Pritam Gurung

Project Assignment

Content Page

Table of Contents

1. Screen Designs

1.1 Screen Designs of 1st Program Menu Program.

For my first program, at the top of the UI shown in Figure 1, it displays the title "Menu Creation System", immediately informing the user of the application's primary function, which is to create and manage the restaurant's menu.

It's been designed to provide a smooth experience for the user while managing a restaurant's menu. The design emphasises ease of navigation and efficient interaction with the program.

1. *The "Display Menu" option is intended to show the user the current list of menu items.*
2. *The "Add Menu Items" allows the user to enter new items into the system, thereby expanding the menu.*
3. *The "Load Menu from File" lets the user populate the menu with a list of items from an external file, enabling quick updates or changes.*
4. *The "Save Menu to File" provides the functionality to save the current state of the menu to a file for later retrieval or record-keeping.*
5. *The "Exist" option gives the user a clear path to close the application once they have completed their task.*

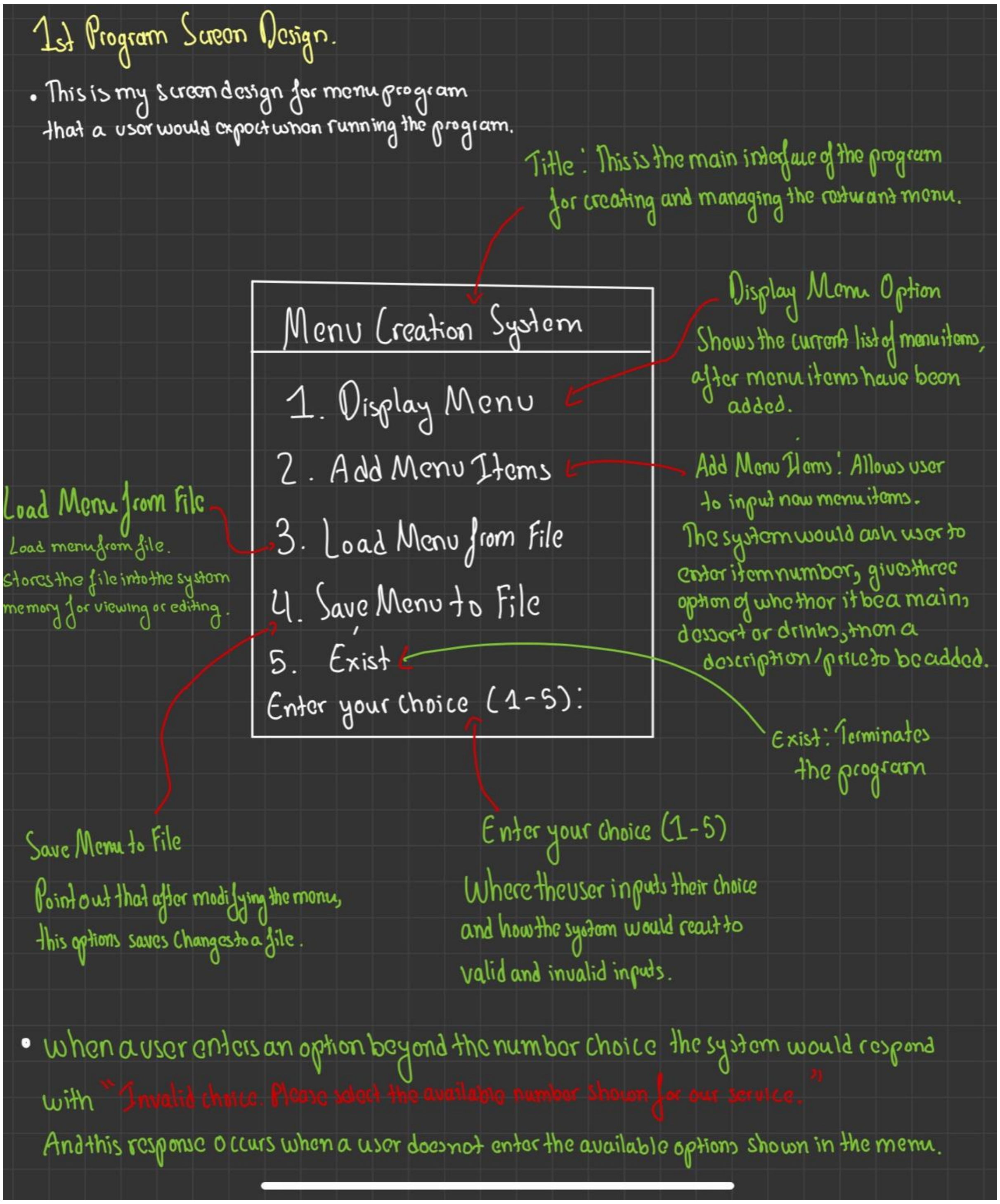
Input Prompt: *Beneath the list of options, the prompt "Enter your choice (1-5):" requests input from the user, thus guiding them on how to select one of the listed functions shown in Figure 1.*

Feedback Mechanism: *The UI design incorporates a feedback mechanism where if the user selects that is not listed (an invalid choice), the system responds with a clear instructive message prompting them to select a valid option within the provided range of (1-5) shown in Figure 1.*

Design Aesthetics: *In my design, the use of contrasting colours (white, green, and red) for text ensures readability and helps to guide the user's attention to different sections of the UI effectively. The key areas of UI are annotated with red arrows and descriptions explaining the purpose of each part of the interface, serving it as a guide for new users interacting with the menu shown in Figure 1.*

Overall, the UI design for the Menu Creation System is practical, aesthetically pleasing, and user-centric which ensures that the users of varying levels of technical skills can navigate around the system and utilise the application easily and effectively.

Figure 1: Menu Creation System Design



1st Program Screen Design.

- This is my screen design for menu program that a user would expect when running the program.

```
Menu Creation System

1. Display Menu
2. Add Menu Items
3. Load Menu from File
4. Save Menu to File
5. Exist
Enter your choice (1-5):
```

Title: This is the main interface of the program for creating and managing the resturant menu.

Display Menu Option
Shows the current list of menu items, after menu items have been added.

Add Menu Items: Allows user to input new menu items. The system would ask user to enter item number, gives three option of whether it be a main, dessert or drinks, then a description/price to be added.

Load Menu from File
Load menu from file. Stores the file into the system memory for viewing or editing.

Exist: Terminates the program

Save Menu to File
Point out that after modifying the menu, this options saves changes to a file.

Enter your Choice (1-5)
Where the user inputs their choice and how the system would react to valid and invalid inputs.

- When a user enters an option beyond the number choice the system would respond with "Invalid choice. Please select the available number shown for our service."
And this response occurs when a user does not enter the available options shown in the menu.

1.2 Screen Design for 2nd Program Billing Checkout

The second screen design is for a billing checkout program that appears as a suite of applications used in a chain of restaurants shown in Figure 2 below.

At the top, there is a title of Menu Items, which sets the context that this is one part of a larger system. It is specifically identified as a billing checkout program, which suggests its use at the point of sale within the restaurant chain.

Menu Display: *A section titled "Menu Items (1-20 menu items)" indicates that the system can display up to 20 menu items at a time. Example menu items are listed sequentially, each with a number, name, and price, such as "1. Main Course – Chicken Pie - $15.99."*

User Prompt for Selection: *There is an instruction for the user: "Please enter the item number and quantities for the order (0 to finish):", which guides the user on how to place an order. The user is expected to input the item number from the menu and specify the quantity for each item they wish to place an order for.*

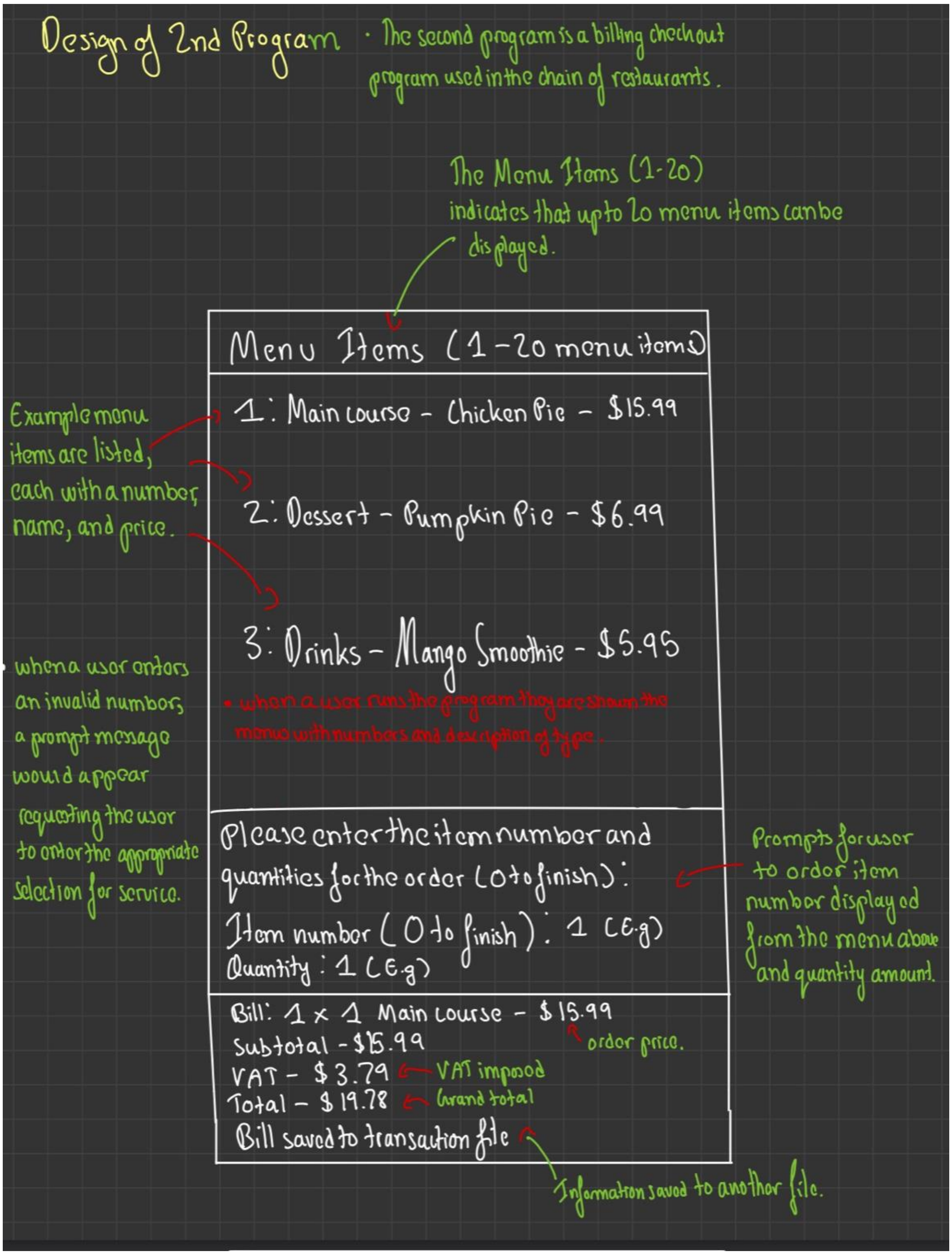
Feedback on Invalid Input: *When a user enters an invalid number, a prompt message from the system is designed to appear, instructing the user to enter the appropriate selection for service. This is crucial for error handling and ensuring the user stays on the correct path during interaction.*

Billing and Calculation Display: *Once the user enters their selections, the system calculates and displays a bill with items, their quantities, prices, a subtotal, VAT imposed, and the grand total. As shown below in Figure 2, an example calculation is displayed showing these calculations, suggesting a real-time update as items are entered.*

Saving the Transactions: *From the design, after finalising the order, there is a note that the bill is saved to a transaction file, which means the system's capability to keep records for each transaction. This is essential for both accounting purposes and potentially for analysing sales data which is perfect for restaurant chains to utilise.*

Overall, the screen design focuses on functionality, ensuring users have a straightforward way to interact with the program during the checkout process as it provides a clear instruction, immediate feedback for errors, and transparent calculations for the customer's order, cumulating in the storage for transaction data which is important for restaurant chains to take utilise for business operations.

Figure 2: Screen Design For Billing Checkout



Design of 2nd Program · The second program is a billing checkout program used in the chain of restaurants.

The Menu Items (1-20) indicates that up to 20 menu items can be displayed.

```
Menu Items (1-20 menu items)

1: Main course - Chicken Pie - $15.99

2: Dessert - Pumpkin Pie - $6.99

3: Drinks - Mango Smoothie - $5.95
```

• when a user runs the program they are shown the menu with numbers and description of type.

Example menu items are listed, each with a number, name, and price.

• when a user enters an invalid number, a prompt message would appear requesting the user to enter the appropriate selection for service.

```
Please enter the item number and quantities for the order (0 to finish):
Item number (0 to finish): 1 (E.g)
Quantity: 1 (E.g)
```

Prompts for user to order item number displayed from the menu above and quantity amount.

```
Bill: 1 x 1 Main Course - $15.99
Subtotal - $15.99
VAT - $3.79
Total - $19.78
Bill saved to transaction file
```

order price.

VAT imposed

Grand total

Information saved to another file.

1.3 Screen Design for Transaction Stats

The third screen design caters to a restaurant owner's need for quick access to daily sales statistics.

Purpose and Functionality: *The design is part of a larger system, specifically a third program that allows a restaurant owner to effortlessly monitor daily sales performance. The focus is on delivering vital statistics in a streamlined and presentable format which can be seen in Figure 3.*

Layouts and Content: *The interface presents the "Daily Statistics" as a header shown in Figure 3, which sets the context for the information displayed. Underneath the header, there are two key pieces of data presented:*

Number of orders for each time: *This provides detailed insights into individual menu item performance, showing the quantity sold. For example, "Item 1: 1 order" shown in Figure 3.*

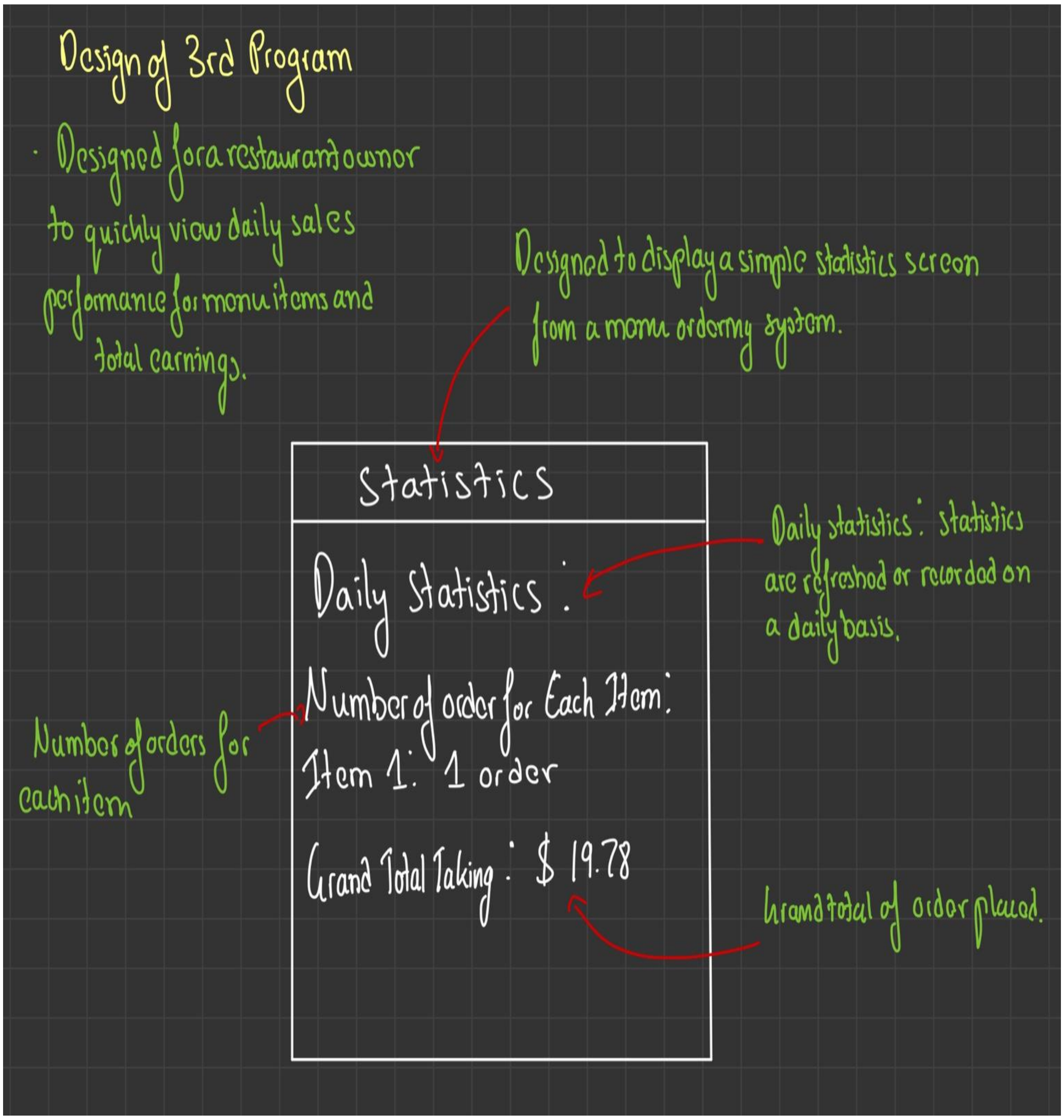
Grand Total Takings: *This is a summary statistic displaying the total revenue generated from the day's sales, e.g., "Grand Total Taking: $19.78" shown in Figure 3.*

Design Aesthetics: *The design follows a clean and organised layout, which is crucial for quick data interpretation and important figures can be highlighted to show significance and to reduce the chance of duplicated data.*

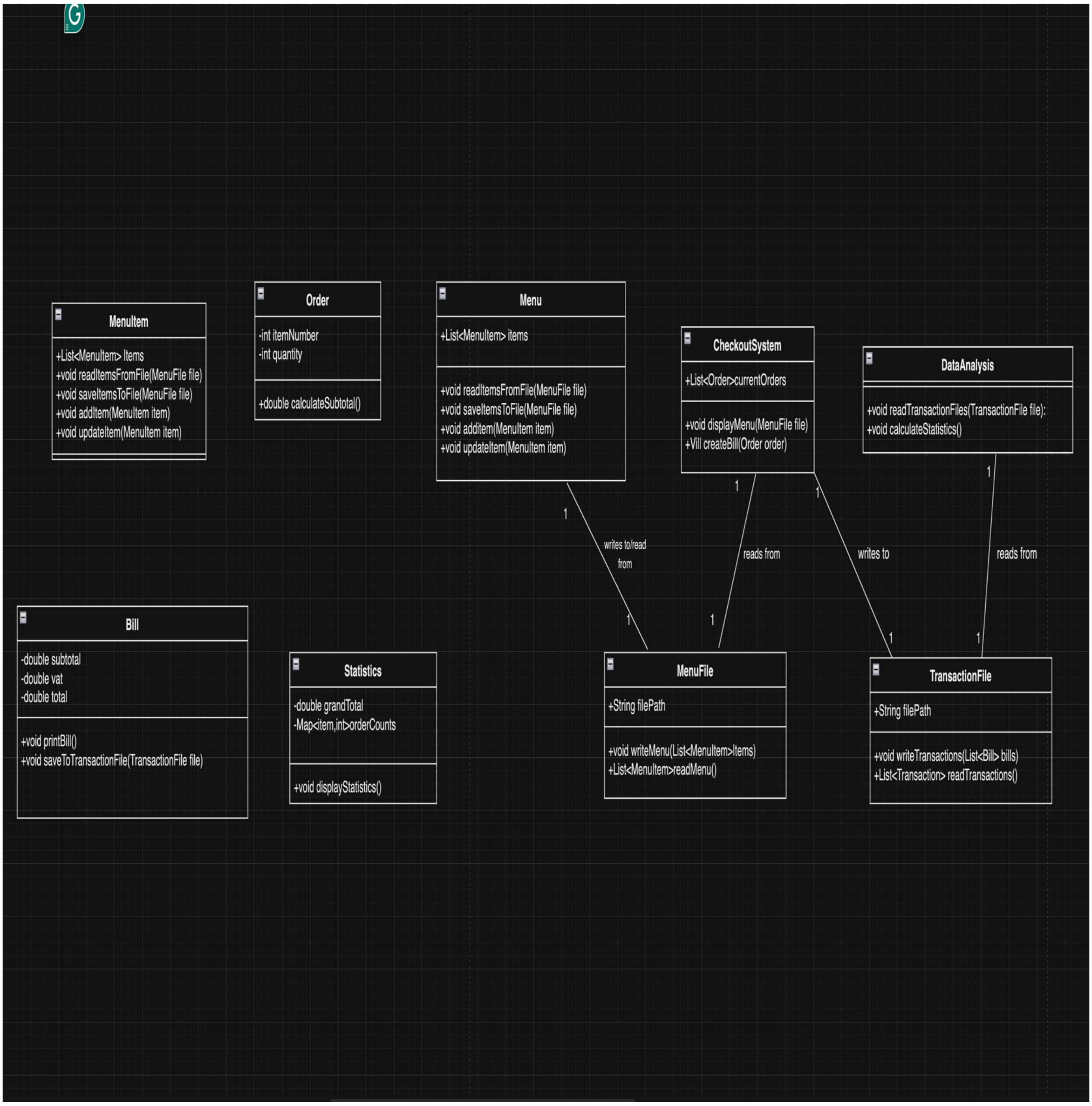
Utility and Efficiency: *By focusing solely on the information, the design ensures efficiency of use-restaurant owners can quickly scan for the day's performance without being bogged down by less relevant data. The design seems to be a part of a daily routine, where updated figures would be available each day, providing the owner with ongoing and actionable insights so an informed decision can be made which can reduce errors.*

Overall, the UI design of the third program aims to present critical sales data clearly and concisely, making it an effective tool for business owners to track daily operations and make informed decisions.

Figure 3 Transaction Screen Design


Design of 3rd Program
· Designed for a restaurant owner to quickly view daily sales performance for menu items and total earnings.
Designed to display a simple statistics screen from a menu ordering system.
Statistics
Daily Statistics :
Number of order for Each Item:
Item 1: 1 order
Grand Total Taking : $ 19.78
Daily statistics: statistics are refreshed or recorded on a daily basis.
Number of orders for each item
Grand total of order placed.

2. Class Diagram



MenuItem
+List<MenuItem> Items
+void readItemsFromFile(MenuFile file)
+void saveItemsToFile(MenuFile file)
+void addItem(MenuItem item)
+void updateItem(MenuItem item)
Order
-int itemNumber
-int quantity
+double calculateSubtotal()
Menu
+List<MenuItem> items
+void readItemsFromFile(MenuFile file)
+void saveItemsToFile(MenuFile file)
+void additem(MenuItem item)
+void updateItem(MenuItem item)
CheckoutSystem
+List<Order>currentOrders
+void displayMenu(MenuFile file)
+Vill createBill(Order order)
DataAnalysis
+void readTransactionFiles(TransactionFile file):
+void calculateStatistics()
1
writes to/read from
1
1
reads from
1
1
writes to
1
1
reads from
1
Bill
-double subtotal
-double vat
-double total
+void printBill()
+void saveToTransactionFile(TransactionFile file)
Statistics
-double grandTotal
-Map<item,int>orderCounts
+void displayStatistics()
MenuFile
+String filePath
+void writeMenu(List<MenuItem>Items)
+List<MenuItem>readMenu()
TransactionFile
+String filePath
+void writeTransactions(List<Bill> bills)
+List<Transaction> readTransactions()

3. Initial Test Plan

3. 1 Test Plan for Program 1: Menu Creation

Objective*: Verify that the program can create, display, and save menu items to a file correctly.*

Features to be Tested:

1. *Menu item creation*
2. *Displaying current menu items.*
3. *Saving menu items to a file.*

Test Data:

- *Predefined menu items data to input.*

Test Cases:

1. *Add New Menu Items:*
- *Input valid menu item data and check if it is added to the vector correctly.*
- *Input invalid data (e.g., negative price) and verify proper handling.*
2. *Display Menu*
- *With no items in the menu, check if the appropriate message is shown.*
- *With items added, check if all items are displayed properly.*
3. *Save to File:*
- *Save current menu items to the file and verify the file contains the correct data.*
- *Test saving when a file cannot be created (e.g., invalid file path).*

Test Procedures:

1. *Run the application and follow the prompts to add, display, and save menu items.*
2. *Use both valid and invalid inputs to check system responses.*

Documentation:

- *Record the inputs given and the corresponding outputs from the program.*
- *Take note of any deviations from the expected behaviour.*

3.2 Test Plan for Program 2: Billing Checkout

Objectives: *Ensure the billing process calculates totals correctly and saves transactions efficiently.*

Features to be Tested:

1. *Reading menu items from a file.*
2. *Adding orders and calculating subtotal, VAT, and total.*
3. *Saving the bill to a transaction file.*

Test Data:

- *Predefined orders and corresponding menu items file.*

Test Cases:

1. *Load Menu Items:*
- *Verify that the menu items are correctly loaded from a file.*
2. *Process Order:*
- *Enter valid order items and quantities and verify the bill totals.*
- *Test with invalid inputs, such as invalid item numbers and negative quantities.*
3. *Save Bill:*
- *Check that the transaction file is updated with the new bill.*
- *Attempt to save to a read-only and verify error handling.*

Test Procedures:

1. *Follow the sequence of ordering items as a customer would.*
2. *Test with both expected and unexpected user behaviour.*

Documentation:

- *Note all order inputs, system outputs, and any errors or unexpected behaviours.*

3.3 Test Plan for Program 3: Daily Sales Statistics

Objective: *Confirm that the program accurately collates daily sales data and provides a summary.*

Features to be Tested:

1. *Reading transaction files and accumulating sales data.*
2. *Displaying daily statistics and grand totals.*

Test Data:

- *Predefined transaction files representing a day's sales.*

Test Cases:

1. *Load and Aggregate Transactions:*
- *Confirm that the data from transaction files are read and summed accurately.*
2. *Display Statistics:*
- *Ensure that the summary statistics displayed are accurate and match the aggregated data.*

Test Procedures:

1. *Stimulate the end-of-day process where transaction files are read.*
2. *Check that the statistics presented are a correct summation of the day's transactions.*

Documentation:

-Document the expected and actual statistics displayed, noting any discrepancies that may appear from the system.

Overall: *For all the test cases, ensure to record the test case description, expected result, actual result, and whether the test passed or failed. Any error should be logged in a defect tracking system or documented for further look and reach for solutions.*

4. Coding

4.1 The first program

```
/**********************************************************************
 * menu.cpp
 * Program for head office to create and save a restaurant menu
 * Pritam Gurung
 * Version 1.0
 **********************************************************************/

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <limits>

struct MenuItem {
    int itemNo;
    std::string category;
    std::string description;
    double price;
};

void displayMenu(const std::vector<MenuItem>& menu);
void addMenuItem(std::vector<MenuItem>& menu);
void loadMenuFromFile(std::vector<MenuItem>& menu, const
std::string& filename);
void saveMenuToFile(const std::vector<MenuItem>& menu, const
std::string& filename);
```

```cpp
int main() {
    std::vector<MenuItem> menu;
    std::string filename = "menu.txt";
    char choice;

    do {
        std::cout << "Menu Creation System\n";
        std::cout << "1. Display Menu\n";
        std::cout << "2. Add Menu Item\n";
        std::cout << "3. Load Menu from File\n";
        std::cout << "4. Save Menu to File\n";
        std::cout << "5. Exit\n";
        std::cout << "Enter your choice (1-5): ";
        std::cin >> choice;
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

        switch (choice) {
            case '1':
                displayMenu(menu);
                break;
            case '2':
                addMenuItem(menu);
                break;
            case '3':
                loadMenuFromFile(menu, filename);
                break;
            case '4':
                saveMenuToFile(menu, filename);
                break;
            case '5':
```

```
            return 0;
        default:
            std::cout << "Invalid choice. Please select the available
number shown for our service. (1-5).\n";
        }
    } while (true);
}

void displayMenu(const std::vector<MenuItem>& menu) {
    std::cout << "\nCurrent Menu:\n";
    for (const auto& item : menu) {
        std::cout << "Item " << item.itemNo << ": "
                  << item.category << " - "
                  << item.description << " - $"
                  << item.price << "\n";
    }
    std::cout << std::endl;
}

void addMenuItem(std::vector<MenuItem>& menu) {
    MenuItem newItem;
    std::cout << "Enter item number: ";
    std::cin >> newItem.itemNo;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::cout << "Enter category (Main Course, Dessert, Drinks): ";
    getline(std::cin, newItem.category);
    std::cout << "Enter item description: ";
    getline(std::cin, newItem.description);
    std::cout << "Enter item price: ";
    std::cin >> newItem.price;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

```cpp
    menu.push_back(newItem);
    std::cout << "Menu item added.\n\n";
}

void loadMenuFromFile(std::vector<MenuItem>& menu, const
std::string& filename) {
    std::ifstream fileIn(filename);
    if (!fileIn) {
        std::cout << "Could not open file for reading.\n";
        return;
    }
    menu.clear();
    std::string line;
    while (getline(fileIn, line)) {
        std::stringstream ss(line);
        MenuItem item;
        std::getline(ss, line, ':');
        item.itemNo = std::stoi(line);
        std::getline(ss, item.category, ':');
        std::getline(ss, item.description, ':');
        std::getline(ss, line);
        item.price = std::stod(line);
        menu.push_back(item);
    }
    fileIn.close();
    std::cout << "Menu loaded from file.\n\n";
}

void saveMenuToFile(const std::vector<MenuItem>& menu, const
std::string& filename) {
    std::ofstream fileOut(filename);
    if (!fileOut) {
        std::cout << "Could not open file for writing.\n";
        return;
    }
    for (const auto& item : menu) {
        fileOut << item.itemNo << ":"
                << item.category << ":"
                << item.description << ":"
                << item.price << "\n";
    }
    fileOut.close();
    std::cout << "Menu saved to file.\n\n";
}
```

4.2 The second program

```
/**********************************************************************
 * billingcheckout.cpp
 * Billing checkout program for restaurant chain
 * Pritam Gurung
 * Version 2.0
 **********************************************************************/

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <iomanip>
#include <sstream>
```

```cpp
struct MenuItem {
    int itemNo;
    std::string name;
    std::string description;
    double price;
};


void loadMenu(std::vector<MenuItem>& menu, const std::string&
filename);
void displayMenu(const std::vector<MenuItem>& menu);
void processOrder(const std::vector<MenuItem>& menu, const
std::string& transactionFile);
double calculateVAT(double amount);
void saveBillToFile(const std::string& transactionFile, const
std::string& billText, double total);

int main() {
    std::vector<MenuItem> menu;
    std::string menuFilename = "menu.txt";
    std::string transactionFilename = "transactions.txt";

    // Load menu from file
    loadMenu(menu, menuFilename);



    processOrder(menu, transactionFilename);

    return 0;
}
```

```
void loadMenu(std::vector<MenuItem>& menu, const std::string&
filename) {
    std::ifstream fileIn(filename);
    if (!fileIn) {
        std::cerr << "Error opening menu file.\n";
        return;
    }

    MenuItem item;
    std::string line;
    while (getline(fileIn, line)) {
        std::stringstream ss(line);
        std::getline(ss, line, ':');
        item.itemNo = stoi(line);
        std::getline(ss, item.name, ':');
        std::getline(ss, item.description, ':');
        std::getline(ss, line);
        item.price = stod(line);
        menu.push_back(item);
    }
    fileIn.close();
}

void displayMenu(const std::vector<MenuItem>& menu) {
    std::cout << "\nMenu Items:\n";
    for (const auto& item : menu) {
        std::cout << item.itemNo << ": " << item.name
                  << " - " << item.description << " - $"
                  << std::fixed << std::setprecision(2)
                  << item.price << "\n";
    }
    std::cout << std::endl;
}

void processOrder(const std::vector<MenuItem>& menu, const
std::string& transactionFile) {
    displayMenu(menu);
    std::cout << "Please enter the item numbers and quantities for
the order (0 to finish):\n";
    int itemNo, quantity;
    double subtotal = 0.0;
    std::string billText;
    while (true) {
        std::cout << "Item number (0 to finish): ";
        std::cin >> itemNo;
        if (itemNo == 0) break;

        std::cout << "Quantity: ";
        std::cin >> quantity;

        bool found = false;
        for (const auto& item : menu) {
            if (item.itemNo == itemNo) {
                double itemTotal = item.price * quantity;
                subtotal += itemTotal;
                billText += std::to_string(item.itemNo) + " x " +
std::to_string(quantity) + " "
                        + item.name + " - $" + std::to_string(itemTotal) +
"\n";
                found = true;
                break;
            }
        }

        if (!found) {
            std::cout << "Invalid item number. Please select an available item number for service.\n";
        }
    }

    double vat = calculateVAT(subtotal);
    double total = subtotal + vat;

    billText += "Subtotal: $" + std::to_string(subtotal) + "\n";
    billText += "VAT: $" + std::to_string(vat) + "\n";
    billText += "Total: $" + std::to_string(total) + "\n";

    std::cout << "\nBill:\n" << billText;
    saveBillToFile(transactionFile, billText, total);
}

double calculateVAT(double amount) {
    const double VAT_RATE = 0.20; // 20% VAT
    return amount * VAT_RATE;
}

void saveBillToFile(const std::string& transactionFile, const std::string& billText, double total) {
    std::ofstream outFile(transactionFile, std::ios::app);
    if (!outFile) {
        std::cerr << "Error opening transaction file.\n";
        return;
    }
    std::time_t now = std::time(nullptr);
    std::string datetime = std::ctime(&now);

    outFile << "\n" << datetime;
    outFile << billText;
    outFile << "Grand Total: $" << total << "\n";
    outFile << "-------------------------\n";

    std::cout << "Bill saved to transactions file.\n";
}
```

4.3 The third program

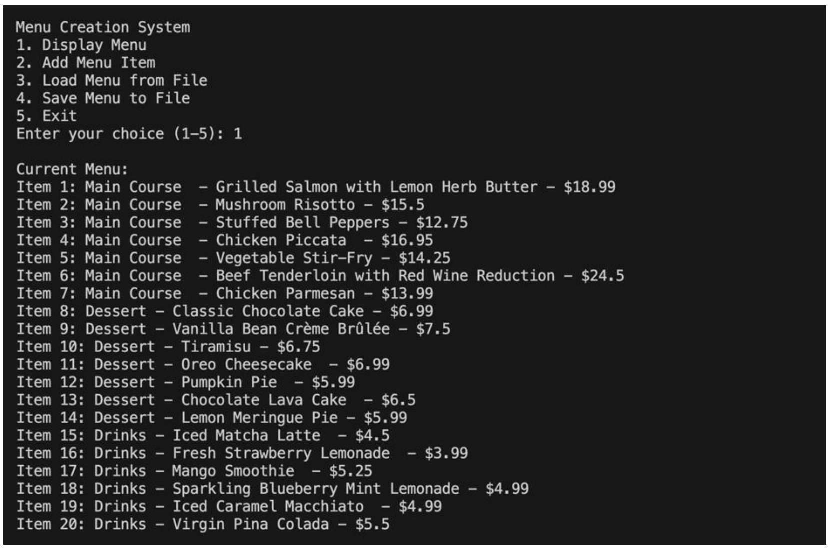
```
/*********************************************************************
 * transactionstatsfile.cpp
 * Program to read and accumulate data from transaction files for
statistics
 * Pritam Gurung
 * Version 3.0
 *********************************************************************/

#include <iostream>
```

```
#include <map>
#include <fstream>
#include <string>
#include <sstream>
#include <iomanip>


void readTransactions(std::map<int, int>& itemOrders, double&
grandTotal, const std::string& filename);
void displayStatistics(const std::map<int, int>& itemOrders, double
grandTotal);

int main() {
    std::map<int, int> itemOrders; // Stores item number and total
quantity ordered
    double grandTotal = 0.0;
    std::string transactionsFilename = "transactions.txt";

    // Read transactions and update the order counts and grand total
    readTransactions(itemOrders, grandTotal, transactionsFilename);

    // Display the statistics
    displayStatistics(itemOrders, grandTotal);

    return 0;
}

void readTransactions(std::map<int, int>& itemOrders, double&
grandTotal, const std::string& filename) {
    std::ifstream fileIn(filename);
    if (!fileIn) {
        std::cerr << "Error opening transactions file.\n";
        return;
    }

    std::string line;
    while (getline(fileIn, line)) {
        if (line.find("Grand Total: $") != std::string::npos) {

            std::size_t pos = line.find("$");
            if (pos != std::string::npos) {
                double total = std::stod(line.substr(pos + 1));
                grandTotal += total;
            }
        } else {

            std::stringstream ss(line);
            int itemNo, quantity;
            char x;
            if ((ss >> itemNo >> x >> quantity) && (x == 'x')) {
                itemOrders[itemNo] += quantity;
            }
        }
    }
    fileIn.close();
}

void displayStatistics(const std::map<int, int>& itemOrders, double
grandTotal) {
    std::cout << "\nDaily Statistics:\n";
    std::cout << "Number of Orders for Each Item:\n";
    for (const auto& order : itemOrders) {
      std::cout << "Item " << order.first << ": " << order.second <<
" orders\n";
   }
   std::cout << "Grand Total Takings: $" << std::fixed <<
std::setprecision(2) << grandTotal << std::endl;
}
```

5. Test Log

Test log for Program 1 of what works.

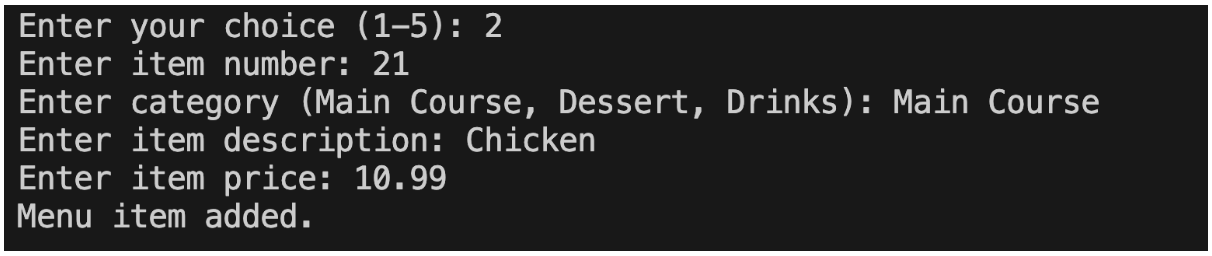
Display Menu: *When the user selects option 1, the program correctly displays the current menu. Each item on the menu is listed with an item number, category, description, and price. This indicates the ‘displayMenu’ function is working as it should be as showin in Figure ‘A’ below. Furthermore, the ‘displayMenu’ function iterates over the ‘menu’ vector, which contains ‘MenuItem’ structs, and outputs the details of each item formatted as "Item [itemNo]: [category] - [description] - $[price]". This shows that the program can correctly access and display the content stored within the vector, proving that it can operate efficiently.*



```
Menu Creation System
1. Display Menu
2. Add Menu Item
3. Load Menu from File
4. Save Menu to File
5. Exit
Enter your choice (1-5): 1

Current Menu:
Item 1: Main Course  - Grilled Salmon with Lemon Herb Butter - $18.99
Item 2: Main Course  - Mushroom Risotto - $15.5
Item 3: Main Course  - Stuffed Bell Peppers - $12.75
Item 4: Main Course  - Chicken Piccata  - $16.95
Item 5: Main Course  - Vegetable Stir-Fry - $14.25
Item 6: Main Course  - Beef Tenderloin with Red Wine Reduction - $24.5
Item 7: Main Course  - Chicken Parmesan - $13.99
Item 8: Dessert - Classic Chocolate Cake - $6.99
Item 9: Dessert - Vanilla Bean Crème Brûlée - $7.5
Item 10: Dessert - Tiramisu - $6.75
Item 11: Dessert - Oreo Cheesecake  - $6.99
Item 12: Dessert - Pumpkin Pie  - $5.99
Item 13: Dessert - Chocolate Lava Cake  - $6.5
Item 14: Dessert - Lemon Meringue Pie - $5.99
Item 15: Drinks - Iced Matcha Latte  - $4.5
Item 16: Drinks - Fresh Strawberry Lemonade  - $3.99
Item 17: Drinks - Mango Smoothie  - $5.25
Item 18: Drinks - Sparkling Blueberry Mint Lemonade - $4.99
Item 19: Drinks - Iced Caramel Macchiato  - $4.99
Item 20: Drinks - Virgin Pina Colada - $5.5
```

Figure ‘A’ showing the capability of the Display function.

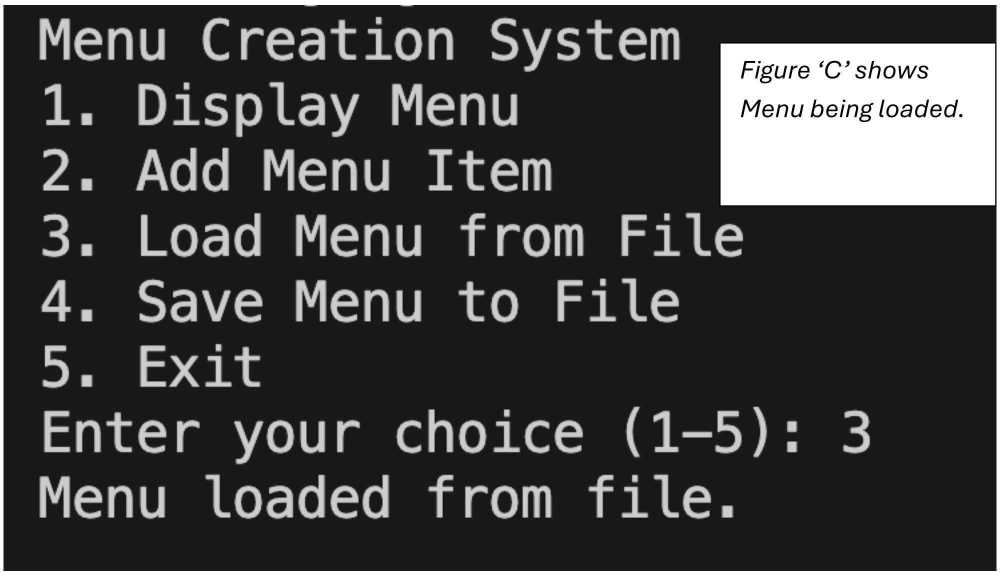
Add Menu Item (Option 2): *Choosing option 2 allows the user to add a new menu item by entering its details, which are then added to the menu. The terminal screenshot shows the program prompts*



```
Enter your choice (1-5): 2
Enter item number: 21
Enter category (Main Course, Dessert, Drinks): Main Course
Enter item description: Chicken
Enter item price: 10.99
Menu item added.
```

Figure ‘B’ showing the Item Added Function *for the item number, category, description, and price, and confirms the addition with "Menu item added." Furthermore, the 'addMenuItem' function collects user input for each field of the 'MenuItem' struct, creates a new 'MenuItem' with the provided details, and then adds this new item the 'menu' vector. The successful addition message indicates that the input was validated correctly and the new item was stored in memory which can be shown in Figure 'B'.*

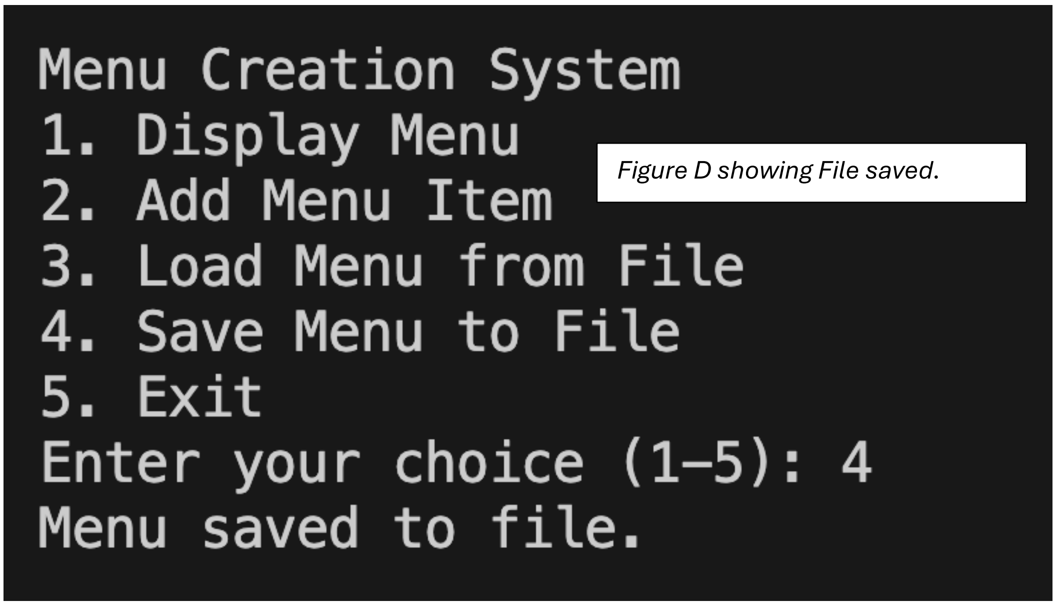
Load Menu from File (Option 3): *When the user selects option 3, the program attempts to load the menu from a file. The terminal output shows the message "Menu loaded from file," which shows that the 'LoadMenuFromFile' function is working shown in Figure 'C'. In addition to this, the 'LoadMenuFromFile' function reads from the specific file, parses each line to create 'MenuItem' objects, clears the exisiting 'menu' vector and fills it with the newly created items. The confirmation message from the system shows that the file was read successfully and that the menu items are loaded into the program's memory without any issues. When the menu has been loaded, the user who types 1, is able to review the menu that was previously created when attempting to run the program.*



```
Menu Creation System
1. Display Menu
2. Add Menu Item
3. Load Menu from File
4. Save Menu to File
5. Exit
Enter your choice (1-5): 3
Menu loaded from file.
```

Figure 'C' shows Menu being loaded.

Save Menu to File (Option 4): *Selecting option 4 causes the program to save the current state of the menu to a file which the output would confirm by displaying "Menu saved to a file" which confirms that the 'saveMenuToFile' function is done efficiently shown in Figure 'D'.*



```
Menu Creation System
1. Display Menu
2. Add Menu Item
3. Load Menu from File
4. Save Menu to File
5. Exit
Enter your choice (1-5): 4
Menu saved to file.
```

Figure D showing File saved.

Test Log for Menu Creation that does not work.

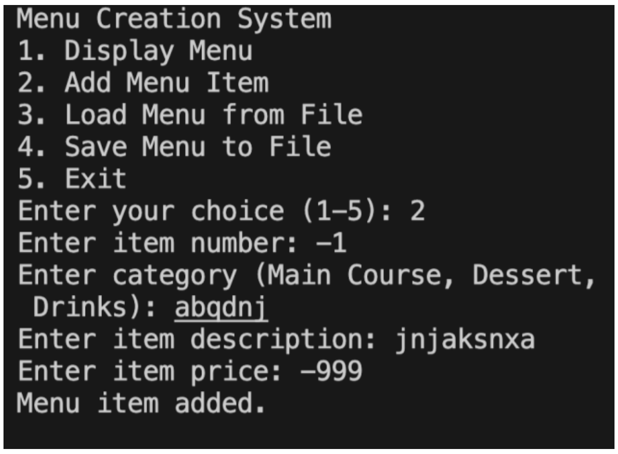
The program's current implementation lacks robust input validation, which leads to the acceptance of any provided values by the user. Even if they are outisde the logical or expected range shown in Figure E.

Item Number Validation*: The system permits negative numbers for the item number, which defies the menu listing convention where items identifies are positive integers.*

Category Validation*: The input for the category is not restricted to a set of predefined valid options (Main Course, Dessert, Drinks). As a result, the system accepts arbitrary strings as category names which can lead to inconsisitencies in the menu and may affect the usability of the menu for both the restaurant staff and customers.*

Price Validation*: The price of an item is a critical piece of information and it's standard to know that no price has a negative value. The current lack of validation means that the suste will accept negative values for prices, which is nonsense in the real world as this would means the restaurant is effectively paying customers to order a dish.*

This limitation in the validation mechanism can lead to a corrupted state of the menu data, where items with negative identifies and prices, as well as nonsensical category names, are stored and treated as valid such data integrity issue can result in erros such as generating bills, inventory management and statistical reporting.



```
Menu Creation System
1. Display Menu
2. Add Menu Item
3. Load Menu from File
4. Save Menu to File
5. Exit
Enter your choice (1-5): 2
Enter item number: -1
Enter category (Main Course, Dessert,
 Drinks): abqdnj
Enter item description: jnjaksnxa
Enter item price: -999
Menu item added.
```

Figure E showing invalid input validation.

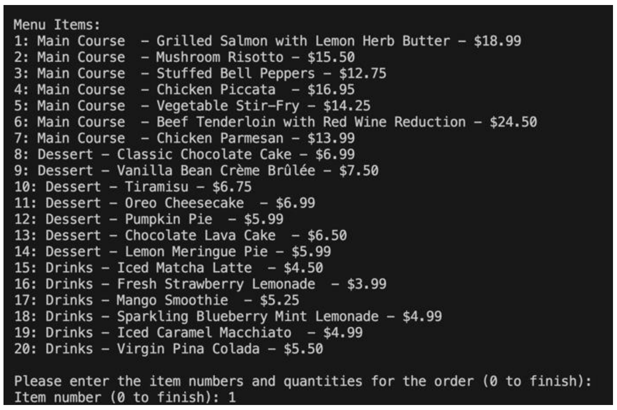
Test Log for Program 2

5.2 Test Log for Program 2, what works.

Displaying Menu Items: The system displays the list of menu items with their item numbers, categories, descriptions, and prices correctly formatted shown in Figure F.

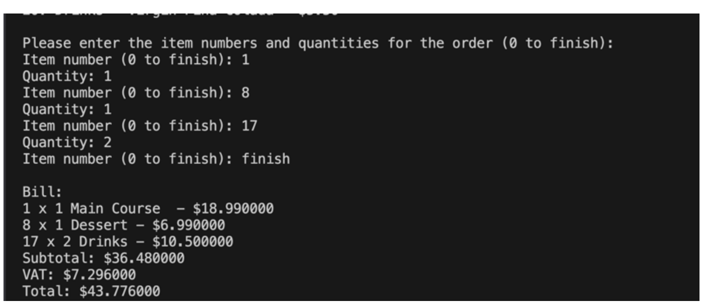
Ordering Process: The user is prompted to enter item numbers and quantities correctly, and the system accepts this input without crashing shown in Figure G.

Calculating Bill: The program correctly calculates and displays the bill with the ordered items, their quantities, prices, subtotal, VAT, and total amount shown in Figure G.



```
Menu Items:
1: Main Course  - Grilled Salmon with Lemon Herb Butter - $18.99
2: Main Course  - Mushroom Risotto - $15.50
3: Main Course  - Stuffed Bell Peppers - $12.75
4: Main Course  - Chicken Piccata  - $16.95
5: Main Course  - Vegetable Stir-Fry - $14.25
6: Main Course  - Beef Tenderloin with Red Wine Reduction - $24.50
7: Main Course  - Chicken Parmesan - $13.99
8: Dessert - Classic Chocolate Cake - $6.99
9: Dessert - Vanilla Bean Crème Brûlée - $7.50
10: Dessert - Tiramisu - $6.75
11: Dessert - Oreo Cheesecake  - $6.99
12: Dessert - Pumpkin Pie  - $5.99
13: Dessert - Chocolate Lava Cake  - $6.50
14: Dessert - Lemon Meringue Pie - $5.99
15: Drinks - Iced Matcha Latte  - $4.50
16: Drinks - Fresh Strawberry Lemonade  - $3.99
17: Drinks - Mango Smoothie  - $5.25
18: Drinks - Sparkling Blueberry Mint Lemonade - $4.99
19: Drinks - Iced Caramel Macchiato  - $4.99
20: Drinks - Virgin Pina Colada - $5.50

Please enter the item numbers and quantities for the order (0 to finish):
Item number (0 to finish): 1
```

Figure F shows the System displaying the list of menu items with their item numbers for customers to order.



```
Please enter the item numbers and quantities for the order (0 to finish):
Item number (0 to finish): 1
Quantity: 1
Item number (0 to finish): 8
Quantity: 1
Item number (0 to finish): 17
Quantity: 2
Item number (0 to finish): finish

Bill:
1 x 1 Main Course  - $18.990000
8 x 1 Dessert - $6.990000
17 x 2 Drinks - $10.500000
Subtotal: $36.480000
VAT: $7.296000
Total: $43.776000
```

Figure G prompting the user to enter item number and quantities along with showing the bill.

5.2.5 What doesn't work for Program 2.

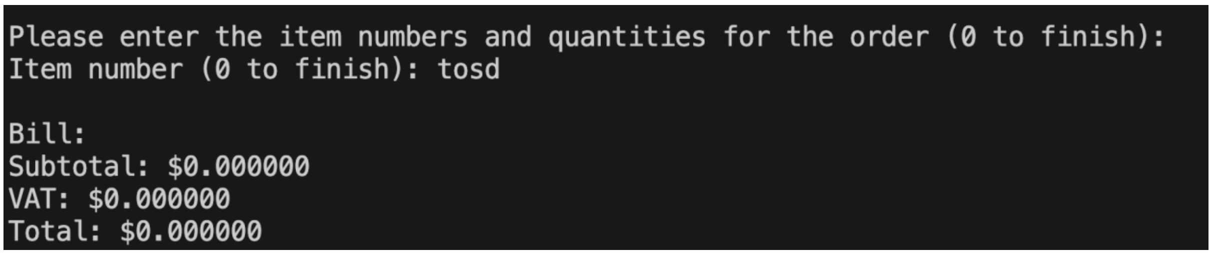
Input Validation: The system doesn't validate the input for item numbers and quantities. For example, the user can enter an item number that does not exist in the menu or input negative numbers, which the system processes without any error message or prompts to re-enter the valid data shown below in Figure H.

Price Precision: The bill displays prices with too many decimal places as shown below in Figure G, which is not typical for a normal billing system and generally prices should be rounded to two decimal places for currency formatting.

VAT Calculations: The VAT is calculated and displayed with excessive precision shown in Figure G, this should also be formatted to two decimal places to match currency standards as well as reduce confusion for both restaurant owners and customers.

Confusing Prompt: The program prompts the user to enter between the value of 0 to finish, instead another unique input could be implemented in the system where an alternative key instead of 'finish' could be used to indicate finish ordering as this can be confusing to some and could be a limitation to of program.

Figure H shows the bill information along with system prompts.



```
Please enter the item numbers and quantities for the order (0 to finish):
Item number (0 to finish): tosd

Bill:
Subtotal: $0.000000
VAT: $0.000000
Total: $0.000000
```

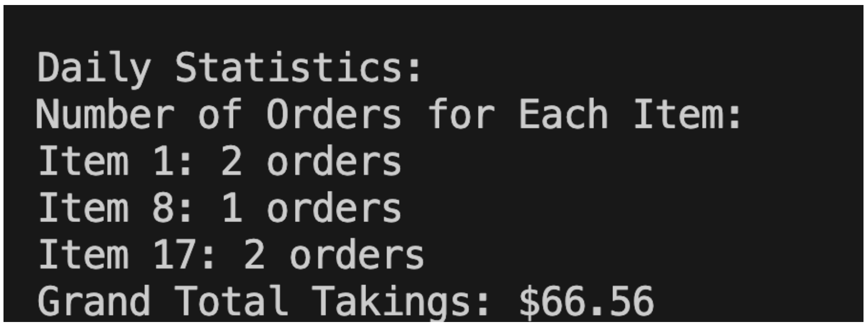
Test Log for Program 3

Test Log for Program 3 What works

Counting Orders: The program successfully counts and displays the numbers of orders for each item, and it correctly identifies items 1, 8, and 17 as having been ordered and gives an accurate count for each shown in Figure I.

Calculating Total Takings: The grand total takins are computed and displayed shown in Figure I, showing that the program can correctly sum up the earnings from the day's orders.

In addition, the program efficiently compiles daily statistics from the transaction records, and it provides a concise summary of the number of orders for each item, which is crucial for inventory tracking and understanding sales patterns. Furthermore, the program displays the total takings, giving a snapshot of the day's revenue shown in Figure I, which is essential for daily business reporting and financial management.



```
Daily Statistics:
Number of Orders for Each Item:
Item 1: 2 orders
Item 8: 1 orders
Item 17: 2 orders
Grand Total Takings: $66.56
```

Figure I show Daily Statistics for the 3rd program.

Test Log for Program 3 for what doesn't work.

Lack of Descriptive details: The statistics display only item numbers without the corresponding item descriptions, which would have been more informative for customers leading to a more informed decision being made and reducing errors being made. For instance, instead of just "Item 1," it would be more descriptive to have "Item 1: Grilled Salmon." This way it's clear, concise, and less prone to errors being made.

Plurality in Labelling: The label "orders" after item counts should be singular or plural depending on the number of orders (e.g., "2 orders" should be "2 orders" but "1 order" should be "1 order"). This way it can prevent confusion between both the customer and restaurant owners as it could lead to more unnecessary items being placed with the use of plural or just general confusion which can cause errors.

Overall, while the program effectively tracks and totals order and revenue, it could be enhanced by refining the output for better readability and clarity. Adjusting plural of orders based on count will improve accuracy and less prone to making mistakes, including item descriptions alongside numbers would make the statistics immediately understandable.

Evaluation

The restaurant billing system consists of the three main components: the menu creation, billing checkout and transaction statistics. Each of these components play a. pivotal role in the restaurant's digital infrastructure.

Menu Creation System: This component is tasked with creating and managing the menu. Users can add new items, display the current menu, load a menu from a file, save updates to a file, or exit the system. The program succeeds in allowing users to input new menu items, display the current menu, and save to and loading from a file. However, the program does not validate input sufficiently. For example, it accepts negative values for item numbers and prices and non-existent categories, which could lead to potential data corruption and confusion. The system would benefit from the implementation of input validation checks against each user input.

Billing Checkout: The billing checkout program enables staff to generate customer bills. It processes orders effectively, including the calculations of the total bull and VAT. However, it displays several limitations. It does not prevent the addition of items not present in the menu or the ordering of items in negative quantities. Additionally, prices and VAT are displayed with an impractical number of decimal places which can cause confusion. Addressing these issues requires formatting outputs to two decimal places and preventing the entry of invalid item numbers or quantities.

Transaction: The daily statistics accumulates and summarise transaction data. It accurately tracks and reports the number of orders per menu items and calulates the day's total earnings. While functional, the program does not dicern between singular and plural forms and lacks a comprehensive display of items that received no orders. Improving these areas will provide more informative and accurate reporting leading to less mistakes.

Input validation and error management were pivotal during development. Grasping C++ file I/O intricacies and stand library utilisation for string and stream operations was possible due to research and in conjunction with lectures. YouTube channels like BroCode, along with resources from GeeksforGeeks and W3Schools, proved vital in overcoming obstacles associated with data manipulation, arithemetic calculations and detailed C++ language documentation, particularly stack overflow were key in tackling coding issues. Peers code reviews were beneficial in identifying and addressing possible mistakes and misunderstanding in the code.

In summary, the three programs form a cohesive system but exhibit areas needing improvement, primarily in user input validation and output formatting. Future work should focus on improving the robustness of input handling and improving the user interface's clarity. These changes will ensure that the system is not only operationanlly effective but also it aligns with industry standards and provides efficient effective experience for users.