**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ВОЛОДИМИРА ДАЛЯ**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ**

**КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ**

**КУРСОВА РОБОТА**

на тему:

Розробка програми з використанням системи класів для продажу автомобілів з дисципліни «Об'єктно-орієнтоване програмування»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент групи  КН – 18Д |  |  |  | Скороход С. Г. |
|  |  | (підпис, дата) |  | (П.І.Б.) |
|  |  |  |  |  |
| Керівник проекту |  |  |  | Щербакова М. Є. |
|  |  | (підпис, дата) |  | (П.І.Б.) |

Сєвєродонецьк

2020

|  |
| --- |
| Східноукраїнський національний університет |
| імені Володимира Даля |

Факультет ІТЕ . Кафедра КНІ .

Напрям 122 Комп’ютерні науки

|  |
| --- |
| ЗАТВЕРДЖУЮ: |
| Зав. кафедри КНІ\_\_\_\_\_\_\_\_\_\_\_\_  І.С.Скарга-Бандурова  «\_\_\_\_\_\_\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  2020р. |

**З А В Д А Н Н Я**

**НА КУРСОВУ РОБОТУ**

|  |
| --- |
| (прізвище, ім'я, по батькові) |

1. Тема проекту (роботи)

Розробка програми з використанням системи класів для продажу автомобілів

затверджена наказом по інституту від « » 2020 р. № .

2. Термін подання студентом закінченого проекту (роботи)

3. Початкові дані до проекту (роботи)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробленню)

5. Перелік графічного матеріалу (з точною вказівкою обов'язкових креслень)

Не потребує

.6. Дата видачі завдання

|  |  |
| --- | --- |
| Керівник проекту |  |
|  | (підпис) |
| Завдання прийняв до виконання |  |
|  | (підпис) |

# КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Найменування етапів дипломного  проекту (роботи) | Термін виконання етапів проекту (роботи) | Примітка |
| 1 | Постановка завдання | 15.04.20 – 16.04.20 |  |
| 2 | Ознайомлення з середовищем програмування | 17.04.20 – 17.04.20 |  |
| 3 | Планування розробки програми | 18.04.20 – 23.04.20 |  |
| 4 | Розробка програми у вибраному середовищі | 23.04.20 – 15.05.20 |  |
| 5 | Написання пояснювальної записки | 16.05.20 – 17.05.20 |  |

|  |  |
| --- | --- |
| Студент |  |
|  | (підпис) |
| Керівник проекту |  |

|  |  |
| --- | --- |
|  | (підпис) |

**ЗМІСТ**

[КАЛЕНДАРНИЙ ПЛАН 3](#_Toc41568951)

[ВСТУП 6](#_Toc41568952)

[1. АНАЛІЗ І ПОСТАНОВКА ЗАДАЧІ 7](#_Toc41568953)

[1.1 Визначення ООП 7](#_Toc41568954)

[1.2 Фундаментальні поняття об’єктно-орієнтованого програмування. 8](#_Toc41568955)

[1.3 Сучасні об'єктно-орієнтовані мови програмування 11](#_Toc41568956)

[1.4 Технічне завдання на розробку 11](#_Toc41568957)

[2. ПОБУДОВА І РЕАЛІЗАЦІЯ ПРОГРАМИ 12](#_Toc41568958)

[2.1 Вибір засобів розробки 12](#_Toc41568959)

[2.2 Реалізація структури( список ) під назвою avt 12](#_Toc41568960)

[2.3 Реалізація структури (списку ) під назвою avtSlot 13](#_Toc41568961)

[2.4 Реалізація структури (списку ) під назвою Klient 13](#_Toc41568962)

[2.5 Реалізація класу Avto 13](#_Toc41568963)

[**2.5.1** Реалізація конструктору класу 13](#_Toc41568964)

[**2.5.2** Реалізація функції CreateAvto 14](#_Toc41568965)

[**2.5.3** Реалізація функції WriteAll 14](#_Toc41568966)

[**2.5.4** Реалізація функції SearchId 15](#_Toc41568967)

[**2.5.5** Реалізація функції SearchMarka 15](#_Toc41568968)

[**2.5.6** Реалізація перегрузки функції SearchMarka 17](#_Toc41568969)

[**2.5.7** Реалізація функції SearchModel 18](#_Toc41568970)

[**2.5.8** Реалізація перегрузки функції SearchModel 19](#_Toc41568971)

[**2.5.9** Реалізація функції SearchStatus 20](#_Toc41568972)

[**2.5.10** Реалізація перегрузки функції SearchStatus 20](#_Toc41568973)

[**2.5.11** Реалізація функції SearchColor 21](#_Toc41568974)

[**2.5.12** Реалізація перегрузки функції SearchColor 22](#_Toc41568975)

[**2.5.13** Реалізація функції AddNewAvto 23](#_Toc41568976)

[**2.5.14** Реалізація функції DelAvto 24](#_Toc41568977)

[**2.5.15** Реалізація деструктору 25](#_Toc41568978)

[2.6 Реалізація класу Client 25](#_Toc41568979)

[**2.6.1** Реалізація конструктору 26](#_Toc41568980)

[**2.6.2** Реалізація функції CreateClient 26](#_Toc41568981)

[**2.6.3** Реалізація функції WriteAllClient 27](#_Toc41568982)

[**2.6.4** Реалізація функції SearchClientId 27](#_Toc41568983)

[**2.6.5** Реалізація функції ByAvto 28](#_Toc41568984)

[**2.6.6** Реалізація деструктору 30](#_Toc41568985)

[2.7 Реалізація класу Prog 30](#_Toc41568986)

[2.8 Реалізація функції Go 30](#_Toc41568987)

[2.9 Реалізація функції Go1 30](#_Toc41568988)

[2.10 Реалізація функції GoAgane 31](#_Toc41568989)

[2.11 Реалізація головної функції main 31](#_Toc41568990)

[3. Загальний вигляд коду 34](#_Toc41568991)

[4. Ісходні дані 62](#_Toc41568992)

[4.1 Файл avto.txt 62](#_Toc41568993)

[4.2 Файл client.txt 62](#_Toc41568994)

[ВИСНОВКИ 63](#_Toc41568995)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64](#_Toc41568996)

# ВСТУП

Об'єктно-орієнтоване програмування— одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua) або ж є цілком об'єктно-орієнтованими (зокрема, Java, C#, C++, Python, Ruby і Objective-C, ActionScript 3, Swift, Vala).

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-тих роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Через ускладнення апаратного та програмного забезпечення було дуже важко зберегти якість програм. Об'єктно-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

# ****АНАЛІЗ І ПОСТАНОВКА ЗАДАЧІ****

## ****Визначення ООП****

**Об'єктно-орієнтоване програмування — це метод програмування, заснований на поданні програми як сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. Програмісти спочатку пишуть клас, а на його основі під час виконання програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів.**

**На думку Алана Кея, розробника мови Smalltalk, якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:**

* **Все є об'єктами.**
* **Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, під час якої один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися під час виконання дії.**
* **Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.**
* **Кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів.**
* **У класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.**
* **Класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.**

**Таким чином, програма є набором об'єктів, що мають стан та поведінку.**

**Об'єкти взаємодіють використовуючи повідомлення. Будується ієрархія об'єктів: програма в цілому — це об'єкт, для виконання своїх функцій вона звертається до об'єктів що містяться у ньому, які у свою чергу виконують запит шляхом звернення до інших об'єктів програми. Звісно, щоб уникнути нескінченної рекурсії у зверненнях, на якомусь етапі об'єкт трансформує запит у повідомлення до стандартних системних об'єктів, що даються мовою та середовищем програмування. Стійкість та керованість системи забезпечуються за рахунок чіткого розподілу відповідальності об'єктів (за кожну дію відповідає певний об'єкт), однозначного означення інтерфейсів міжоб'єктної взаємодії та повної ізольованості внутрішньої структури об'єкта від зовнішнього середовища (інкапсуляції).**

## ****Фундаментальні поняття об’єктно-орієнтованого програмування.****

В результаті дослідження Дебори Дж. Армстронг (англ. Deborah J. Armstrong) комп'ютерної літератури, що була видана протягом останніх 40 років, вдалось відокремити фундаментальні поняття (принципи), використані у переважній більшості визначень об'єктно-орієнтованого програмування. До них належить:

**Клас**

Клас визначає абстрактні характеристики деякої сутності, включно з характеристиками самої сутності (її атрибутами або властивостями) та діями, які вона здатна виконувати (її поведінкою, методами або можливостями). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єктно-орієнтовану програму. Зазвичай клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його членами.

**Об'єкт**

Окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу). Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт Сірко є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Об'єкт Сірко є екземпляром (примірником) класу Собака. Сукупність значень атрибутів окремого об'єкта називається станом. На основі класу Собака можна, також, створити інший об'єкт Дружок, який відрізнятиметься від об'єкта Сірко своїм станом (наприклад кольором хутра). Обидва об'єкта (Сірко і Дружок) є екземплярами класу Собака.

**Метод**

Можливості об'єкта. Оскільки Сірко — Собака, він може гавкати. Тому гавкати() є одним із методів об'єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об'єкт; всі Собаки можуть гавкати, але треба щоб гавкав лише один окремий собака.

**Обмін повідомленнями**

«Передача даних від одного процесу іншому, або надсилання викликів методів.»

**Успадкування (наслідування)**

Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування. Наприклад, клас Собака може мати підкласи Коллі, Пекінес, Вівчарка тощо. Так, Сірко може бути екземпляром класу Вівчарка. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в Java дозволене лише одинарне успадкування, а в С++ і те і інше.

**Приховування інформації (інкапсуляція)**

Приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення. Так, наприклад, клас Собака має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути() а потім видихнути() на обраній частоті та гучності). Петро, хазяїн пса Сірка, не повинен знати як він гавкає. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта. Як наслідок, кожен об'єкт надає кожному іншому класу певний інтерфейс — члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дасть змогу полегшити внесення змін без потреби змінювати і користувачів інтерфейсу. Наприклад, інтерфейс може гарантувати, що щенята можуть додаватись лише до об'єктів класу Собака кодом самого класу. Часто, члени класу позначаються як публічні (англ. public), захищені (англ. protected) та приватні (англ. private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено. Деякі мови програмування йдуть ще далі: Java використовує ключове слово private для обмеження доступу, що буде дозволений лише з методів того самого класу, protected — лише з методів того самого класу і його нащадків та з класів із того ж самого пакету, C# та VB.NET відкривають деякі члени лише для класів із тієї ж збірки шляхом використання ключового слова internal (C#) або Friend (VB.NET), а Eiffel дозволяє вказувати які класи мають доступ до будь-яких членів.

**Абстрагування**

Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Сірко більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи коллі — як Коллі і як Тварина (можливо, батьківський клас Собака) під час підрахунку тварин Петра.

**Поліморфізм**

Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. Наприклад, якщо Собака отримує команду голос(), то у відповідь можна отримати Гав; якщо Свиня отримує команду голос (), то у відповідь можна отримати Рох-рох. На практиці - це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. Залежно від того, що передається, і вибирається відповідна підпрограма.

## Сучасні об'єктно-орієнтовані мови програмування

Зі всіх об'єктно-орієнтованих мов програмування найчастіше використовується мова C++. Мова програмування C++ отримала набагато більшу популярність, ніж такі мови як Pascal, Visual Basic чи C, бо вона стала потужним інструментом для розроблення складного програмного забезпечення. У синтаксичному плані мови C і C++ дуже схожі між собою. Понад це, мова програмування C++ є надбудовою мови C. Завдяки цьому відпала потреба перед вивченням мови програмування C++ вивчати мову С. Ті студенти, які мають навички роботи мовою C, можуть знехтувати частиною матеріалу, викладеного на початкову цього посібника, проте решта інформації буде для них новою і корисною для засвоєння.

## Технічне завдання на розробку

Розробити на мові С ++ систему класів про продаж автомобілів. система повинна забезпечувати ведення списку нових і старих автомобілів, ведення списку покупців, автоматизований підбір варіантів для покупця. Написати програму, яка демонструвала б роботу з створеними класами.

# ПОБУДОВА І РЕАЛІЗАЦІЯ ПРОГРАМИ

## Вибір засобів розробки

Найбільш зручним у використанні в даний час є інтегроване середовище розробки програмного забезпечення **Microsoft Visual Studio**.

Visual Studio включає Visual C++.

Microsoft Visual C++ (MSVC) - інтегроване середовище розробки додатків на мові C++, розроблена корпорацією Microsoft і поставляється або як частина комплекту Microsoft Visual Studio, або окремо у вигляді безкоштовного функціонально обмеженого комплекту Microsoft Visual Studio Community Edition (раніше Visual C++ Express Edition). Змінила інтегроване середовище розробки Microsoft QuickC.

Visual C++ підтримує перелік додатків як на Managed C++ і C++ / CLI, так і на звичайному C++, і тим самим дозволяє генерувати код як для платформи .NET Framework, так і для виконання в середовищі «чистої» Windows. В цьому відношенні Visual C++ є унікальним серед інших мовних засобів, що надаються середовищем Visual Studio, оскільки ні Visual Basic .NET, ні Visual J # не здатні генерувати код для чистого Win32, на відміну від попередніх версій (Visual Basic і Visual J++ відповідно).

Остання версія Visual C ++ 2019 16.2.0 входить в комплект Visual Studio 2019.

Для створення програми знадобиться консольний додаток С ++.

Звичайною відправною точкою для програміста на C ++ є додаток "Hello World", що виконується в командному рядку.

## Реалізація структури( список ) під назвою avt

// Список полной базы авто

struct avt

{

string id; // зберігання номеру авто

string marka;// марка авто

string model; // модель авто

string status; // статус авто (нова чи бу)

string color; // колір авто

avt \*next, \*pred;

};

Було обрано двусвязний список для зручності видалення в подальшій роботі.

## Реалізація структури (списку ) під назвою avtSlot

struct avtSlot

{

string id; // зберігання номеру авто

string marka; // марка авто

string model; // модель авто

string status; // статус авто (нова чи бу)

string color; // колір авто

avtSlot \*next;

};

Список було створено для реалізації багаторівневого пошуку авто. Він зберігає результати попередніх пошуків .

## Реалізація структури (списку ) під назвою Klient

struct Klient

{

string name;// ім’я клієнта (формат Прізвище.Ім’я. По батькові)

string id;// номеру авто

string marka; // марка авто

string model; // модель авто

string status; // статус авто (нова чи бу)

string color; // колір авто

Klient \*next;

};

Список зберігає базу всіх клієнтів які купили авто.

## Реалізація класу Avto

class Avto

{

public:

friend class Client; // Avto передає доступ до закритих полів класу Client

private:

avt \*Head, \*Tail; //зберігання початку та кінця списку avt

avtSlot \*HeadSlot, \*TailSlot; //зберігання початку та кінця списку avtSlot

int Count; // кількість авто в списку avt

int CountSlot; ; // кількість авто в списку avtSlot

};

### Реалізація конструктору класу

Avto()

{

Head = Tail = 0;

HeadSlot = TailSlot = 0;

Count = 0;

CountSlot = 0;

}

### Реалізація функції CreateAvto

Функція здійснює первинну ініціалізацію списку avt з файлу бази. Повертає 0 якщо база має хоча б одне авто, 1 якщо база порожня.

int CreateAvto()

{

ifstream avtoIn("e:\\avto.txt");// открытие файла

if (!avtoIn.is\_open() ) // когда файл не удалось открыть

{

cout << "Файл не найден"; return 1;

}

if (Count > 0)return 0;// Если база не пуста не проводить инициализацию базы с файла

while (!avtoIn.eof()) // пока не конец файла

{

// построчно считывем и записываем данные с файла в список

avt \*temp = new avt;

avtoIn >> temp->id >> temp->marka >> temp->model >> temp->status >> temp->color;

if (temp->id != "")

{

Count++;

temp->next = 0; temp->pred = 0;

if (Tail) { Tail->next = temp; temp->pred = Tail; }

if (!Head)Head = temp;

Tail = temp;

}

else break;

}

avtoIn.close(); // закрываем файл

if (Count == 0) return 1;// Предупреждает о пустоте файла

return 0; // успешная инициализация

}

### Реалізація функції WriteAll

Функція здійснює повний вивід списку avt в консоль.

void WriteAll()

{

if (Count != 0)// если список не пуст

{

avt \*temp = Head;

while (temp != 0)

{

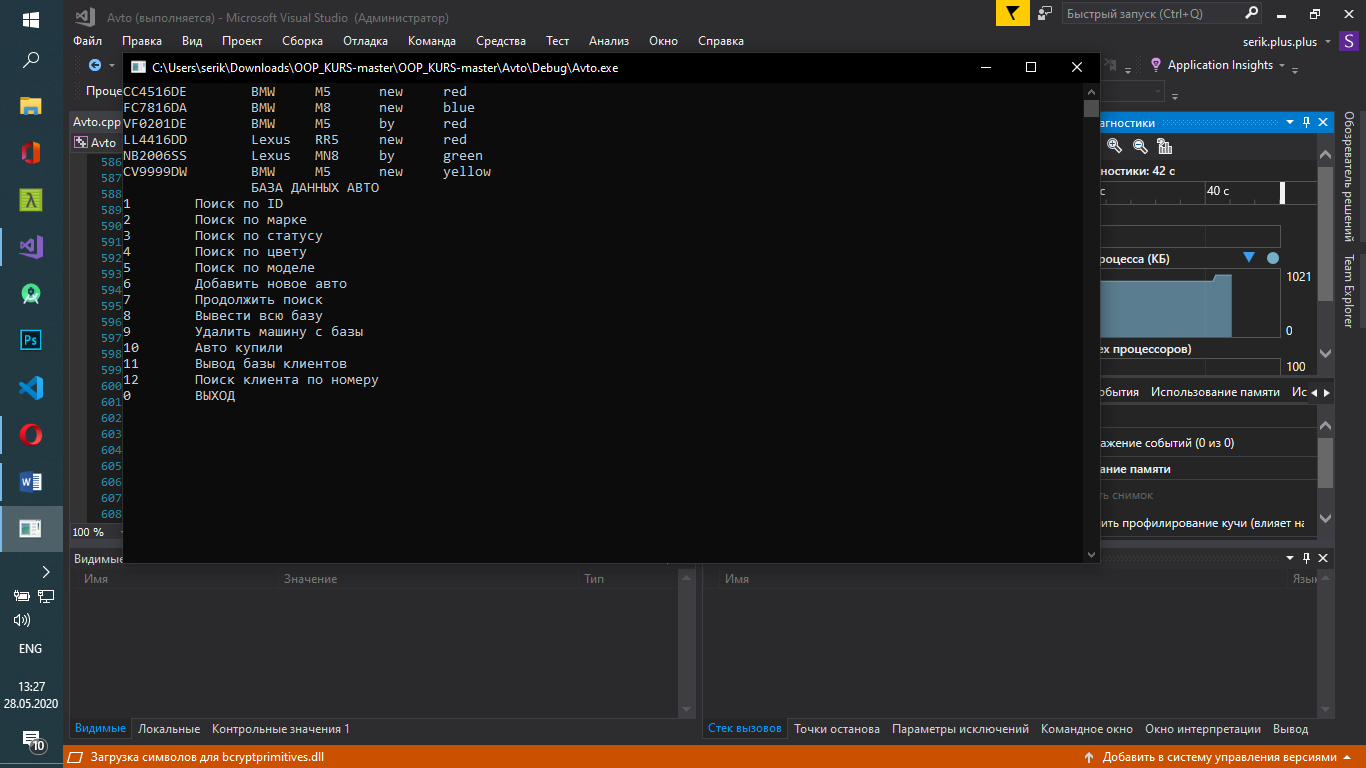
cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

}

}

}



### Реалізація функції SearchId

Функція здійснює пошук авто по номеру.

void SearchId()

{

if (Count != 0)

{

string id; // ID для поиска

int chec = 0;// флаг для обозначиния соотвецтвий

cout << "Укажите ID"<<endl;

cin >> id;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)// пока не конец структуры

{

if (temp->id == id)

{

// Вывод полей поиска

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

chec++;

break;

}

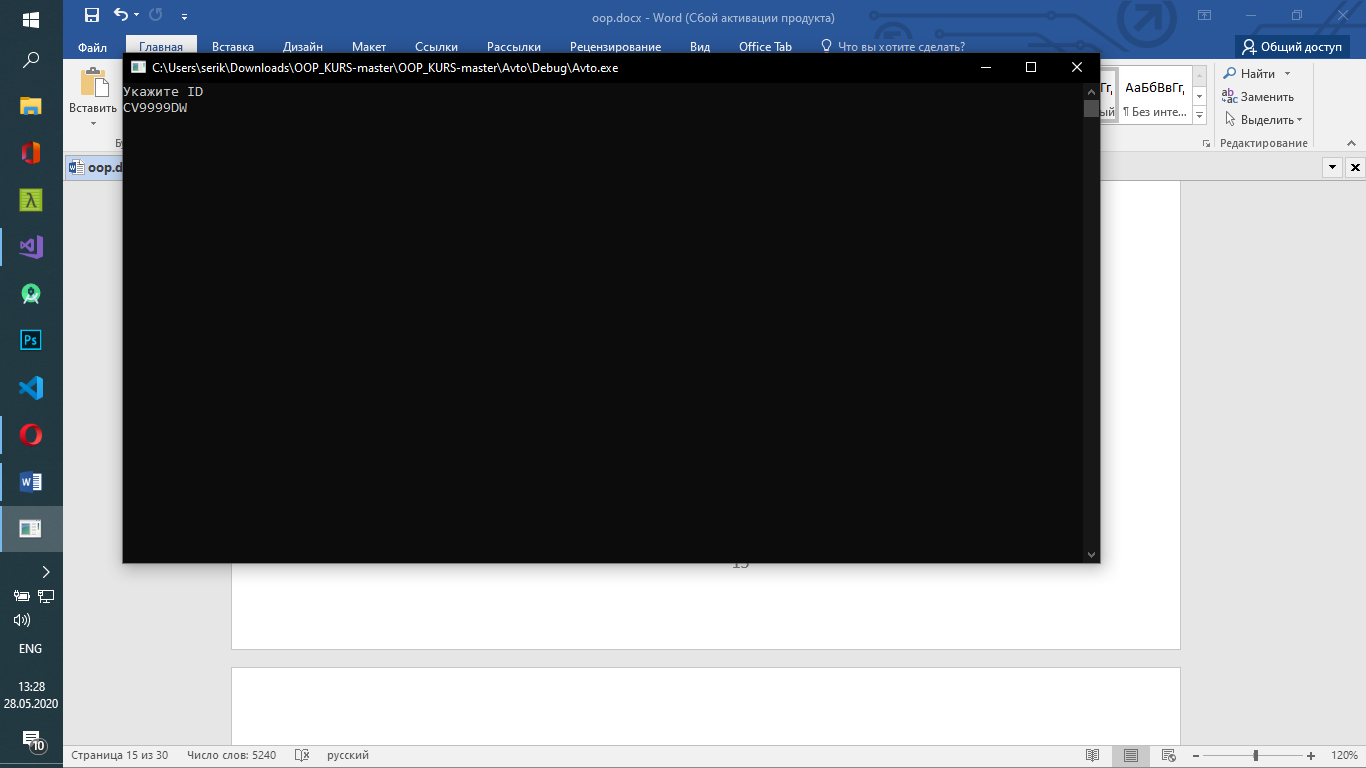
else temp = temp->next;

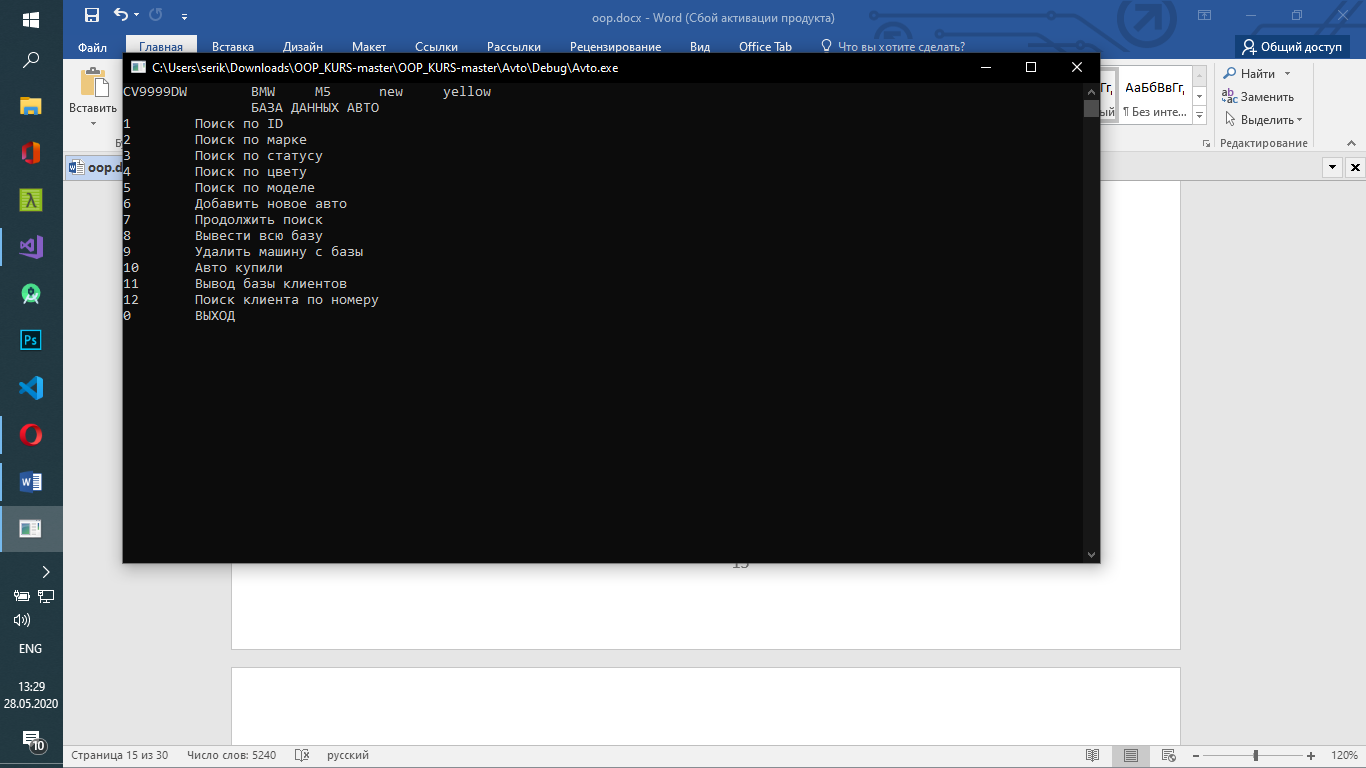
}

if (chec == 0)cout << "Такой авто нет" << endl;

}

}





### Реалізація функції SearchMarka

Функція здійснює пошук авто за маркою, вивід всіх результатів, та запис їх в стек-список avtSlot для можливого продовження пошуку.

void SearchMarka()

{

if (Count != 0)

{

string marka;

int chec = 0;

cout << "Укажите марку авто" << endl;

cin >> marka;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->marka == marka)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

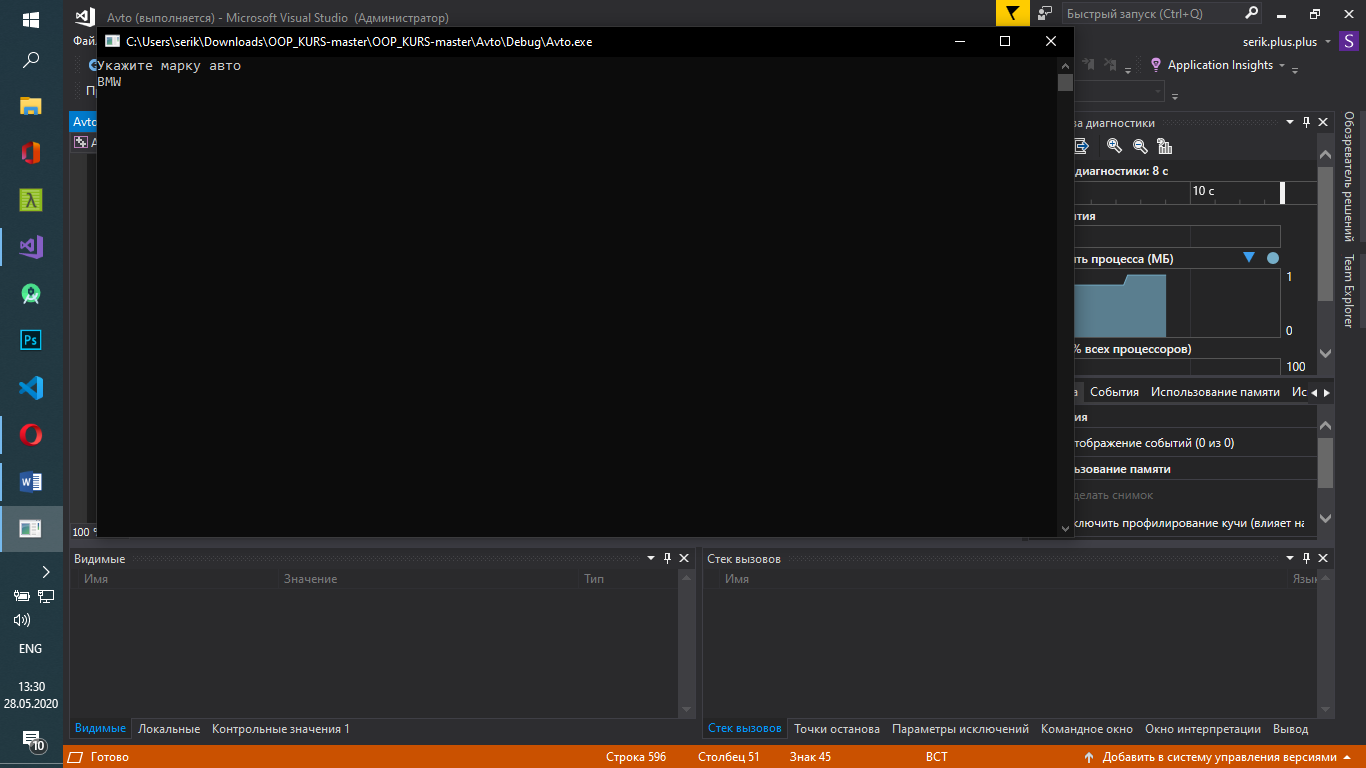
else temp = temp->next;

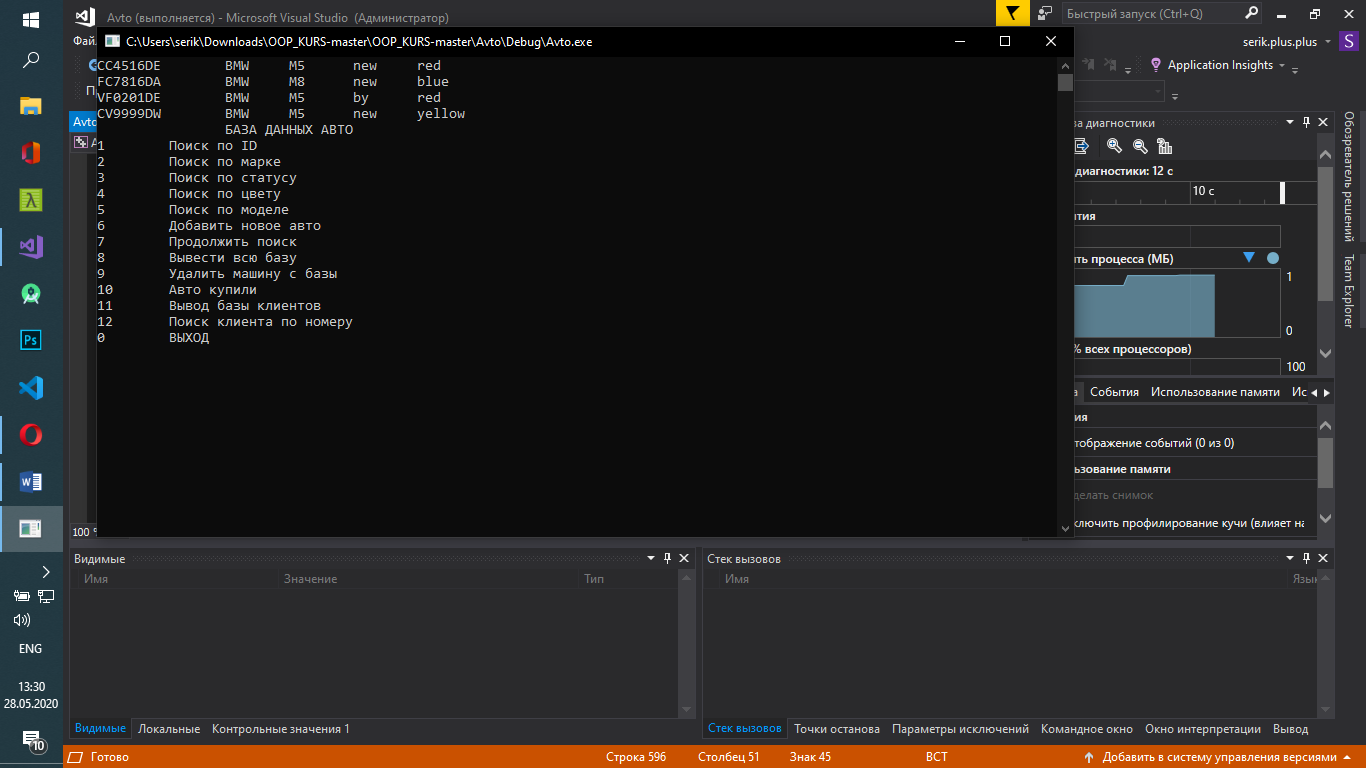
}

if (chec == 0)cout << "Такой марки нет" << endl;

}

}





Було здійснено пошук авто за маркою. Інші типи пошуку виконуються за такою ж технологією.

### Реалізація перегрузки функції SearchMarka

Здійснює продовження минулого пошуку для відокремлення авто з певною маркою

void SearchMarka(int n)

{

if (CountSlot != 0)

{

string marka;

int chec = 0;

cout << "Укажите марку авто" << endl;

cin >> marka;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->marka == marka)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

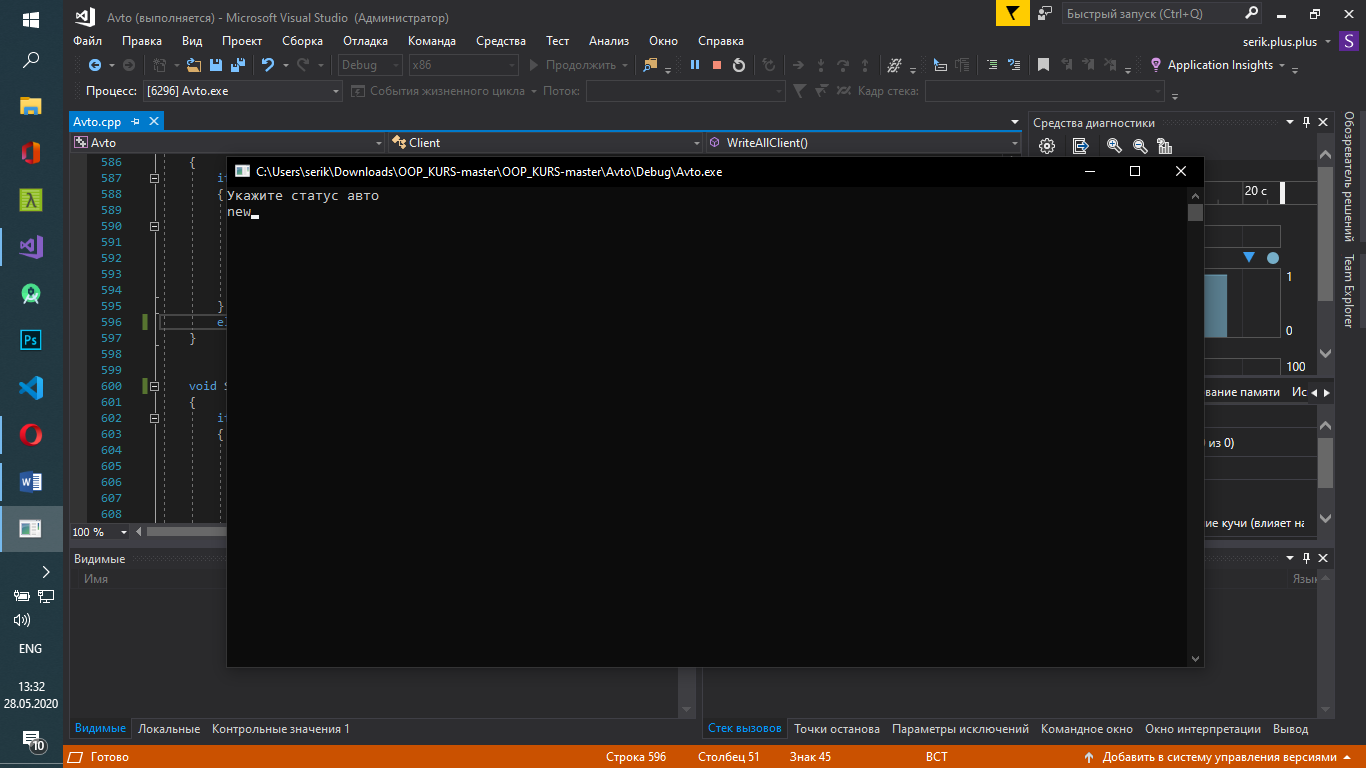
else temp = temp->next;

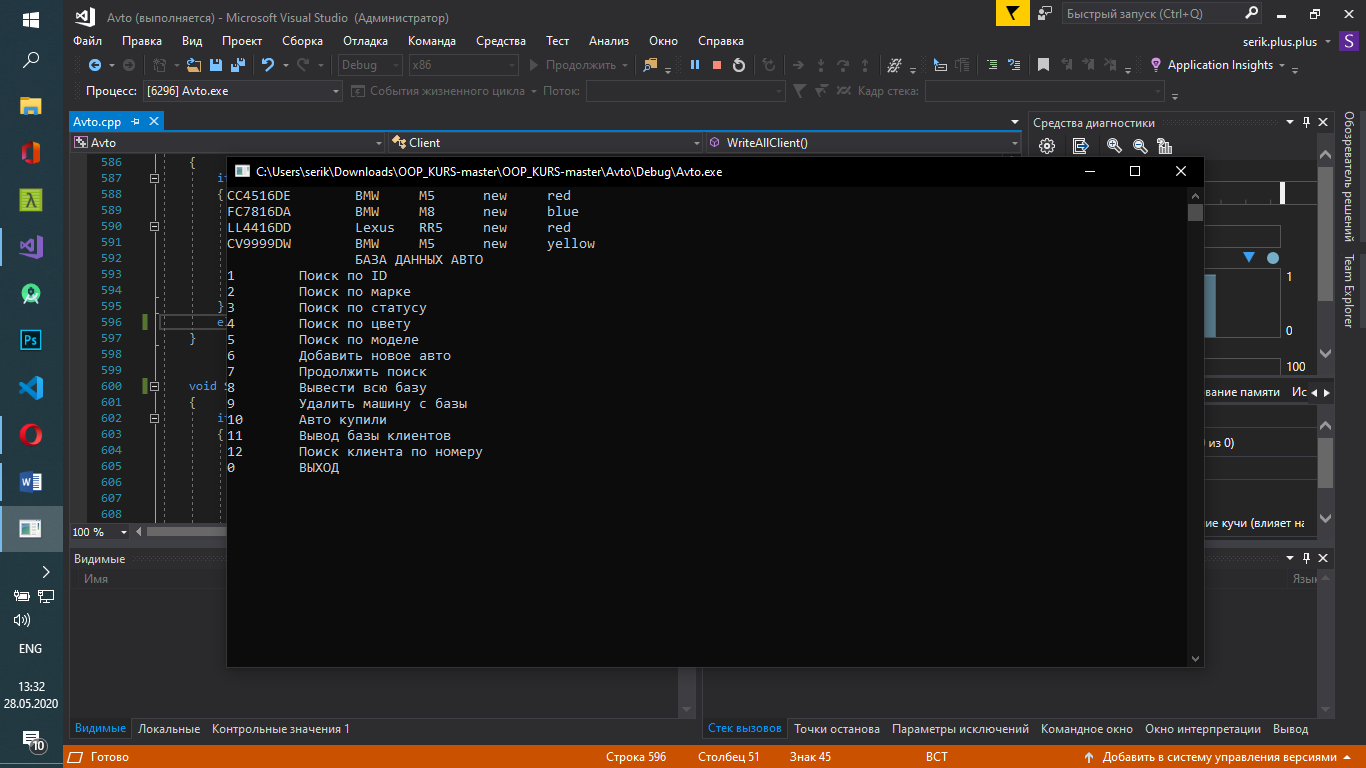
}

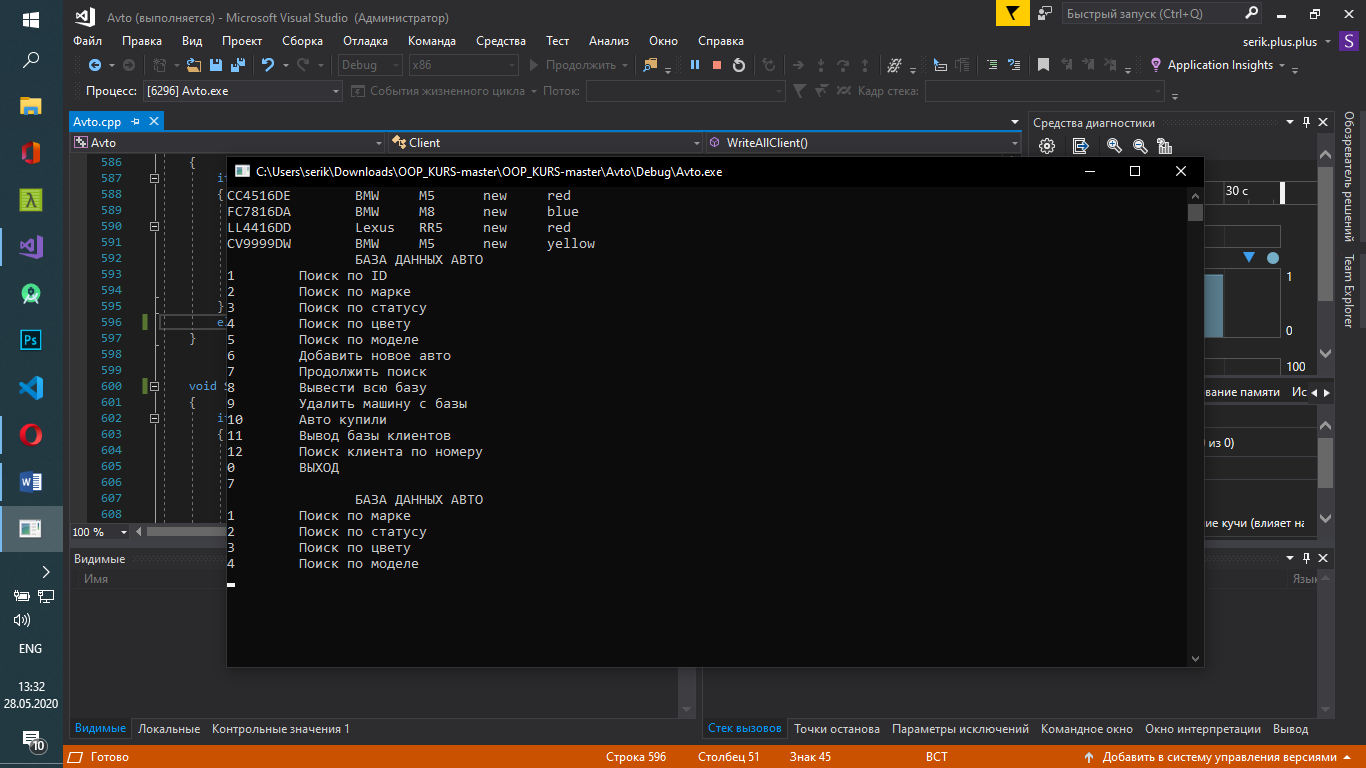
if (chec == 0)cout << "Такой машины нет" << endl;

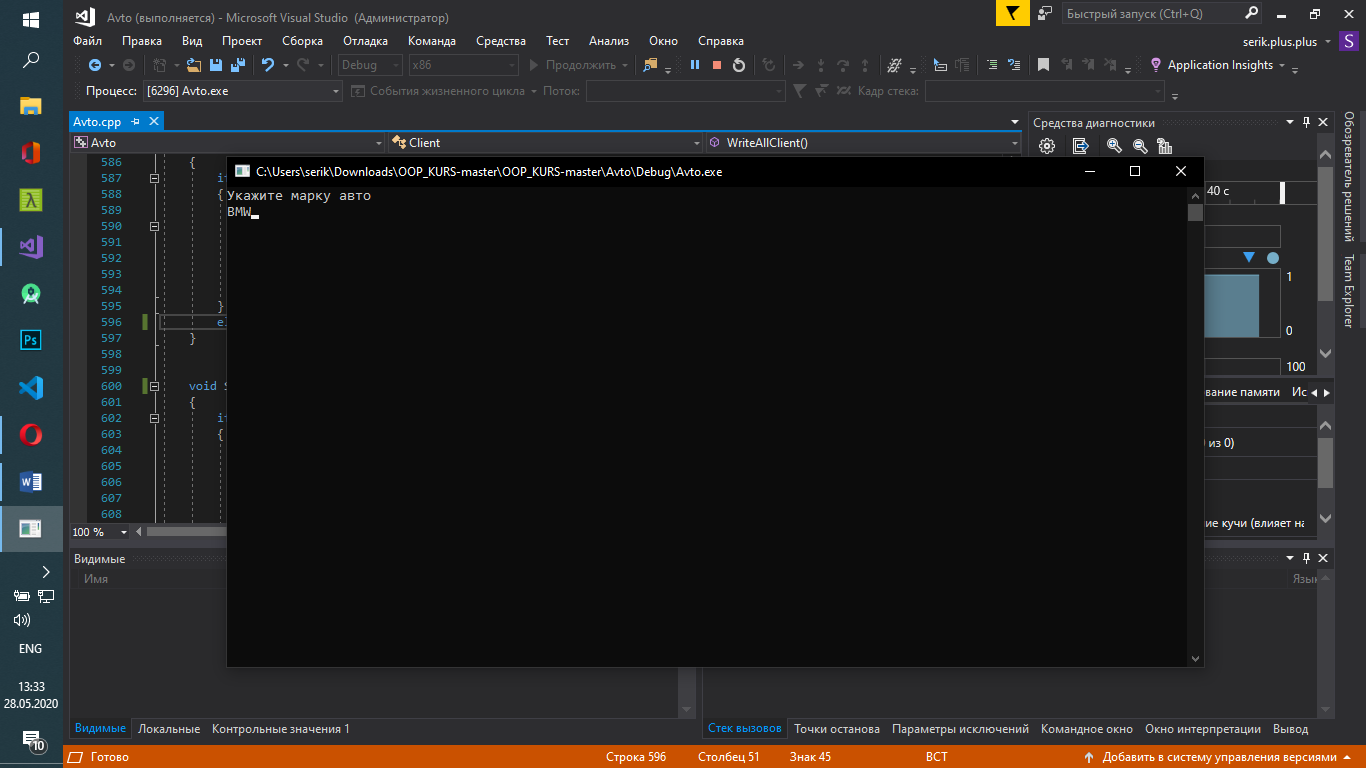
}

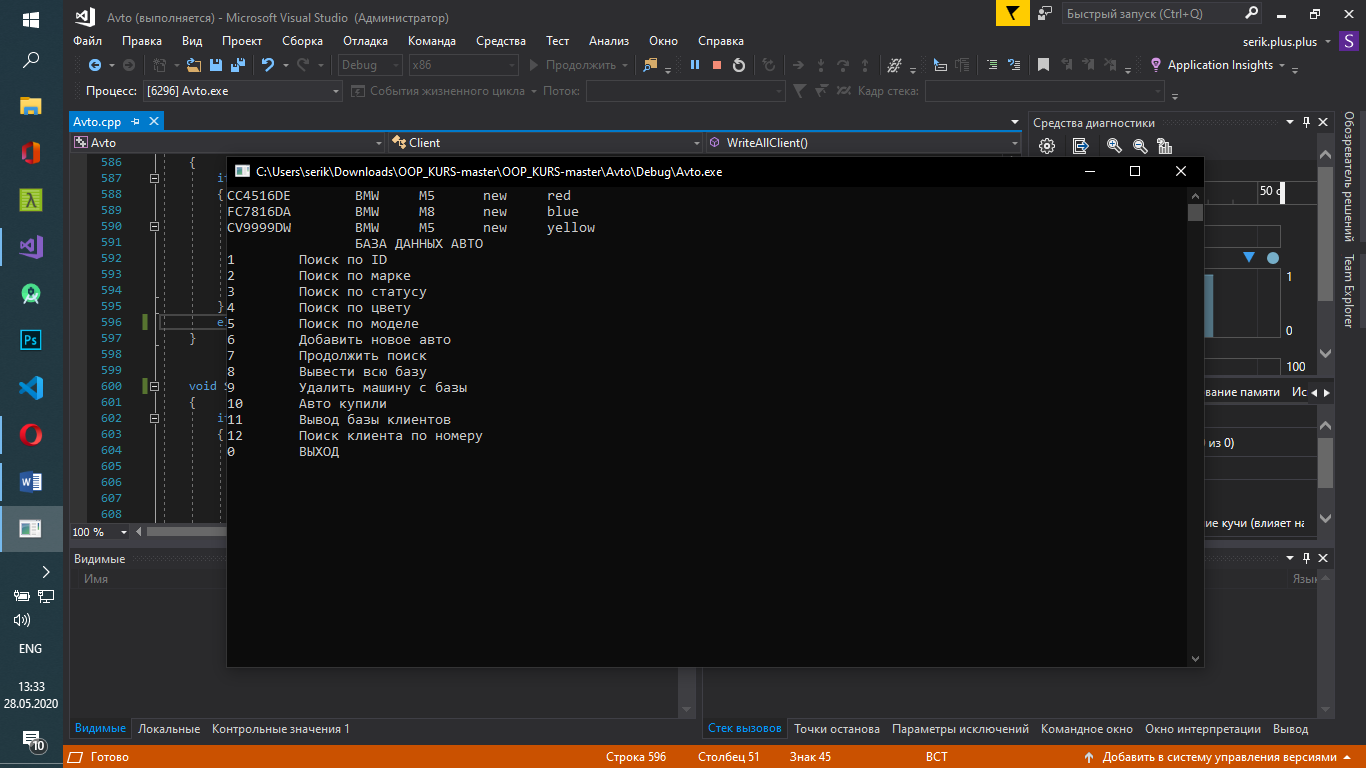
}











Було здійснено спочатку пошук за статусом після продовжено пошук за маркою. Результат – авто певної марки з вказаним статусом. Інші типи багаторівневого пошуку виконуються за такою ж технологією. Кількість рівнів пошуку не обмежено.

Порядок виконання багаторівневого пошуку:

* Поиск по статусу
* Продолжить поиск
* Поиск по марке

Для продовження додавання рівнів пошуку слід знов звернутися до дії – Продолжить поиск

### Реалізація функції SearchModel

Функція здійснює пошук авто за моделлю , вивід всіх результатів, та запис їх в стек-список avtSlot для можливого продовження пошуку.

void SearchModel()

{

if (Count != 0)

{

string model;

int chec = 0;

cout << "Укажите модель авто" << endl;

cin >> model;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->model == model)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такой модели нет" << endl;

}

}

### Реалізація перегрузки функції SearchModel

Здійснює продовження минулого пошуку для відокремлення авто з певною моделлю

void SearchModel(int n)

{

if (CountSlot != 0)

{

string model;

int chec = 0;

cout << "Укажите модель авто" << endl;

cin >> model;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->model == model)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такой модели нет" << endl;

}

}

### Реалізація функції SearchStatus

Функція здійснює пошук авто за статусом ( нове чи бу) , вивід всіх результатів, та запис їх в стек-список avtSlot для можливого продовження пошуку.

void SearchStatus()

{

if (Count != 0)

{

string status;

int chec = 0;

cout << "Укажите статус авто" << endl;

cin >> status;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->status == status)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого статуса нет" << endl;

}

}

### Реалізація перегрузки функції SearchStatus

Здійснює продовження минулого пошуку для відокремлення авто з певним статусом

void SearchStatus(int n)

{

if (CountSlot != 0)

{

string status;

int chec = 0;

cout << "Укажите статус авто" << endl;

cin >> status;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->status == status)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого статуса нет" << endl;

}

}

### Реалізація функції SearchColor

Функція здійснює пошук авто за кольором , вивід всіх результатів, та запис їх в стек-список avtSlot для можливого продовження пошуку.

void SearchColor()

{

if (Count != 0)

{

string color;

int chec = 0;

cout << "Укажите цвет авто" << endl;

cin >> color;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->color == color)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого цвета нет" << endl;

}

}

### Реалізація перегрузки функції SearchColor

Здійснює продовження минулого пошуку для відокремлення авто з певним кольором

void SearchColor(int n)

{

if (CountSlot != 0)

{

string color;

int chec = 0;

cout << "Укажите цвет авто" << endl;

cin >> color;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->color == color)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого цвета нет" << endl;

}

}

### Реалізація функції AddNewAvto

Функція здійснює вставку нового авто до бази.

void AddNewAvto()

{

ofstream avtoOut("e:\\avto.txt", ios::app);

avt \*temp = new avt;

temp->next = 0;

cout << "Номер: " << endl;

cin >> temp->id;

avtoOut << temp->id << "\t";

cout << "Марка: " << endl;

cin >> temp->marka;

avtoOut << temp->marka<<"\t";

cout << "Модель: " << endl;

cin >> temp->model;

avtoOut << temp->model << "\t";

cout << "Статус: " << endl;

cin >> temp->status;

avtoOut << temp->status << "\t";

cout << "Цвет: " << endl;

cin >> temp->color;

avtoOut << temp->color<<"\n";

temp->pred = Tail;

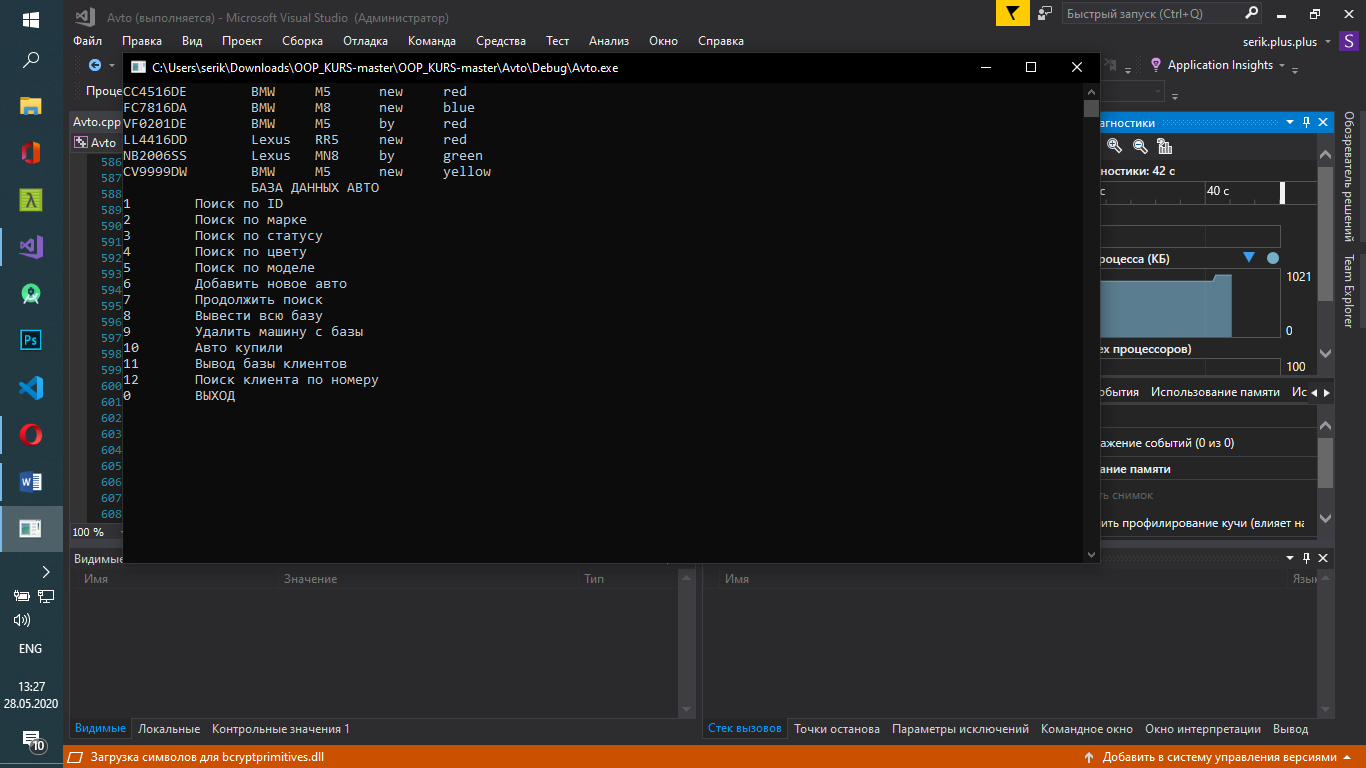
if (Tail != 0)Tail->next = temp;

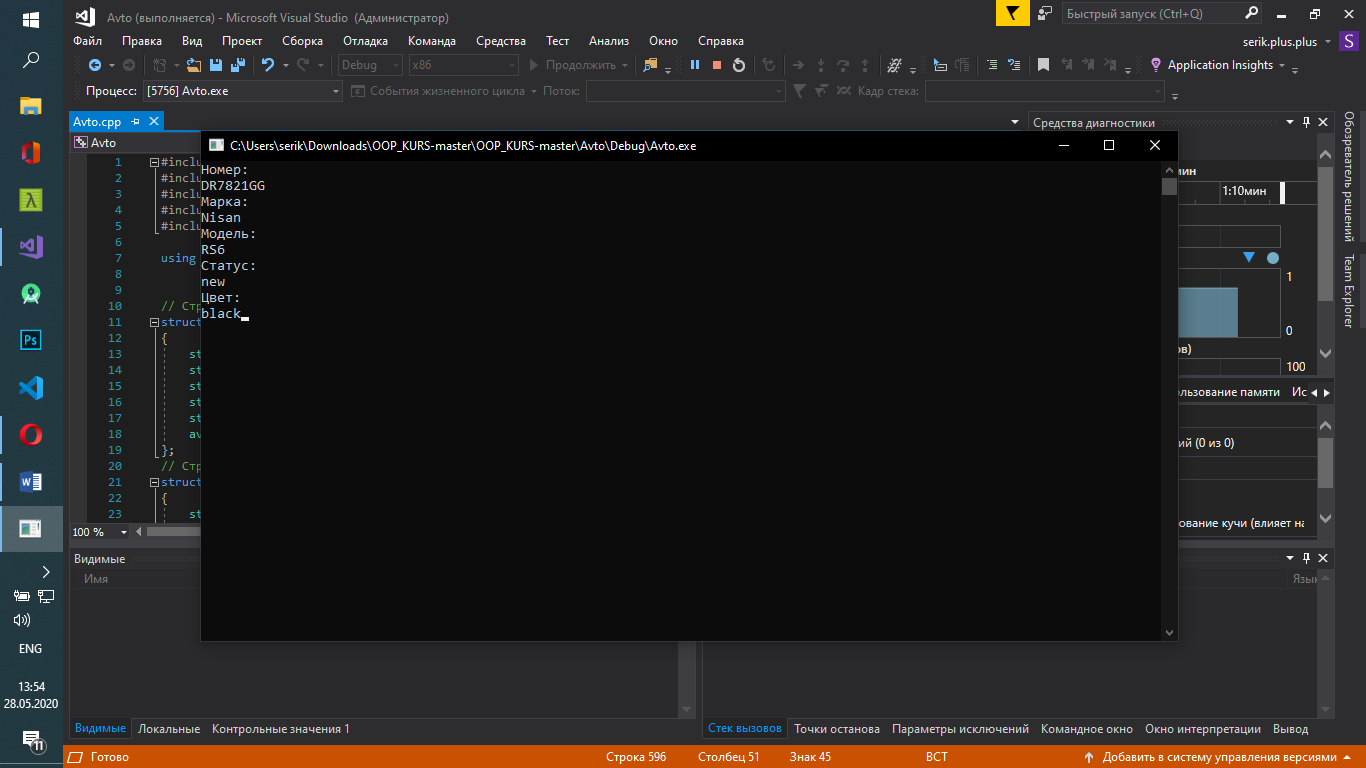
if (Count == 0)Head = Tail = temp;

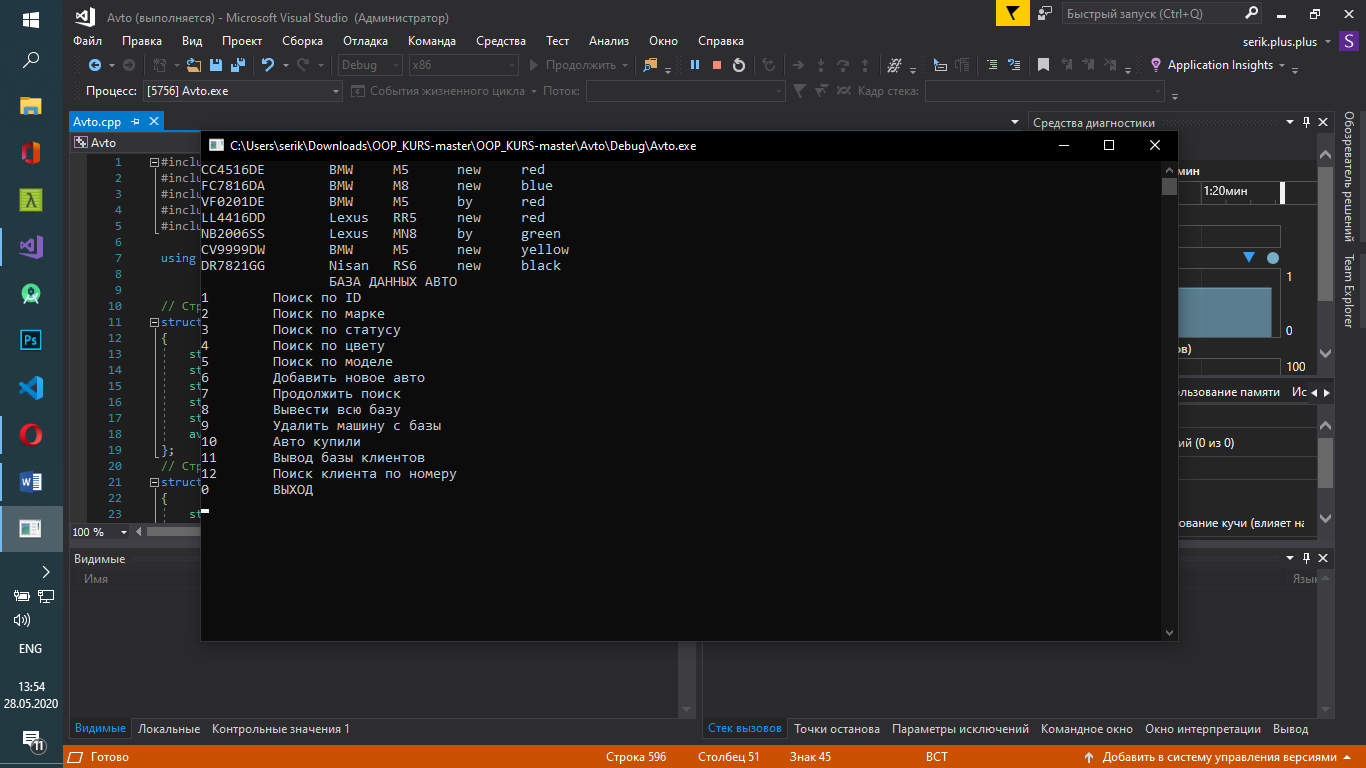
Tail = temp; ++Count;

avtoOut.close();

}







### Реалізація функції DelAvto

Функція здійснює видалення авто з бази за номером та перезапис бази.

int DelAvto()

{

ofstream avtoDel("e:\\avto.txt");

if (Count != 0)

{

string id;

cout << "Введите номер: ";

cin >> id;

int n = 1;

avt \*Del = Head;

while (Del->id != id)

{

Del = Del->next;

++n;

}

// Доходим до элемента,

// который предшествует удаляемому

avt \*PredDel = Del->pred;

// Доходим до элемента, который следует за удаляемым

avt \*AfterDel = Del->next;

// Если удаляем не голову

if (PredDel != 0 && Count != 1)

PredDel->next = AfterDel;

// Если удаляем не хвост

if (AfterDel != 0 && Count != 1)

AfterDel->pred = PredDel;

// Удаляются крайние?

if (n == 1)

Head = AfterDel;

if (n == Count)

Tail = PredDel;

// Удаление элемента

delete Del;

Count--;

if (Count != 0)

{

avt \*temp = Head;

while (temp != 0)

{

avtoDel << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << endl;

temp = temp->next;

}

};

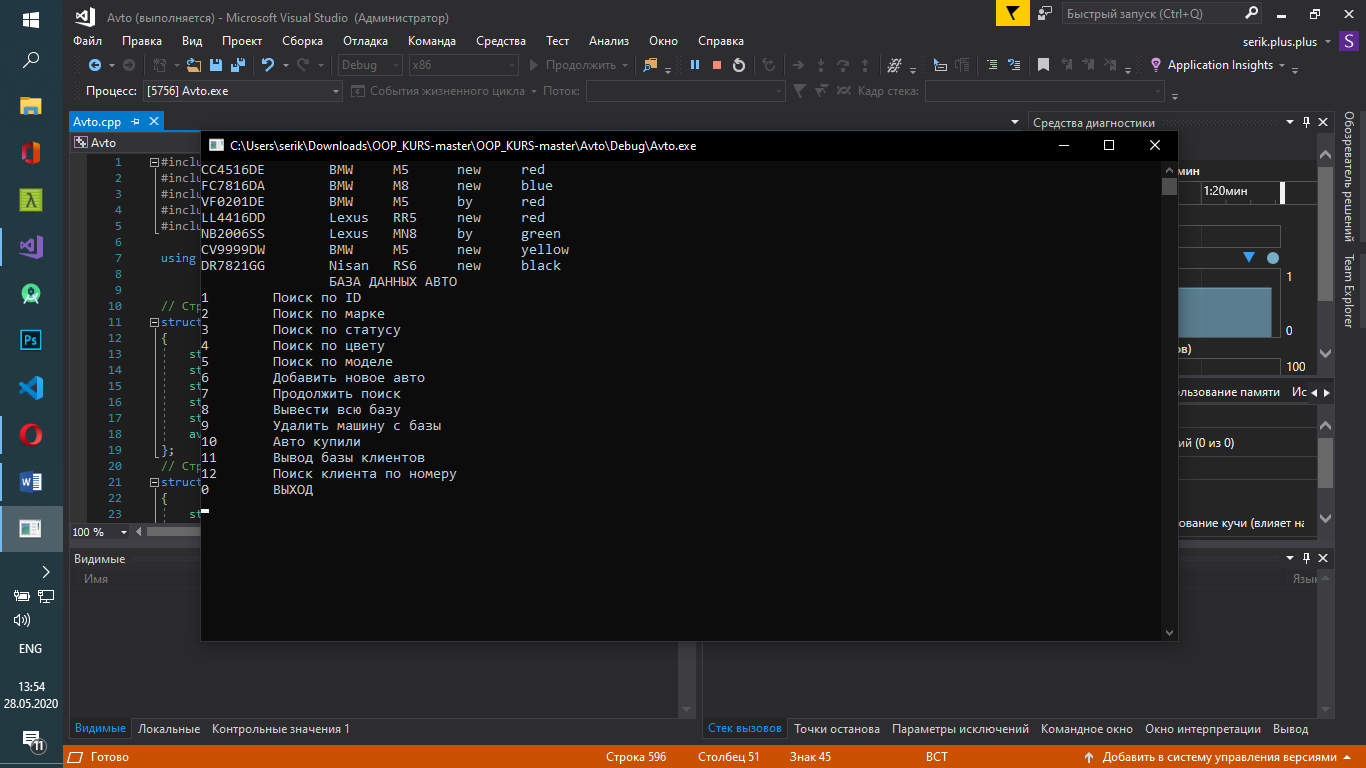
}

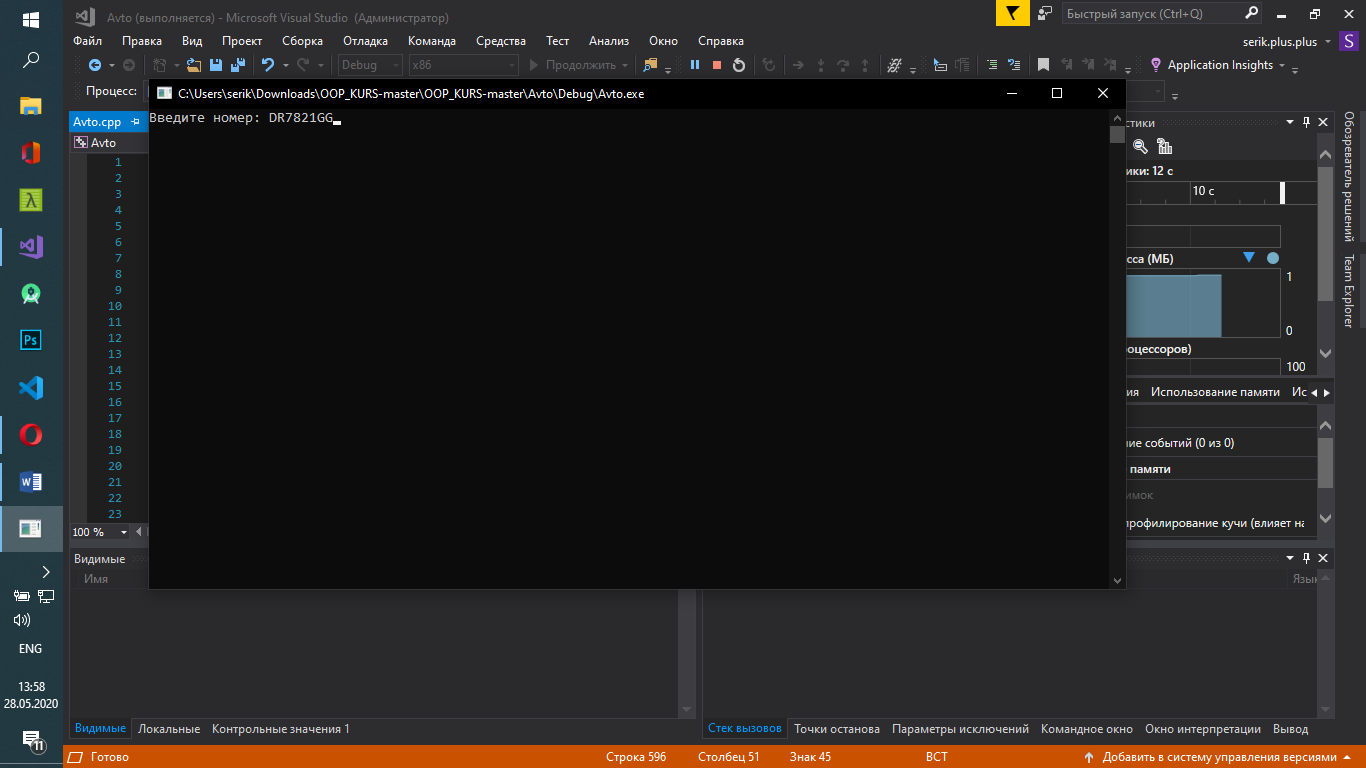
else cout << "Такого номера нет";

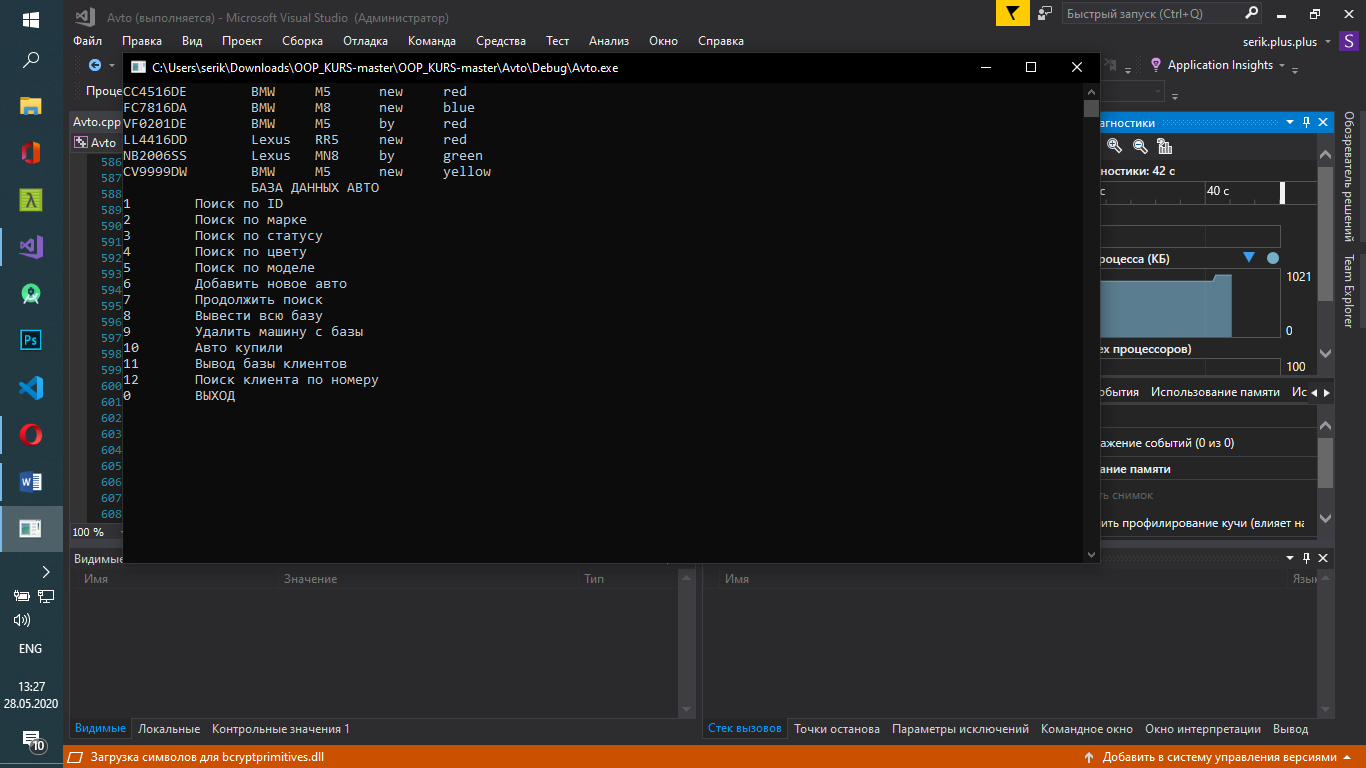
avtoDel.close();

return 0;

}







### Реалізація деструктору

Видалення всіх динамічних змінних класу.

~Avto()

{

delete Head, Tail, HeadSlot, TailSlot;

}

## Реалізація класу Client

class Client : public Avto

{

public:

private:

Klient \*HeadClient,\*TailClient; //зберігання початку та кінця списку Klient

int CountClient; // зберігання кількості клієнтів

};

### Реалізація конструктору

Client()

{

HeadClient = TailClient = 0;

CountClient = 0;

}

### Реалізація функції CreateClient

Функція здійснює первинну ініціалізацію списку Klient з файлу бази. Повертає 0 якщо база має хоча б одного клієнта, 1 якщо база порожня

int СreateClient()

{

ifstream clientIn("e:\\client.txt");

if (CountClient > 0)return 0;

while (!clientIn.eof())

{

Klient \*client= new Klient;

clientIn >> client->name >> client->id >> client->marka >> client->model >> client->status >> client->color;

if (client->name != "")

{

CountClient++;

client->next = 0;

if (TailClient){ TailClient->next = client;}

if (!HeadClient)HeadClient = client;

TailClient = client;

}

else break;

}

clientIn.close();

if (CountClient == 0) return 1;

return 0;

}

void WriteAllClient()

{

if (CountClient != 0)

{

Klient \*temp = HeadClient;

while (temp != 0)

{

cout <<temp->name<<"\t" <<temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

}

}

else cout << "База клиентов пуста";

}

### Реалізація функції WriteAllClient

Функція здійснює повний вивід списку Klient в консоль.

void WriteAllClient()

{

if (CountClient != 0)

{

Klient \*temp = HeadClient;

while (temp != 0)

{

cout <<temp->name<<"\t" <<temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

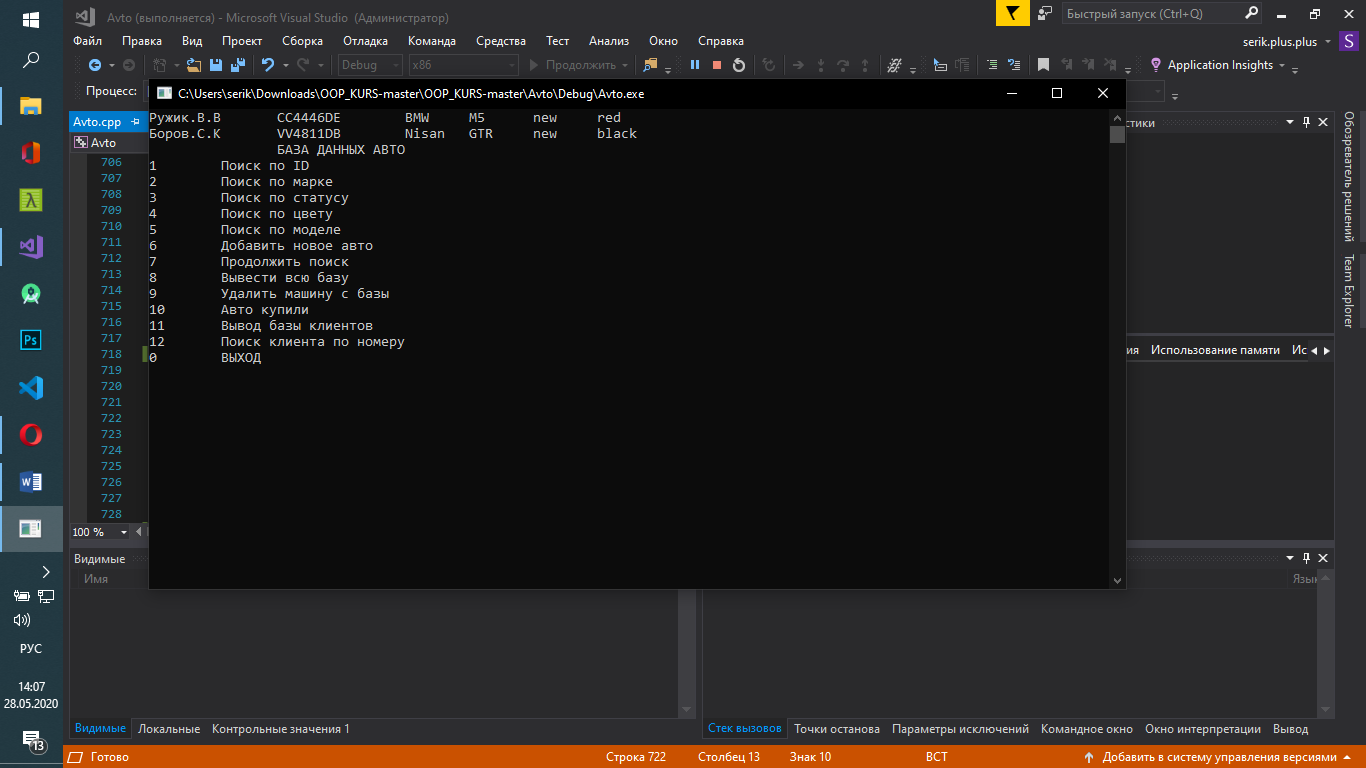
temp = temp->next;

}

}

else cout << "База клиентов пуста";

}



### Реалізація функції SearchClientId

Функція здійснює пошук клієнта за номером його авто.

void SearchClientId()

{

if (CountClient != 0)

{

string id;

int chec = 0;

cout << "Укажите ID" << endl;

cin >> id;

system("cls");

Klient \*temp = HeadClient;

while (temp != 0)

{

if (temp->id == id)

{

cout <<temp->name<<"\t"<< temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

chec++;

break;

}

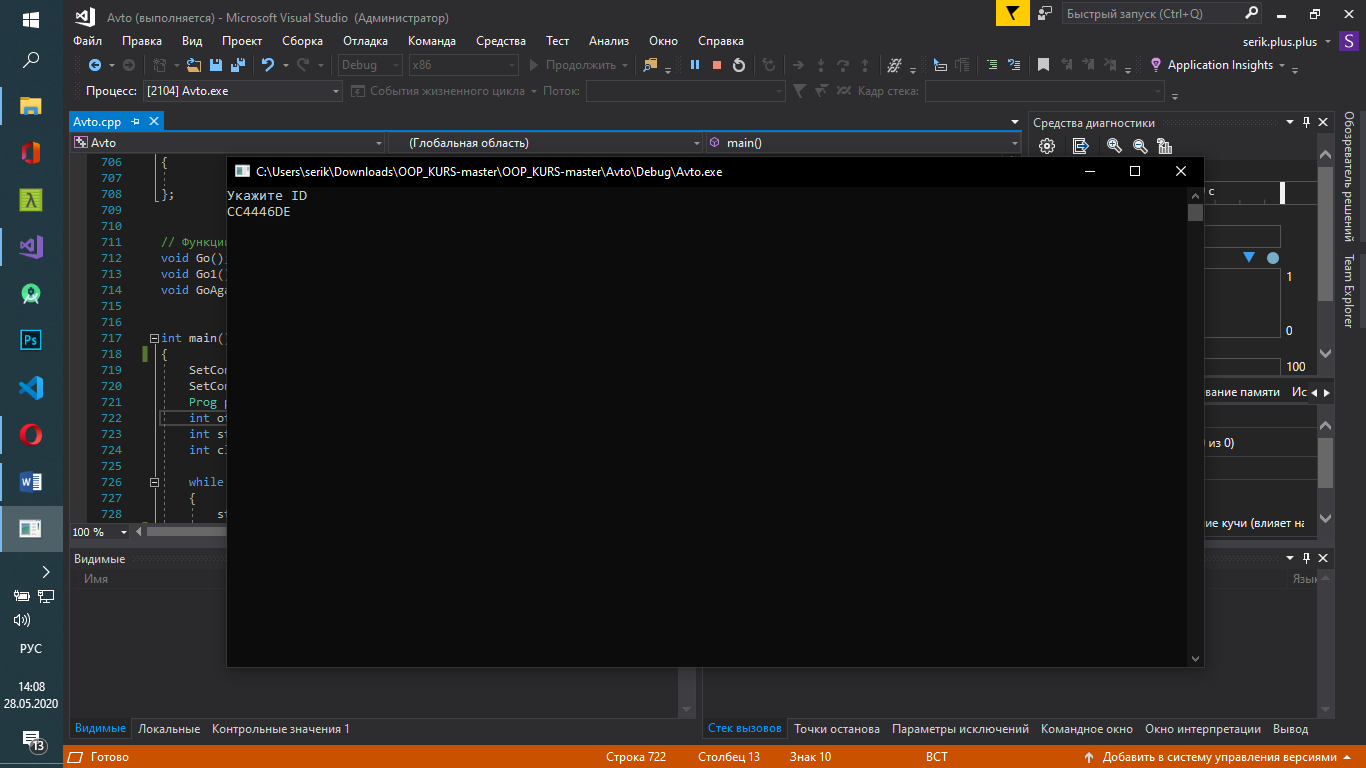
else temp = temp->next;

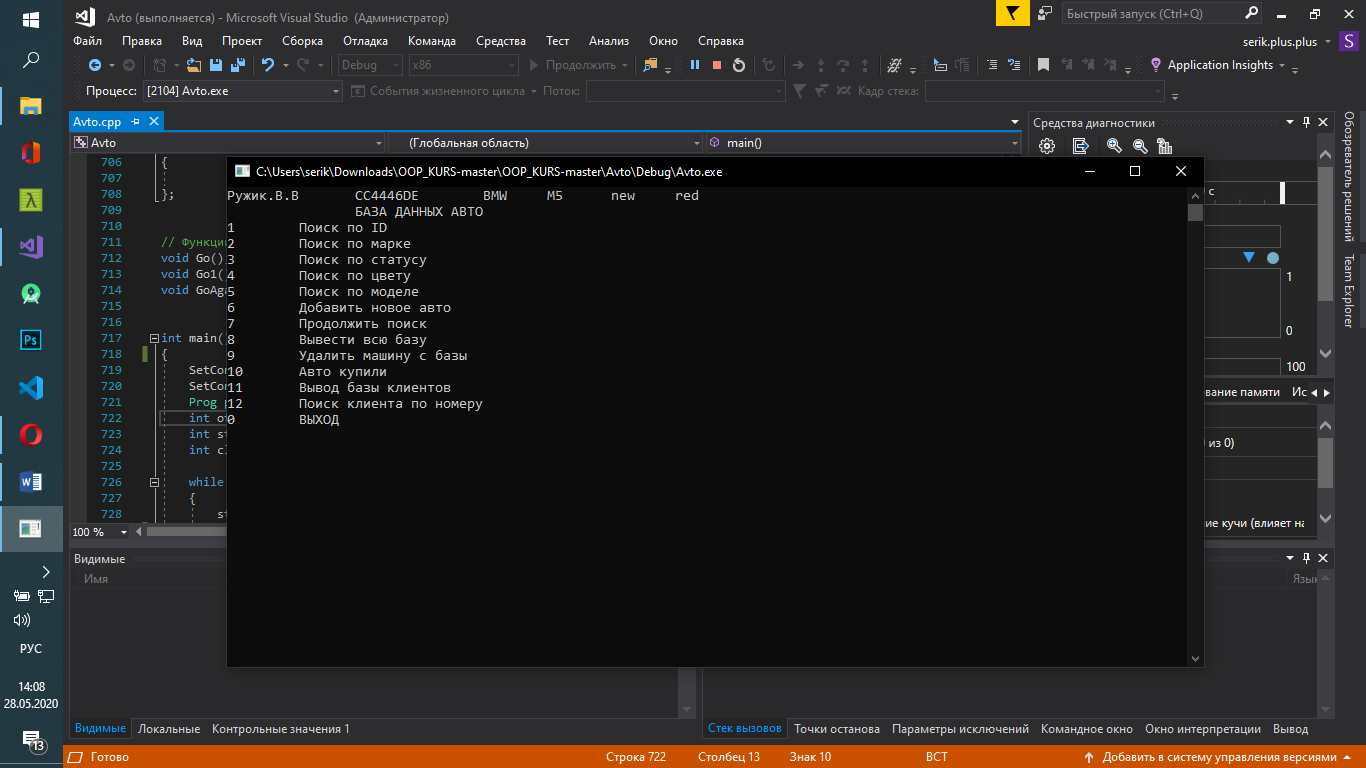
}

if (chec == 0)cout << "Такого клиента в базе нет";

}

}





### Реалізація функції ByAvto

Функція здійснює реалізацію купівлі авто. Видалення авто з бази авто та внесення її в базу клієнтів з додатком ініціалів покупця.

void ByAvto()

{

int v = Count;

if (v != 0)

{

ofstream clientOut("e:\\client.txt", ios::app);

ofstream avtoDel("e:\\avto.txt");

string id;

cout << "Введите номер: ";

cin >> id;

cout << "Введите имя: ";

string name;

cin >> name;

int n = 1;

avt \*Del = Head;

Klient \*client = new Klient;

while (Del->id != id)

{

Del = Del->next;

++n;

}

client->next = 0;

client->name = name;

client->id = Del->id;

client->color = Del->color;

client->marka = Del->marka;

client->model = Del->model;

client->status = Del->status;

if (TailClient != 0)TailClient->next = client;

if (CountClient == 0)HeadClient = TailClient = client;

TailClient = client; CountClient++;

clientOut << client->name <<"\t"<< client->id << "\t" << client->marka << "\t" << client->model << "\t" << client->status << "\t" << client->color << endl;

// Доходим до элемента,

// который предшествует удаляемому

avt \*PredDel = Del->pred;

// Доходим до элемента, который следует за удаляемым

avt \*AfterDel = Del->next;

// Если удаляем не голову

if (PredDel != 0 && (Count != 1))

PredDel->next = AfterDel;

// Если удаляем не хвост

if (AfterDel != 0 && (Count != 1))

AfterDel->pred = PredDel;

// Удаляются крайние?

if (n == 1)

Head = AfterDel;

if (n == Count)

Tail = PredDel;

// Удаление элемента

delete Del;

if (v != 0)

{

avt \*temp = Head;

while (temp != 0)

{

avtoDel << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << endl;

temp = temp->next;

}

};

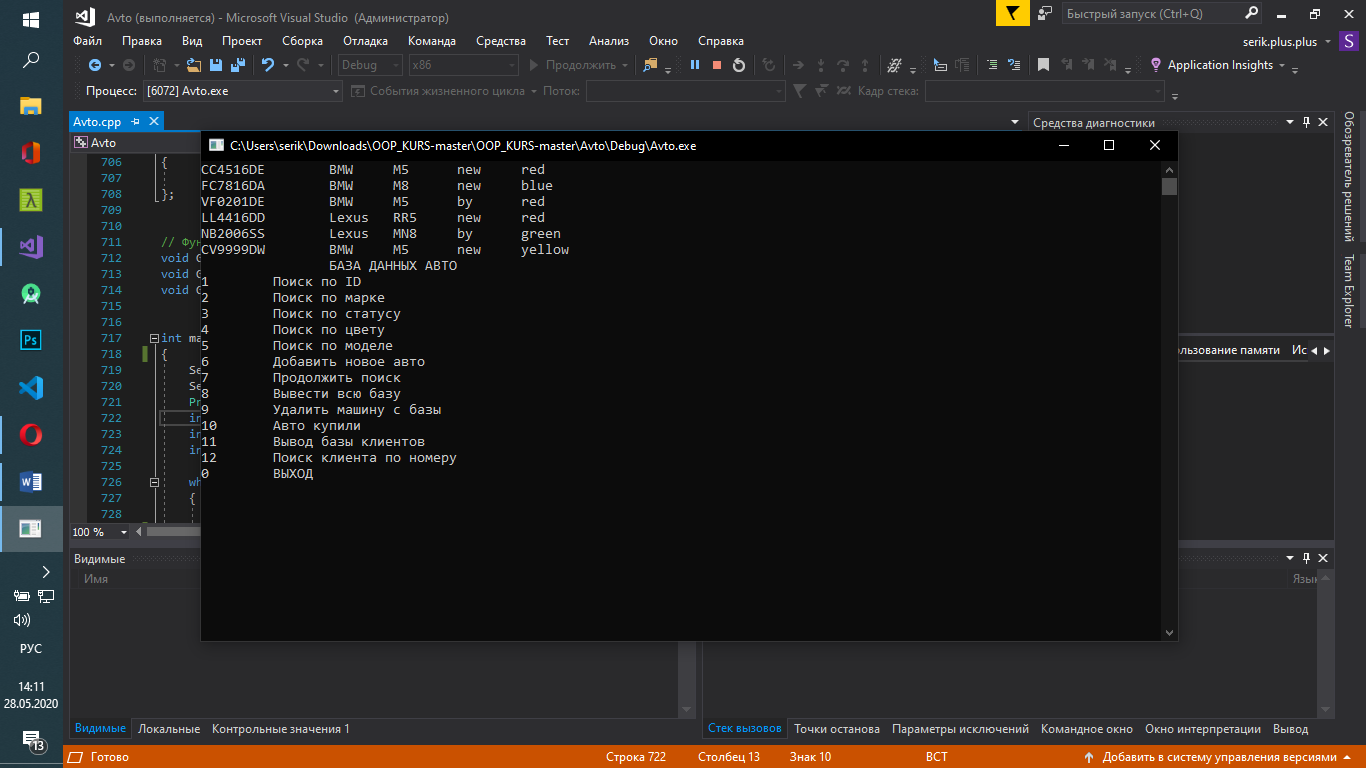
Count--;

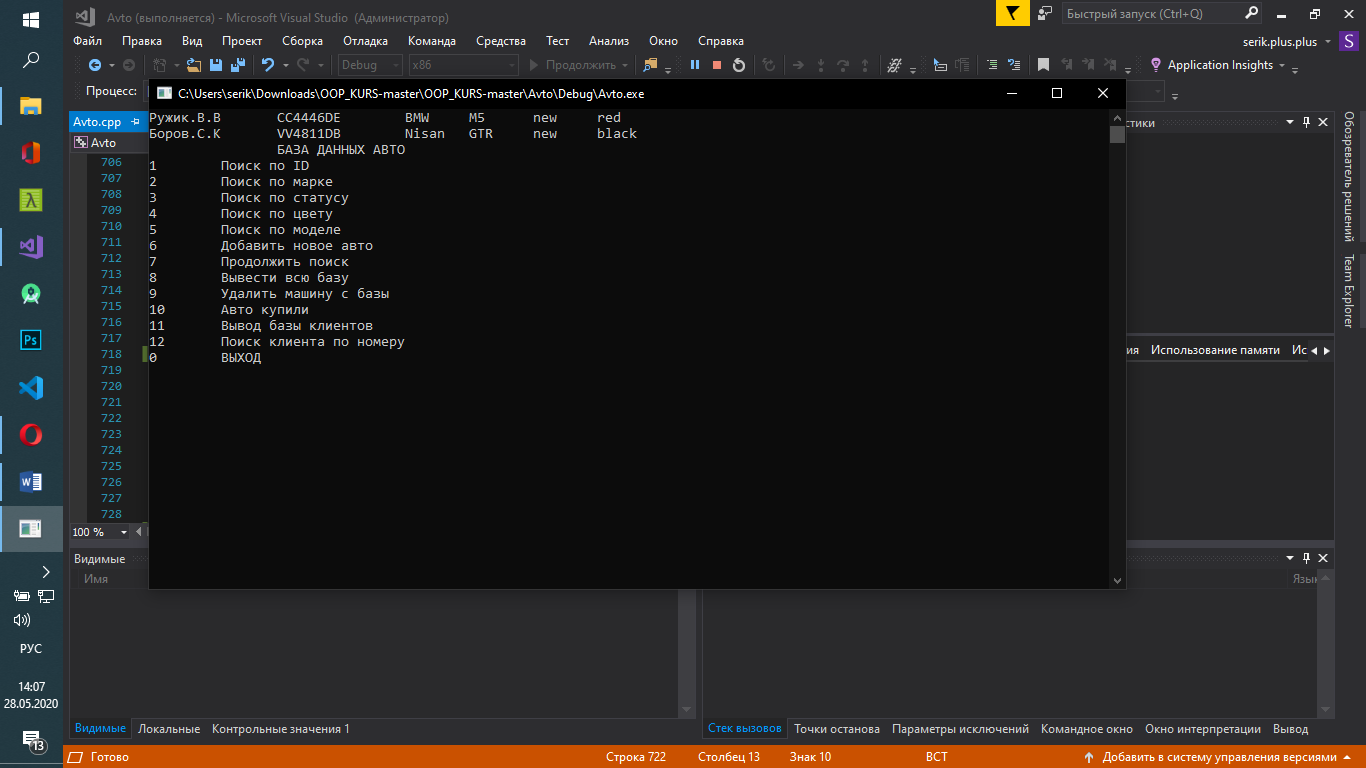
avtoDel.close();

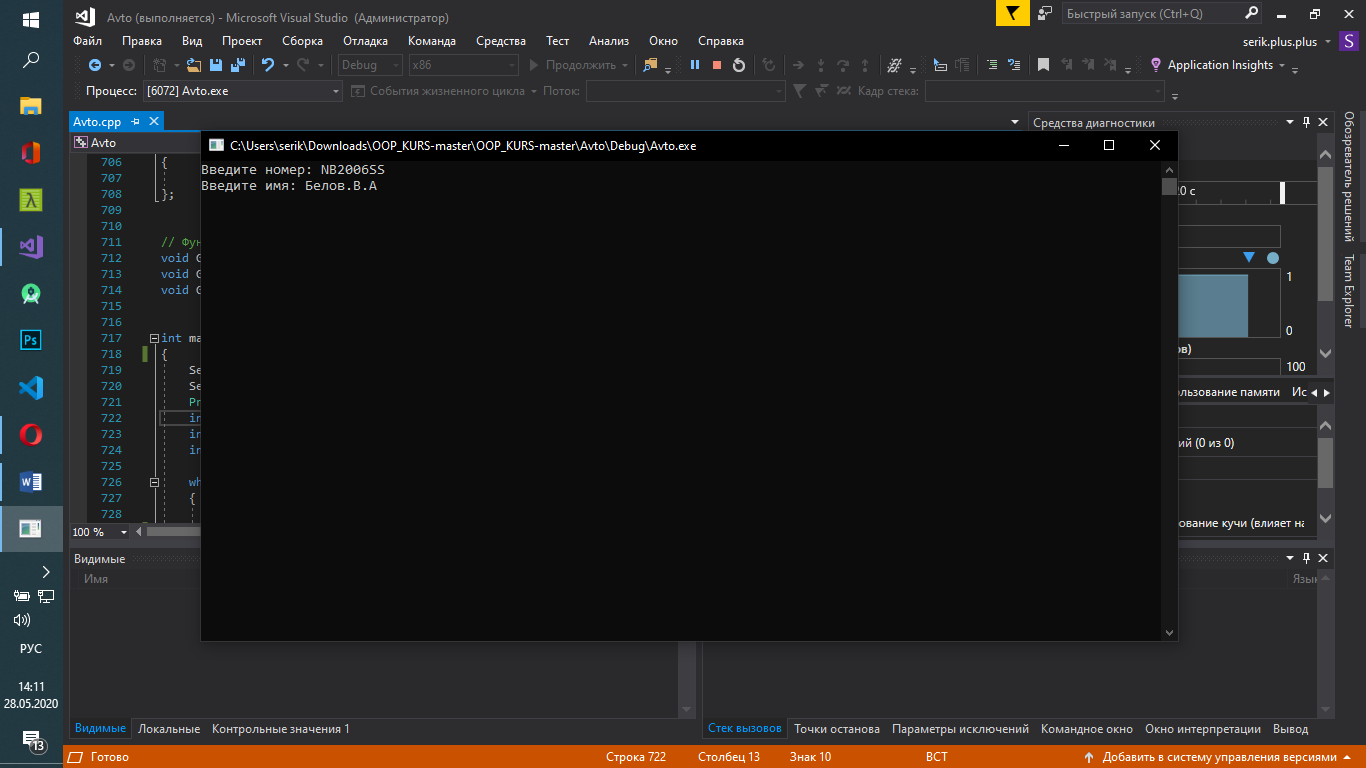
clientOut.close();

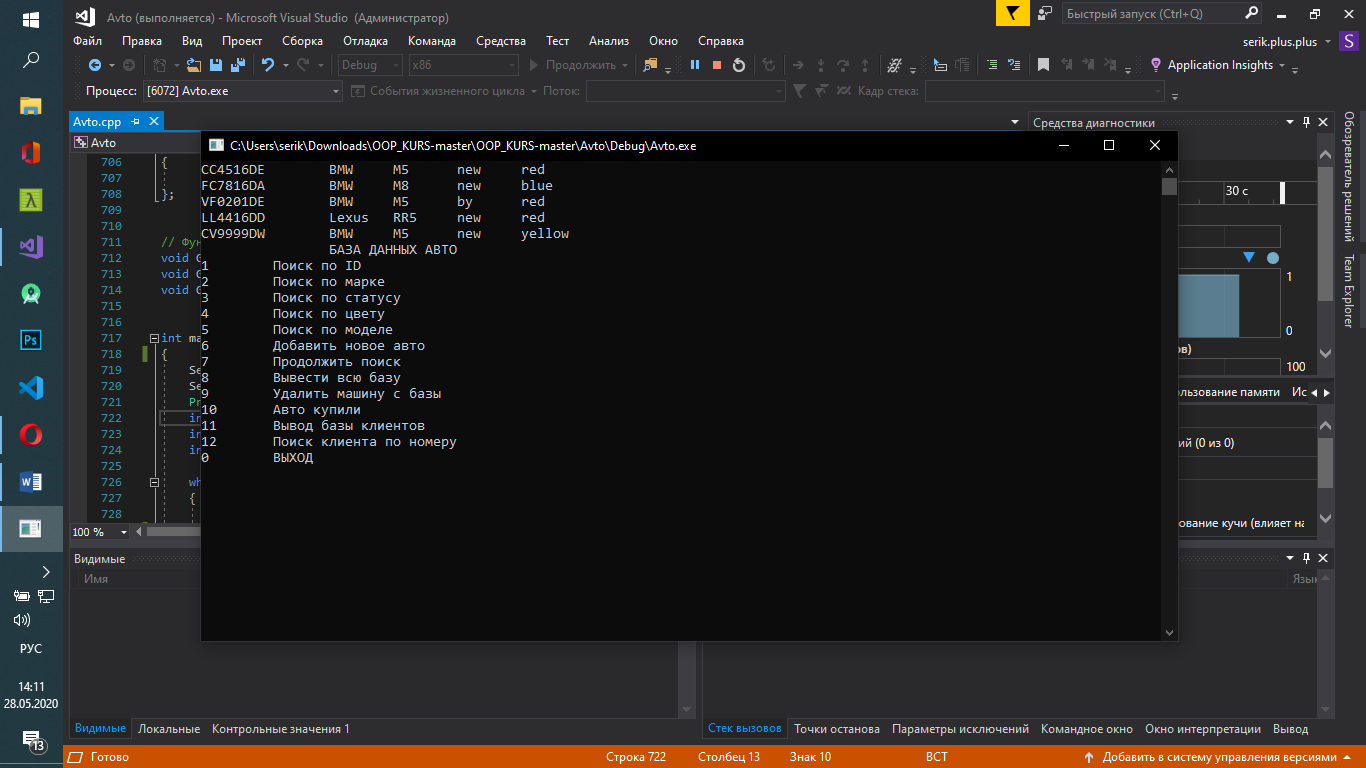
}

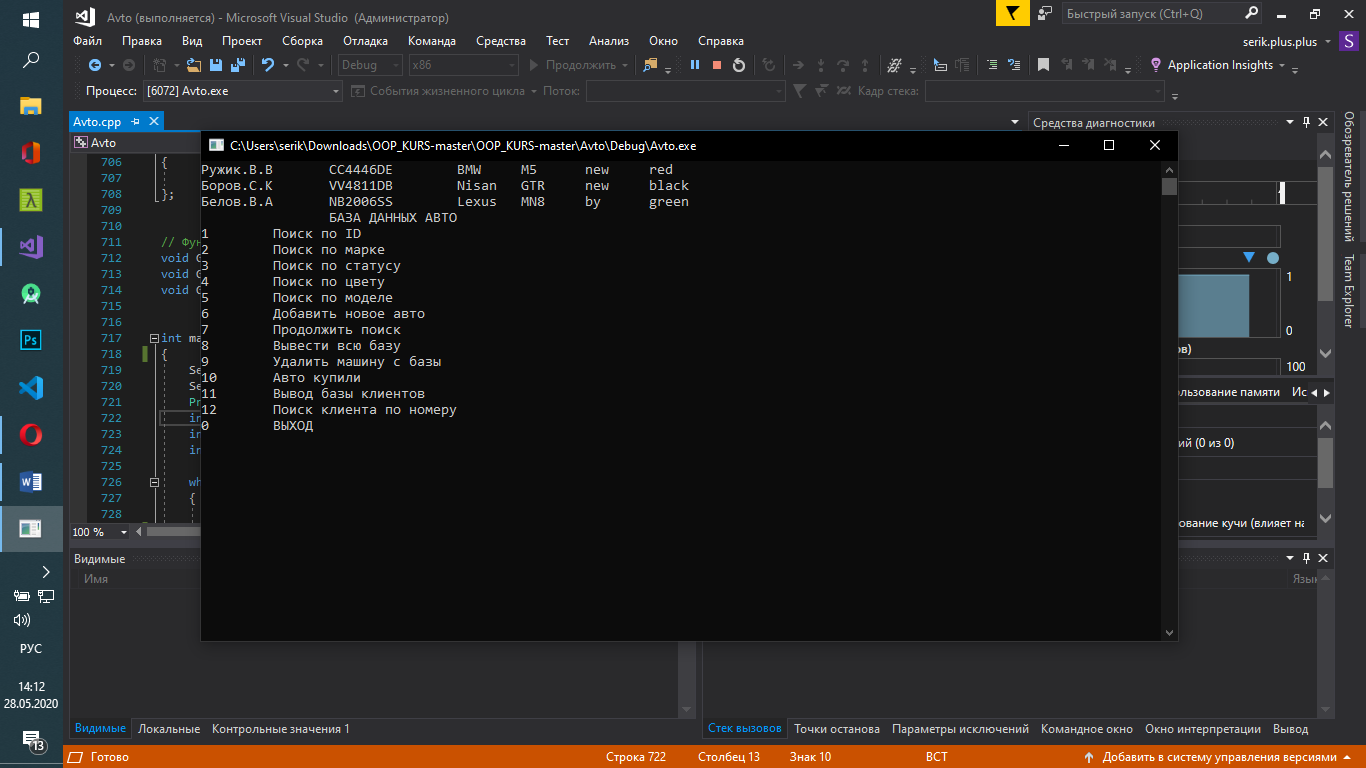
}











### Реалізація деструктору

Видалення всіх динамічних змінних класу.

~Client()

{

delete HeadClient, TailClient;

}

## Реалізація класу Prog

Клас створено для зручності виклику решти функцій програми.

class Prog :public Client

{

};

## Реалізація функції Go

Функція здійснює вивід підказок що до управління програмою.

void Go()

{

cout << "\t\tБАЗА ДАННЫХ АВТО" << endl

<< "1\t Поиск по ID" << endl

<< "2\t Поиск по марке" << endl

<< "3\t Поиск по статусу" << endl

<< "4\t Поиск по цвету" << endl

<< "5\t Поиск по моделе" << endl

<< "6\t Добавить новое авто" << endl

<< "7\t Продолжить поиск" << endl

<< "8\t Вывести всю базу" << endl

<< "9\t Удалить машину с базы" << endl

<< "10\t Авто купили" << endl

<< "11\t Вывод базы клиентов" << endl

<< "12\t Поиск клиента по номеру" << endl

<< "0\t ВЫХОД" << endl;

}

## Реалізація функції Go1

Функція здійснює вивід підказок що до управління програмою.

void Go1()

{

cout << "\t\tБАЗА ДАННЫХ АВТО" << endl

<< "6\t Добавить новое авто" << endl

<< "0\t ВЫХОД" << endl;

}

## Реалізація функції GoAgane

Функція здійснює вивід підказок для додаткового пошуку що до управління програмою.

void GoAgane()

{

cout << "\t\tБАЗА ДАННЫХ АВТО" << endl

<< "1\t Поиск по марке" << endl

<< "2\t Поиск по статусу" << endl

<< "3\t Поиск по цвету" << endl

<< "4\t Поиск по моделе" << endl;

}

## Реалізація головної функції main

SetConsoleCP(1251); // кирилица для вывода

SetConsoleOutputCP(1251); // кирилица для ввода

Prog prog; // обьект управления программою

int otv; // переменная контроля вводу действий в программе

int stbase; //переменная для хранения ответа функции CreateAvto

int clientbase; //переменная для хранения ответа функции СreateClient

while (true) // пезпрерывное действие программы

{

stbase = prog.CreateAvto(); // создаем список авто и сохраняем ответ

clientbase = prog.СreateClient();// создаем список клиентов и сохраняем ответ

switch (stbase) // Анализ ответа функции CreateAvto

{

case 1: // если база пуста

cout << "База пуста" << endl;

Go1();

switch (clientbase) // Анализ ответа функции СreateClient

{

case 1: // если база пуста

cout << "База клиентов пуста" << endl;

break;

case 0: // база имеет поля

cout << "11\t Вывод базы клиентов" << endl;

break;

default:

break;

}

break;

case 0: // база имеет поля

Go();

break;

default:

break;

}

cin >> otv; // ввод числа для действия в программе

switch (otv) // Анализ ввода действия от пользователя

{

case 1:

system("cls");

prog.SearchId();

break;

case 2:

system("cls");

prog.SearchMarka();

break;

case 3:

system("cls");

prog.SearchStatus();

break;

case 4:

system("cls");

prog.SearchColor();

break;

case 5:

system("cls");

prog.SearchModel();

break;

case 6:

system("cls");

prog.AddNewAvto();

break;

case 7:

int ov; // переменная контроля вводу действий в дополнительном поиске

GoAgane();

cin >> ov;

switch (ov) // анализ ввода действий

{

case 1:

system("cls");

prog.SearchMarka(1);

break;

case 2:

system("cls");

prog.SearchStatus(1);

break;

case 3:

system("cls");

prog.SearchColor(1);

break;

case 4:

system("cls");

prog.SearchModel();

break;

default:

break;

}

break;

case 8:

system("cls");

prog.WriteAll();

break;

case 9:

system("cls");

prog.DelAvto();

break;

case 10:

system("cls");

prog.ByAvto();

break;

case 11:

system("cls");

prog.WriteAllClient();

break;

case 12:

system("cls");

prog.SearchClientId();

break;

case 0:

system("cls");

cout << "By";

return 0;

break;

default:

break;

}

}

# Загальний вигляд коду

#include "pch.h"

#include <iostream>

#include <fstream>

#include <string>

#include < windows.h>

using namespace std;

struct avt

{

string id;

string marka;

string model;

string status;

string color;

avt \*next, \*pred;

};

struct avtSlot

{

string id;

string marka;

string model;

string status;

string color;

avtSlot \*next;

};

struct Klient

{

string name;

string id;

string marka;

string model;

string status;

string color;

Klient \*next;

};

class Avto

{

public:

Avto()

{

Head = Tail = 0;

HeadSlot = TailSlot = 0;

Count = 0;

CountSlot = 0;

}

int CreateAvto()

{

ifstream avtoIn("e:\\avto.txt");

if (!avtoIn.is\_open() )

{

cout << "Файл не найден"; return 1;

}

if (Count > 0)return 0;

while (!avtoIn.eof())

{

avt \*temp = new avt;

avtoIn >> temp->id >> temp->marka >> temp->model >> temp->status >> temp->color;

if (temp->id != "")

{

Count++;

temp->next = 0; temp->pred = 0;

if (Tail) { Tail->next = temp; temp->pred = Tail; }

if (!Head)Head = temp;

Tail = temp;

}

else break;

}

avtoIn.close();

if (Count == 0) return 1;

return 0;

}

void WriteAll()

{

if (Count != 0)

{

avt \*temp = Head;

while (temp != 0)

{

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

}

}

}

void SearchId()

{

if (Count != 0)

{

string id;

int chec = 0;

cout << "Укажите ID"<<endl;

cin >> id;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)// пока не конец структуры

{

if (temp->id == id)

{

// Вывод полей и запись их в стековую структуру поиска

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

chec++;

break;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такой авто нет" << endl;

}

}

// Поиск по марке (во всей базе)

void SearchMarka()

{

if (Count != 0)

{

string marka;

int chec = 0;

cout << "Укажите марку авто" << endl;

cin >> marka;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->marka == marka)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такой марки нет" << endl;

}

}

// Поиск по марке (в временной структуре поиска)

void SearchMarka(int n)

{

if (CountSlot != 0)

{

string marka;

int chec = 0;

cout << "Укажите марку авто" << endl;

cin >> marka;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->marka == marka)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такой машины нет" << endl;

}

}

// Поиск по моделе (во всей базе)

void SearchModel()

{

if (Count != 0)

{

string model;

int chec = 0;

cout << "Укажите модель авто" << endl;

cin >> model;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->model == model)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такой модели нет" << endl;

}

}

// Поиск по моделе (в временной структуре поиска)

void SearchModel(int n)

{

if (CountSlot != 0)

{

string model;

int chec = 0;

cout << "Укажите модель авто" << endl;

cin >> model;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->model == model)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такой модели нет" << endl;

}

}

// Поиск по статусу (во всей базе)

void SearchStatus()

{

if (Count != 0)

{

string status;

int chec = 0;

cout << "Укажите статус авто" << endl;

cin >> status;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->status == status)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого статуса нет" << endl;

}

}

// Поиск по статусу (в временной структуре поиска)

void SearchStatus(int n)

{

if (CountSlot != 0)

{

string status;

int chec = 0;

cout << "Укажите статус авто" << endl;

cin >> status;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->status == status)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого статуса нет" << endl;

}

}

// Поиск по цвету (во всей базе)

void SearchColor()

{

if (Count != 0)

{

string color;

int chec = 0;

cout << "Укажите цвет авто" << endl;

cin >> color;

system("cls");

avt \*temp = Head;

CountSlot = 0;

while (temp != 0)

{

if (temp->color == color)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого цвета нет" << endl;

}

}

// Поиск по цвету (в временной структуре поиска)

void SearchColor(int n)

{

if (CountSlot != 0)

{

string color;

int chec = 0;

cout << "Укажите цвет авто" << endl;

cin >> color;

system("cls");

avtSlot \*temp = HeadSlot;

CountSlot = 0;

while (temp != 0)

{

if (temp->color == color)

{

avtSlot \*slot = new avtSlot;

slot->id = temp->id; slot->marka = temp->marka; slot->model = temp->model; slot->status = temp->status; slot->color = temp->color;

cout << slot->id << "\t" << slot->marka << "\t" << slot->model << "\t" << slot->status << "\t" << slot->color << "\t" << endl;

temp = temp->next;

slot->next = 0;

if (CountSlot == 0)

{

HeadSlot = TailSlot = slot;

}

else {

TailSlot->next = slot;

TailSlot = slot;

}

++CountSlot;

chec++;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого цвета нет" << endl;

}

}

// Добавление авто в базу

void AddNewAvto()

{

ofstream avtoOut("e:\\avto.txt", ios::app);

avt \*temp = new avt;

temp->next = 0;

cout << "Номер: " << endl;

cin >> temp->id;

avtoOut << temp->id << "\t";

cout << "Марка: " << endl;

cin >> temp->marka;

avtoOut << temp->marka<<"\t";

cout << "Модель: " << endl;

cin >> temp->model;

avtoOut << temp->model << "\t";

cout << "Статус: " << endl;

cin >> temp->status;

avtoOut << temp->status << "\t";

cout << "Цвет: " << endl;

cin >> temp->color;

avtoOut << temp->color<<"\n";

temp->pred = Tail;

if (Tail != 0)Tail->next = temp;

if (Count == 0)Head = Tail = temp;

Tail = temp; ++Count;

avtoOut.close();

}

int DelAvto()

{

ofstream avtoDel("e:\\avto.txt");

if (Count != 0)

{

string id;

cout << "Введите номер: ";

cin >> id;

int n = 1;

avt \*Del = Head;

while (Del->id != id)

{

Del = Del->next;

++n;

}

// Доходим до элемента,

// который предшествует удаляемому

avt \*PredDel = Del->pred;

// Доходим до элемента, который следует за удаляемым

avt \*AfterDel = Del->next;

// Если удаляем не голову

if (PredDel != 0 && Count != 1)

PredDel->next = AfterDel;

// Если удаляем не хвост

if (AfterDel != 0 && Count != 1)

AfterDel->pred = PredDel;

// Удаляются крайние?

if (n == 1)

Head = AfterDel;

if (n == Count)

Tail = PredDel;

// Удаление элемента

delete Del;

Count--;

if (Count != 0)

{

avt \*temp = Head;

while (temp != 0)

{

avtoDel << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << endl;

temp = temp->next;

}

};

}

else cout << "Такого номера нет";

avtoDel.close();

return 0;

}

~Avto()

{

delete Head, Tail, HeadSlot, TailSlot;

}

friend class Client;

private:

avt \*Head, \*Tail;

avtSlot \*HeadSlot, \*TailSlot;

int Count;

int CountSlot;

};

class Client : public Avto

{

public:

Client()

{

HeadClient = TailClient = 0;

CountClient = 0;

}

int СreateClient()

{

ifstream clientIn("e:\\client.txt");

if (CountClient > 0)return 0;

while (!clientIn.eof())

{

Klient \*client= new Klient;

clientIn >> client->name >> client->id >> client->marka >> client->model >> client->status >> client->color;

if (client->name != "")

{

CountClient++;

client->next = 0;

if (TailClient){ TailClient->next = client;}

if (!HeadClient)HeadClient = client;

TailClient = client;

}

else break;

}

clientIn.close();

if (CountClient == 0) return 1;

return 0;

}

void WriteAllClient()

{

if (CountClient != 0)

{

Klient \*temp = HeadClient;

while (temp != 0)

{

cout <<temp->name<<"\t" <<temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

}

}

else cout << "База клиентов пуста" << endl;

}

void SearchClientId()

{

if (CountClient != 0)

{

string id;

int chec = 0;

cout << "Укажите ID" << endl;

cin >> id;

system("cls");

Klient \*temp = HeadClient;

while (temp != 0)

{

if (temp->id == id)

{

cout <<temp->name<<"\t"<< temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << "\t" << endl;

temp = temp->next;

chec++;

break;

}

else temp = temp->next;

}

if (chec == 0)cout << "Такого клиента в базе нет";

}

}

void ByAvto()

{

int v = Count;

if (v != 0)

{

ofstream clientOut("e:\\client.txt", ios::app);

ofstream avtoDel("e:\\avto.txt");

string id;

cout << "Введите номер: ";

cin >> id;

cout << "Введите имя: ";

string name;

cin >> name;

int n = 1;

avt \*Del = Head;

Klient \*client = new Klient;

while (Del->id != id)

{

Del = Del->next;

++n;

}

client->next = 0;

client->name = name;

client->id = Del->id;

client->color = Del->color;

client->marka = Del->marka;

client->model = Del->model;

client->status = Del->status;

if (TailClient != 0)TailClient->next = client;

if (CountClient == 0)HeadClient = TailClient = client;

TailClient = client; CountClient++;

clientOut << client->name <<"\t"<< client->id << "\t" << client->marka << "\t" << client->model << "\t" << client->status << "\t" << client->color << endl;

// Доходим до элемента,

// который предшествует удаляемому

avt \*PredDel = Del->pred;

// Доходим до элемента, который следует за удаляемым

avt \*AfterDel = Del->next;

// Если удаляем не голову

if (PredDel != 0 && (Count != 1))

PredDel->next = AfterDel;

// Если удаляем не хвост

if (AfterDel != 0 && (Count != 1))

AfterDel->pred = PredDel;

// Удаляются крайние?

if (n == 1)

Head = AfterDel;

if (n == Count)

Tail = PredDel;

// Удаление элемента

delete Del;

if (v != 0)

{

avt \*temp = Head;

while (temp != 0)

{

avtoDel << temp->id << "\t" << temp->marka << "\t" << temp->model << "\t" << temp->status << "\t" << temp->color << endl;

temp = temp->next;

}

};

Count--;

avtoDel.close();

clientOut.close();

}

}

~Client()

{

delete HeadClient, TailClient;

}

private:

Klient \*HeadClient,\*TailClient;

int CountClient;

};

class Prog :public Client

{

};

// Функции вывода

void Go();

void Go1();

void GoAgane();

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Prog prog;

int otv;

int stbase;

int clientbase;

while (true)

{

stbase = prog.CreateAvto();

clientbase = prog.СreateClient();

switch (stbase)

{

case 1:

cout << "База пуста" << endl;

Go1();

switch (clientbase)

{

case 1:

cout << "База клиентов пуста" << endl;

break;

case 0:

cout << "11\t Вывод базы клиентов" << endl;

break;

default:

break;

}

break;

case 0:

Go();

break;

default:

break;

}

cin >> otv;

switch (otv)

{

case 1:

system("cls");

prog.SearchId();

break;

case 2:

system("cls");

prog.SearchMarka();

break;

case 3:

system("cls");

prog.SearchStatus();

break;

case 4:

system("cls");

prog.SearchColor();

break;

case 5:

system("cls");

prog.SearchModel();

break;

case 6:

system("cls");

prog.AddNewAvto();

break;

case 7:

int ov;

GoAgane();

cin >> ov;

switch (ov)

{

case 1:

system("cls");

prog.SearchMarka(1);

break;

case 2:

system("cls");

prog.SearchStatus(1);

break;

case 3:

system("cls");

prog.SearchColor(1);

break;

case 4:

system("cls");

prog.SearchModel();

break;

default:

break;

}

break;

case 8:

system("cls");

prog.WriteAll();

break;

case 9:

system("cls");

prog.DelAvto();

break;

case 10:

system("cls");

prog.ByAvto();

break;

case 11:

system("cls");

prog.WriteAllClient();

break;

case 12:

system("cls");

prog.SearchClientId();

break;

case 0:

system("cls");

cout << "By";

return 0;

break;

default:

break;

}

}

}

void Go()

{

cout << "\t\tБАЗА ДАННЫХ АВТО" << endl

<< "1\t Поиск по ID" << endl

<< "2\t Поиск по марке" << endl

<< "3\t Поиск по статусу" << endl

<< "4\t Поиск по цвету" << endl

<< "5\t Поиск по моделе" << endl

<< "6\t Добавить новое авто" << endl

<< "7\t Продолжить поиск" << endl

<< "8\t Вывести всю базу" << endl

<< "9\t Удалить машину с базы" << endl

<< "10\t Авто купили" << endl

<< "11\t Вывод базы клиентов" << endl

<< "12\t Поиск клиента по номеру" << endl

<< "0\t ВЫХОД" << endl;

}

void GoAgane()

{

cout << "\t\tБАЗА ДАННЫХ АВТО" << endl

<< "1\t Поиск по марке" << endl

<< "2\t Поиск по статусу" << endl

<< "3\t Поиск по цвету" << endl

<< "4\t Поиск по моделе" << endl;

}

void Go1()

{

cout << "\t\tБАЗА ДАННЫХ АВТО" << endl

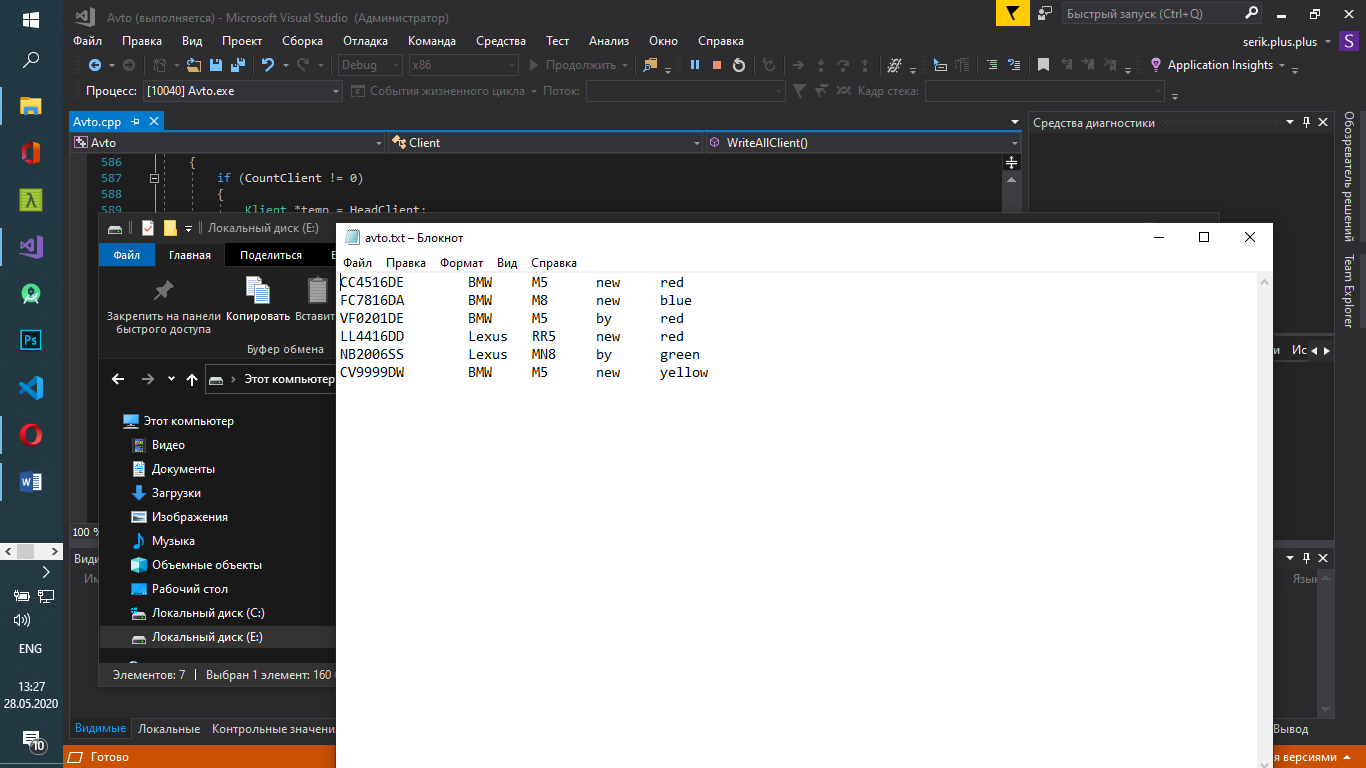
<< "6\t Добавить новое авто" << endl

<< "0\t ВЫХОД" << endl;

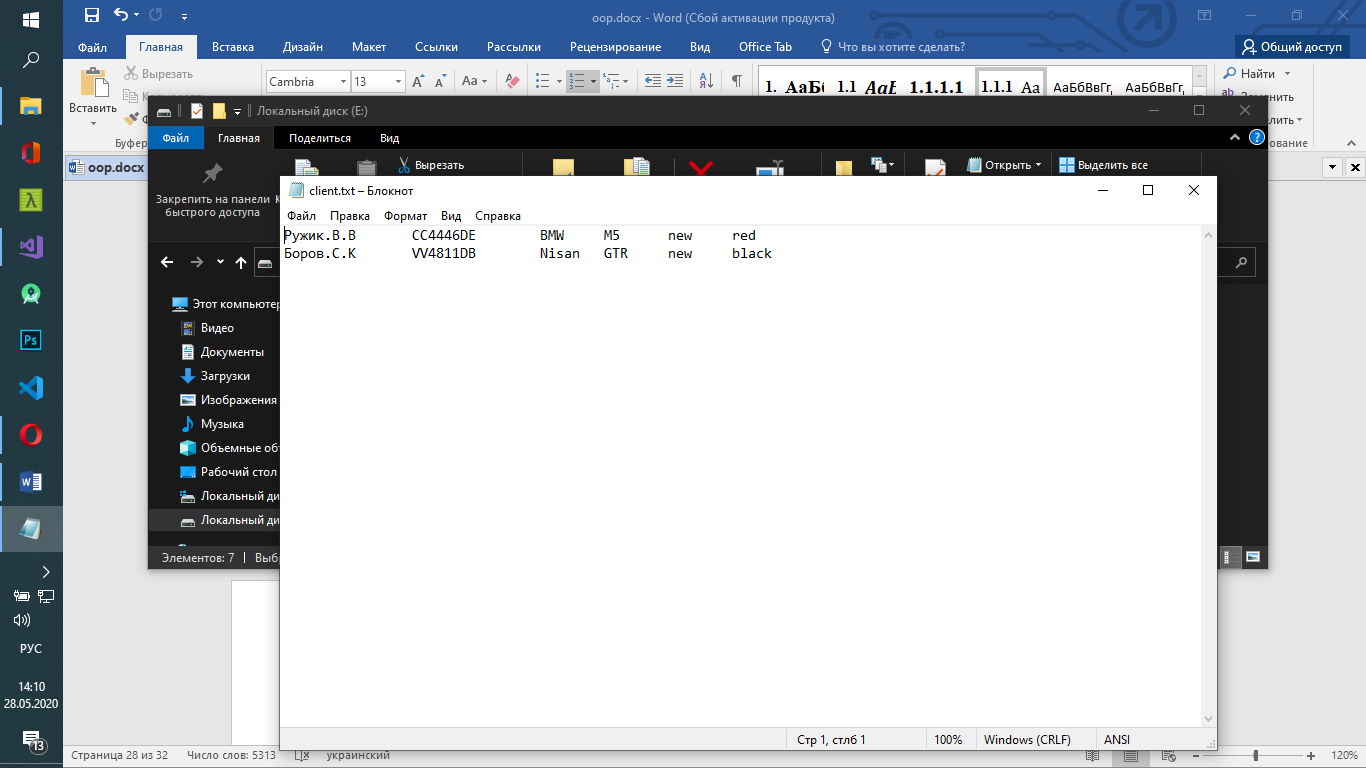
}

# Ісходні дані

## Файл avto.txt



## Файл client.txt



# ВИСНОВКИ

У результаті даної курсової роботи було розроблено програму, з допомогою якої можна вести базу авто та клієнтів. Виконувати різноманітні пошуки авто для підбору потрібного авто для клієнта. Реалізовувати видалення з бази авто які були зняті з продажу. Автоматизовано купівлю авто ( видалення з бази та додавання в базу клієнтів разом з ініціалами покупця). Створено зручний консольний інтерфейс. При зміні баз не треба виходити з програми для підтвердження дій, вона сама автоматично здійснить оновлення даних.

Дані можливості розроблені для максимальної зручності використання.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Грэди Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на С++ = Object-Oriented Analysis and Design with Applications / Пер. И.Романовский, Ф.Андреев. — 2-е изд. — М., СПб.: «Бином», «Невский диалект», 1998. — 560 с. — 6000 экз. — ISBN 5-7989-0067-3.
2. Лука Карделли[en]. Typeful programming ( (англ.)) // IFIP State-of-the-Art Reports. — New York: Springer-Verlag, 1991. — Вып. Formal Description of Programming Concepts.
3. Лука Карделли[en], Peter Wegner. On Understanding Types, Data Abstraction, and Polymorphism // ACM Computing Surveys. — New York, USA: ACM, 1985. — Т. 17, вып. 4. — С. 471–523. — ISSN 0360-0300. — doi:10.1145/6041.6042.
4. Martín Abadi, Лука Карделли[en]. A Semantics of Object Types ( (англ.)). — LICS[en], 1994.
5. Benjamin Pierce. Types and Programming Languages (неопр.). — MIT Press, 2002. — ISBN 0-262-16209-1.
6. Перевод на русский язык: Бенджамин Пирс. Типы в языках программирования (неопр.). — Добросвет, 2012. — 680 с. — ISBN 978-5-7913-0082-9.
7. Design Patterns
8. Иан Грэхем. Объектно-ориентированные методы. Принципы и практика = Object-Oriented Methods: Principles & Practice. — 3-е изд. — М.: «Вильямс», 2004. — С. 880. — ISBN 0-201-61913-X.
9. Антони Синтес. Освой самостоятельно объектно-ориентированное программирование за 21 день = Sams Teach Yourself Object-Oriented Programming in 21 Days. — М.: «Вильямс», 2002. — С. 672. — ISBN 0-672-32109-2.
10. Matt Weisfeld. The Object-Oriented Thought Process. — Fourth Edition. — Addison-Wesley Professional, 2013. — 336 с. — ISBN 978-0-321-86127-6.
11. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированный анализ и проектирование с примерами приложений. — 3-е издание. — «Вильямс», 2010. — ISBN 978-5-8459-1401-9. — ISBN 0-201-89551-X.