**Pharmacy Management System**

****

Session: 2021 – 2025

**Submitted by:**

**Syed Abdul-Rehman 2021-CS-62**

**Supervised by:**

**Dr. Awais Hassan**

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**CONTENTS**

* **Abstract**
* **Functional Requirements**
* **Wireframes**
* **Data Structures**
* **Function Prototypes**
* **Flowchart**
* **Complete Code**
* **Test Cases**

**Pharmacy Management System**

**Abstract:**

Pharmacies are critical to human life. The involve people from all walks of life. They are usually managed by not-so-technical people and since the lockdowns due to COVID-19, ability to buy things through websites has increased exponentially. Medicines are no different in this case. My Goal with this system is to minimize fancy menus and technical details the user has to deal with in order to operate it and allow customers to buy things using self-service channels. With this constraint, I tried my best to include all factors that a pharmacy might have to deal with on a day-to-day basis.

**Functional Requirements:**

There are three users of this system:

* Manager
* Receptionist
* Customer

1. **Manager:**

As manager, I can:

* 1. Check status of stocks
  2. View transactions
  3. Add new stock
  4. Add new item
  5. Show drug dealer license
  6. Read feedbacks
  7. View product return

1. **Receptionist:**

As a receptionist, I can:

* 1. Place customer’s order
  2. Decrease stocks (maybe due to expiry)
  3. Increase cash amount
  4. Let customers use redeem coupon
  5. Generate coupon

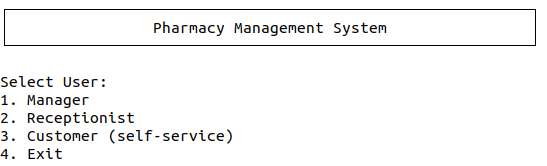
1. **Customer:**

As a customer, I can use self-service channels to:

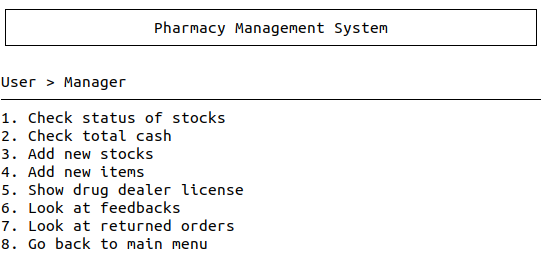
* 1. Buy a product
  2. Ask for drug dealer license
  3. Redeem coupon
  4. Return product
  5. Check price of a medicine
  6. View list of items available
  7. Record feedback

**Wireframes:**

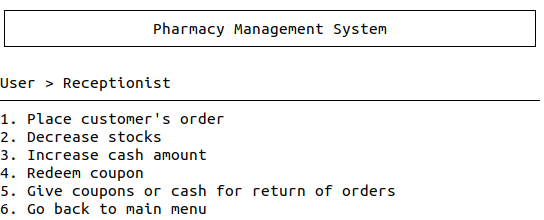
**Main Menu:**



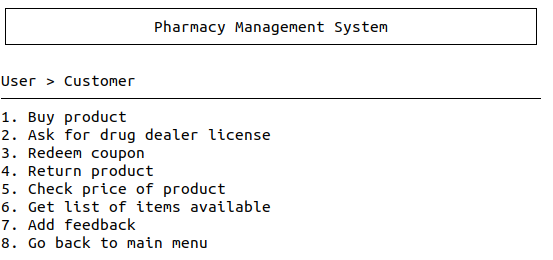
**Manager Menu:**



**Receptionist Menu:**

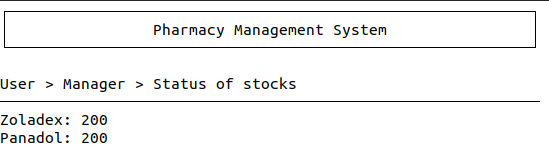


**Customer Menu:**

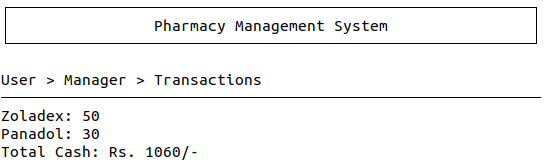


**Manager Options:**

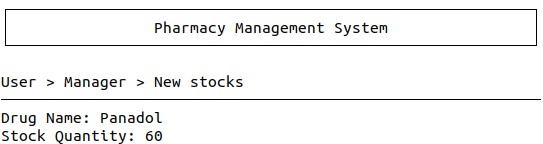
* **Check status of stock:**



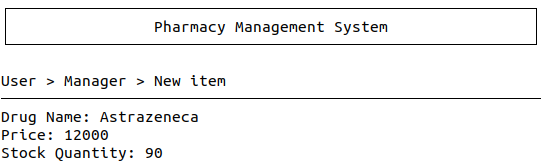
* **View transactions:**



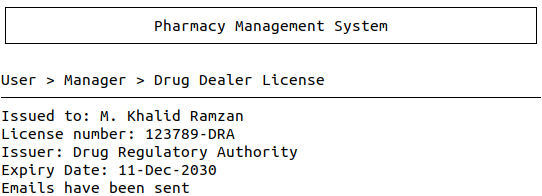
* **Add new stock:**

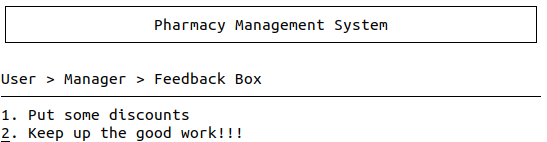
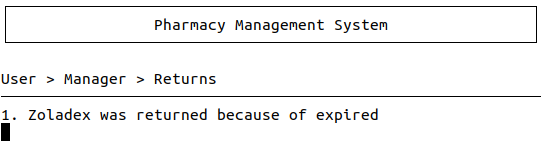


* **Add new item:**



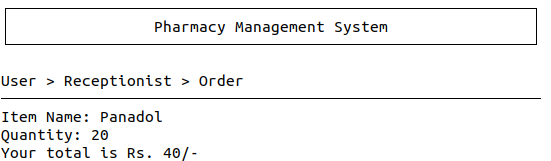
* **Show/Send drug dealer license:**

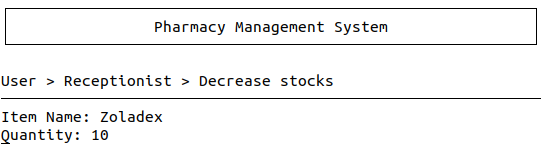


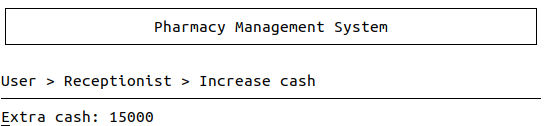
* **View feedbacks:**
* **View product returns:**

**Receptionist Options:**

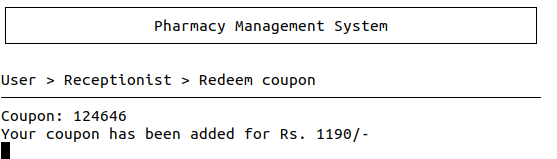
* **Customer’s order:**



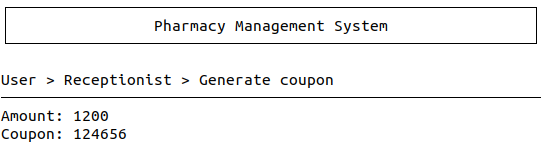
* **Decrease stock:** 
* **Add cash:**



* **Redeem customer’s coupon:**

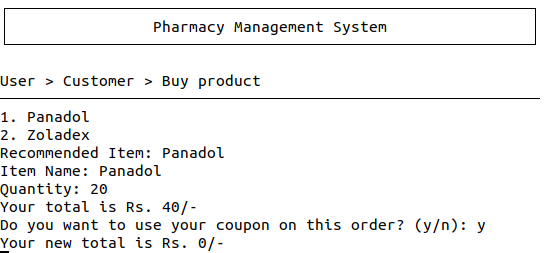


* **Generate coupon:**

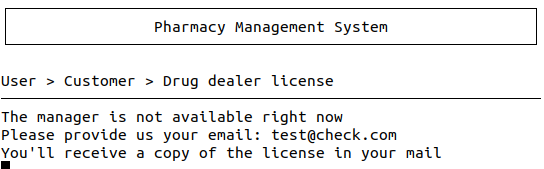


**Customer Options:**

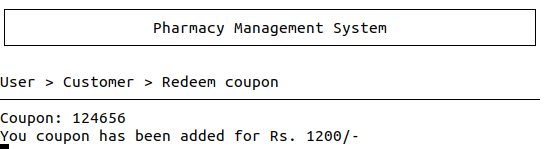
* **Buy product:**



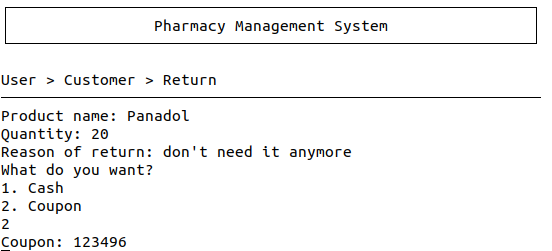
* **Ask for drug dealer license:**

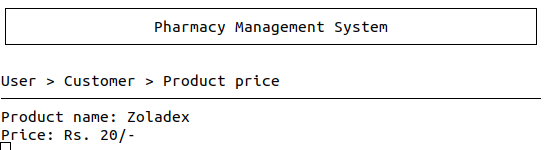


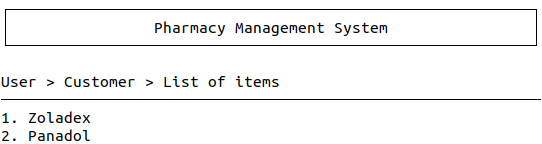
* **Redeem Coupon:**



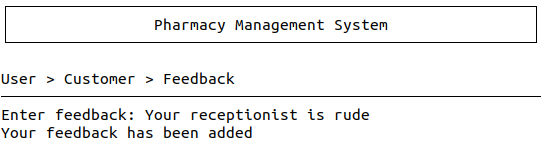
* **Return product:**



* **Product price:**
* **List items:**



* **Add feedback:**



**Data Structures:**

**Files:**

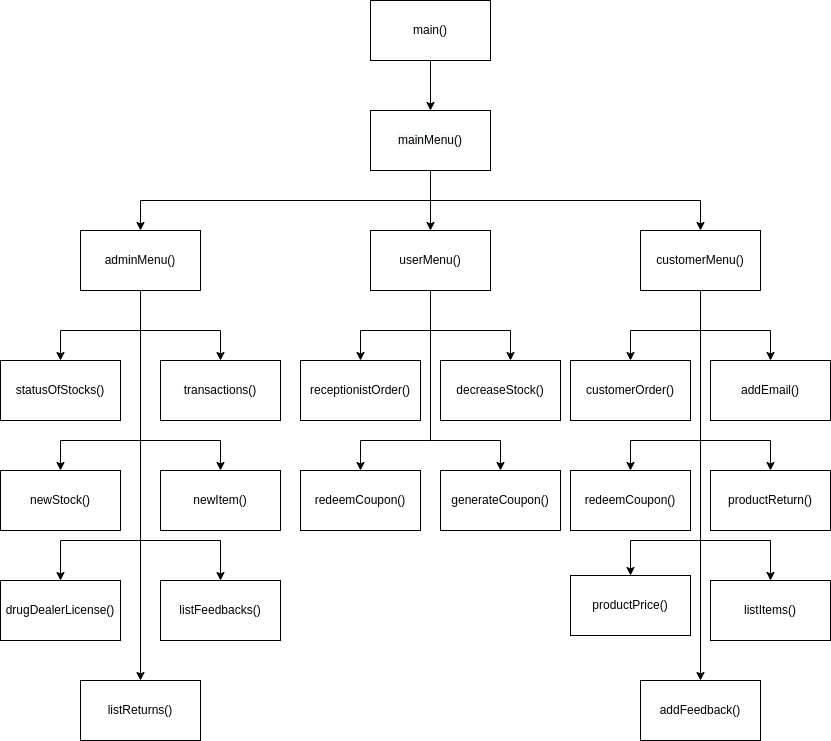
* projectData.txt

**Arrays:**

* string itemNames[MAX\_ITEMS] (For names of items)
* int itemPrices[MAX\_ITEMS] (For prices of items)
* int itemCounts[MAX\_ITEMS] (For number of items available in stock)
* int itemSold[MAX\_ITEMS] (For number of items sold)
* string feedbackBox[MAX\_ITEMS] (For feedbacks)
* string orderReturns[MAX\_ITEMS] (For reasons of product returns)
* string emails[MAX\_ITEMS] (For emails address to send license)

**Functions:**

1. void header();
2. void pause();
3. void banner(string);
4. int mainMenu();
5. int adminMenu();
6. int userMenu();
7. int customerMenu();
8. void statusOfStocks();
9. void transactions();
10. bool newStock(string, int);
11. bool newItem(string, int, int);
12. bool isValidEMAIL(string);
13. bool addEmail(string);
14. void receptionistOrder(string, int);
15. bool decreaseStock(string, int);
16. void customerOrder(string, int);
17. void productReturn(string, int, bool);
18. int productPrice(string);
19. void listItems();
20. bool addFeedback(string);
21. bool addReturn(int, string);
22. int getItemIndex(string);
23. int getMaxIndex(int[], int);
24. void sortArray(int[]);
25. void swap(int, int);
26. void load();
27. void store();
28. string getField(string, int);
29. int convertToInt(string);
30. void drugDealerLicense();
31. void listFeedbacks();
32. void listReturns();
33. bool redeemCoupon(int);
34. int generateCoupon(int);

**Flowchart:**

**Code:**

#include <iostream>

#include <fstream>

#include <conio.h>

using namespace std;

void header();

void pause();

void banner(string);

int mainMenu();

int adminMenu();

int userMenu();

int customerMenu();

void statusOfStocks();

void transactions();

bool newStock(string, int);

bool newItem(string, int, int);

bool isValidEMAIL(string);

bool addEmail(string);

void receptionistOrder(string, int);

bool decreaseStock(string, int);

void customerOrder(string, int);

void productReturn(string, int, bool, string);

int productPrice(string);

void listItems();

bool addFeedback(string);

bool addReturn(int, string);

int getItemIndex(string);

int getMaxIndex(int[], int);

void sortArray(int[]);

void swap(int, int);

void load();

void store();

string getField(string, int);

int convertToInt(string);

void drugDealerLicense();

void listFeedbacks();

void listReturns();

bool redeemCoupon(int);

int generateCoupon(int);

const int MAX\_ITEMS = 30;

string itemNames[MAX\_ITEMS];

int itemPrices[MAX\_ITEMS];

int itemCounts[MAX\_ITEMS];

int itemSold[MAX\_ITEMS];

int nStocks = 0;

string feedbackBox[MAX\_ITEMS];

int feedbackCount = 0;

string orderReturns[MAX\_ITEMS];

int returnCount = 0;

string emails[MAX\_ITEMS];

int emailCount = 0;

int tmpInt, totalCash = 0, couponAmout = 0;

string tmpString, reasonReturn;

int main(){

    int option;

string reasonReturn;

    load();

    while(true){

        header();

        option = mainMenu();

        if(option == 1){

            while(true){

                option = adminMenu();

                if(option == 1){

                    banner("User > Manager > Status of stocks");

                    statusOfStocks();

                }

                else if(option == 2){

                    banner("User > Manager > Transactions");

                    transactions();

                }

                else if(option == 3){

                    banner("User > Manager > New stocks");

                    cout << "Drug Name: ";

                    cin >> tmpString;

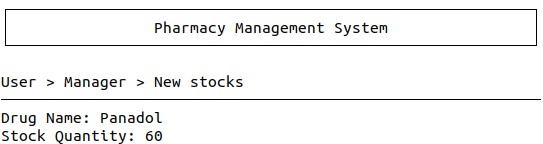
                    cout << "Stock Quantity: ";

                    cin >> tmpInt;

                    if(!newStock(tmpString, tmpInt)){

                        cout << tmpString << " not found in stock" << endl;

                    }



}

                else if(option == 4){

                    banner("User > Manager > New item");

                    cout << "Drug Name: ";

                    cin >> tmpString;

                    cout << "Price: ";

                    cin >> tmpInt;

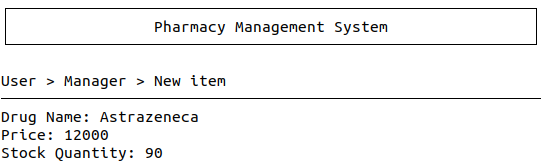
                    cout << "Stock Quantity: ";

                    cin >> option;

                    if(!newItem(tmpString, tmpInt, option)){

                        cout << "We cannot add more new items" << endl;

                    }



                }

                else if(option == 5){

                    banner("User > Manager > Drug Dealer License");

                    drugDealerLicense();

                }

                else if(option == 6){

                    banner("User > Manager > Feedback Box");

                    listFeedbacks();

                }

                else if(option == 7){

                    banner("User > Manager > Returns");

                    listReturns();

                }

                else if(option == 8) {

                    break;

                }

                else {

                    cout << "Invalid option" << endl;

                }

                pause();

            }

        }

        else if(option == 2){

            while(true){

                option = userMenu();

                if(option == 1){

                    banner("User > Receptionist > Order");

                    cout << "Item Name: ";

                    cin >> tmpString;

                    tmpInt = getItemIndex(tmpString);

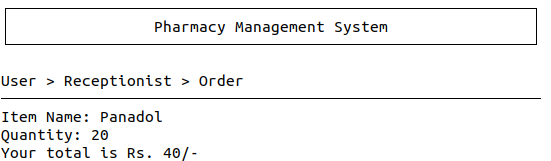
                    if(tmpInt != -1){

                        cout << "Quantity: ";

                        cin >> tmpInt;

                        receptionistOrder(tmpString, tmpInt);

                    }



                    else {

                        cout << tmpString << " is not available here" << endl;

                    }

                }

                else if(option == 2){

                    banner("User > Receptionist > Decrease stocks");

                    cout << "Item Name: ";

                    cin >> tmpString;

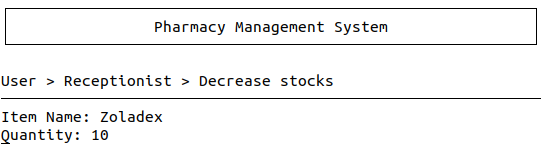
                    cout << "Quantity: ";

                    cin >> tmpInt;

                    if(!decreaseStock(tmpString, tmpInt)){

                        cout << tmpString << " not found in stock" << endl;

                    }



                }

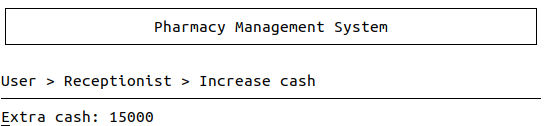
                else if(option == 3){

                    banner("User > Receptionist > Increase cash");

                    cout << "Extra cash: ";

                    cin >> tmpInt;

                    totalCash += tmpInt;



                }

                else if(option == 4){

                    banner("User > Receptionist > Redeem coupon");

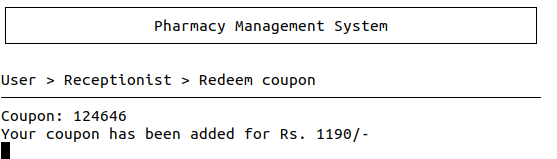
                    cout << "Coupon: ";

                    cin >> tmpInt;

                    if(redeemCoupon(tmpInt)){

                        cout << "Your coupon has been added for Rs. " << couponAmout << "/-" << endl;

                    }



                    else {

                        cout << "Invalid coupon" << endl;

                    }

                }

                else if(option == 5){

                    banner("User > Receptionist > Generate coupon");

                    cout << "Amount: ";

                    cin >> tmpInt;

                    tmpInt = generateCoupon(tmpInt);

                    if(tmpInt){

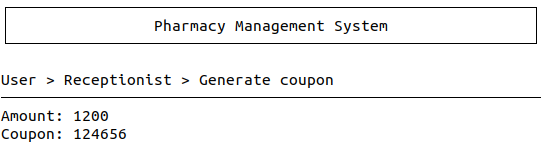
                        cout << "Coupon: " << tmpInt << endl;

                    }

                    else {

                        cout << "Invalid amout" << endl;

                    }



                }

                else if(option == 6){

                    break;

                }

                else {

                    cout << "Invalid option" << endl;

                }

                pause();

            }

        }

        else if(option == 3){

            while(true){

                option = customerMenu();

                if(option == 1){

                    banner("User > Customer > Buy product");

                    listItems();

                    sortArray(itemSold);

                    if(nStocks > 0){

                        cout << "Recommended Item: " << itemNames[0] << endl;

                    }

                    cout << "Item Name: ";

                    cin >> tmpString;

                    tmpInt = getItemIndex(tmpString);

                    if(tmpInt != -1){

                        cout << "Quantity: ";

                        cin >> tmpInt;

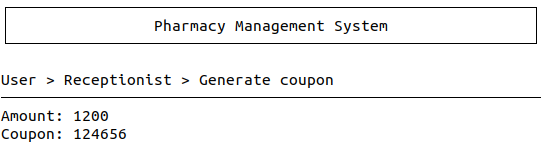
                        customerOrder(tmpString, tmpInt);

                    }

                    else {

                        cout << tmpString << " is not available here" << endl;

                    }



                }

                else if(option == 2){

                    banner("User > Customer > Drug dealer license");

                    cout << "The manager is not available right now" << endl;

                    cout << "Please provide us your email: ";

                    cin >> tmpString;

                    if(!addEmail(tmpString)){

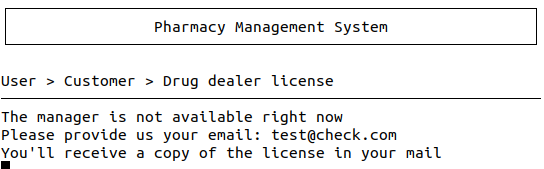
                        cout << "Something went wrong" << endl;

                    }

                    else {

                        cout << "You'll receive a copy of the license in your mail" << endl;

                    }



                    pause();

                }

                else if(option == 3){

                    banner("User > Customer > Redeem coupon");

                    cout << "Coupon: ";

                    cin >> tmpInt;

                    if(redeemCoupon(tmpInt)){

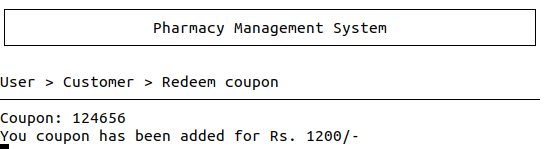
                        cout << "Your coupon has been added for Rs. " << couponAmout << "/-" << endl;

                    }

                    else {

                        cout << "Invalid coupon" << endl;

                    }



                }

                else if(option == 4){

                    banner("User > Customer > Return");

                    cout << "Product name: ";

                    cin >> tmpString;

                    tmpInt = getItemIndex(tmpString);

                    if(tmpInt != -1){

                        cout << "Quantity: ";

                        cin >> tmpInt;

                        cin.get(); // to read troublesome newline

                        cout << "Reason of return: ";

                        getline(cin, reasonReturn);

                        cout << "What do you want?" << endl

                            << "1. Cash" << endl

                            << "2. Coupon" << endl;

                        cin >> option;

                        if(option == 1){

                            productReturn(tmpString, tmpInt, true, reasonReturn);

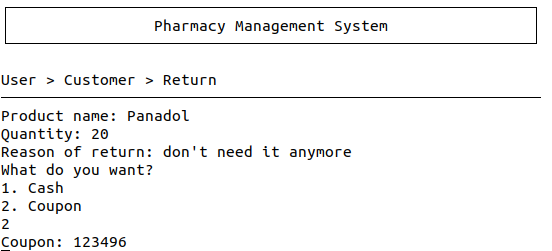
                        }

                        else {

                            productReturn(tmpString, tmpInt, false, reasonReturn);

                        }

                    }



                    else {

                        cout << "We don't sale " << tmpString << endl;

                    }

                }

                else if(option == 5){

                    banner("User > Customer > Product price");

                    cout << "Product name: ";

                    cin >> tmpString;

                    tmpInt = productPrice(tmpString);

                    if(tmpInt != 0){

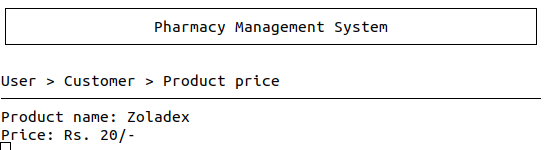
                        cout << "Price: Rs. " << tmpInt << "/-" << endl;

                    }

                    else {

                        cout << "We don't have " << tmpString << endl;

                    }



                }

                else if(option == 6){

                    banner("User > Customer > List of items");

                    listItems();

                }

                else if(option == 7){

                    banner("User > Customer > Feedback");

                    cin.get(); // to read troublesome newline

                    cout << "Enter feedback: ";

                    getline(cin, tmpString);

                    if(!addFeedback(tmpString)){

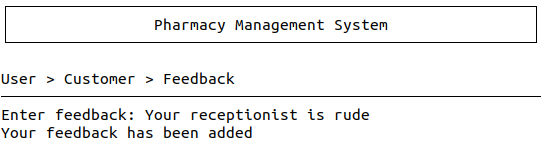
                        cout << "Something went wrong" << endl;

                    }

                    else {

                        cout << "Your feedback has been added" << endl;

                    }



                }

                else if(option == 8){

                    break;

                }

                else {

                    cout << "Invalid option";

                }

                pause();

            }

        }

        else if(option == 4){

            break;

        }

        else {

            cout << "Invalid option" << endl;

        }

        store();

        pause();

    }

}

void pause(){

    getch();

}

void header(){

    system("cls");

    wcout << L"\u00da\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00bf" << endl

         << L"\u00b3                Pharmacy Management System                \u00b3" << endl

         << L"\u00c0\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00d9" << endl << endl;

}

void banner(string s){

    header();

    cout << s << endl;

    wcout << L"\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4\u00c4" << endl;

}

int mainMenu(){

    int opt;

    cout << "Select User: " << endl

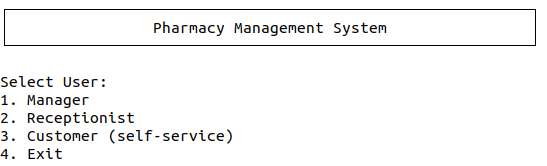
        << "1. Manager" << endl

        << "2. Receptionist" << endl

        << "3. Customer (self-service)" << endl

        << "4. Exit" << endl;

    cin >> opt;



    return opt;

}

int adminMenu(){

    int opt;

    banner("User > Manager");

    cout << "1. Check status of stocks" << endl

        << "2. Check total cash" << endl

        << "3. Add new stocks" << endl

        << "4. Add new items" << endl

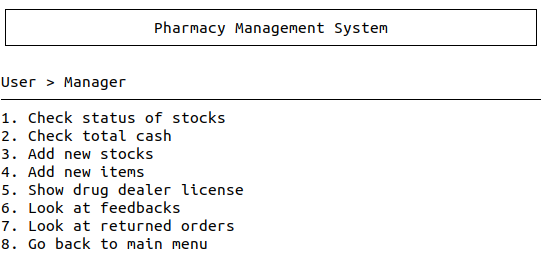
        << "5. Show drug dealer license" << endl

        << "6. Look at feedbacks" << endl

        << "7. Look at returned orders" << endl

        << "8. Go back to main menu" << endl;

    cin >> opt;



    return opt;

}

int userMenu(){

    int opt;

    banner("User > Receptionist");

    cout << "1. Place customer's order" << endl

        << "2. Decrease stocks" << endl

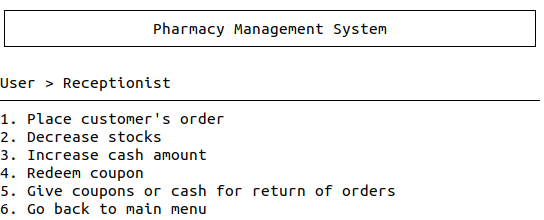
        << "3. Increase cash amount" << endl

        << "4. Redeem coupon" << endl

        << "5. Give coupons or cash for return of orders" << endl

        << "6. Go back to main menu" << endl;

    cin >> opt;



    return opt;

}

int customerMenu(){

    int opt;

    banner("User > Customer");

    cout << "1. Buy product" << endl

        << "2. Ask for drug dealer license" << endl

        << "3. Redeem coupon" << endl

        << "4. Return product" << endl

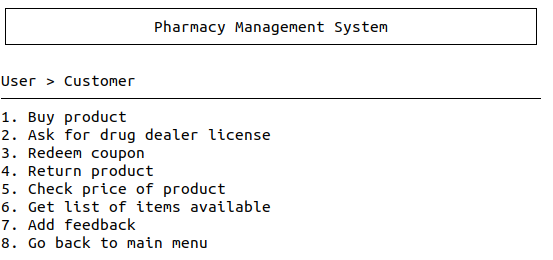
        << "5. Check price of product" << endl

        << "6. Get list of items available" << endl

        << "7. Add feedback" << endl

        << "8. Go back to main menu" << endl;

    cin >> opt;



    return opt;

}

void statusOfStocks(){

    if(nStocks != 0){

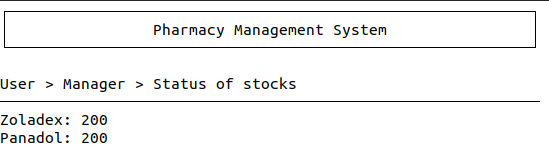
        sortArray(itemCounts);

        for(int i = 0; i < nStocks; i++){

            cout << itemNames[i] << ": " << itemCounts[i] << endl;

        }

    }



    else {

        cout << "No item in stock" << endl;

    }

}

void transactions(){

    if(nStocks != 0){

        sortArray(itemSold);

        for(int i = 0; i < nStocks; i++){

            cout << itemNames[i] << ": " << itemSold[i] << endl;

        }

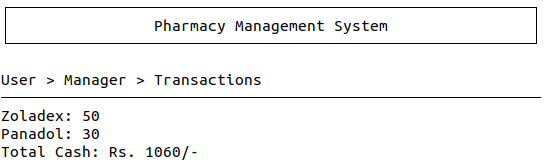
    }

    else {

        cout << "No item sold" << endl;

    }

    cout << "Total Cash: Rs. " << totalCash << "/-" << endl;



}

bool newStock(string drugName, int drugCount){

    int idx = getItemIndex(drugName);

    if(idx != -1){

        itemCounts[idx] += drugCount;

        return true;

    }

    else {

        return false;

    }

}

bool newItem(string drugName, int price, int count){

    if(nStocks < MAX\_ITEMS){

        itemNames[nStocks] = drugName;

        itemPrices[nStocks] = price;

        itemCounts[nStocks] = count;

        nStocks++;

        return true;

    }

    else {

        return false;

    }

}

bool isValidEMAIL(string text){

    int atIndex = -1, dotIndex = -1;

    for(int i = 0; i < text.length(); i++){

        if(text[i] == '@'){

            atIndex = i;

            break;

        }

    }

    if(atIndex != -1){

        for(int i = atIndex; i < text.length(); i++){

            if(text[i] == '.'){

                dotIndex = i;

                break;

            }

        }

    }

    if(atIndex != -1 && dotIndex != -1){

        return true;

    }

    return false;

}

bool addEmail(string email){

    if(isValidEMAIL(email) && emailCount < MAX\_ITEMS){

        emails[emailCount] = email;

        emailCount++;

        return true;

    }

    else {

        return false;

    }

}

void receptionistOrder(string drugName, int qty){

    int idx = getItemIndex(drugName);

    int total = 0;

    if(idx != -1){

        if(qty > itemCounts[idx]){

            cout << "We don't have that much in stock" << endl;

        }

        else {

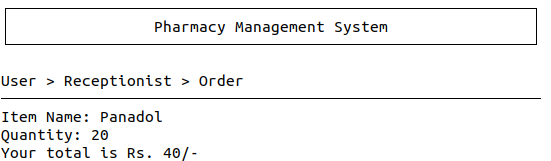
            itemCounts[idx] -= qty;

            itemSold[idx] += qty;

            total = qty \* itemPrices[idx];

        }

    }



    if(total != 0){

        cout << "Your total is Rs. " << total << "/-"<< endl;

        if(couponAmout != 0){

            cout << "Do you want to use your coupon on this order? (y/n): ";

            cin >> tmpString;

            if(tmpString == "y"){

                if(couponAmout > total){

                    couponAmout -= total;

                    total = 0;

                }

                else {

                    total -= couponAmout;

                    couponAmout = 0;

                }

                cout << "Your new total is Rs. " << total << "/-"<< endl;

            }

        }

        totalCash += total;

    }

}

bool decreaseStock(string drugName, int qty){

    int idx = getItemIndex(drugName);

    if(idx != -1){

        itemCounts[idx] -= qty;

        return true;

    }

    else {

        return false;

    }

}

void customerOrder(string drugName, int qty){

    int idx = getItemIndex(drugName);

    int total = 0;

    if(idx != -1){

        if(qty > itemCounts[idx]){

            cout << "We don't have that much in stock" << endl;

        }

        else {

            itemCounts[idx] -= qty;

            itemSold[idx] += qty;

            total = qty \* itemPrices[idx];

        }

    }

    if(total != 0){

        cout << "Your total is Rs. " << total << "/-"<< endl;

        if(couponAmout != 0){

            cout << "Do you want to use your coupon on this order? (y/n): ";

            cin >> tmpString;

            if(tmpString == "y"){

                if(couponAmout > total){

                    couponAmout -= total;

                    total = 0;

                }

                else {

                    total -= couponAmout;

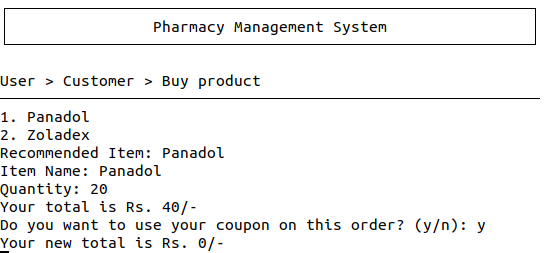
                    couponAmout = 0;

                }

                cout << "Your new total is Rs. " << total << "/-"<< endl;

            }

        }



        totalCash += total;

    }

}

void productReturn(string drugName, int qty, bool cash, string reason){

    int idx = getItemIndex(drugName);

    if(idx != -1){

        qty \*= itemPrices[idx];

        if(cash == true){

        cout << "Here's your Rs. " << qty << "/- return amount" << endl;

        totalCash -= qty;

        }

        else {

            qty += 123456;

            cout << "Coupon: " << qty << endl;

        }

        addReturn(idx, reason);

    }

}

bool addReturn(int index, string reason){

    if(returnCount < MAX\_ITEMS){

        orderReturns[returnCount] = itemNames[index] + " was returned because of " + reason;

        returnCount++;

        return true;

    }

    else {

        return false;

    }

}

void listItems(){

    sortArray(itemCounts);

    tmpInt = 0;

    for(int i = 0; i < nStocks; i++){

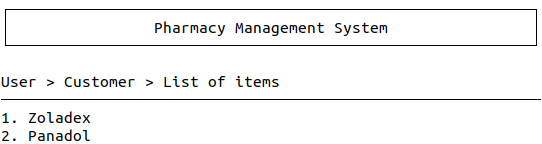
        if(itemCounts[i] != 0){

            cout << tmpInt + 1 << ". " << itemNames[i] << endl;

            tmpInt++;

        }

    }



}

int productPrice(string drugName){

    int idx = getItemIndex(drugName);

    if(idx != -1){

        return itemPrices[idx];

    }

    return 0;

}

bool addFeedback(string feedback){

    if(feedbackCount < MAX\_ITEMS){

        feedbackBox[feedbackCount] = feedback;

        feedbackCount++;

        return true;

    }

    else {

        return false;

    }

}

int getItemIndex(string name){

    for(int i = 0; i < nStocks; i++){

        if(itemNames[i] == name){

            return i;

        }

    }

    return -1;

}

void sortArray(int array[]){

    for(int i = 0; i < nStocks; i++){

        swap(i, getMaxIndex(array, i));

    }

}

int getMaxIndex(int array[], int start){

    int max = start;

    for(int i = start; i < nStocks; i++){

        if(array[i] > array[max]){

            max = i;

        }

    }

    return max;

}

void swap(int idx1, int idx2){

    tmpString = itemNames[idx1];

    itemNames[idx1] = itemNames[idx2];

    itemNames[idx2] = tmpString;

    tmpInt = itemPrices[idx1];

    itemPrices[idx1] = itemPrices[idx2];

    itemPrices[idx2] = tmpInt;

    tmpInt = itemCounts[idx1];

    itemCounts[idx1] = itemCounts[idx2];

    itemCounts[idx2] = tmpInt;

    tmpInt = itemSold[idx1];

    itemSold[idx1] = itemSold[idx2];

    itemSold[idx2] = tmpInt;

}

void store(){

    fstream file;

    file.open("projectData.txt", ios::out);

    for (int i = 0; i < nStocks; i++) {

        file << itemNames[i] << ","

            << itemPrices[i] << ","

            << itemCounts[i] << ","

            << itemSold[i] << endl;

    }

    file << endl;

    for (int i = 0; i < feedbackCount; i++) {

        file << feedbackBox[i] << endl;

    }

    file << endl;

    for (int i = 0; i < returnCount; i++){

        file << orderReturns[i] << endl;

    }

    file << endl;

    for (int i = 0; i < emailCount; i++){

        file << emails[i] << endl;

    }

    file.close();

}

void load(){

    fstream file;

    string line = "";

    nStocks = 0;

    feedbackCount = 0;

    returnCount = 0;

    emailCount = 0;

    file.open("projectData.txt", ios::in);

    while(!file.eof()){

        getline(file, line);

        if(line == ""){

            break;

        }

        itemNames[nStocks] = getField(line, 0);

        itemPrices[nStocks] = convertToInt(getField(line, 1));

        itemCounts[nStocks] = convertToInt(getField(line, 2));

        itemSold[nStocks] = convertToInt(getField(line, 3));

        totalCash += itemSold[nStocks] \* itemPrices[nStocks];

        nStocks++;

    }

    while(!file.eof()){

        getline(file, line);

        if(line == ""){

            break;

        }

        feedbackBox[feedbackCount] = line;

        feedbackCount++;

    }

    while(!file.eof()){

        getline(file, line);

        if(line == ""){

            break;

        }

        orderReturns[returnCount] = line;

        returnCount++;

    }

    while(!file.eof()){

        getline(file, line);

        if(line == ""){

            break;

        }

        emails[emailCount] = line;

        emailCount++;

    }

    file.close();

}

string getField(string record, int field){

    int seps = 0;

    string ret = "";

    char sep = ',';

    for(int i = 0; i < record.length(); i++){

        if(record[i] == sep){

            seps++;

            continue;

        }

        else if(seps == field){

            ret += record[i];

        }

        else if(seps > field){

            break;

        }

    }

    return ret;

}

int convertToInt(string orig){

    int n = 0, idx = 0;

    int sign = 1;

    bool signChecked = false;

    while(orig[idx] != '\0'){

        if(orig[idx] == ' '){

            continue;

        }

        if((orig[idx] == '-' || orig[idx] == '+') && !signChecked){

            if(orig[idx] == '-'){

                sign = -1;

            }

            signChecked = true;

        }

        else if(orig[idx] >= '0' && orig[idx] <= '9'){

            n = n \* 10 + orig[idx] - '0';

        }

        else {

            break;

        }

        idx++;

    }

    return sign \* n;

}

void drugDealerLicense(){

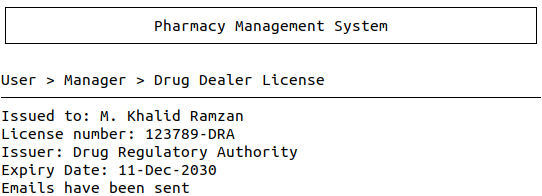
    cout << "Issued to: M. Khalid Ramzan" << endl

        << "License number: 123789-DRA" << endl

        << "Issuer: Drug Regulatory Authority" << endl

        << "Expiry Date: 11-Dec-2030" << endl;

    cout << "Emails have been sent" << endl;



}

void listFeedbacks(){

    if(feedbackCount > 0){

        for(int i = 0; i < feedbackCount; i++){

            cout << i + 1 << ". " << feedbackBox[i] << endl;

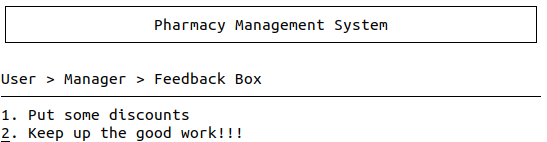
        }

    }

    else {

        cout << "No feedbacks yet" << endl;

    }



}

void listReturns(){

    if(returnCount > 0){

        for(int i = 0; i < returnCount; i++){

            cout << i + 1 << ". " << orderReturns[i] << endl;

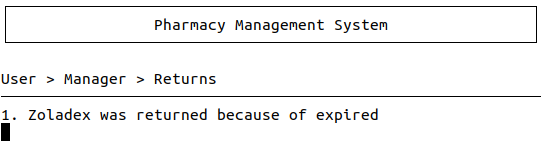
        }

    }

    else {

        cout << "No product has been returned yet" << endl;

    }



}

bool redeemCoupon(int coupon){

    coupon -= 123456;

    if(coupon < 0){

        return false;

    }

    couponAmout += coupon;

    return true;

}

int generateCoupon(int amount){

    if(amount > 0){

        return 123456 + amount;

    }

    return 0;

}

**Test Cases:**

* banner(“User > Manager”) ┌────────────────────────────────────────────────┐

│ Pharmacy Management System │

└────────────────────────────────────────────────┘

User > Manager

──────────────────────────────────────────────────

* isValidEMAIL(“test@abc.com”) -> true
* isValidEMAIL(“test.com”) -> false
* addEmail(“test.com”) -> false
* addEmail(“test@abc.com”) -> true

**Weakness:**

* The program cannot handle products more than MAX\_ITEMS.
* It doesn’t allow handling of multiple receptionists and their individual productivity.
* It doesn’t have any functionality for discounts.
* It doesn’t save information regarding each customer

**Conclusion:**

Through this project I learnt about handling of shared resources in a multi-user environment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting **Grade:** | A | Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | B | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | A | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | A | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | A | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure (Arrays)  **Grade:** | A | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | A | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity  **Grade:** | A | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | A | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendation Feature | A | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and Demo  **Grade:** | A | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | A | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | All Code style criteria is followed | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure (Arrays)  **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | Sort working 100% and generating useful report | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendation Feature | Proper meaning full recommendation is present into system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |

**2021-CS-62:**   **Syed Abdul Rehman**

|  |  |
| --- | --- |
| **Checked by:** |  |