Project Final Report

Only for course Teacher						
		Needs Improvement	Developing	Sufficient	Above Average	Total Mark
Allocate mark & Percentage		**25%**	**50%**	**75%**	**100%**	**40**
Problem Understanding	**10**					
Analysis	**15**					
Implementation	**10**					
Task Efficiency	**5**					
					Total obtained mark	
Comments						

Semester: Spring 2024

Group name: Team Titan

Group Details: -

Student name: Chowdhury Nabil Ahmed**;** **Student ID:** 0242310005341011

Student name: Md Ismail Hossain; **Student ID:** 0242310005341098

Student name: Nusrat Jahan Fariha; **Student ID:** 0242310005341127

Batch: 40 **Section:** B1

Course Code: SE133 **Course Name:** Software Development Capstone Project

Course Teacher Name: Md. Abdul Hye Zebon

Designation: Lecturer

Submission Date: 11/06/2024

Introduction:

In today's competitive healthcare environment, efficient inventory control and streamlined sales processes are crucial for medical stores. The Medical Store Management System project is designed to modernize how pharmacies manage their stock and customer transactions. This system aims to provide a simple yet effective solution for tracking inventory, automating tasks, and improving overall efficiency, allowing pharmacies to focus on providing excellent customer service.

The Medical Store Management System boasts a user-centric interface designed to streamline medication management. Users can easily add, view, search, update, and adjust stock levels of medical supplies. This automation eliminates time-consuming manual tasks and minimizes the risk of errors in inventory management. As a result, pharmacists can focus on customer needs while maintaining accurate and reliable stock data, ensuring patients have uninterrupted access to necessary medications.

The Medical Store Management System can extend beyond just a practical business tool. It can also function as a valuable training resource for aspiring pharmacy technicians or those interested in healthcare operations. By working with the system, users can gain practical experience in essential tasks like inventory management, customer interaction (simulated sales), and record-keeping. This hands-on learning experience reinforces key concepts in pharmacy management and prepares users for the realities of the healthcare field.

In Summary, the Medical Store Management System is a comprehensive and innovative solution designed to modernize pharmacy operations. This user-centric system streamlines inventory control, automates tasks, and fosters improved efficiency. It empowers pharmacists to focus on patient care and customer service. Additionally, the system transcends its practical use by serving as a valuable training tool for aspiring pharmacy technicians and those interested in healthcare operations. By providing hands-on experience in crucial tasks, the system equips users with the necessary skills to excel in the dynamic healthcare environment.

Objective:

The objective of the Medical Store Management System project is to develop a user-friendly software application for pharmacies. This system aims to streamline medication management by automating tasks like adding, displaying, searching, updating, and adjusting stock levels of medical supplies. By automating these processes, the system will improve inventory control, reduce manual work, and minimize errors. This translates to increased efficiency and accuracy in managing pharmacy stock, while ensuring all necessary medications are readily available for patients.

This project is designed for beginner-level students to learn by doing. Through a user-friendly interface, students will gain hands-on experience with managing a medical store. This practical experience includes working with data structures to organize medicine information, file I/O to store and retrieve data, and basic algorithms to manage tasks like inventory control and billing. These skills are fundamental building blocks for a successful career in healthcare IT.

Additionally, the system aims to:

- **Improve Data Management:** The technology minimizes human mistake and ensures accurate, up-to-date information by automating stock tracking and recordkeeping.
- **Enhance Learning:** By putting theoretical knowledge to use in a real-world context, the project helps students better comprehend fundamental programming concepts and acts as an instructional tool.
- **Facilitate Scalability:** Now it can handle only 100 medicine items**.** The system is built with scalability in mind as such as dynamic memory allocation and more advanced data handling techniques.
- **Promote Good Programming Practices:** All prospective programmers should understand the value of modular functions, organized code, and fundamental error handling, which will be taught to the students.

Students will gain a greater understanding of the complexities of software development as well as a working software application by finishing this project. In their academic and professional pursuits, they'll be more equipped to take on more difficult assignments and obstacles.

Library:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_ITEMS 100
```

The provided code includes necessary header files and defines a constant for managing student records in a C program. These inclusions and definitions set up the foundational components needed for the Student Management System, allowing the program to handle input/output operations, memory management, string operations, and limit the number of students to 100.

Structure Declare:

```
struct Item
{
    char id[50];
    char name[50];
    char company[50];
    char drug[50];
    float price;
    int quantity;
};
struct Item items[MAX_ITEMS];
int itemCount = 0;
int databaseLoaded = 0;
```

The provided code defines a structure `Item` in C, which groups together related data about a medicine. The structure contains the following members:

- `char id[50]`: An array to store the medicine's ID.
- `char name[50]`: An array to store the medicine's name.
- `char company[50]`: An array to store the medicine's company.
- `char drug[50]`: An array to store the medicine's drug.
- `float price`: An float to store the medicine's price.
- `int quantity`: An integer to store the medicine's quantity.
- `struct Item items[MAX_ITEMS]`: An array to store up to `MAX_ITEMS` medicine records.
- `int itemCount = 0`: A variable initialized to 0 that keeps track of the current number of medicine records in the array.
- `int databaseLoaded = 0`: A variable initialized to 0 that keeps track of the database of medicine.

This structure allows you to create variables that can hold and manage all these related pieces of information together, simplifying data handling in your program. These variables facilitate adding, showing, updating, and deleting medicine records within the program.

Main Function

```
int main()
{

    printf("\n~~~~~~~~~~~~~~~~ WELCOME ~~~~~~~~~~~~~~~~\n");
    printf("\n~~~ Medical Store Management System ~~~\n");
    printf("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

    int loadChoice;
    printf("\nDo you want to load Previous Database? (1 for Yes, 0 
for No): ");
    scanf("%d", &loadChoice);

    if (loadChoice == 1)
    {
        loadItemsFromFile();
        if (itemCount == 0)
        {
            printf("\nNo data found in the Database. Starting with 
the Main Menu-\n");
        }
        else
        {
            printf("\nDatabase loaded Successfully.\n");
        }
    }
    else
    {
        printf("\nDatabase will not be loaded. Starting with the Main 
Menu- \n");
        itemCount = 0;
```

```
}

int choice;
while (1)
{

    printf("\n~~~~~~~~~~ Main Menu ~~~~~~~~~~\n\n");
    printf("1. Add Medicine\n");
    printf("2. Show Medicine List\n");
    printf("3. Update Medicine\n");
    printf("4. Remove Medicine\n");
    printf("5. Calculate bill\n");
    printf("0. Exit\n\n~");
    printf("\nEnter your Choice: ");

    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        addMedicine();
        saveItemsToFile();
        break;
    case 2:
        showList();
        break;
    case 3:
        updateMedicine();
        break;
    case 4:
        removeMedicine();
```

```
            break;
        case 5:
            calculateBill();
            break;
        case 0:
            printf("\nThank You. Come Again!\n");
            exit(0);
        default:
            printf("Invalid choice. Try again.\n\n");
            break;
        }
    }
    return 0;
}
```

The provided code is for a basic "Medical Store Management System" implemented in C. It uses if-else, nested if-else, a loop and a switch statement to provide a menu-driven interface for various operations related to student records. Here's a breakdown of the code:

1. **Function Definition**: `int main() { ... }`: This defines the main function, which is the entry point of the program.
2. **Variable Declaration**: `int loadChoice;` This variable stores the database download choice.
3. **Conditionals**:
    - If the input of the `loadChoice` is '1' it undergoes another Conditional.
        - If '`itemCount==0`', it prints "`No data found in the Database. Starting with the Main Menu`"
        - Else it prints "Database loaded Successfully."
    - Else it prints, "`Database will not be loaded. Starting with the Main Menu-`"
4. **Variable Declaration:** `int choice;` This variable stores the user's menu choice.
5. **Infinite Loop**: `while(1) { ... }`: This creates an infinite loop, which continuously displays the menu until the user chooses to exit.
6. **Menu Display**: `printf` statements: These print the menu options to the console.
7. **User Input**: `scanf("%d", &choice);`: This reads the user's choice from the console.

8. **Switch Statement**: `switch (choice) { ... }`: This handles the different user choices. Each case corresponds to a menu option and calls the appropriate function:

    - `case 1: addMedicine();`
      `saveItemsToFile();`
    - `case 2: showList()`
    - `case 3: updateMedicine();`
    - `case 4: removeMedicine();`
    - `case 5: calculateBill();`
    - `case 0:` Prints a Thank you message and exits the program using `exit(0)`.
    - `default:` Prints an error message for invalid choices.

The "Add Medicine" function:

```
void addMedicine()
{
    printf("\nAdding Item\n");
    struct Item newItem;
    getchar();
    printf("Enter ID: ");
    gets(newItem.id);
    printf("Enter Name: ");
    gets(newItem.name);
    printf("Enter Company Name: ");
    gets(newItem.company);
    printf("Enter Drug: ");
    gets(newItem.drug)
    printf("Enter Price: ");
    scanf("%f", &newItem.price);
    printf("Enter Quantity: ");
    scanf("%d", &newItem.quantity);
    items[itemCount++] = newItem;
    printf("\n---New Item Added Successfully---\n");
}
```

The `addMedicine` function adds a new medicine record to the medicine array if the maximum limit has not been reached.

- **Check Limit**: Verifies if the maximum number of medicine (`MAX_ITEMS`) has been reached. If so, prints an error message and exits the function.
- **Input New Medicine**: Prompts the user to enter details for a new medicine:

    - ID
    - Name
    - Company name
    - Drug
    - Price
    - Quantity

- **Store Medicine**: Adds the new medicine to the medicine array and increments `itemCount.` Then saves the medicine details in `‘saveItemsToFile’` file.
- **Confirmation**: Prints a success message.

The “Show List” function

```
void showList()

{

    printf("\n~~~Viewing Items:-\n");

    if (itemCount == 0)

    {

        printf("No medicine found.\n");

    }

    else

    {

        for (int i = 0; i < itemCount; i++)

        {

printf("\nID: %s || Name: %s || Company: %s || Drug: %s || Price:
%.2f || Quantity: %d\n", items[i].id, items[i].name,
items[i].company, items[i].drug, items[i].price, items[i].quantity);

        }

    }

}
```

The `showList` function shows the Medicines by their ID and displays their information if found.

- **Check Medicine Count**: If no Medicines are available (`itemCount == 0`) it prints a message and returns.
- **Print Header:** Displays the headers for the table with medicine information, formatted for readability.
- **Loop Through Medicines**: Iterates through each medicine in the `medicine` array and prints their details in a formatted manner:
    - `ID`
    - `Name`
    - `Company`
    - `Drug`
    - `Price`
    - `Quantity`

The “Update Medicine” function

```
{
    char id[50];
    printf("Enter Medicine ID to update: ");
    scanf("%s", id);
    for (int i = 0; i < itemCount; i++)
    {
        if (strcmp(items[i].id, id) == 0)
        {
            printf("Enter Updated Price: ");
            scanf("%f", &items[i].price);
            printf("Enter Updated Quantity: ");
            scanf("%d", &items[i].quantity);
            printf("\n---Item Updated Successfully---\n");
            saveItemsToFile();
            return;
        }
    }
    printf("---Medicine not found!---\n");
}
```

The `updateMedicine` function edits the Medicine’s details by their ID and stores it in the file and array.

- **Array Declaration:** `char id[50];` This variable stores the id that has been searched.

- **Display a message:** This shows ‘`Enter Medicine ID to update:`’ to take medicine’s id.
- **User Input**: `scanf("%s", id); This reads the user's choice from the console.`
- **Loop Through Medicines:** Iterates through each medicine by id in the `medicine` array.
- **Compares ID’s: (**`strcmp(items[i].id, id) == 0)` compares the id that user wants to update.
- **Update Details:** If the id matches, it asks for the latest Price and Quantity
- **Save to File**: Latest Details are saved into file.

The “Remove Medicine” function

```
void removeMedicine()
{
    printf("\nDeleting Item\n");
    char idToDelete[50];
    printf("Enter ID of item to delete: ");
    scanf("%s", idToDelete);

    int found = 0;
    for (int i = 0; i < itemCount; i++)
    {
        if (strcmp(items[i].id, idToDelete) == 0)
        {
            found = 1;
            for (int j = i; j < itemCount - 1; j++)
            {
                items[j] = items[j + 1];
            }
            itemCount--;
            break;
        }
    }

    if (found)
    {
        printf("\n---Item Removed Successfully---\n");
        saveItemsToFile();
    }
    else
    {
        printf("---Item with ID %s not found.---\n", idToDelete);
    }
}
```

The `removeMedicine` function deletes the Medicine’s details by their ID and stores the latest list it in the file and array.

- **Display a message:** This shows ‘`Deleting Item:`’
- **Array Declaration:** `idToDelete[50]` This variable stores the id that has been searched for deletion.

- **User Input**: `scanf("%s", id);` This reads the user's choice from the console.
- **Loop Through Medicines:** Iterates through each medicine by id in the `medicine` array.
- **Compares ID's: (**`strcmp(items[i].id, id) == 0)` compares the id that user wants to delete from the database.
- **Delete Items:** If the id matches remove it from the database.
- **Save to File**: Latest Details are saved into file.

The “Calculate Bill” function

```
void calculateBill()
{
    printf("\nCalculating Bill:-\n");
    float totalBill = 0.0;

    while (1)
    {
        printf("\nEnter the ID of the item to purchase: ");
        char itemID[50];
        scanf("%s", itemID);

        printf("Enter the quantity of the item: ");
        int quantity;
        scanf("%d", &quantity);

        int found = 0;
        for (int j = 0; j < itemCount; j++)
        {
            if (strcmp(items[j].id, itemID) == 0)
            {
                found = 1;
                if (quantity <= items[j].quantity)
                {
                    printf("Item %s: %s x %d = %.2f\n", itemID,
items[j].name, quantity, items[j].price * quantity);

                    totalBill = totalBill + (items[j].price *
quantity);

                    items[j].quantity = items[j].quantity - quantity;
                }
                else
                {
                    printf("---Stock not available. Only %d units
available.---\n", items[j].quantity);
                }
                break;
            }
        }
        if (!found)
        {
```

```
            printf("---Item with ID %s not found.---\n", itemID);
        }

        printf("\n-----Total Bill: %.2f-----\n", totalBill);

        char again;
        printf("\nDo you want to calculate bill for another item? (1
for Yes, 0 for No):  ");
        scanf(" %c", &again);
        if (again != 1)
        {
            break;
        }
    }
}
```

The `calculateBill` function calculates total bill for a particular medicine then shows it in the console.

- **Initialize Variables:** One Float variable '`totalBill',` one array `char 'itemID[50]',`one integer `'again',` one Integer variable '`quantity'` declared to stores the ID's and the Quantity of the items being purchased. Then the total bill is shown.
- **Infinite Loop**: `while(1) { ... }`: This creates an infinite loop, which continuously displays the bill menu until the user chooses to exit.
- **Loop Through Medicines:** Iterates through each medicine by id in the `medicine` array.
- **Compares ID's: (**`strcmp(items[i].id, itemId) == 0)` compares the id that user wants to buy from the database that is available.
    - If id matches and stock is available, '`price * quantity'` calculates and gives the total bill of a medicine and updates the database by subtracting the quantity.
    - If id doesn't match it shows, `'Item with ID [particular id] not found'.`
    - If the demanded quantity of the medicine is not available then shows `'Stock not available'`
- **Print the Total Bill:** After processing all items, the function prints the total bill amount.

Output

1. This is the main menu. As shown in the picture, the main menu consists of a welcome message and it is asked if the user wants to load previous Database.

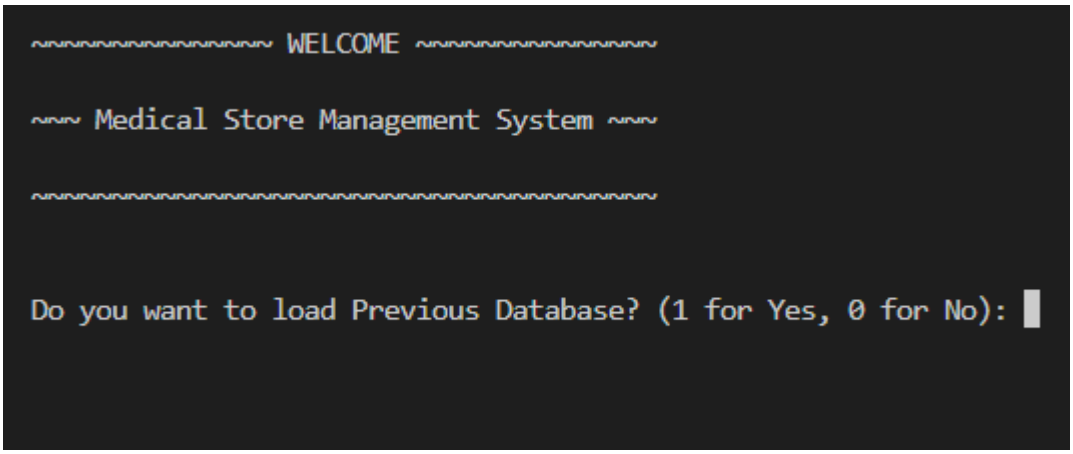

Fig-1: Welcome message

2. If the user gives an input 1 and there is a previous database stored, the system will show the Fig-2. If the user If the user gives an input 1 and there is no previous database stored, the system will show the Fig-3.

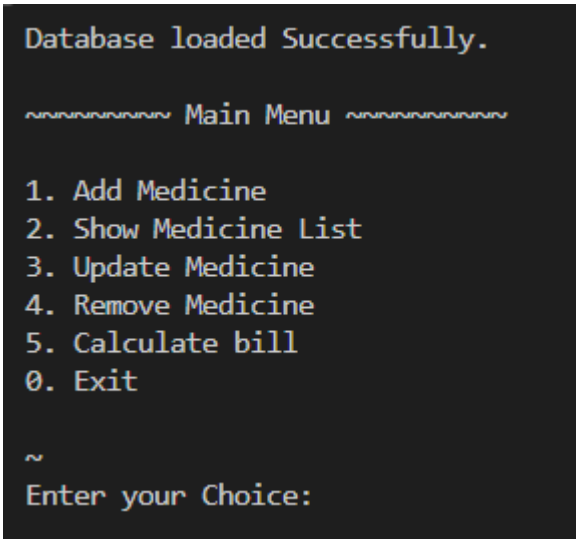

Fig-2: Main menu with database

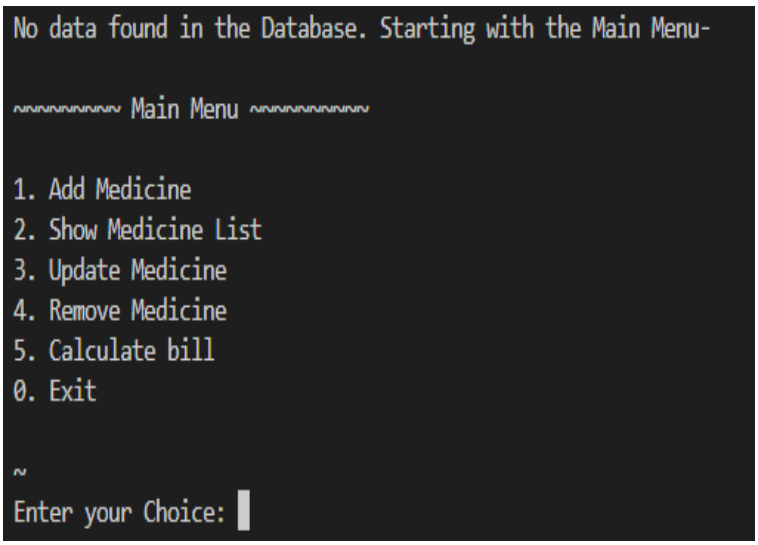

Fig-3: Main menu without database

3. While loading the main menu if a user gives an input other than 0-5, the system will show an error message and will the load the main menu again.

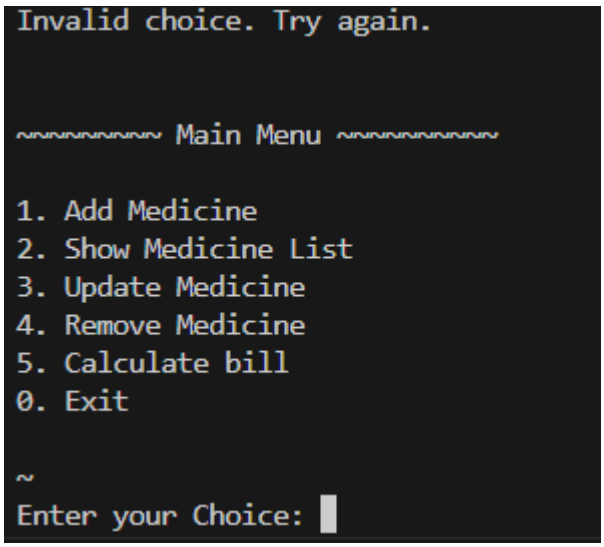

Fig-4: Invalid Input

4. Option-1 of the main menu adds a new medicine in the database. If user selects this option, the details of the medicine is asked as input. Here, a medicine with proper details has been added. At the end a successful message is shown.

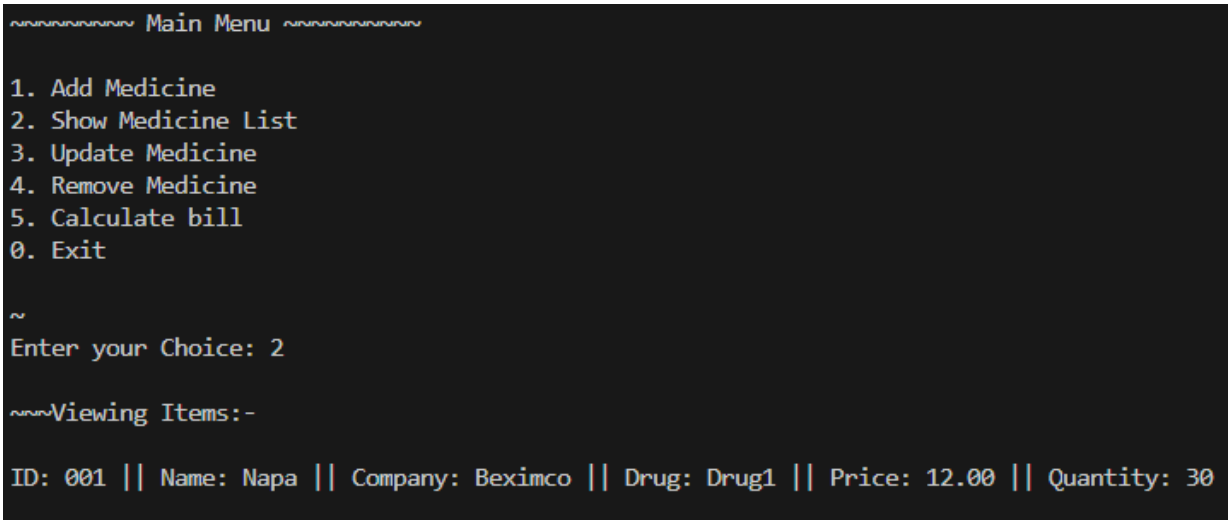

Fig-5: Item add

5. At some point the price could change of a medicine or the stock could be updated then this option is used. For example, here the price of napa decreased to 10 and the stocks goes down to 20. At the end it shows a message.

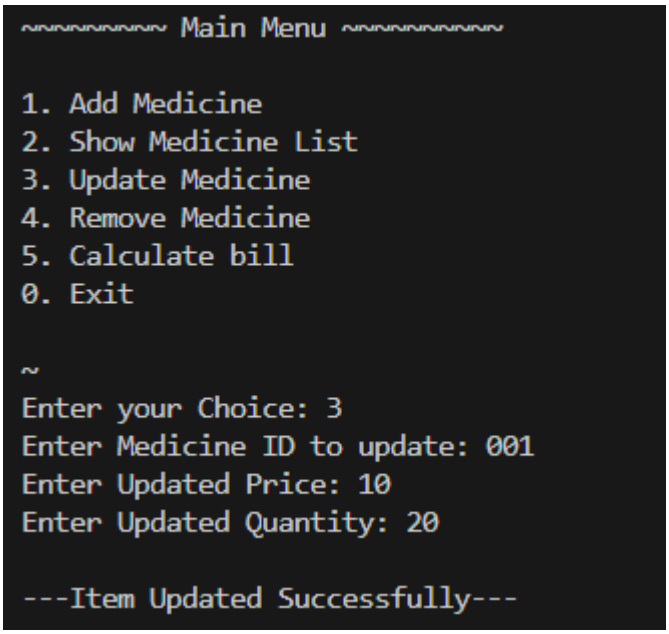

Fig-5: Update item

6. If a customer comes for a medicine the Calculate bill option is used. This options askes for id and quantity. If the stock is available then the bill is shown. If the stock is not available then the system shows stock not available message. At last, it asks if asks if another bill has to be created.

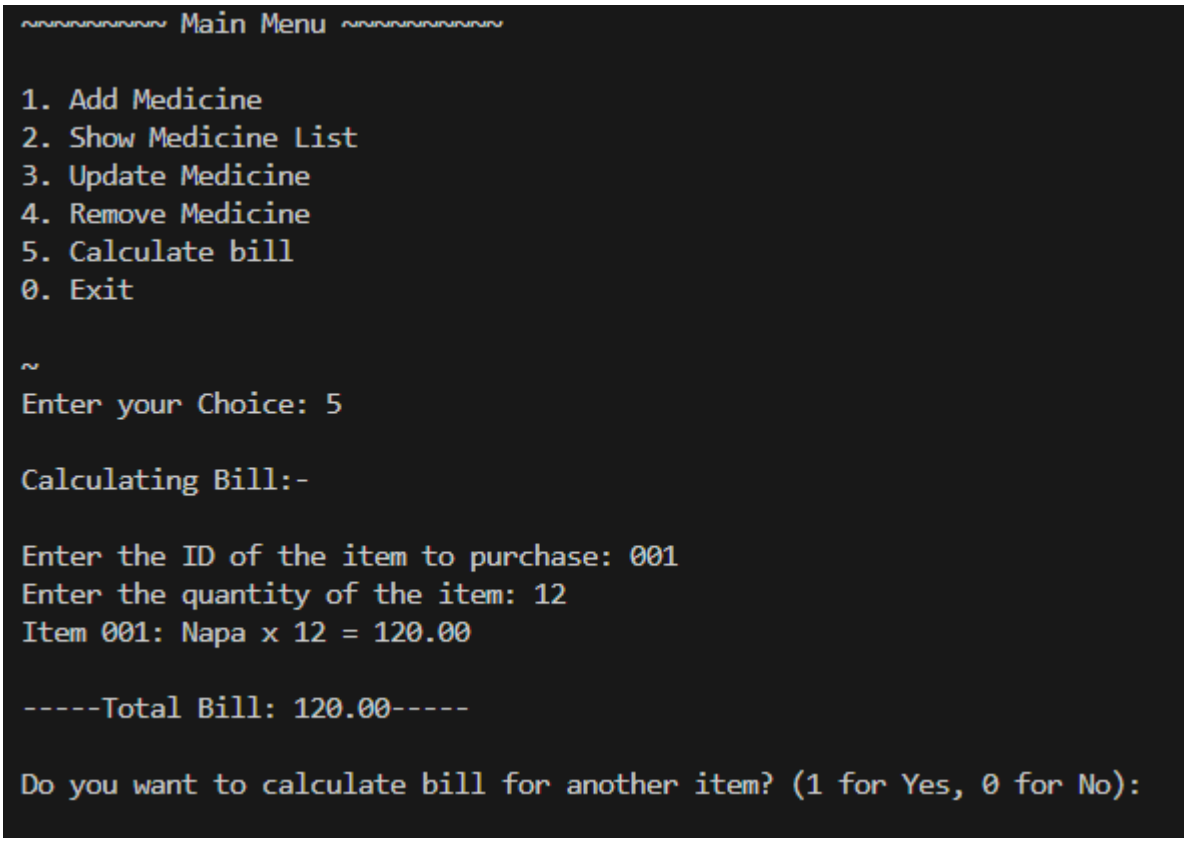

Fig-6: Calculating Bill

Limitations:

There are various restrictions with the Medical Store Management System. For user input, it uses gets, which is prone to security flaws and buffer overflows. Because it makes use of system("cls"), a Windows-specific command for screen cleaning, the software isn't cross-platform. Furthermore, the system might only be able to handle up to 100 items at a time, without having the dynamic memory allocation flexibility needed to deal with an inventory that changes frequently. Additionally, there's a chance that the file import and export features won't handle corrupted data gracefully, which could cause problems with data integrity. Ultimately, code maintainability and scalability are hampered for managing a larger and more complex pharmaceutical business due to the reliance on global variables and the lack of a modular design.

Conclusion:

In conclusion, the Medical Store Management System project is a commendable effort in enhancing the efficiency and accuracy of inventory management in medical retail environments. By automating key processes such as adding, displaying, searching, updating, and deleting medications, this system significantly reduces the workload on store staff and minimizes the chances of human error. It serves not only as a practical solution for current medical store management challenges but also as an educational tool for beginner-level programming students, providing them with hands-on experience in fundamental programming concepts like data structures, file I/O, and basic algorithms. Despite its limitations, including a fixed medication limit, platform dependency, and basic error handling, the project lays a solid foundation for future improvements and scalability. These constraints also offer valuable learning opportunities, encouraging users to think critically about potential enhancements and to explore more advanced programming techniques. Overall, the Medical Store Management System is an excellent starting point for those looking to delve into software development for medical retail management, preparing them for more complex projects in the future.