95              }
96              //Lists all active customers
97              for (int c = 0; c < customerCount; c++)
98                  cout << c + 1 << ". Customer " << c + 1 << endl;
99              int index;
100
101             cout << "Select which customer order to remove: ";
102             cin >> index;
103
104             //Checks if it's a valid customer number
105             if (index >= 1 && index <= customerCount) {
106                 for (int i = index - 1; i < customerCount - 1; i++)
107                     customers[i] = customers[i + 1];
108                 customerCount--;
109                 cout << "Customer order removed.\n";
110             } else {
111                 cout << "Invalid selection.\n";
112             }
113         }
114     } while (choice != 3 && programRunning); //Keep menu running until logout
115 }
```

Figure 12. Employee Menu

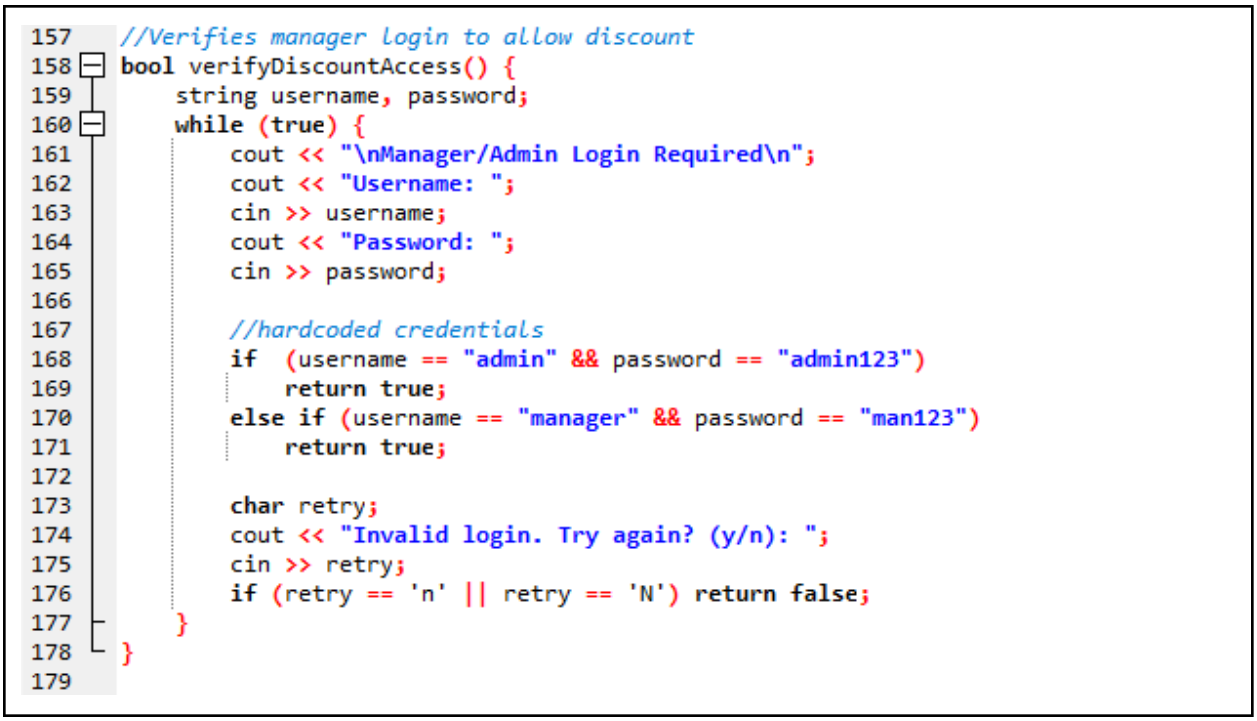
Figure 6 shows the menu available to employees, including viewing active orders and removing completed orders. Making it possible to manage orders without needing for manager login.



```
118  //Prints out and handles manager menu
119  void adminMenu() {
120      int choice;
121      do {
122          cout << "==========================\n";
123          cout << "||      MANAGER MENU      ||\n";
124          cout << "==========================\n";
125          cout << "|| 1. Change Stocks      ||\n";
126          cout << "|| 2. Shutdown Program   ||\n";
127          cout << "|| 3. Logout             ||\n";
128          cout << "==========================\n";
129          cout << "Select: ";
130          cin >> choice;
131
132          //Allows admin to change stock quantities
133          if (choice == 1) {
134              for (int i = 0; i < 3; i++)
135                  cout << "(" << i + 1 << ") " << product[i] << " - Stocks: " << stocks[i] << endl;
136
137              int item, newStock;
138              cout << "Select which item (1-3): ";
139              cin >> item;
140
141              if (item >= 1 && item <= 3) {
142                  cout << "Enter new stock: ";
143                  cin >> newStock;
144                  if (newStock >= 0) stocks[item - 1] = newStock;
145              } else {
146                  cout << "Invalid item.\n";
147              }
148
149          //Shutdowns system
150          } else if (choice == 2) {
151              programRunning = false;
152              cout << "\nSystem will shutdown...\n";
153          }
154      } while (choice != 3 && programRunning); //Keep menu running until logout
155  }
156
```

Figure 13. Manager Menu

This figure shows the manager menu, which allows stock management, program shutdown and logout. Providing a higher level of control over the system compared to the employee menu.



```
157  //Verifies manager login to allow discount
158  bool verifyDiscountAccess() {
159      string username, password;
160      while (true) {
161          cout << "\nManager/Admin Login Required\n";
162          cout << "Username: ";
163          cin >> username;
164          cout << "Password: ";
165          cin >> password;
166
167          //hardcoded credentials
168          if  (username == "admin" && password == "admin123")
169              return true;
170          else if (username == "manager" && password == "man123")
171              return true;
172
173          char retry;
174          cout << "Invalid login. Try again? (y/n): ";
175          cin >> retry;
176          if (retry == 'n' || retry == 'N') return false;
177      }
178  }
179
```

Figure 14. Discount Access

Figure 8 shows the discount verification interface for managers or admins. Allows authorized personnel to apply discounts to customer orders, ensuring that discounts are validated.

```
180  //Staff login menu and handles login for both manager and employee
181  void staffLoginFlow() {
182      int sel;
183          cout << "==========================\n";
184          cout << "||       STAFF LOGIN      ||\n";
185          cout << "==========================\n";
186          cout << "|| 1. Manager Login       ||\n";
187          cout << "|| 2. Employee Login      ||\n";
188          cout << "|| 3. Cancel              ||\n";
189          cout << "========================\n";
190          cout << "Select: ";
191          cin >> sel;
192
193      //Checks credentials for manager login
194      if (sel == 1) {
195          string u, p;
196          cout << "Username: ";
197          cin >> u;
198          cout << "Password: ";
199          cin >> p;
200          if (u == "manager" && p == "man123") adminMenu();
201          else cout << "Invalid admin credentials.\n";
202      }
203      //Checks credentials for employee login
204      else if (sel == 2) {
205          string u, p;
206          cout << "Username: ";
207          cin >> u;
208          cout << "Password: ";
209          cin >> p;
210          if (u == "employee" && p == "emp123") employeeMenu();
211          else cout << "Invalid employee credentials.\n";
212      }
213
214
215      if (!programRunning) return; //Stop if system is shutting down
216  }
217
```

Figure 15. Staff Login Menu

This figure shows the login options for the staff. Ensuring that only authorized users can access the restricted functions.

```
//Customer order menu
void customerOrder() {
    CustomerOrder currentCustomer;
    currentCustomer.itemCount = 0;
    currentCustomer.discount = 1.0f;

    int item_number = -1;
    int quantity;
    string addMore;

        while (programRunning) {
        cout << "\n================ MENU ==============\n";

        // Find longest product name for aligned menu
        int maxLen = 0;
        for (int i = 0; i < 3; i++)
            if ((int)product[i].length() > maxLen)
                maxLen = product[i].length();

        // Print products with aligned stocks counts
        for (int i = 0; i < 3; i++) {
            cout << "|| (" << i + 1 << ") "
                 << left << setw(maxLen) << product[i]
                 << " | Stocks: " << setw(2) << stocks[i] << " ||" << endl;
        }

        // Bottom border
        cout << string(6 + maxLen + 16, '=') << endl;

        cout << "\nEnter item number (or 0 for Staff Login): ";
        cin >> item_number;


        //Runs Staff Login Menu when 0 is entered
        if (item_number == 0) {
            staffLoginFlow();

            if (!programRunning) return;
            continue;
        }

        if (item_number < 1 || item_number > 3) {
            cout << "Invalid item number. Try again.\n";
            continue;
        }

        cout << "Enter quantity (or 0 to cancel): ";
        cin >> quantity;
        if (quantity <= 0) {
            cout << "returning to main menu. \n";
            continue;
        }
        if (quantity > stocks[item_number - 1]) {
            cout << "Not enough stock. Available: " << stocks[item_number - 1] << endl;
            continue;
        }


        //Updates stock and add item to customer order
        stocks[item_number - 1] -= quantity;
        currentCustomer.items[currentCustomer.itemCount].product = product[item_number - 1];
        currentCustomer.items[currentCustomer.itemCount].qty = quantity;
        currentCustomer.items[currentCustomer.itemCount].price = price[item_number - 1] * quantity;
        currentCustomer.items[currentCustomer.itemCount].prepTime = avg_time[item_number - 1] * quantity;
        currentCustomer.itemCount++;

        cout << "Add another item? (y/n): ";
        cin >> addMore;
        if (addMore != "y" && addMore != "Y") break;
    }

    if (!programRunning) return;
```

Figure 16. Customer Order Menu

Figure 10 represents the menu customers interact with when placing orders. It shows the product names, prices, stock counts.

```
293         //Checks if the user wants to apply discount
294         string hasDiscount;
295         cout << "\nDo you have a discount? (y/n): ";
296         cin >> hasDiscount;
297         if (hasDiscount == "y" || hasDiscount == "Y") {
298             int discType;
299             cout << "\nSelect discount type:\n";
300             cout << "1. PWD / Senior (20%)\n";
301             cout << "2. Coupon (10%)\n";
302             cin >> discType;
303 
304             if (verifyDiscountAccess()) {
305                 if (discType == 1) currentCustomer.discount = 0.80f;
306                 else if (discType == 2) currentCustomer.discount = 0.90f;
307             } else {
308                 cout << "Discount not authorized.\n";
309             }
310         }
311 
312         //Saves customer order
313         customers[customerCount] = currentCustomer;
314         customerCount++;
315 
316         cout << "\nCustomer order saved.\n";
317         //Prints out all active orders
318         showAllOrders();
319 }
```

Figure 17. Discount Checker

This figure displays the process of verifying and applying a discount to a customer's order. Ensuring that the discount is correctly applied and calculated in the total price.

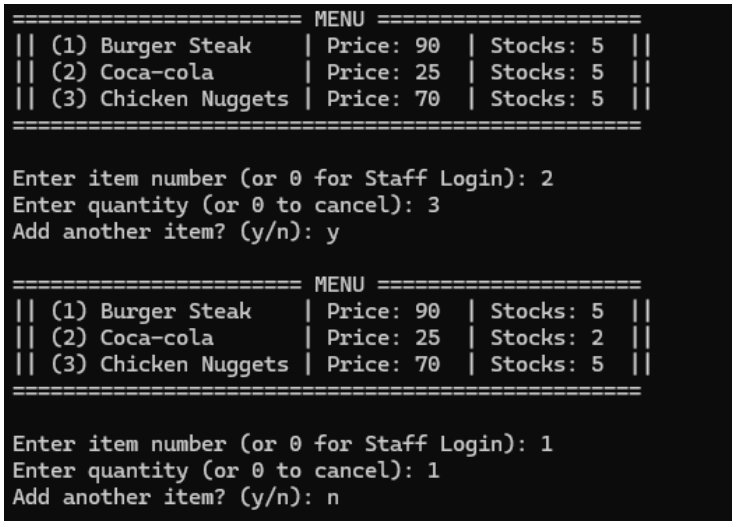
```
321 int main() {
322     while (programRunning) {
323         customerOrder();
324     }
325 
326     cout << "\nSystem stopped by Manager.\n";
327     return 0;
328 }
```

Figure 18. Program Flow

This figure shows the overall program flow, calling the customerOrder functions while the programRunninng boolean is set to true, and if set to false it will print out "System stopped by Manager."

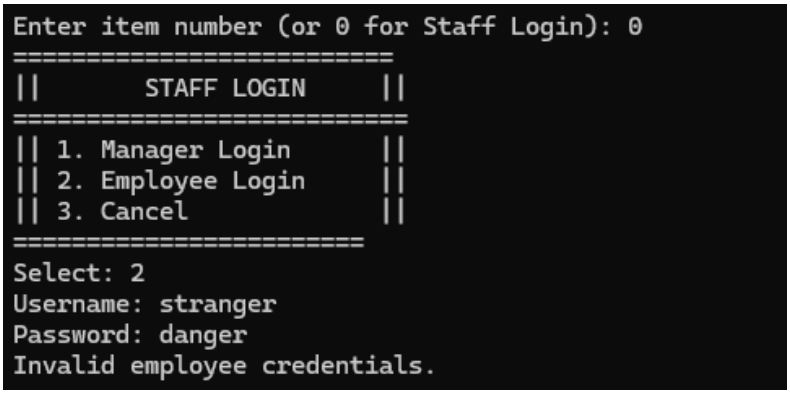
Results and Discussion

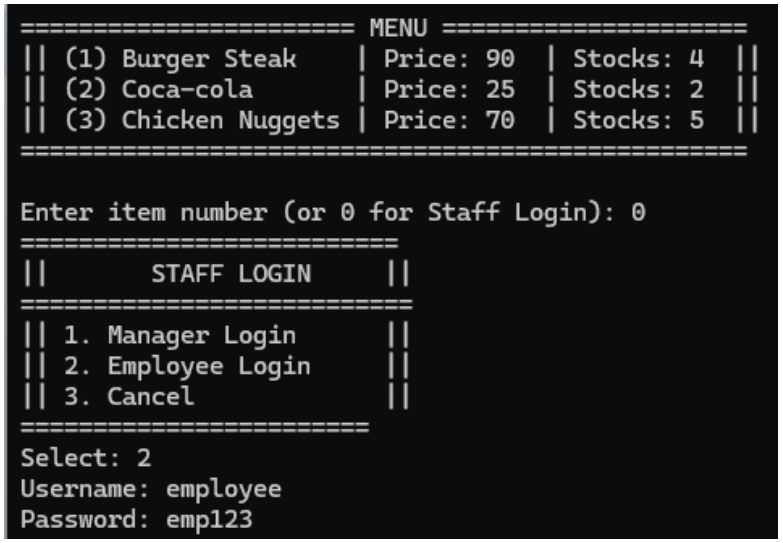
This part aims to show the results and discussion of the output of the system.



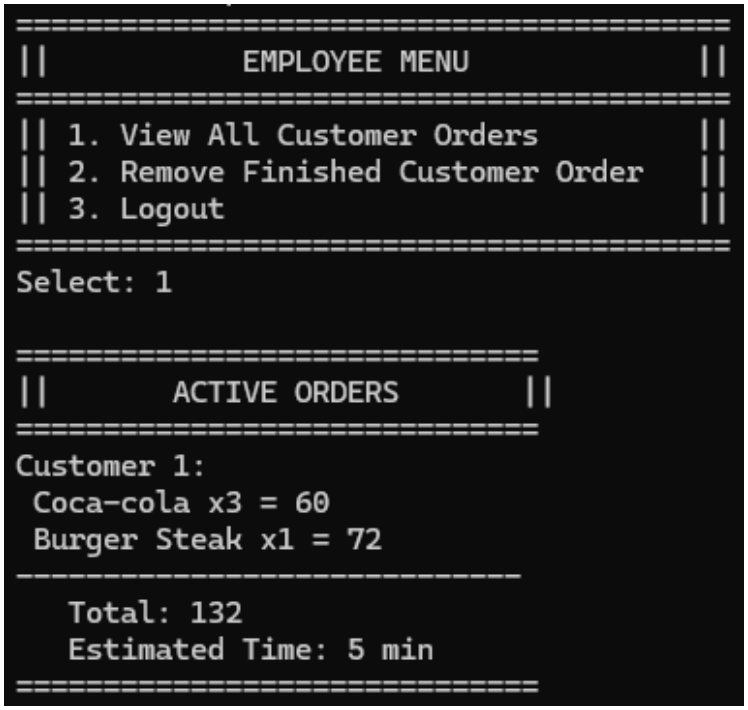
This output presents the customer menu interface, demonstrating how customers select items, view prices and stock, and input their designed quantities.

The output focuses on the discount part of the system, where a manager verifies the discount to ensure it is valid and properly applied to the customer's order. It also prints out the order summary with the items selected, quantity, price with or without discount, total and estimated time for preparation.

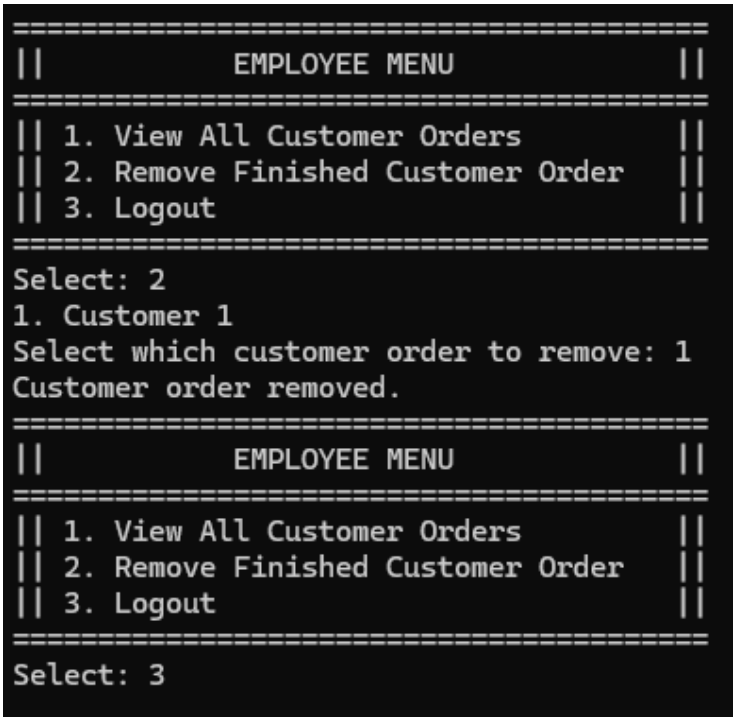




This screenshot focuses on the staff login for employees, ensuring that the credentials entered are correct before granting access to the staff menu



The output demonstrates one of the functions of the employee menu, viewing all active customer orders.



The output demonstrates the second function of the employee menu, removing finished customer orders.

```
====================== MENU ====================
|| (1) Burger Steak    | Price: 90  | Stocks: 4  ||
|| (2) Coca-cola       | Price: 25  | Stocks: 2  ||
|| (3) Chicken Nuggets | Price: 70  | Stocks: 5  ||
================================================

Enter item number (or 0 for Staff Login): 0
=========================
||       STAFF LOGIN     ||
=========================
|| 1. Manager Login      ||
|| 2. Employee Login     ||
|| 3. Cancel             ||
=========================
Select: 1
Username: manager
Password: man123
```

This output demonstrates the manager login.

```
=========================
||     MANAGER MENU      ||
=========================
|| 1. Change Stocks      ||
|| 2. Shutdown Program   ||
|| 3. Logout             ||
=========================
Select: 1
(1) Burger Steak - Stocks: 4
(2) Coca-cola - Stocks: 2
(3) Chicken Nuggets - Stocks: 5
Select which item (1-3): 2
Enter new stock: 10
=========================
||     MANAGER MENU      ||
=========================
|| 1. Change Stocks      ||
|| 2. Shutdown Program   ||
|| 3. Logout             ||
=========================
Select: 2

System will shutdown...

System stopped by Admin.
```

The output demonstrates the functions of the admin, changing the stocks and shutting the system down.

Conclusion

The fast food ordering system shows how programming concepts by using arrays, loops can be put to solve real life problems in the food service industry. Based on the result and discussion, the system performed all its main functions effectively including customers ordering, discount verification, staff log in, and manager controls. It also provides an accurate total price, discounts, and estimated waiting time, making transactions faster and more organized.

It's recommended to enhance the system by improving its interface and adding more features for real-world application. Future versions may include online ordering through mobile or web platforms, real-time sales and stock monitoring, and automated receipt printing for better customer convenience.

References

Pangilinan, B. (2025, May 28). Digital transformation of the order-taking process in fast food restaurants in the Philippines. Digital Archives @ UP Diliman. https://digitalarchives.upd.edu.ph/item/62984/1091?fbclid=IwY2xjawOBi05leHRuA2FlbQIxMABicmlkETFwOFdtQWJ5cHJvN0g1ME5Ec3J0YwZhcHBfaWQQMjIyMDM5MTc4ODIwMDg5MgABHp5N7Uny7zI9_rPTm5IQa05roj18UJElHd_xTa23e0Ap9v4sA_uEuI1DEeKv_aem_WK3yjthc4Ueorb5zsFUKsA

https://www.w3schools.com/cpp/cpp_variables.asp

W3Schools.com. https://www.w3schools.com/cpp/default.asp