Картина, която съдържа текст, графична колекция

Описанието е генерирано автоматичноТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „КНТ“

СЕМЕСТРИАЛНА ДОМАШНА РАБОТА

по дисциплина “Базово програмиране” на тема “Информационна система Коли под наем”

Вариант 39

Изготвил: Цветан Цветанов Проверил:

Специалност: КСТ

Група: II

Факултетен номер: 23621411

2024

Съдържание:  
I. Задание на проекта - 3,4 стр  
II. Анализ на решението - 5, до 15стр  
1. Константи, Изброими типове и Структури за данните в програмата  
2. Реализация на условие A  
3. Реализация на условие B   
Реализация на условие C   
Реализация на условие D   
Реализация на условие E   
Реализация на условие F  
Реализация на условие … - допълнение първо  
Реализация на условие … - допълнение второ.   
Реализация на допълнение трето   
III. Примерно действие на програмата. – 15, до 26стр  
Условие A  
Условие B  
Условие C  
Условие D  
Условие E  
Условие F   
Допълнение първо   
Допълнение второ  
Допълнение трето  
IV. ПРИЛОЖЕНИЕ 1 - от 27 до 41  
Изходен код на програмата

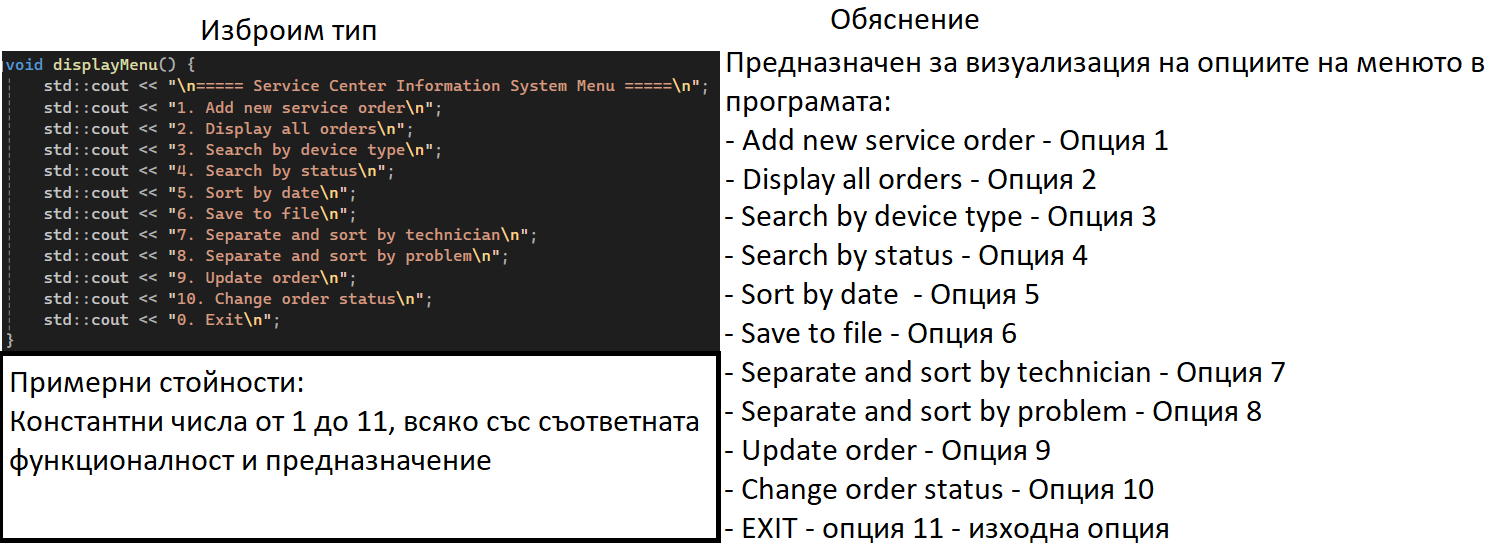
**ЗАДАНИЕ НА ПРОЕКТА**   
Информационна система Сервиз за техника  
- Да се напише компютърна програма, реализираща информационна система за сервиз за техника. Програмата съхранява информация за извършените сервизни поръчки в рамките на даден месец (номер на поръчката, ден от месеца, клиент, тип устройство (телефон, лаптоп, телевизор и др.), сериен номер на устройството, проблем според клиента (до 50 символа), име на сервизния техник, извършен ремонт (до 50 символа), цена, дни престой в сервиза, статус: приета/отказана/завършена). Предполагаемите проблеми и извършените ремонти да бъдат предварително декларирани в отделни списъци с максимален брой по 10 за всеки списък. Максималния брой сервизни поръчки е 100.   
Базова задача   
A. Меню за избор на функциите от програмата Функции от програмата са:   
B. Добавяне на нова сервизна поръчка: a. Добавяне на няколко поръчки, като не трябва да се превишава максималният брой (100). Пример: Добавяне на списък от поръчки. Въвежда се цяло число n и след него n на брой поръчки. n не може да надвишава свободните елементи в списъка с поръчки. \*При добавяне на поръчка се въвежда част от информацията (номер на поръчка, вид устройство, номер, и т.н.). Име на сервизен техник, извършен ремонт и т.н. се актуализират след извършения ремонт!   
C. Извеждане на всички устройства на екрана: a. Извеждане на всички устройства, чрез подходящо форматиране в таблица   
D. Търсене и извеждане на екрана: a. Търсене и извеждане на устройства по вид b. Търсене и извеждане на устройства по статус   
E. Сортиране и извеждане на поръчки на екран: a. Сортиране на поръчките по дата на приемане   
F. Работа с външен файл(двоичен) a. Извеждане на масива с поръчки във файл. b. Въвеждане на масива с поръчки от файл.   
Допълнение първо (+ базова задача):  
G. Да се създаде подменю към основното с нови функции за:   
a. Отделяне на завърщените поръчки на определен техник и сортирани по ден на приемане.   
  
b. Отделяне на поръчките с определен проблем за дадено устройство (въвежда се от потребителя) и сортиране в намаляващ ред по тип на устройството.  
   
Допълнение второ (+ базова задача)   
H. Въвеждане/Актуализация на данни за сервизна поръчка: a. Добаване в структурата на поле вид на поръчката (нормална – за 10 дена, бърза за 5 дена, експресна за 2 дена) b. Въвеждане/Актулизацията на данни (за извършен ремонт) става по въведен номер на сервизната поръчйа.   
Ако поръчката присъства в списъка(масива):   
a. Проверява се статуса й: Приета поръчка – се актуализира информацията за извършеният ремонт и се пресмята цената (Изчисляване на цена: всеки тип ремонт има цена (в лв.) + процент от цената на ремонта за вида поръчка (експресна - 50%, бърза - 20%, нормална - 0% от цената на ремонт). Отказана/завършена – НЕ може да се актуалириза информация   
b. Ако не присъства в масива, се извежда пояснително съобщение на екрана. I. Смяна на статус на поръчка: Въвежда се номерът на поръчката (ако я има се въвежда и новият статус). Ако статусът е приета се сменя с новият. Ако статусът е отказана/завършена НЕ СЕ допуска промяна! Допълнение трето (+ базова задача)   
J. Данните в програмата да се попълват автоматично от файл при стартиране и да се записват автоматично във файл при затваряне на програмата.

**АНАЛИЗ НА РЕШЕНИЕТО**1. Константи, Изброими типове и Структури за данните в програмата

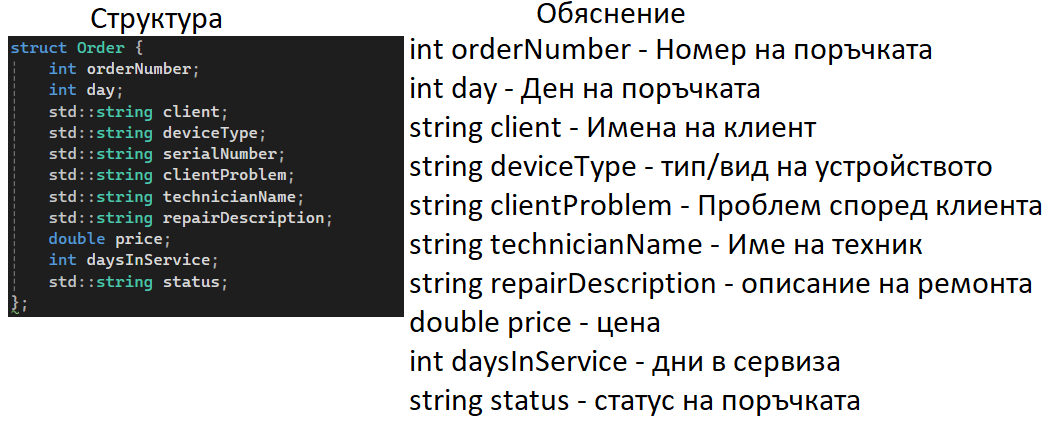
const int MAX\_ORDERS = 100; - Константа за максимален брой поръчки

const int MAX\_PROBLEMS = 10; - Константа за максимален брой проблеми

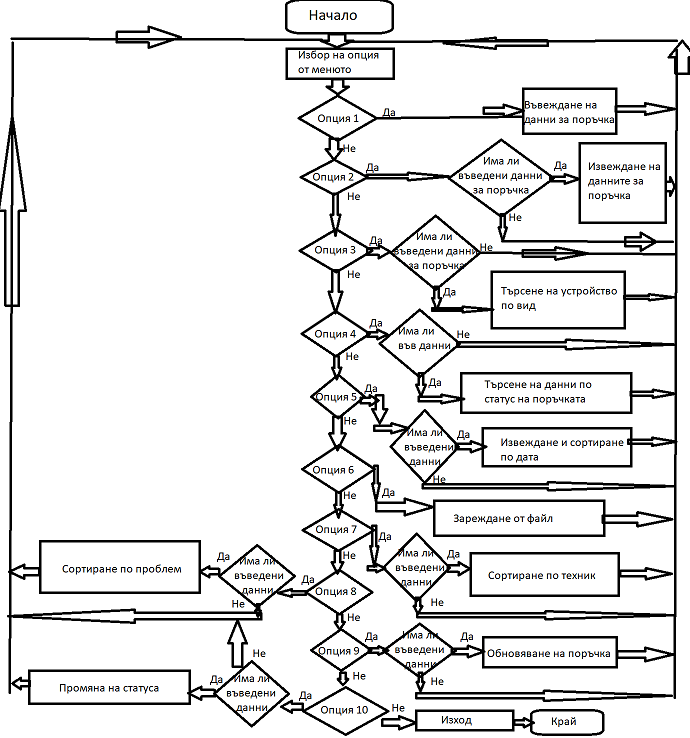
const int MAX\_REPAIRS = 10; - Константа за максимален брой ремонти

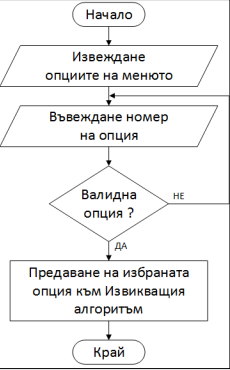


Структури



**Реализация на Условие А**

****

****

Функция с която е реализиран алгоритъма:  
int main()

Изходни данни на функцията:  
Главната функция организира “display menu“ функция и възприема изборът на потребителя от клавиатурата, следователно спрямо избора се извиква допълнителната функция, която ще се употребява.

* displayMenu()

int choice – избор на клиента спрямо опциите в менюто. Има и валидация за „коректен“ избор.

**Реализация на условие B**

Алгоритъм за добавяне на поръчка:  
Изисква се потребителят да въведе данни за поръчка описани в струкура Order:  
Въвеждат се, номер на поръчката, вид на устройството, сериен номер и поръчката е добавена

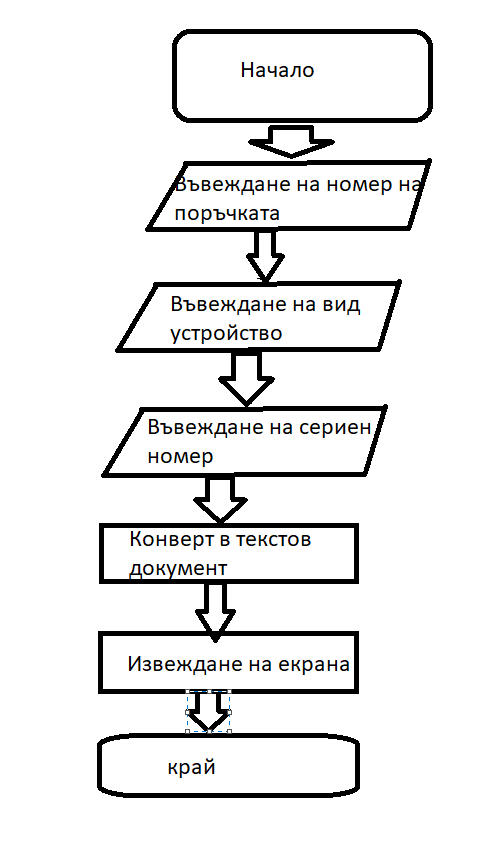
Функция за реализацията:  


Входни данни на функцията:

Order orders[] – променлива тип Структура Order

Int& orderCount – брояч на поръчките за да няма надвишавания в броя им

Следва блок схема на реализацията:

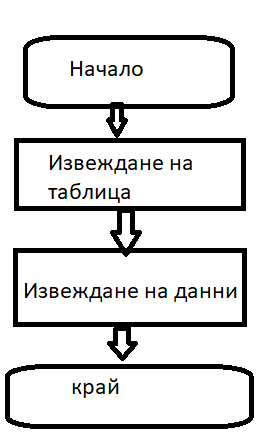


**Реализация на условие C**

При добавени данни, те се извеждат под формата на таблица на екрана. Ако няма добавени данни това което ще се изведе ще бъде празна бланка със сервизните поръчки, защото няма такива и е необходимо добавянето им.

За форматирането в таблица и извеждането отговаря следната функция:  


Блок схема на алгоритъма



Входни данни на функцията:

Const Order orders[] - списък с поръчките

Int orderCount – брояч на поръчките

Няма изходни данни, функцията отпечатва поръчките в табличен вид.

**Реализация на условие D**

Функции използвани за сортиране на поръчките по:

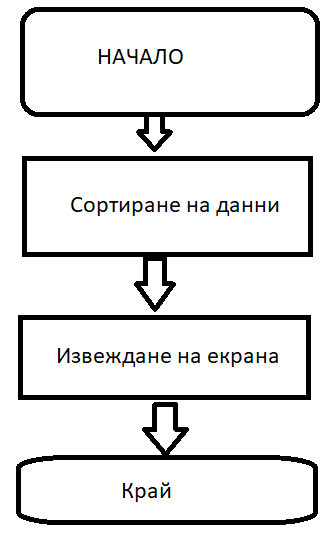


* Имена на техник



* По вид на проблема

Блок схема на алгоритъма



**Реализация на условие Е**

От меню опциите се избира опция за съответната сортировка,  
номер 5 за сортиране по дата.  
Следователно данните се извеждат таблично като показва броят на поръчките и датата им.

Функция за реализацията на алгоритъма

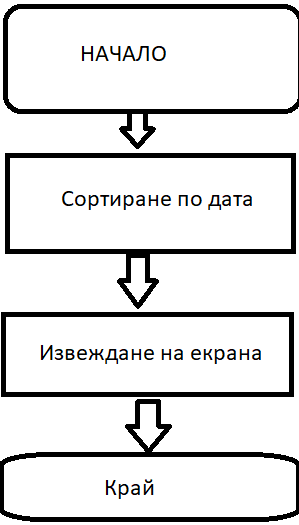


Order orders [] – броят на въведените поръчки

Int orderCount – брояч на въведените поръчки

Функцията няма изходни данни, извежда данните таблично по дата на прием.

Блок схема



**Реализация на условие F**

Алгоритъм за извеждане на поръчките във Файл

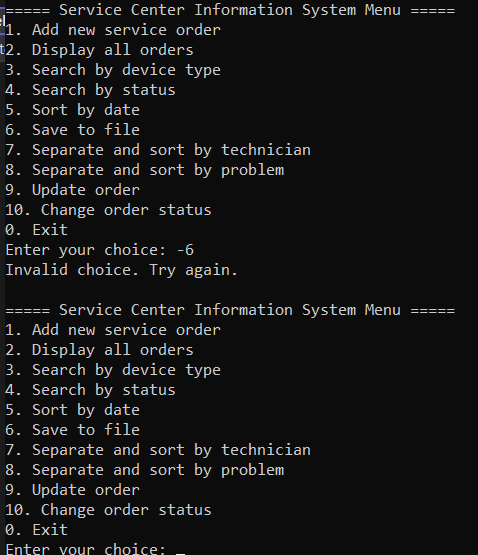
Данните се записват в текстов файл чрез избор на опция от главното меню

Функция която се използва:



**Примерно действие на програмата**

**A)**

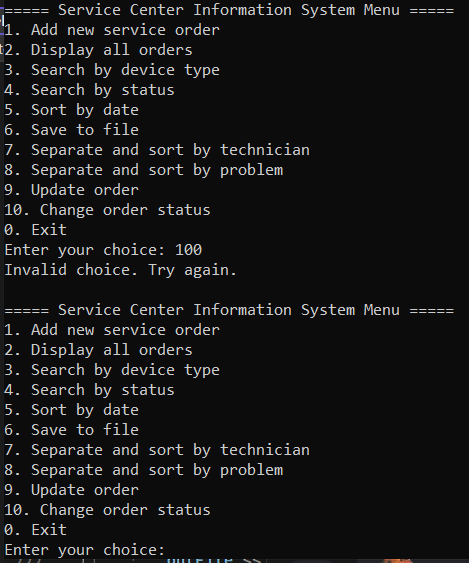
Възможни случаи: 

Въвеждане на негативна стойност: Отрицателно число

В този случай програмата ще върне отговор: Инвалиден избор моля опитайте отново. Докато не се въвете коректен избор.

Случай 2:

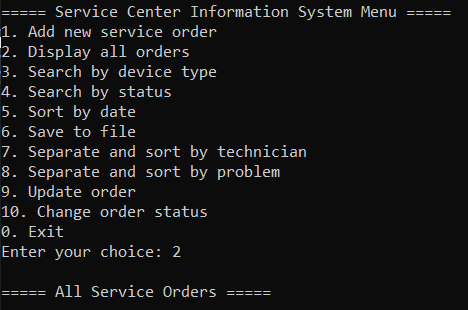
Въвеждане на негативна стойност: Х(във число) > от максимума



В този случай програмата ще върне отговор: Инвалиден избор моля опитайте отново. Докато не се въвете коректен избор.

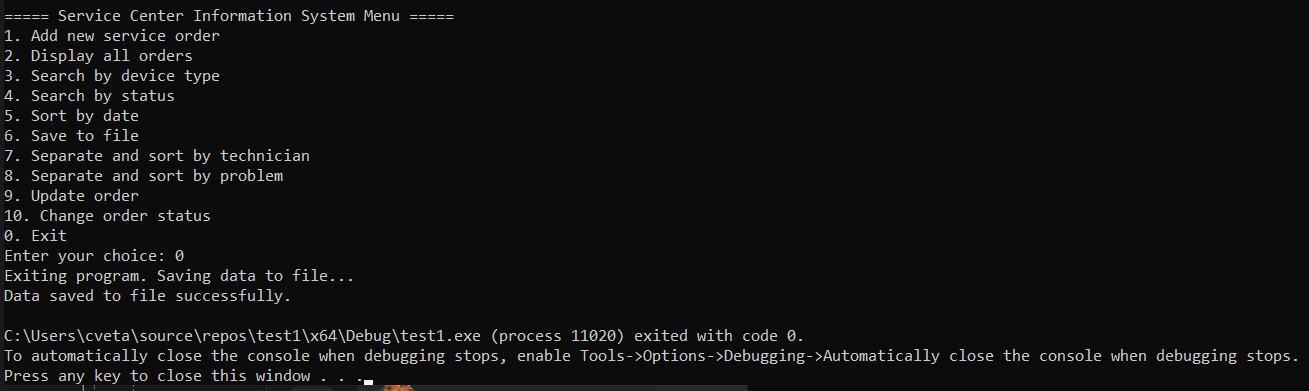
Случай 3:

Въвеждане на регламентирано число от 1 до 10



Програмата връща отговор със съответния избор и продължава спрямо развитието на ситуацията.

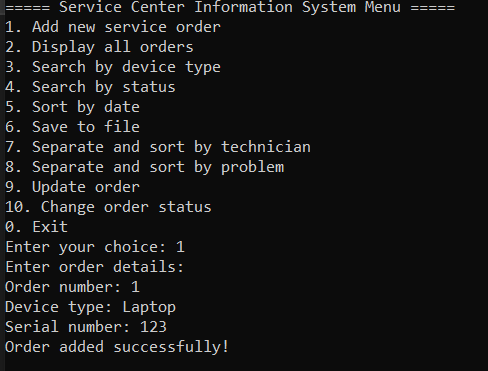
Случай 4:



Изходен вариант – 0.

Изход от програмата

**Условие B)**



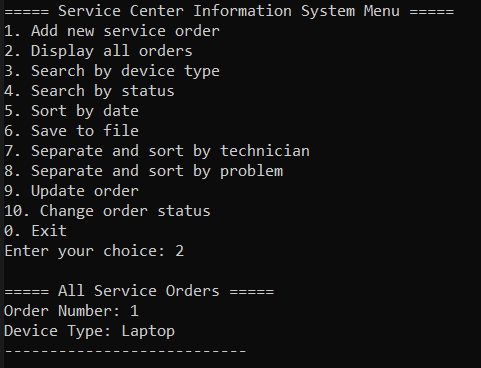
Необходими са данни за въвеждане, при което поръчката се добавя:

Примерни данни:

Номер на поръчка – 1,  
Вид на устройство – Лаптоп,  
Сериен номер – 123,

**Условие C)**

**Случай 1:**

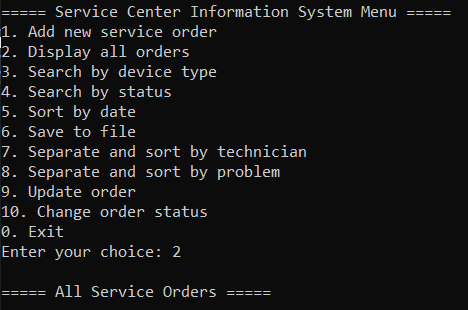


При избор за извеждане на направените поръчки се извеждат таблично една под друга.

Като са използвани примерни стойности за реализацията

Номер на поръчка: 1  
Вид на техниката: Лаптоп

Случай 2:

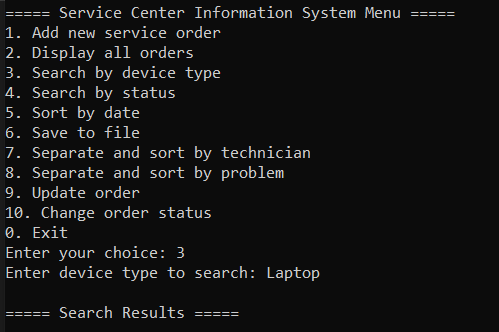


Когато няма налични въведени данни за поръчки.

И е необходимо въвеждане на поръчки за да се изразят таблично,

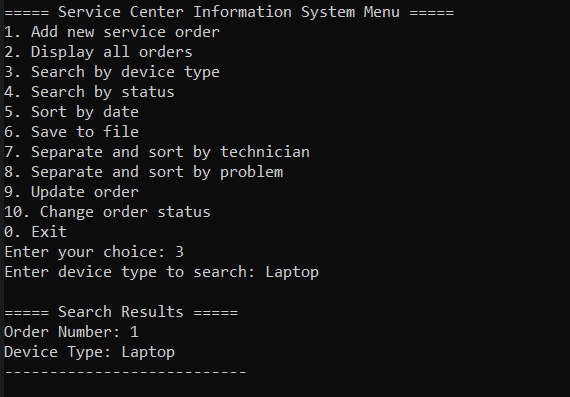
Реализация на условие D)

Случай 1:



Когато няма налични поръчки

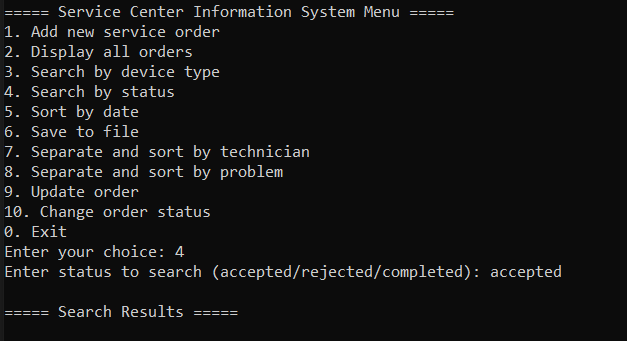
Случай 2:



Когато има налични поръчки ги извежда.

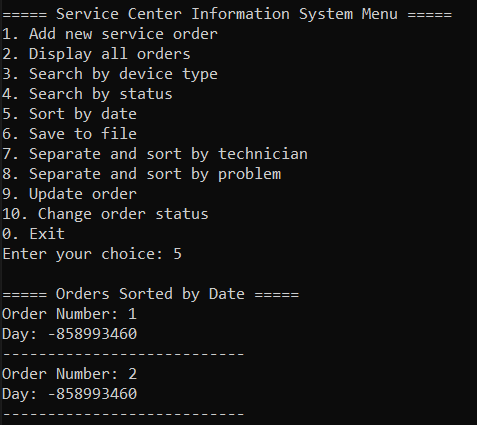
Въвежда се търсения вид на устройството.

Примерни данни: Laptop

Случай 3: 

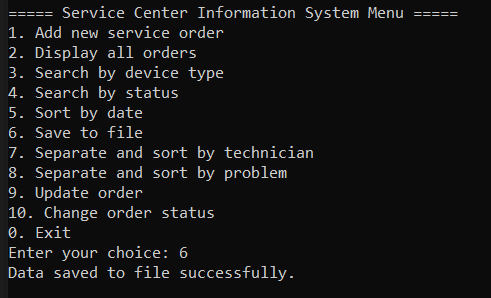
Търсене по статус: Въвежда се търсения статус на поръчката.

Реализация на условие Е)



Сортиране по ден на приемане на поръчката

Реализация на условие Ф)



Запазване във Файл

ПРИЛОЖЕНИЕ:

#include <iostream>

#include <fstream>

const int MAX\_ORDERS = 100;

const int MAX\_PROBLEMS = 10;

const int MAX\_REPAIRS = 10;

struct Order {

int orderNumber;

int day;

std::string client;

std::string deviceType;

std::string serialNumber;

std::string clientProblem;

std::string technicianName;

std::string repairDescription;

double price;

int daysInService;

std::string status;

};

// Lists of predefined problems and repairs

std::string problems[MAX\_PROBLEMS] = { "Screen issue", "Battery problem", "Software error", "Hardware failure", "Other" };

std::string repairs[MAX\_REPAIRS] = { "Replace component", "Software fix", "Hardware repair", "Cleaning", "Other" };

// Function prototypes

void displayMenu();

void addOrder(Order orders[], int& orderCount);

void displayAllOrders(const Order orders[], int orderCount);

void searchByDeviceType(const Order orders[], int orderCount);

void searchByStatus(const Order orders[], int orderCount);

void sortByDate(Order orders[], int orderCount);

void saveToFile(const Order orders[], int orderCount);

void loadFromFile(Order orders[], int& orderCount);

void separateAndSortByTechnician(const Order orders[], int orderCount);

void separateAndSortByProblem(const Order orders[], int orderCount);

void updateOrder(Order orders[], int orderCount);

void changeOrderStatus(Order orders[], int orderCount);

int main() {

Order orders[MAX\_ORDERS];

int orderCount = 0;

// Load data from file on startup

loadFromFile(orders, orderCount);

int choice;

do {

displayMenu();

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1:

addOrder(orders, orderCount);

break;

case 2:

displayAllOrders(orders, orderCount);

break;

case 3:

searchByDeviceType(orders, orderCount);

break;

case 4:

searchByStatus(orders, orderCount);

break;

case 5:

sortByDate(orders, orderCount);

break;

case 6:

saveToFile(orders, orderCount);

break;

case 7:

separateAndSortByTechnician(orders, orderCount);

break;

case 8:

separateAndSortByProblem(orders, orderCount);

break;

case 9:

updateOrder(orders, orderCount);

break;

case 10:

changeOrderStatus(orders, orderCount);

break;

case 0:

std::cout << "Exiting program. Saving data to file...\n";

// Save data to file before exiting

saveToFile(orders, orderCount);

break;

default:

std::cout << "Invalid choice. Try again.\n";

}

} while (choice != 0);

return 0;

}

void displayMenu() {

std::cout << "\n===== Service Center Information System Menu =====\n";

std::cout << "1. Add new service order\n";

std::cout << "2. Display all orders\n";

std::cout << "3. Search by device type\n";

std::cout << "4. Search by status\n";

std::cout << "5. Sort by date\n";

std::cout << "6. Save to file\n";

std::cout << "7. Separate and sort by technician\n";

std::cout << "8. Separate and sort by problem\n";

std::cout << "9. Update order\n";

std::cout << "10. Change order status\n";

std::cout << "0. Exit\n";

}

void addOrder(Order orders[], int& orderCount) {

if (orderCount < MAX\_ORDERS) {

std::cout << "Enter order details:\n";

Order newOrder;

// Get partial information from the user

std::cout << "Order number: ";

std::cin >> newOrder.orderNumber;

std::cout << "Device type: ";

std::cin >> newOrder.deviceType;

std::cout << "Serial number: ";

std::cin >> newOrder.serialNumber;

// Add other fields as needed

// Add the order to the array

orders[orderCount++] = newOrder;

std::cout << "Order added successfully!\n";

}

else {

std::cout << "Maximum number of orders reached!\n";

}

}

void displayAllOrders(const Order orders[], int orderCount) {

std::cout << "\n===== All Service Orders =====\n";

// Display orders in a formatted table

// Modify this part based on your specific data structure and fields

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

std::cout << "Order Number: " << orders[i].orderNumber << "\n";

std::cout << "Device Type: " << orders[i].deviceType << "\n";

// Add other fields as needed

std::cout << "---------------------------\n";

}

}

void searchByDeviceType(const Order orders[], int orderCount) {

std::string targetDeviceType;

std::cout << "Enter device type to search: ";

std::cin >> targetDeviceType;

std::cout << "\n===== Search Results =====\n";

// Search and display orders with matching device type

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

if (orders[i].deviceType == targetDeviceType) {

std::cout << "Order Number: " << orders[i].orderNumber << "\n";

std::cout << "Device Type: " << orders[i].deviceType << "\n";

// Add other fields as needed

std::cout << "---------------------------\n";

}

}

}

void searchByStatus(const Order orders[], int orderCount) {

std::string targetStatus;

std::cout << "Enter status to search (accepted/rejected/completed): ";

std::cin >> targetStatus;

std::cout << "\n===== Search Results =====\n";

// Search and display orders with matching status

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

if (orders[i].status == targetStatus) {

std::cout << "Order Number: " << orders[i].orderNumber << "\n";

std::cout << "Status: " << orders[i].status << "\n";

// Add other fields as needed

std::cout << "---------------------------\n";

}

}

}

void sortByDate(Order orders[], int orderCount) {

// Bubble sort based on order date (day)

for (int i = 0; i < orderCount - 1; ++i) {

for (int j = 0; j < orderCount - i - 1; ++j) {

if (orders[j].day > orders[j + 1].day) {

// Swap orders

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

std::cout << "\n===== Orders Sorted by Date =====\n";

// Display orders after sorting

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

std::cout << "Order Number: " << orders[i].orderNumber << "\n";

std::cout << "Day: " << orders[i].day << "\n";

// Add other fields as needed

std::cout << "---------------------------\n";

}

}

void saveToFile(const Order orders[], int orderCount) {

std::ofstream outFile("service\_orders.txt");

if (!outFile.is\_open()) {

std::cout << "Error opening file for writing.\n";

return;

}

// Write orders to the file

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

outFile << orders[i].orderNumber << " ";

outFile << orders[i].day << " ";

outFile << orders[i].client << " ";

outFile << orders[i].deviceType << " ";

outFile << orders[i].serialNumber << " ";

outFile << orders[i].clientProblem << " ";

outFile << orders[i].technicianName << " ";

outFile << orders[i].repairDescription << " ";

outFile << orders[i].price << " ";

outFile << orders[i].daysInService << " ";

outFile << orders[i].status << "\n";

// Add other fields as needed

}

std::cout << "Data saved to file successfully.\n";

outFile.close();

}

void loadFromFile(Order orders[], int& orderCount) {

std::ifstream inFile("service\_orders.txt");

if (!inFile.is\_open()) {

std::cout << "File not found. Starting with an empty list.\n";

return;

}

// Read orders from the file

while (inFile >> orders[orderCount].orderNumber >> orders[orderCount].day >> orders[orderCount].client

>> orders[orderCount].deviceType >> orders[orderCount].serialNumber >> orders[orderCount].clientProblem

>> orders[orderCount].technicianName >> orders[orderCount].repairDescription >> orders[orderCount].price

>> orders[orderCount].daysInService >> orders[orderCount].status) {

orderCount++;

if (orderCount >= MAX\_ORDERS) {

std::cout << "Maximum number of orders reached. Loading stopped.\n";

break;

}

}

std::cout << "Data loaded from file successfully.\n";

inFile.close();

}

void separateAndSortByTechnician(const Order orders[], int orderCount) {

// Display the results accordingly

std::cout << "Separate and sort by technician functionality not implemented yet.\n";

}

void separateAndSortByProblem(const Order orders[], int orderCount) {

// Display the results accordingly

std::cout << "Separate and sort by problem functionality not implemented yet.\n";

}

void updateOrder(Order orders[], int orderCount) {

int orderNumber;

std::cout << "Enter order number to update: ";

std::cin >> orderNumber;

// Find the order with the given number

int index = -1;

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

if (orders[i].orderNumber == orderNumber) {

index = i;

break;

}

}

if (index != -1) {

// Order found

if (orders[index].status == "Accepted") {

// Update repair information and calculate price

std::cout << "Enter technician name: ";

std::cin >> orders[index].technicianName;

std::cout << "Enter repair description: ";

std::cin >> orders[index].repairDescription;

// Calculate price based on repair type

// Add your logic here

std::cout << "Order updated successfully!\n";

}

else {

std::cout << "Order cannot be updated. It is already rejected/completed.\n";

}

}

else {

std::cout << "Order not found.\n";

}

}

void changeOrderStatus(Order orders[], int orderCount) {

int orderNumber;

std::string newStatus;

std::cout << "Enter order number: ";

std::cin >> orderNumber;

// Find the order with the given number

int index = -1;

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

if (orders[i].orderNumber == orderNumber) {

index = i;

break;

}

}

if (index != -1) {

// Order found

std::cout << "Enter new status (Accepted/Rejected/Completed): ";

std::cin >> newStatus;

if (orders[index].status == "Accepted") {

// Status can be changed only if the order is still in the "Accepted" state

orders[index].status = newStatus;

std::cout << "Status updated successfully!\n";

}

else {

std::cout << "Status cannot be changed. It is already rejected/completed.\n";

}

}

else {

std::cout << "Order not found.\n";

}

}