#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdbool.h>  
#include <time.h>  
#include <errno.h>  
#include "main.h"  
  
Car \*cars; // intitialise the cars from struct  
Sale \*sales; // same with sales  
int salesCount = 0; // counter for the number of sales recorded  
bool needSorting = true; // a bool used to determine if the list needs sorting  
 // off by default  
  
int main() {  
 //memory allocation for the cars  
 cars = (Car \*)malloc(sizeof(Car) \* MAX\_CARS);  
 //and for the sales  
 sales = (Sale \*)malloc(sizeof(Sale) \* MAX\_SALES);  
 //checks to make sure it worked, if not it informs the user  
 // and shuts down the program  
 if (cars == NULL || sales == NULL) {  
 printf("Error: Memory allocation failed.\n");  
 return 1;  
 }  
  
 initialiseCars(); // initialize the car stock data  
 loadSalesData();  
 // load existing sales data from file  
 printf("\nWelcome to Los Pollos Hermanos!\n");  
 char choice;  
 do {  
 clearConsole(); // for user benefit  
 // Display the main menu  
 printf("\nMenu:\n");  
 printf("1. View Cars Stock\n");  
 printf("2. Buy Car\n");  
 printf("3. View Sales Data\n");  
 printf("4. Leave Feedback\n");  
 printf("5. Exit\n");  
  
 //input validation variable  
 bool validInput = false;  
 //does this until the user selects a valid input  
 do {  
 printf("Choose an option: ");  
 // if the input is not in the correct parameters the user is told  
 if (scanf(" %c", &choice) != 1 || (choice < '1') || (choice > '5')) {  
 printf("Invalid input. Please try again.\n");  
 } else { // otherwise if its correct  
 validInput = true; // bool variable changed  
 clearInputBuffer();  
 clearConsole();  
 break; // and code breaks  
 }  
 }while (validInput == false);  
  
 // Handle the user's choice  
 switch (choice) {  
 case '1':  
 viewCarStock(); // show available car stock  
 break;  
 case '2':  
 buyCar(); // handle car purchase  
 break;  
 case '3':  
 viewSalesData(); // display sales data  
 break;  
 case '4':  
 leaveFeedback(); // collect user feedback  
 break;  
 case '5':  
 // save sales data to file before exiting  
 printf("Exiting program. Goodbye!\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 pauseProgram();  
 }  
 } while (choice != '5'); // loop until the user chooses to exit  
 //free the memory  
 free(cars);  
 free(sales);  
  
 return 0;  
}  
  
// initialize the car stock with predefined values  
// follows the struct formmat of index, name, year,  
// price and stock  
void initialiseCars() {  
 strcpy(cars[0].model, "Audi A3");  
 cars[0].year = 2019;  
 cars[0].price = 20000.0f;  
 cars[0].remaining = 5;  
 strcpy(cars[1].model, "BMW X1");  
 cars[1].year = 2020;  
 cars[1].price = 25000.0f;  
 cars[1].remaining = 3;  
 strcpy(cars[2].model, "Mercedes C-Class");  
 cars[2].year = 2018;  
 cars[2].price = 30000.0f;  
 cars[2].remaining = 2;  
 strcpy(cars[3].model, "Volkswagen Golf");  
 cars[3].year = 2017;  
 cars[3].price = 18000.0f;  
 cars[3].remaining = 4;  
 strcpy(cars[4].model, "Ford Focus");  
 cars[4].year = 2019;  
 cars[4].price = 17000.0f;  
 cars[4].remaining = 6;  
 strcpy(cars[5].model, "Toyota Corolla");  
 cars[5].year = 2021;  
 cars[5].price = 22000.0f;  
 cars[5].remaining = 5;  
 strcpy(cars[6].model, "Honda Civic");  
 cars[6].year = 2020;  
 cars[6].price = 21000.0f;  
 cars[6].remaining = 7;  
 strcpy(cars[7].model, "Nissan Qashqai");  
 cars[7].year = 2018;  
 cars[7].price = 19000.0f;  
 cars[7].remaining = 3;  
 strcpy(cars[8].model, "Kia Sportage");  
 cars[8].year = 2021;  
 cars[8].price = 23000.0f;  
 cars[8].remaining = 4;  
 strcpy(cars[9].model, "Hyundai Tucson");  
 cars[9].year = 2020;  
 cars[9].price = 24000.0f;  
 cars[9].remaining = 5;  
}  
  
// load sales data from a file  
void loadSalesData() {  
 //opens file in read only mode  
 FILE \*file = fopen(SALES\_FILE, "r");  
 // if it cant open it, lets the user know and throws error code  
 if (file == NULL) {  
 printf("Warning: Could not open sales data file (%s). Reason: %s\n", SALES\_FILE, strerror(errno));  
 pauseProgram();  
 return;  
 }  
 // this is a command to read the file  
 // i saw a line of code in the week 6 dealing with user input that was %[^\n] and  
 // researched it and found more commands like the ones i am using. so i decided to use  
 // it and its set out to read until a comma and until a new line. i just thought it was  
 // cool  
 while (fscanf(file, "%[^,],%d,%[^,],%f,%d,%d,%[^\n]\n",  
 //reads all until the end of file  
 sales[salesCount].customerName, &sales[salesCount].customerAge,  
 sales[salesCount].carModel, &sales[salesCount].totalPrice,  
 &sales[salesCount].discountGiven, &sales[salesCount].numberOfCars,  
 sales[salesCount].date) != EOF) {  
 // decrypt the data after reading  
 handleEncryption(sales[salesCount].customerName);  
 handleEncryption(sales[salesCount].carModel);  
 handleEncryption(sales[salesCount].date);  
  
 salesCount++; // increases the index  
  
 // if it exceeds max it lets the user know and breaks the while loop  
 if (salesCount >= MAX\_SALES) {  
 printf("Warning: Sales data exceeds maximum limit.\n");  
 break;  
 }  
 }  
 //closes the file to prevent issues trying to access later  
 fclose(file);  
}  
  
// save sales data to a file  
void saveSalesData() {  
 // file is opened in write mode  
 // if the file is not there its automatically created  
 FILE \*file = fopen(SALES\_FILE, "w");  
 //if the file for some reason doesnt work it informs the user  
 if (file == NULL) {  
 printf("Error: Could not save sales data.\n");  
 pauseProgram();  
 return;  
 }  
 // iterates through all the data  
 for (int i = 0; i < salesCount; i++) {  
 //encrypts it  
 handleEncryption(sales[i].customerName);  
 handleEncryption(sales[i].carModel);  
 handleEncryption(sales[i].date);  
 //writes it to file  
 fprintf(file, "%s,%d,%s,%.2f,%d,%d,%s\n",  
 sales[i].customerName, sales[i].customerAge,  
 sales[i].carModel, sales[i].totalPrice,  
 sales[i].discountGiven, sales[i].numberOfCars,  
 sales[i].date);  
  
 // decrypts it back for readability  
 handleEncryption(sales[i].customerName);  
 handleEncryption(sales[i].carModel);  
 handleEncryption(sales[i].date);  
 }  
 //closes the file so it can be opened again  
 fclose(file);  
}  
  
// display the current car stock  
void viewCarStock() {  
 sortCarsByYearDescending(); // sort cars by year (newest first)  
 printf("\nCar Stock:\n");  
 for (int i = 0; i < MAX\_CARS && strlen(cars[i].model) > 0; i++) {  
 printf("Model: %s, Year: %d, Price: %.2f, Remaining: %d\n", cars[i].model, cars[i].year, cars[i].price, cars[i].remaining);  
 }  
}  
  
// handle a car purchase  
void buyCar() {  
 char customerName[MAX\_TEXT\_LENGTH];  
 getValidatedString("Enter your name: ", customerName, MAX\_TEXT\_LENGTH); // get customers name  
 int customerAge = getValidatedInt("Enter your age: ", 0, 120); // get customers age  
  
 // show available cars using this instead of the method  
 // this is because when they are added in they arent sorted in the struct so if i  
 // were to sort them here then the index wouldnt be in order confusing the user  
 printf("\nAvailable Cars:\n");  
 for (int i = 0; i < MAX\_CARS && strlen(cars[i].model) > 0; i++) {  
 printf("%d. %s (%d) - $%.2f, Remaining: %d\n", i + 1, cars[i].model, cars[i].year, cars[i].price, cars[i].remaining);  
 }  
  
 // get the car choice and number of cars to buy  
 int carChoice = getValidatedInt("Choose a car to buy (enter number): ", 1, MAX\_CARS) - 1;  
 // if the car is out of stock then it stops  
 if (cars[carChoice].remaining <= 0) {  
 printf("Sorry, that car is out of stock.\n");  
 pauseProgram();  
 return;  
 }  
  
 clearConsole(); // for user benefit  
  
 // gets the number of cars the user would like to buy  
 int numberOfCars = getValidatedInt("How many cars would you like to buy? ", 1, cars[carChoice].remaining);  
  
 clearConsole(); // celars console for user benefit  
  
 // calculate total price and apply discount if eligible  
 float totalPrice = cars[carChoice].price \* numberOfCars;  
 printf("Total Price: %.2f\n", totalPrice); // prints price before any discounts  
  
 // checks against the min and max to aget to see if they receive a discount  
 bool discountGiven = (customerAge >= DISCOUNT\_MIN\_AGE && customerAge <= DISCOUNT\_MAX\_AGE);  
 // if discount is given it informs the user  
 if (discountGiven) {  
 //walks the user step by step through the discount  
 totalPrice \*= (1 - DISCOUNT\_PERCENTAGE);  
 printf("You received a discount!\n");  
 printf("Discount for %.2f applied \n", DISCOUNT\_PERCENTAGE);  
 printf("Final Price after discount %.2f\n", totalPrice);  
 }  
 else {  
 printf("You have not received a discount. \n");  
 }  
  
 // update car stock and add the sale record  
 cars[carChoice].remaining -= numberOfCars;  
  
 // record the sale date  
 time\_t t = time(NULL);  
 struct tm tm = \*localtime(&t);  
 char date[MAX\_TEXT\_LENGTH];  
 sprintf(date, "%02d-%02d-%d", tm.tm\_mday, tm.tm\_mon + 1, tm.tm\_year + 1900);  
  
 // save the sale details  
 Sale newSale = {"", customerAge, "", totalPrice, discountGiven, numberOfCars, ""};  
 strcpy(newSale.customerName, customerName);  
 strcpy(newSale.carModel, cars[carChoice].model);  
 strcpy(newSale.date, date);  
 sales[salesCount++] = newSale;  
  
 clearConsole(); //for user benefit  
 printf("Purchase successful!\n");  
 saveSalesData(); // save sale data after every purchase  
}  
  
// view all recorded sales data  
void viewSalesData() {  
 sortSalesByTotalPriceDescending(); // sort sales by total price (highest first)  
 //checks to see there actually is some sales data  
 if (salesCount == 0) {  
 // if not it informs the user  
 printf("\nNo sales data available.\n");  
 return;  
 }  
  
 // if there is sales data its sorted by car model using this  
 printf("\nSales Data by Car Model:\n");  
 for (int i = 0; i < salesCount; i++) {  
 printf("Customer: %s, Car: %s, Total Price: %.2f, Date: %s\n",  
 sales[i].customerName, sales[i].carModel, sales[i].totalPrice, sales[i].date);  
 }  
}  
  
// collect feedback from the user  
void leaveFeedback() {  
 // because feedback is added on the file is opened in append mode  
 FILE \*file = fopen(FEEDBACK\_FILE, "a");  
 // if it cannot be opened it informs the user  
 if (file == NULL) {  
 printf("Error: Could not save feedback.\n");  
 pauseProgram();  
 return;  
 }  
  
 // this is so it can be associated to a car model  
 int carChoice;  
 //iterates through every car model  
 printf("\nAvailable Cars:\n");  
 for (int i = 0; i < MAX\_CARS && strlen(cars[i].model) > 0; i++) {  
 printf("%d. %s \n", i + 1, cars[i].model, cars[i].year);  
 }  
 //takes the car choice  
 carChoice = getValidatedInt("To select a car to leave feedback enter a number: ", 1, MAX\_CARS - 1);  
  
 //takes feedback using max text length  
 char feedback[MAX\_TEXT\_LENGTH];  
 getValidatedString("Please enter a short feedback: ", feedback, MAX\_TEXT\_LENGTH);  
 //writes to file then closes  
 fprintf(file, "Model: %s, Feedback: %s\n", cars[carChoice].model, feedback);  
 fclose(file);  
  
 printf("Thank you for your feedback!\n");  
 printf("The model selected was %s!\n", cars[carChoice].model);  
}  
  
// validate and get an integer input from the user  
int getValidatedInt(const char \*prompt, int min, int max) {  
 int value;  
 do {  
 printf("%s", prompt);  
 //if input is invalid the user is told  
 if (scanf("%d", &value) != 1) {  
 printf("Invalid input. Please enter a number.\n");  
 clearInputBuffer();  
 continue;  
 }  
 //same with if the input is out of range  
 if (value < min || value > max) {  
 printf("Input out of range. Please enter a value between %d and %d.\n", min, max);  
 }  
 } while (value < min || value > max);  
 clearInputBuffer(); //clears the buffer  
 return value;  
}  
  
// validate and get a string input from the user  
void getValidatedString(const char \*prompt, char \*input, int length) {  
 // do while loop to ensure the input is correct  
 do {  
 //prints the prompt  
 printf("%s", prompt);  
  
 //if its not null it proceeds  
 if (fgets(input, length,stdin)!= NULL) {  
 size\_t len = strcspn(input, "\n"); // find newline character  
 // if its empty it informs the user  
 if (len == 0) {  
 printf("Input can't be empty, try again \n");  
 continue;  
 }  
 //if its too long it informs the user  
 if (len>= length -1) {  
 printf("Input too long, try again \n");  
 clearInputBuffer();  
 continue;  
 }  
 input[len] = '\0'; // null termination  
 break;  
 }  
 }while(1); // this ensures the loop runs until a valid input is provided  
 // AI taught me this method of loop as its supposedly more efficient than declaring a bool, it  
 // saves the management of the varible as this does the same thing  
}  
  
  
// sort cars by year in descending order using a bubble sort  
//using Raz's simple method  
void sortCarsByYearDescending() {  
 // check to see if it needs sorting  
 if (!needSorting) {  
 return; // if it doesnt it stops  
 }  
 for (int i = 0; i < MAX\_CARS - 1; i++) {  
 for (int j = i + 1; j < MAX\_CARS; j++) {  
 if (cars[j].year > cars[i].year) {  
 Car temp = cars[i];  
 cars[i] = cars[j];  
 cars[j] = temp;  
 }  
 }  
 }  
 // after the sort considering that its stored in the array its marked as sorted  
 needSorting = false;  
}  
  
// sort sales by total price in descending order  
// also credit to Raz  
void sortSalesByTotalPriceDescending() {  
 for (int i = 0; i < salesCount - 1; i++) {  
 for (int j = i + 1; j < salesCount; j++) {  
 if (sales[j].totalPrice > sales[i].totalPrice) {  
 Sale temp = sales[i];  
 sales[i] = sales[j];  
 sales[j] = temp;  
 }  
 }  
 }  
}  
  
//method to handle encryption and decryption  
// too much research went into this  
void handleEncryption(char \*data) {  
 //takes the length to check if theres data in to encrypt  
 size\_t dataLen = strlen(data);  
  
 // check if data or dataSize is valid  
 // if not the user is informd  
 if (data == NULL || dataLen == 0) {  
 fprintf(stderr, "Error: Invalid data or size for encryption.\n");  
 pauseProgram();  
 return;  
 }  
  
 // xor encryption  
 size\_t keyLength = strlen(SECUREKEY);  
 unsigned char \*byteData = (unsigned char \*)data;  
  
 //stack overflow and gpt failed to explain this in a way i understand  
 for (size\_t i = 0; i < dataLen; i++) {  
 byteData[i] ^= SECUREKEY[i % keyLength];  
 }  
}  
  
// clear the input buffer to handle extraneous characters  
void clearInputBuffer() {  
 int c;  
 while ((c = getchar()) != '\n' && c != EOF) {  
 // discards the remaining characters in the buffer  
 }  
}  
  
//clear the console  
void clearConsole() {  
 system("clear");  
}  
void pauseProgram() {  
 printf("\n\nPress Enter to return to the Menu...");  
 getchar();  
}