**Вступ**

Темою моєї курсової роботи є автоматизоване робоче адміністратора стоматологічної клініки.

Програмне забезпечення є необхідним в сучасному світі. Не винятком є і програмне забезпечення для робити стоматологічної клініки. Це набагато полегшує роботу: дозволяє вводити дані, зберігати їх, редагувати та видаляти, та виводити їх на екран.

В управлінні клініки важливим фактором успішної діяльності є наявність системи обліку, яка б надавала адміністратору контроль за роботою. Дана програма дозволяє вести облік замовлень клієнтів.

Вся інформація зберігається в файловій системі, в якій крім замовлень клієнтів зберігається і інша важлива довідкова інформація: перелік послуг клініки, дані про клієнтів та стоматологів стомат. кабінету.

Важливо розуміти, що програма не є повністю закінченим рішенням. Це лише приклад можливої ​​конфігурації. У всіх організаціях своя специфіка ведення справ і програму можна доопрацювати під потреби конкретної організації прибравши непотрібний функціонал і додавши відсутній.

# **Аналіз специфікації вимог технічного завдання**

Під час проведення аналізу специфікації вимог технічного завдання було поставлено певні цілі, які потрібно було досягти.

Написати програму на мові С++ яка зберігає дані. Програма повинна містити основні функції: додавання запису, видалення запису, редагування запису, пошук запису за заданими параметрами та виведення певних даних на екран.

Функціональне призначення програми:

Дана програма дозволяє зберігати інформацію про послуги, клієнтів, представників та замовлення фотостудії. Програма дозволяє використовувати наступні можливості: додавання нових даних у файлову систему. видалення записів, редагування наявних записів, пошук існуючих записів по даті народження.

Також передбачена перевірка даних, що вводяться, на ідентичність, при введенні ідентичних даних користувачеві буде видано запобігання і дана можливість ввести нові дані. Програма використовує зручний і зрозумілий інтерфейс, організована система меню і підменю.

# **Об’єктно - орієнтований аналіз досліджуваної проблеми та методів моделювання й програмування**

Програма розроблялася на мові програмування С++ у седовищі Code::Blocks.

**C++** (Сі-плюс-плюс) — мова програмування високого рівня з підтримкою кількох парадигмпрограмування: об'єктно-орієнтованої, узагальненої та процедурної.

**Code::Blocks**  — вільне багатоплатформне середовище розробки програмного забезпечення. Code::Blocks написане на [C++](https://uk.wikipedia.org/wiki/C%2B%2B).

Розробка даної програми здійснювалася з дотриманням усіх концепцій об'єктно-орієнтованого програмування. В програмі використовується велика кількість функцій, простих методів, віртуальних методів, булевих методів. Об’єктно орієнтоване прорамування(ООП) – це модель програмування яка базується на ствердженні того, що програма це сукупність об’єктів, які взаємодіють між собою. Кожен об’єкт в цій моделі є незалежним, і він здатний отримувати, оброблляти дані та відправляти ці дані іншим об’єктам. В ООП використано моделі успадкування, поліморфізму та інкапсуляції.

* Інкапсуляція - це механізм, який об’єднує дані і код, який маніпулює цими даними, а також захищає їх від зовнішнього втручання або неправильного використання.
* Поліморфізм **-** це властивість класів, що дозволяє використовувати об’єкти класів з однаковим інтерфейсом без інформаці про тип і внутрішню структуру об’єкта.
* Наслідування **–** це процес за допомогою якогоодтн об’єкт може набути властивості іншого, тобто об’єкт може успадковувати основні властивості іншого об’єкта і додавати до них риси характерні тільки для нього.
* Бінарний (двійоковий) файл— [файл](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB), що містить послідовність довільних [байтів](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82). Назва пов'язана з тим, що байти складаються з [біт](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D1%82), тобто двійкових [цифр](https://uk.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%B8).
* UML-діаграма – це спеціалізований мова графічного опису, призначений для об'єктного моделювання в сфері розробки різного програмного забезпечення. Ця мова має широкий профіль і являє собою відкритий стандарт, в якому використовуються різні графічні позначення, щоб створити абстрактну модель системи. Діаграма класів UML являє собою статичну структурну діаграму, призначену для опису структури системи, а також демонстрації атрибутів, методів і залежностей між кількома різними класами.

# **Процес розробки програми**

**Процес розробки програмного забезпечення** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *software development process*, *software process*) — [сукупність ряду послідовних дій](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81), спрямованих на розробку [програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) (ПЗ).

Існує кілька моделей такого процесу, кожна з яких описує свій підхід, у вигляді завдань і/або діяльності, які мають місце в ході процесу. Вибір тієї або іншої моделі здійснюється відповідно до обраної [методології розробки програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F).

Процес розробки складається з безлічі підпроцесів. Існує 3 моделі процесу розробки програми: водоспадна, ітераційна та спіральна. Модель вибрана мною є ітераційною.

### Ітераційна модель

Альтернативою послідовної моделі є так звана [модель ітеративної та інкрементальної розробки](https://uk.wikipedia.org/wiki/%D0%86%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0_%D1%82%D0%B0_%D1%96%D0%BD%D0%BA%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%BD%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0) ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *iterative and incremental development, IID*), отримала також від Т. Ґілба в 1970-ті назву *еволюційної моделі*. Також цю модель називають *ітеративної моделлю* та і*нкрементальною моделлю*.

Модель IID передбачає розбиття життєвого циклу проекту на послідовність ітерацій, кожна з яких нагадує «міні-проект», включаючи всі процеси розробки в застосуванні до створення менших фрагментів функціональності, порівняно з проектом в цілому. Мета кожної *ітерації* — отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим змістом усіх попередніх і поточної ітерації. Результат фінальної ітерації містить всю необхідну функціональність продукту. Таким чином, із завершенням кожної ітерації продукт отримує приріст — *інкремент* — до його можливостей, які, отже, розвиваються *еволюційно*. Ітеративності, инкрементальность і еволюційність в даному випадку є вислів одного і того ж сенсу різними словами зі злегка різних точок зору.

За висловом Ґілба, «еволюція — прийом, призначений для створення видимості стабільності. Шанси успішного створення складної системи будуть максимальними, якщо вона реалізується у серії невеликих кроків і якщо кожен крок містить у собі чітко визначений успіх, а також можливість „відкату“ до попереднього успішному етапу в разі невдачі. Перед тим, як пустити в справу всі ресурси, призначені для створення системи, розробник має можливість одержувати з реального світу сигнали зворотного зв'язку і виправляти можливі помилки в проекті».

Підхід IID має і свої негативні сторони, які, по суті, — зворотній бік чеснот. По-перше, цілісне розуміння можливостей і обмежень проекту дуже довгий час відсутня. По-друге, при ітераціях доводиться відкидати частину раніше зробленої роботи. По-третє, сумлінність фахівців при виконанні робіт все ж знижується, що психологічно зрозуміло, адже над ними постійно тяжіє відчуття, що «все одно все можна буде переробити і поліпшити пізніше».

# **Розробка системи класів**

Програма складається з 8 класів, кожен з яких має свої змінні та методи.

|  |
| --- |
| **class Client** |
| string name,  lastname,  email,  tel; |
| void message(){}  void getCl(){}  void zdClient(){}  void write\_to\_file(){}  void read\_from\_file(){} |

|  |
| --- |
| **class Dantist** |
| string NameDantist,  SNameDantist,  adress,  posada,  email,  pd,  tel,  salary; |
| void message(){}  void getRl(){}  void zkDantist(){}  void write\_to\_file(){}  void read\_from\_file(){} |

|  |
| --- |
| **class table\_client** |
| vector <Client> vec |
| void print\_all\_clients(){}  void add\_new\_client(){}  int search\_client(){}  void poshuk(){}  void edit\_client(){}  void delete\_client(){}  void write\_clients\_to\_file(){}  void read\_clients\_from\_file(){}  void menu(){}  table\_client(){}  ~table\_client(){} |

|  |
| --- |
| **class table\_dantist** |
| vector <Dantist> vec |
| void print\_all\_dantists(){}  void add\_new\_dantists(){}  int search\_dantists(){}  void poshuk(){}  void edit\_dantists(){}  void delete\_dantists(){}  void write\_dantists\_to\_file(){}  void read\_dantists\_from\_file(){}  void menu\_dantists(){}  table\_dantist (){}  ~table\_dantist(){} |

|  |
| --- |
| **class Services** |
| string services;  int price; |
| void message(){}  void getSer(){}  void zdServices(){}  void write\_to\_file(){}  void read\_from\_file(){} |

|  |
| --- |
| **class table\_services** |
| vector <Services> vec |
| void print\_all\_services(){}  void add\_new\_services(){}  int search\_services(){}  void poshuk(){}  void edit\_services (){}  void delete\_services(){}  void write\_services\_to\_file(){}  void read\_services\_from\_file(){}  void menuServices(){}  table\_services(){}  ~table\_services(){} |

|  |
| --- |
| **class Order** |
| string orderData,  dataOfIssue;  int client\_id,  representative\_id,  service\_id; |
| void message(){}  void getOrder(){}  void zdOrder(){}  void write\_to\_file(){}  void read\_from\_file(){} |

|  |
| --- |
| **class table\_order** |
| vector <Order> vec |
| void print\_all\_orders(){}  void add\_new\_orders (){}  int search\_orders(){}  void poshuk(){}  void edit\_orders (){}  void delete\_orders(){}  void write\_orders\_to\_file(){}  void read\_orders\_from\_file(){}  void menuOrder(){}  table\_orders(){}  ~table\_orders(){} |

Клас Client було створено для зберігання даних про окремого клієнта. У вищевказаному класі було передбачено поля, які зберігають загальні дані, такі як ім’я і прізвище, email та номер телефону. В цьому класі містяться методи, які дозволяють вивести дані про клієнта на екран або зчитати дані з консолі.

Клас table\_client зберігає дані про всіх клієнтів у векторі, реалізує можливість вивести всіх клієнтів, знайти клієнта, редагувати дані, видалити клієнта, а також організовує меню клієнтів.

Клас Dantist було створено для зберігання даних про окремого представника фотостудії. У вищевказаному класі було передбачено змінні, які являють собою загальні дані, такі як ім’я і прізвище, email, паспортні дані, номер телефону, адресу, посаду та зарплату працівника які потрібні для бази даних. В цьому класі містяться методи, які дозволяють вивести дані про представника на екран або зчитати дані з консолі.

Клас table\_dantists зберігає дані про всіх клієнтів у векторі, реалізує можливість вивести всіх предтавників салону, знайти представника, редагувати дані, видалити їх, а також організовує меню представників.

Клас Services було створено для зберігання даних про окрему послугу студії. У вищевказаному класі було передбачено змінні, які являють собою загальні дані, такі як назву послуги та її ціну. В цьому класі містяться методи, які дозволяють вивести дані про послугу на екран або зчитати дані з консолі.

Клас table\_services зберігає дані про всі послуги у векторі, реалізує можливість вивести їх, знайти, редагувати та видалити дані, а також організовує меню послуг.

Клас Order було створено для зберігання даних про окреме замовлення. У вищевказаному класі було передбачено змінні, які являють собою загальні дані, такі як дату замовлення, дату видачі замовлення, ім’я та прізвище клієнта та представника та назву послуги. В цьому класі містяться методи, які дозволяють вивести дані про замовлення на екран або зчитати дані з консолі.

Клас table\_order зберігає дані про всі замовлення у векторі, реалізує можливість виведення їх, знаходження, редагування та видалення даних, а також організовує меню замовлень.

# **Розробка методів**

**Ме́тод** в об'єктно-орієнтованому програмуванні — підпрограма (процедура, функція), що використовується виключно разом із класом (методи класу) або з об'єктом(методи екземпляра).

В даній програмі ми розглянемо приклади методів реалізованих в класі table\_client.

void print\_all\_clients()

{

for(int i=0; i<vec.size(); i++)

{

vec[i].getCl(); /\*cout<<endl;\*/

}

}

void add\_new\_client()

{

Client temp;

temp.zdClient();

vec.push\_back(temp);

}

int search\_client(bool mozhna\_povernutus=true)

{

string firstNameC;

string lastNameC;

int id=-1;

while(true)

{

cout << "\n\tIм'я: ";

cin >> firstNameC;

cout << "\tПрiзвище: ";

cin >> lastNameC;

for(int i=0; i<vec.size(); i++)

{

if((vec[i].getfNameC()==firstNameC)&&(vec[i].getlNameC()==lastNameC))id=i;

}

if(id==-1)

{

if(!mozhna\_povernutus)

{

cout << "\tКлієнта не знайдено. Спробуйте ще раз." << endl;

continue;

}

cout<<"\tКлієнта не знайдено. Спробуйте ще раз |1|, або поверніться в попереднє меню |2|" << endl;

int in;

cin >> in;

switch(in)

{

case 1:

break;

case 2:

return -1;

break;

}

}

else return id;

}

}

void poshuk()

{

int id;

id=search\_client();

if(id!=-1)vec[id].getCl();

}

void edit\_client()

{

int id;

id=search\_client();

if(id!=-1)vec[id].zdClient();

}

void delete\_client()

{

int id;

id=search\_client();

if(id!=-1)

{

vec.erase(vec.begin()+id);

cout << "\tКлiєнта видалено" <<endl;

}

else cout << "\tКлієнта не видалено" <<endl;

}

void write\_clients\_to\_file()

{

ofstream fClient("client1.txt");

for(int i=0; i<vec.size(); i++) vec[i].write\_to\_file(fClient);//для кожного елемента викликати функцію яка записує у файл

fClient.close();

}

void read\_clients\_from\_file()

{

ifstream fClient("client1.txt");

if(fClient.is\_open())

{

Client temp;

while(!fClient.eof())

{

temp.read\_from\_file(fClient);

vec.push\_back(temp);

}

vec.erase(vec.end()); //видалення останнього об'єкта, бо його дублює

fClient.close();

}

}

void menu()

{

while(1)

{

cout <<"\n\tВивести всiх клiєнтiв------>|1|\n\tЗнайти клiєнта------------->|2|\n\tДодати клiєнта------------->|3|\n\tРедагувати данi про клiєнта>|4|\n\tВидалити клiєнта----------->|5|\n\tВийти---------------------->|6|" << endl;

int inC;

cin >> inC;

switch(inC)

{

case 1:

print\_all\_clients();

break;

case 2:

poshuk();

break;

case 3:

add\_new\_client();

break;

case 4:

edit\_client();

break;

case 5:

delete\_client();

break;

case 6:

return;

}

}

}

table\_client()

{

read\_clients\_from\_file(); //конструктор

}

~table\_client()

{

write\_clients\_to\_file(); //деструктор

}

};

Метод void void print\_all\_clients за домопомогою циклу виводить всі дані про всіх клієнтів;

Метод void void add\_new\_client створюєм тимчасовий об’єкт класу Client, викликаєм для нього метод введення даних з клавіатури;

Метод int search\_client знаходить індекс клієнта у векторі за іменем та прізвищем;

Метод void poshuk викликає метод search\_client, який знаходить індекс клієнта і для об’єкта з цим індексом викликає метод виведення на екран;

Метод void edit\_client викликає метод search\_client, який знаходить індекс клієнта і для об’єкта з цим індексом викликає метод введення з консолі (тобто змінює дані);

Метод void delete\_client викликає метод search\_client, який знаходить індекс клієнта і об’єкт з цим індексом видаляє з вектора

Метод void write\_clients\_to\_file для запису даних про клієнтів в файл “Client.txt”;

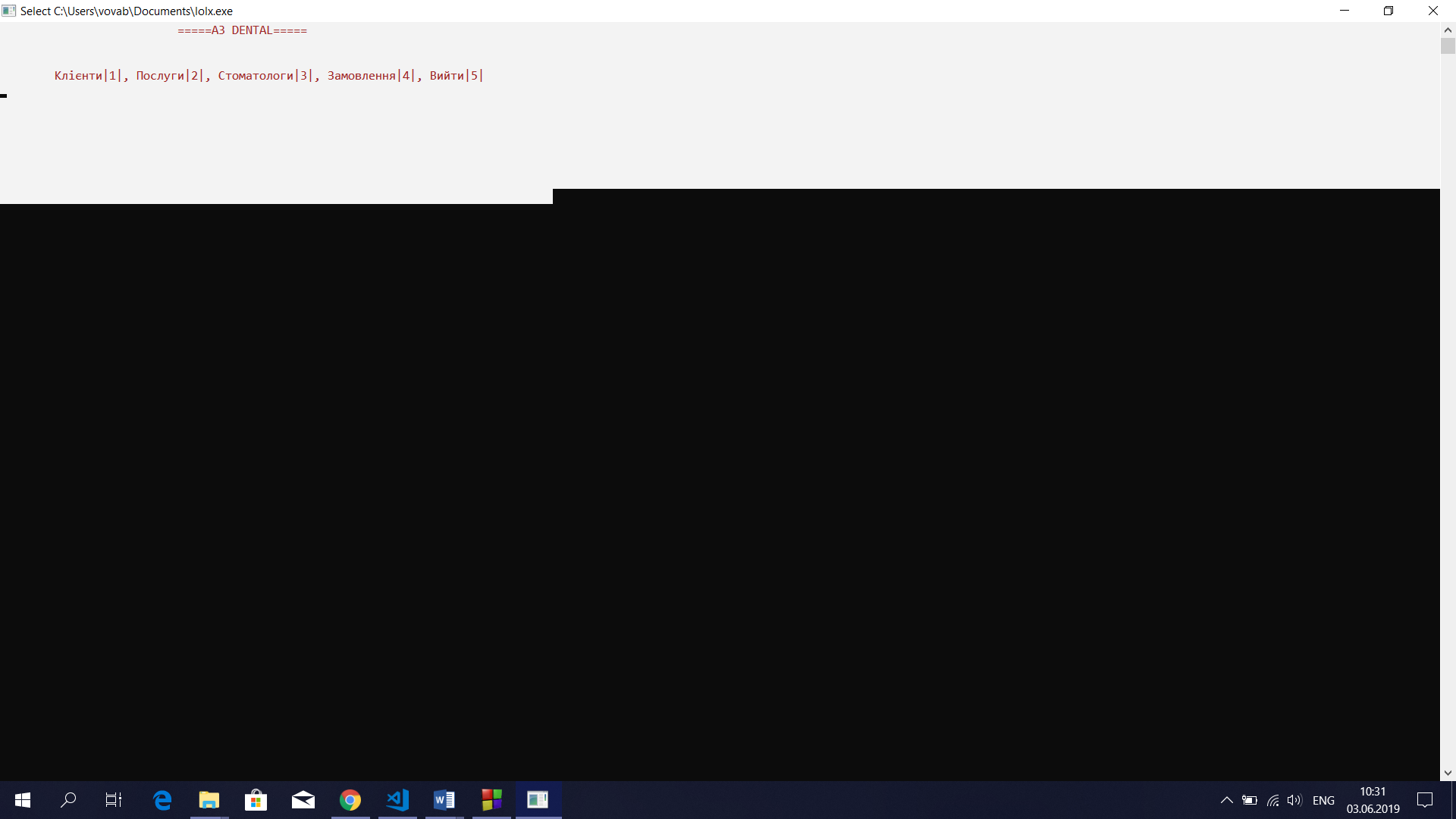
Метод void read\_clients\_from\_file для зчитування даних про клієнтів з файлу “Client.txt”;

Метод void menu реалізує підменю умовної таблиці клієнти;

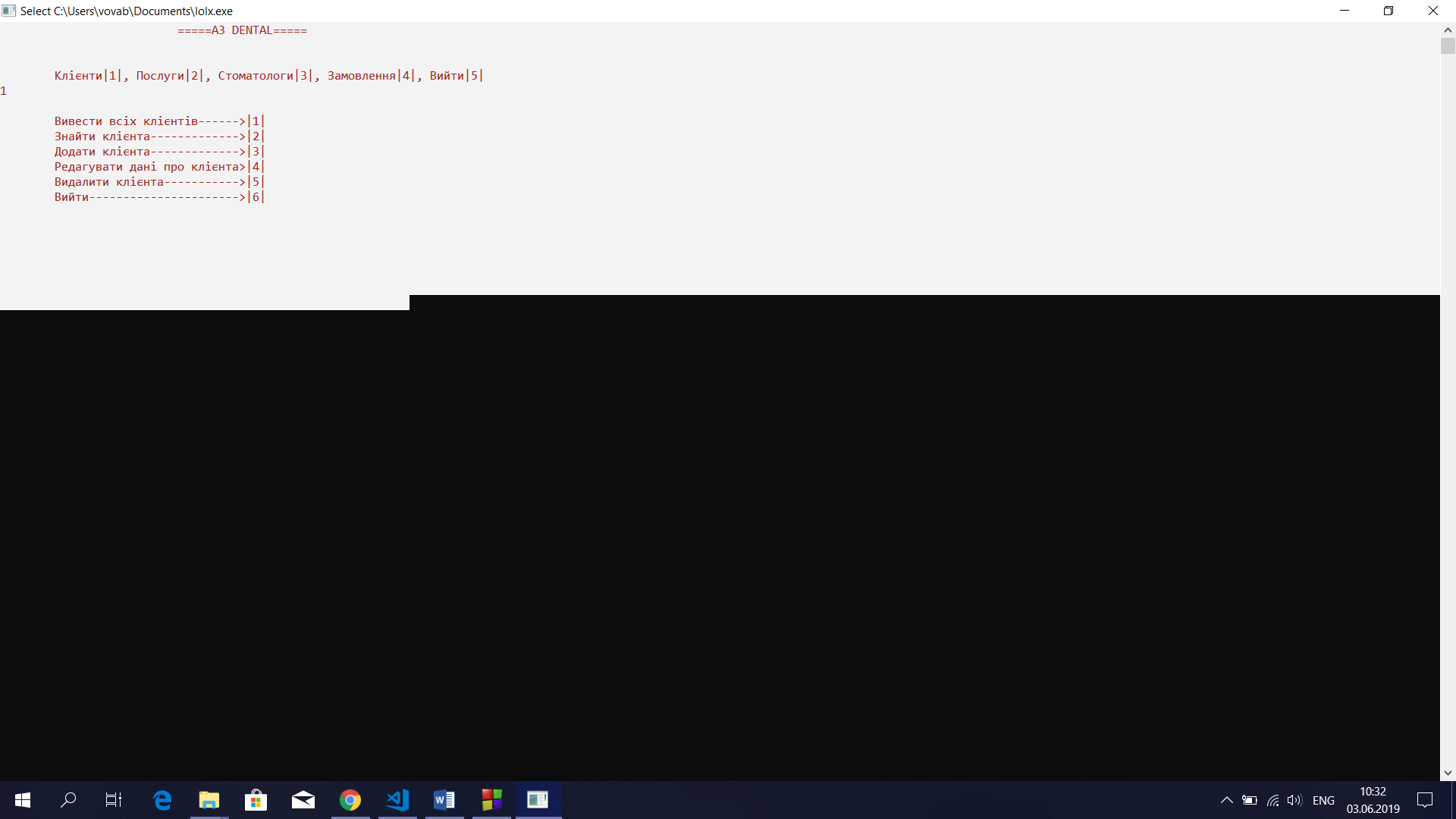
За таким принципом реалізовані методи інших класів.

* 1. **Створення об’єктів, розробка головної програми та опис файлів даних. Інтерфейс прорами.**

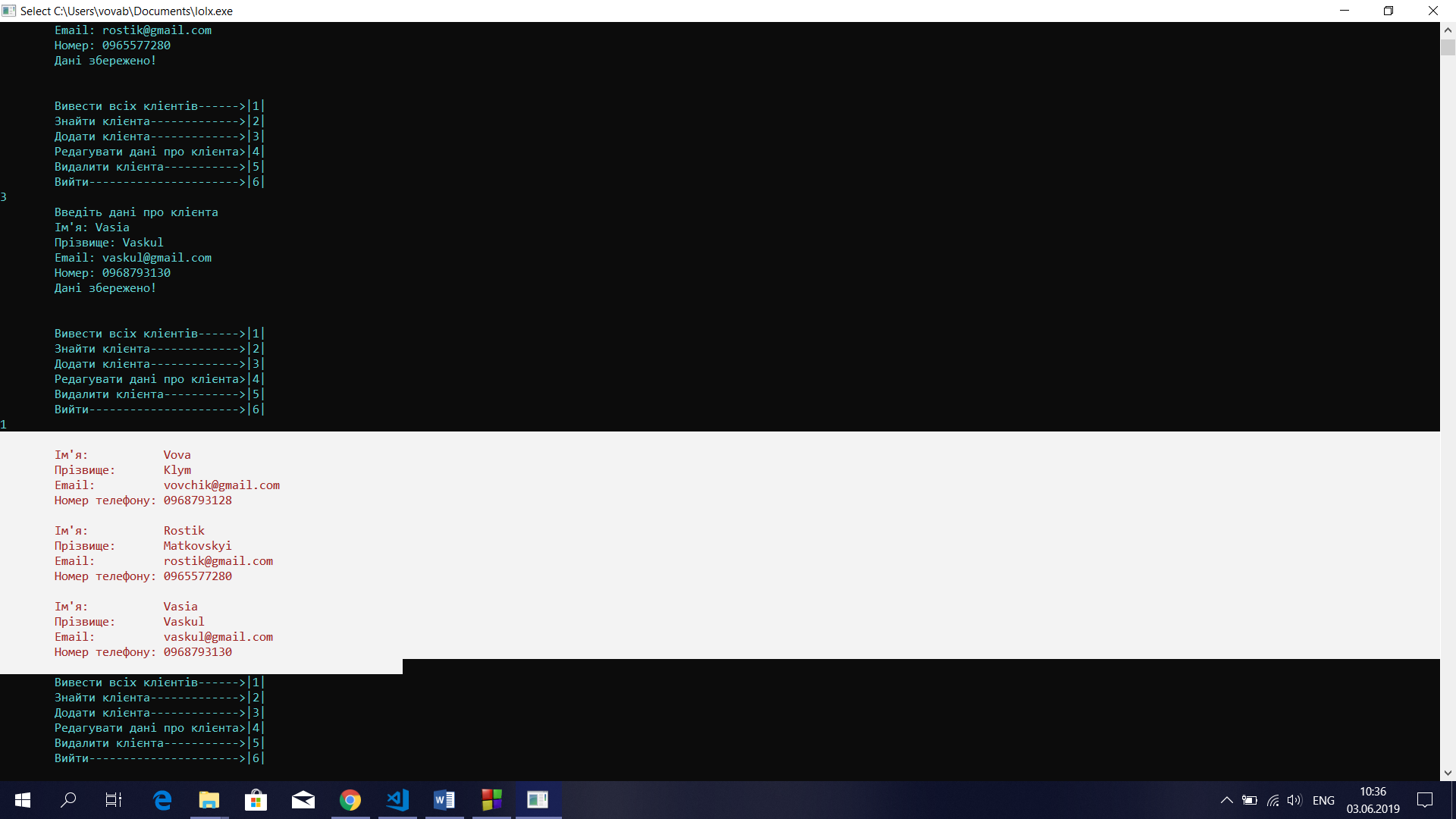
Програма використовує зручний і зрозумілий інтерфейс, організована система меню і підменю.



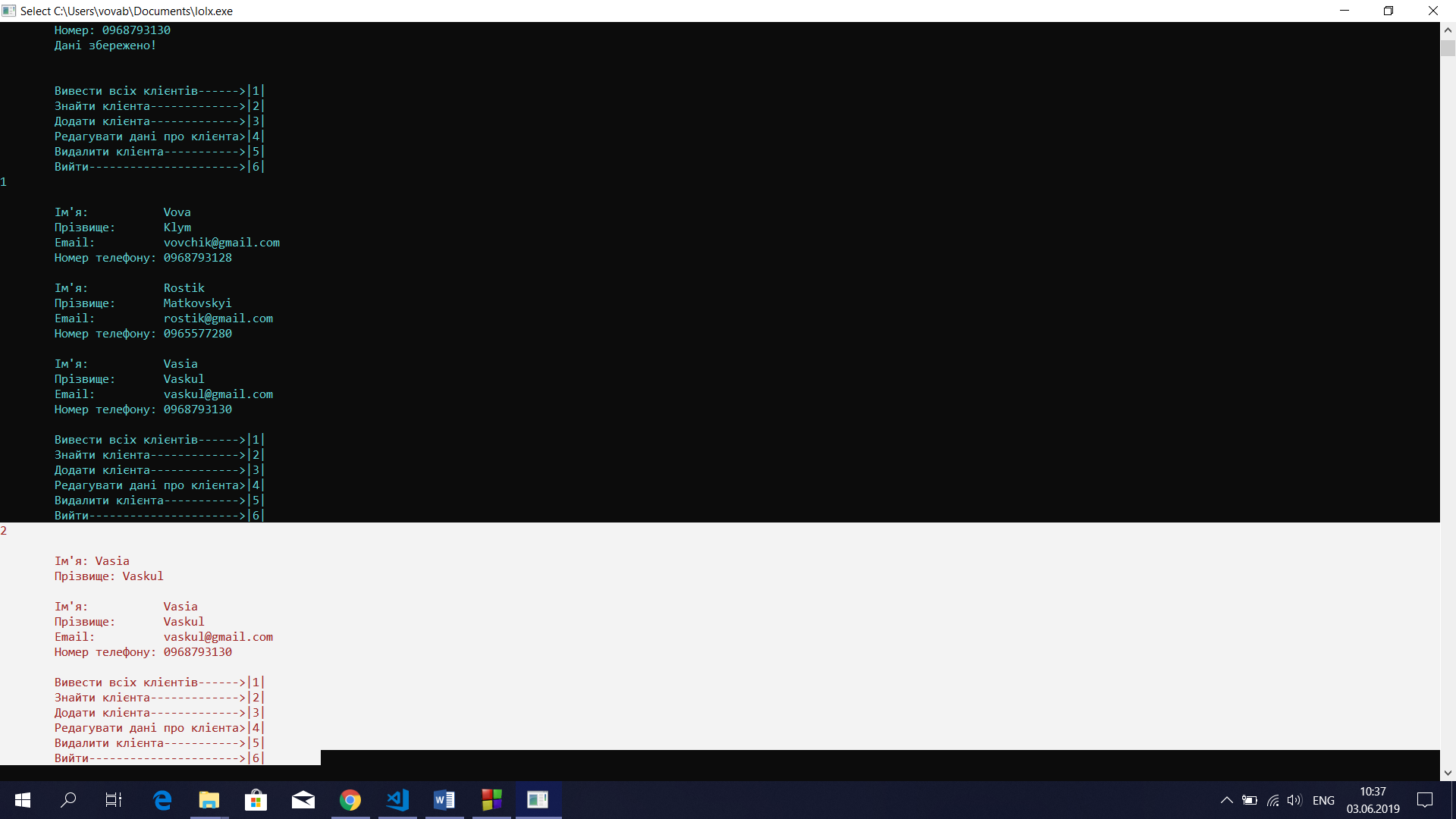
При виборі пункту |1|:



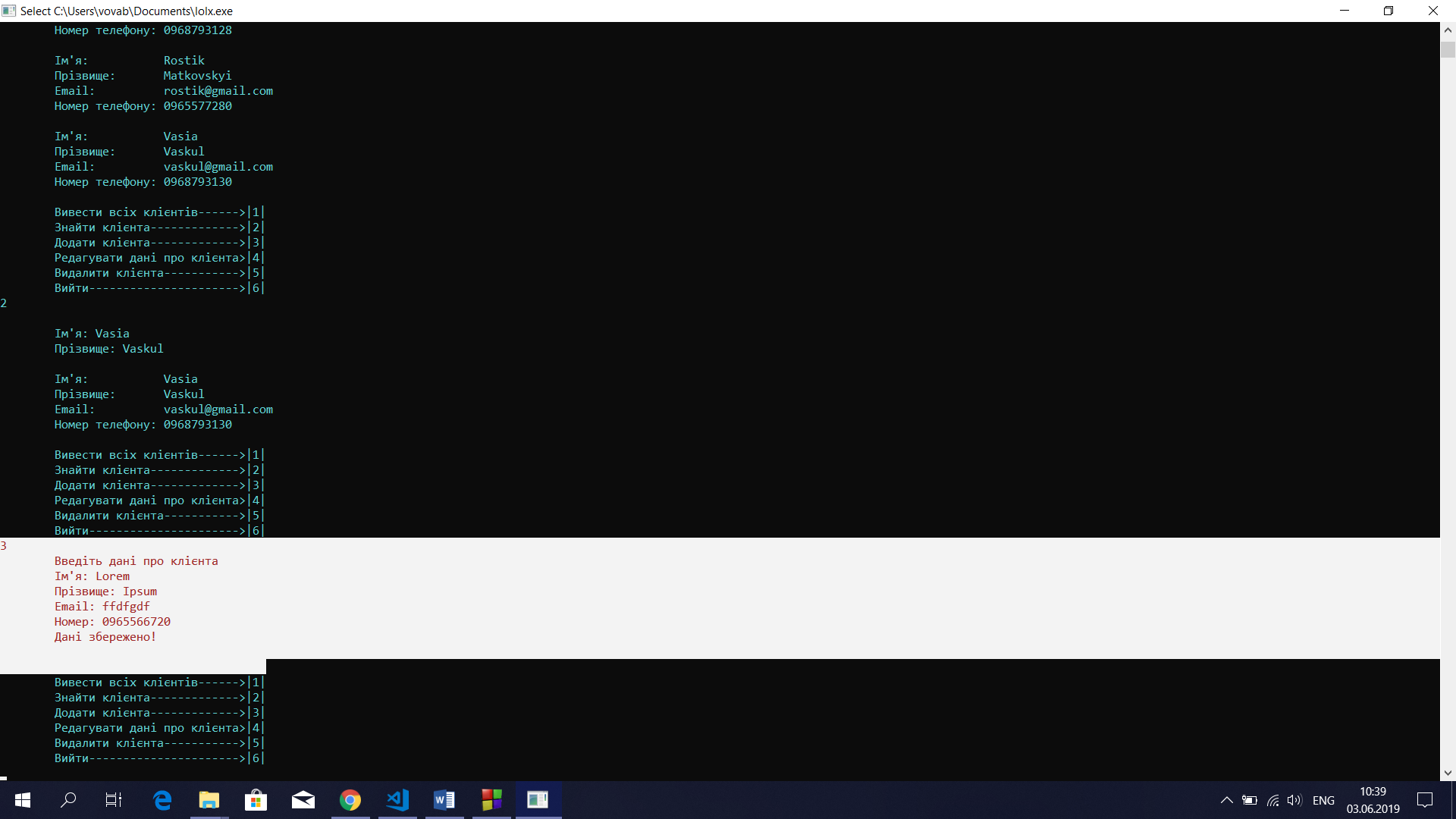
Дані підменю дає змогу вивести всіх клієнтів|1|:



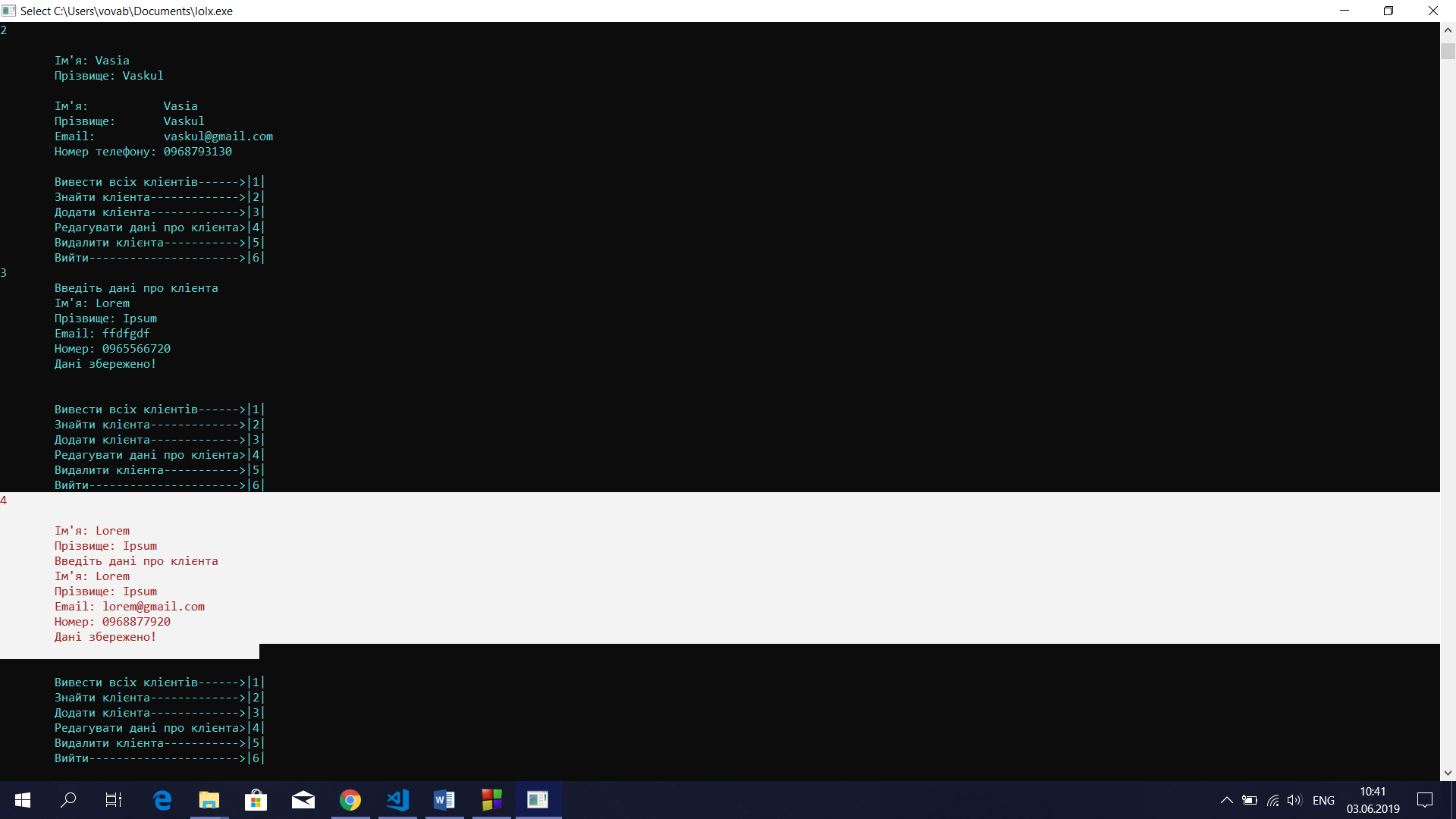
Також знайти дані про клієнта за іменем та прізвищем|2|:



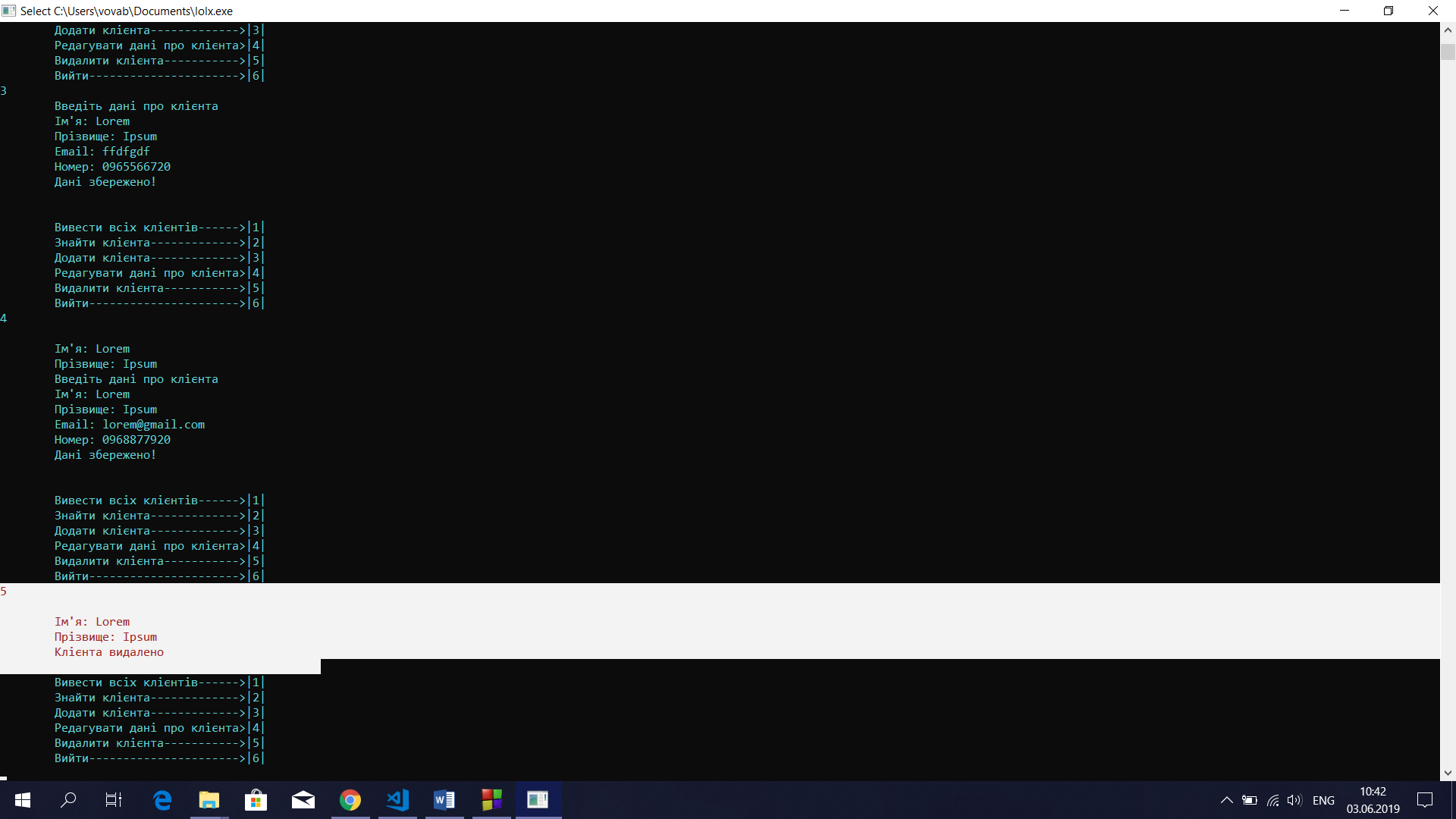
Пунк |3| додає нового клієнта:



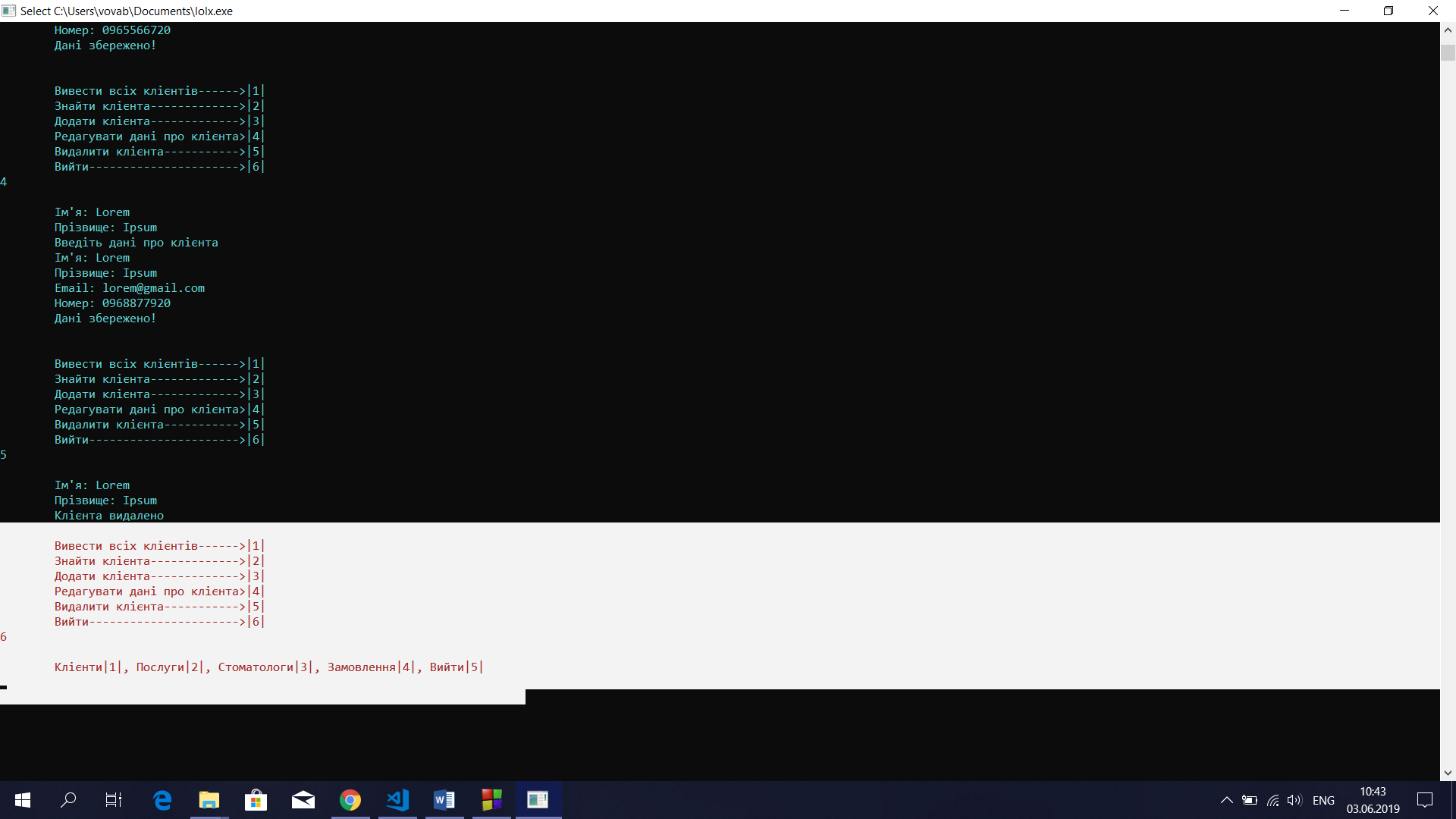
Пунк |4| дає змогу редагувати дані про клієнта:



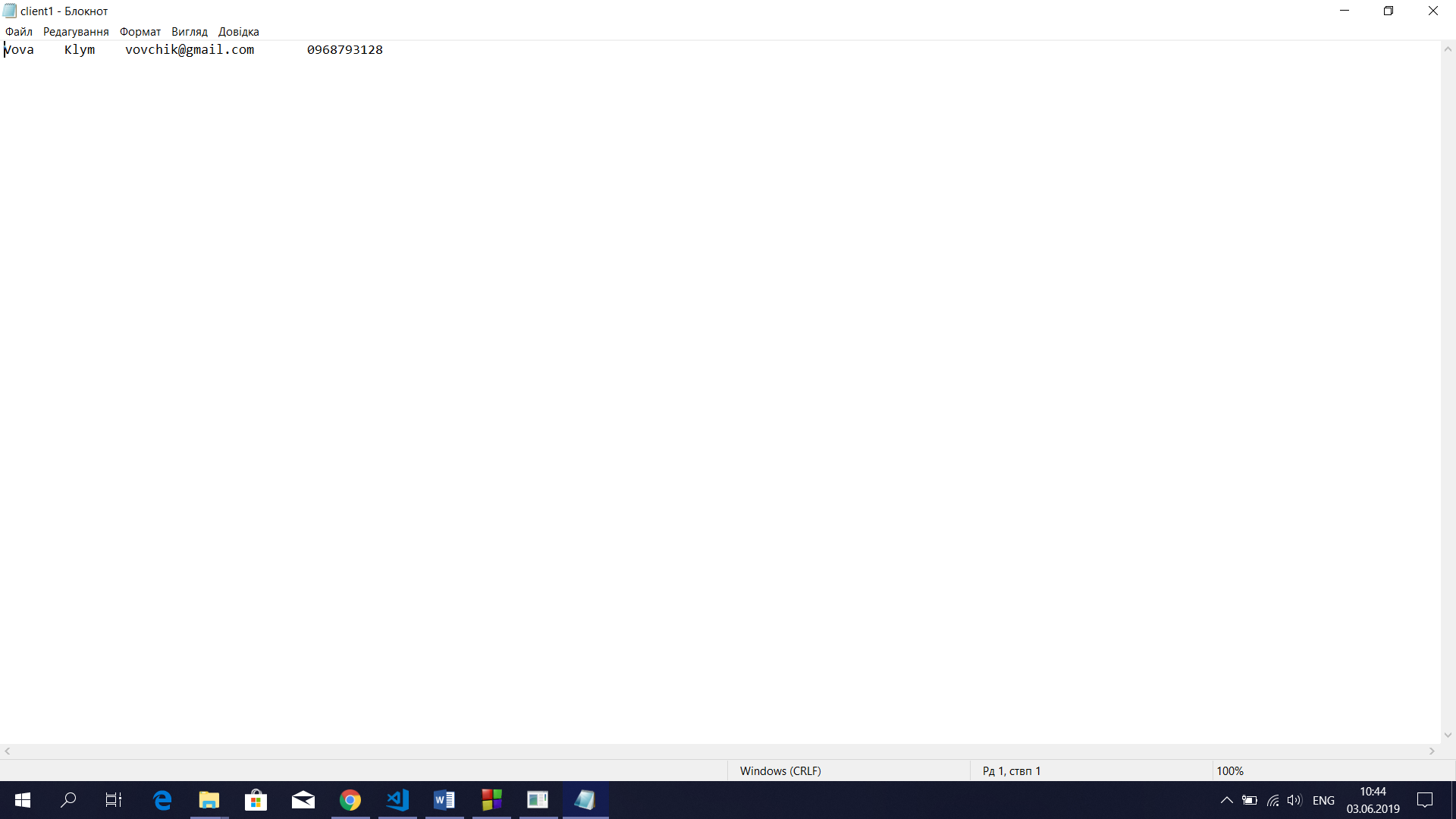
Пунк |5| знаходить клієнта за іменем та прізвищем і видаляє його:



Пунк |6| повертається в головне меню:

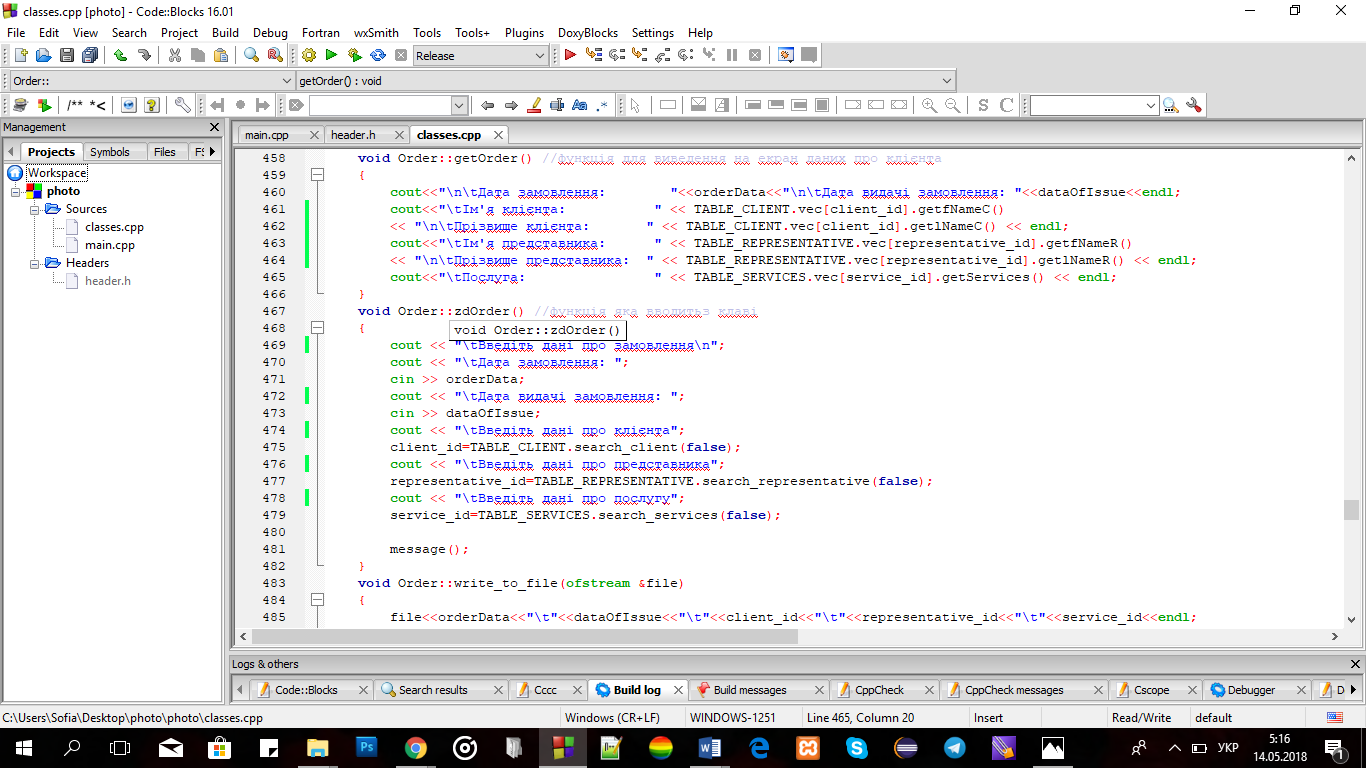


Після завершення робити з програмою всі дані зберігаються в файловій системі.



За таким принципом працюють інші умовні таблиці.

Дана програма створена як багатофайловий проект, вона складається з 3 файлів:



У файлі main.cpp реалізоване головне меню і створене об’єкти деяких класів;

У файлі header.h оголошуються класи;

У файлі classes.cpp реалізовані методи;

Бібліотеки які використовувалися в програмі:

**#include<iostream>** - стандатрна бібліотека, для введення-виведення в мові програмувння С++.

**#include<string.h>** - бібліотека необхідна для роботи з рядками(string).

**#include<fstream>** - стандартна бібліотека для роботи з файлами.

**#include<vector>** - стандартна бібліотека для роботи з векторами.

**#include<cstdlib>** - бібліотека для exit

1. **Тестування програми і результат її виконання**

Тестування програмного забезпечення —процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись

Оцінюється:

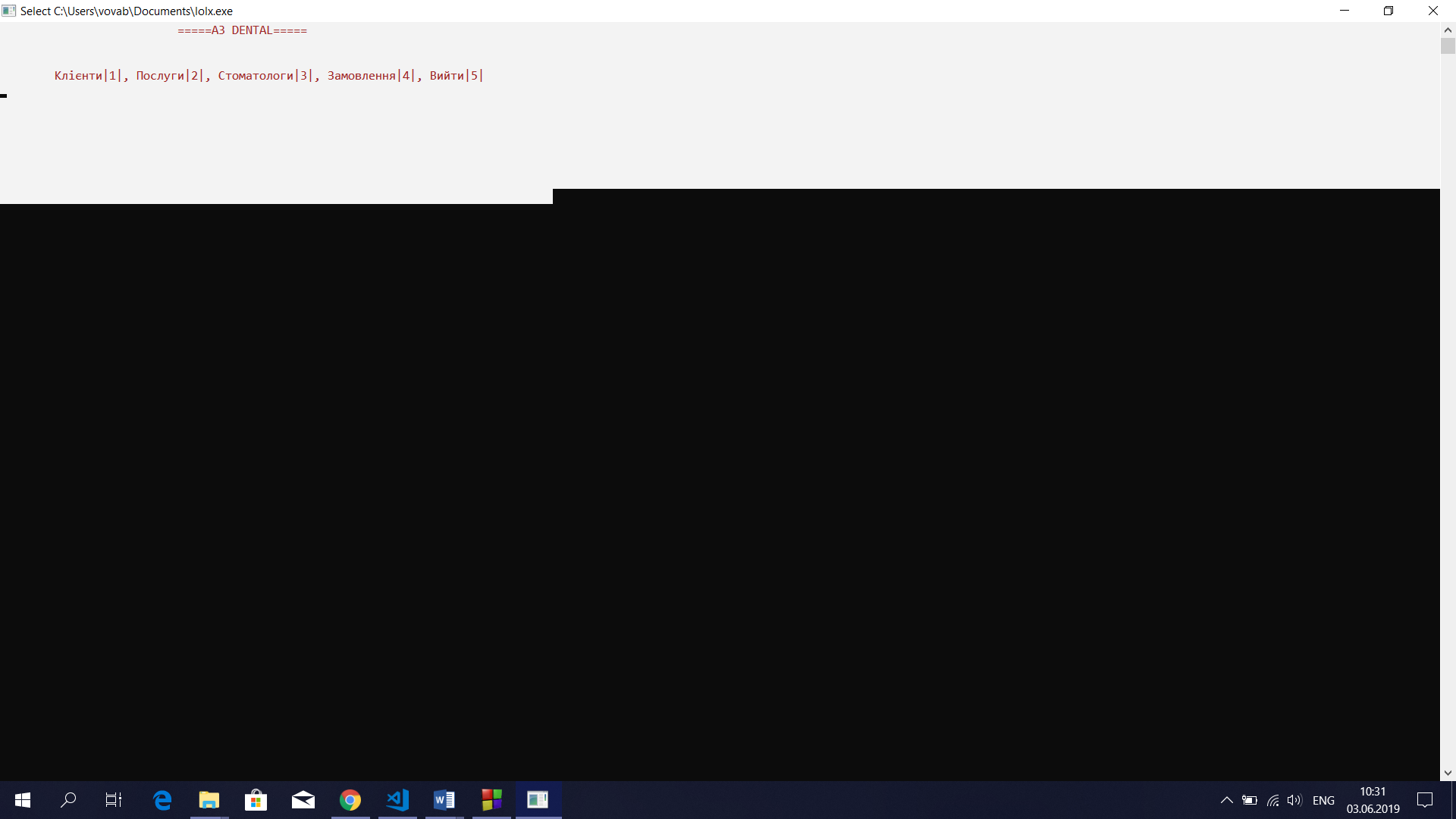
* відповідність вимогам, якими керувалися проектувальники та розробники
* правильна відповідь для усіх можливих вхідних даних
* виконання функцій за прийнятний час
* практичність
* сумісність з програмним забезпеченням та операційними системами
* відповідність задачам замовника.

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершено).

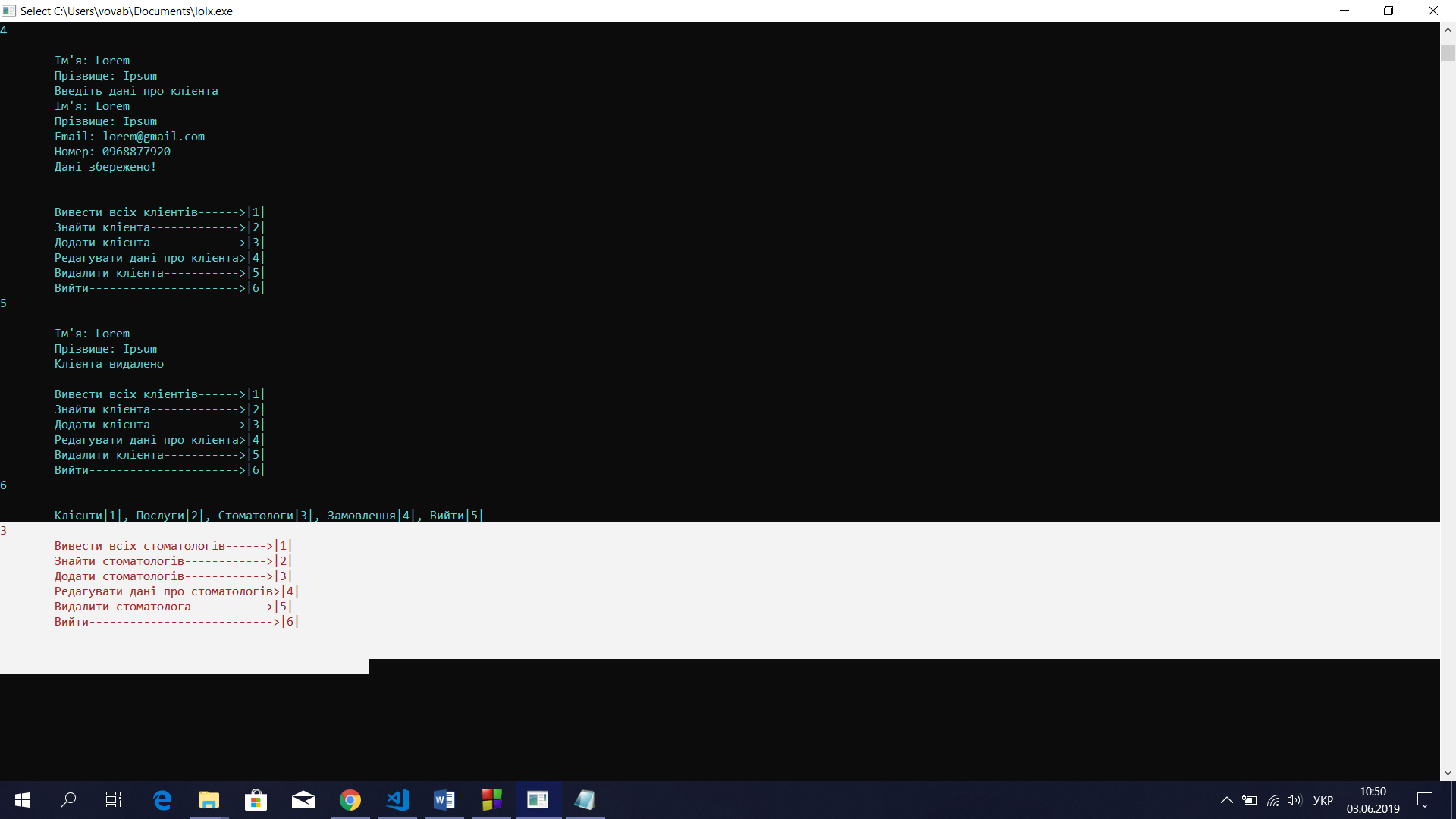
Тестування програми для стоматологічної клініки:

Для початку роботи з програмою достатньо запустити її виконуваний файл. Після запуску цього файлу перед користувачем появиться вікно з головним меню.

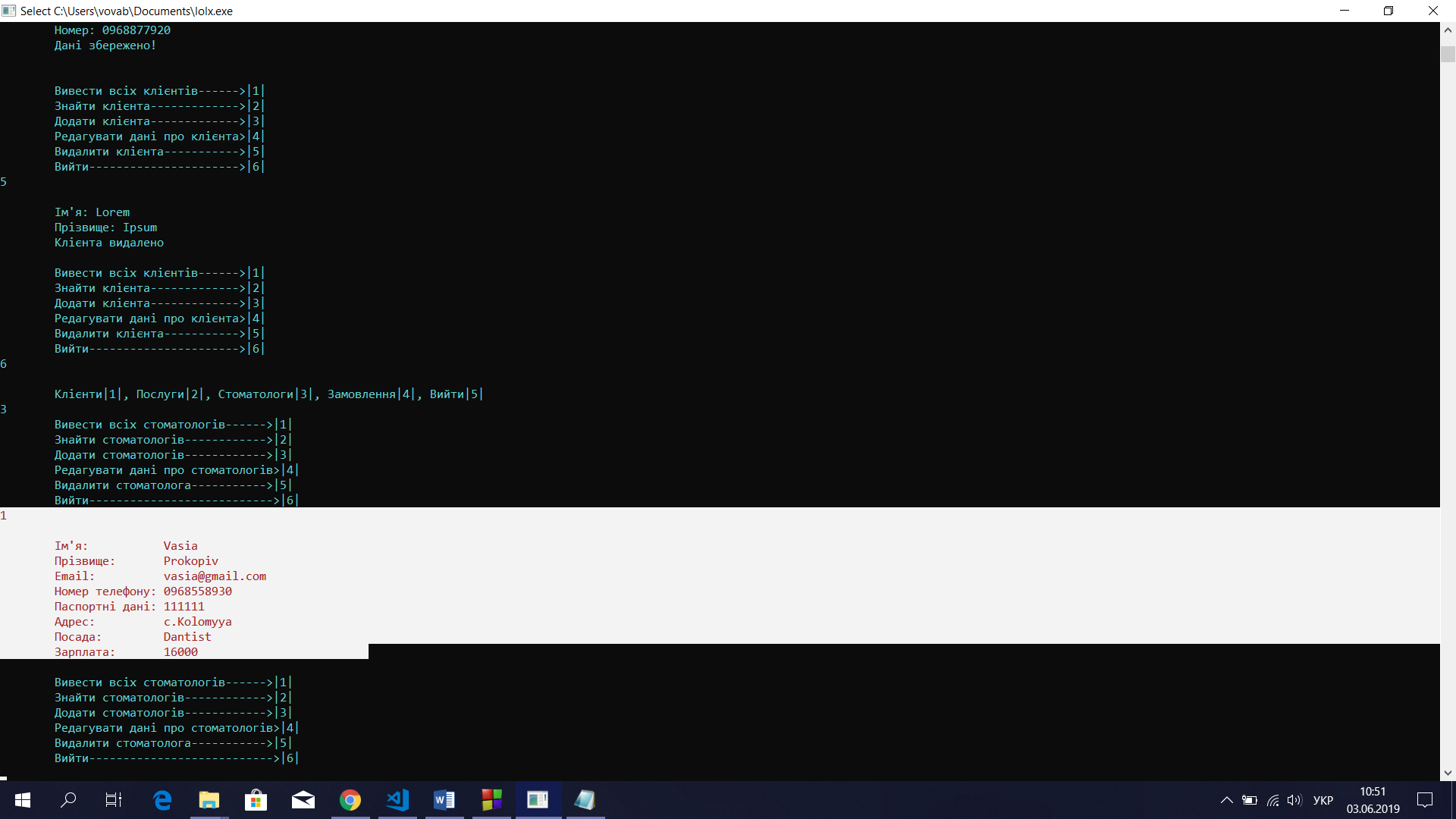


В даному вікні потрібно ввести відповідну цифру з меню та натиснути Enter,

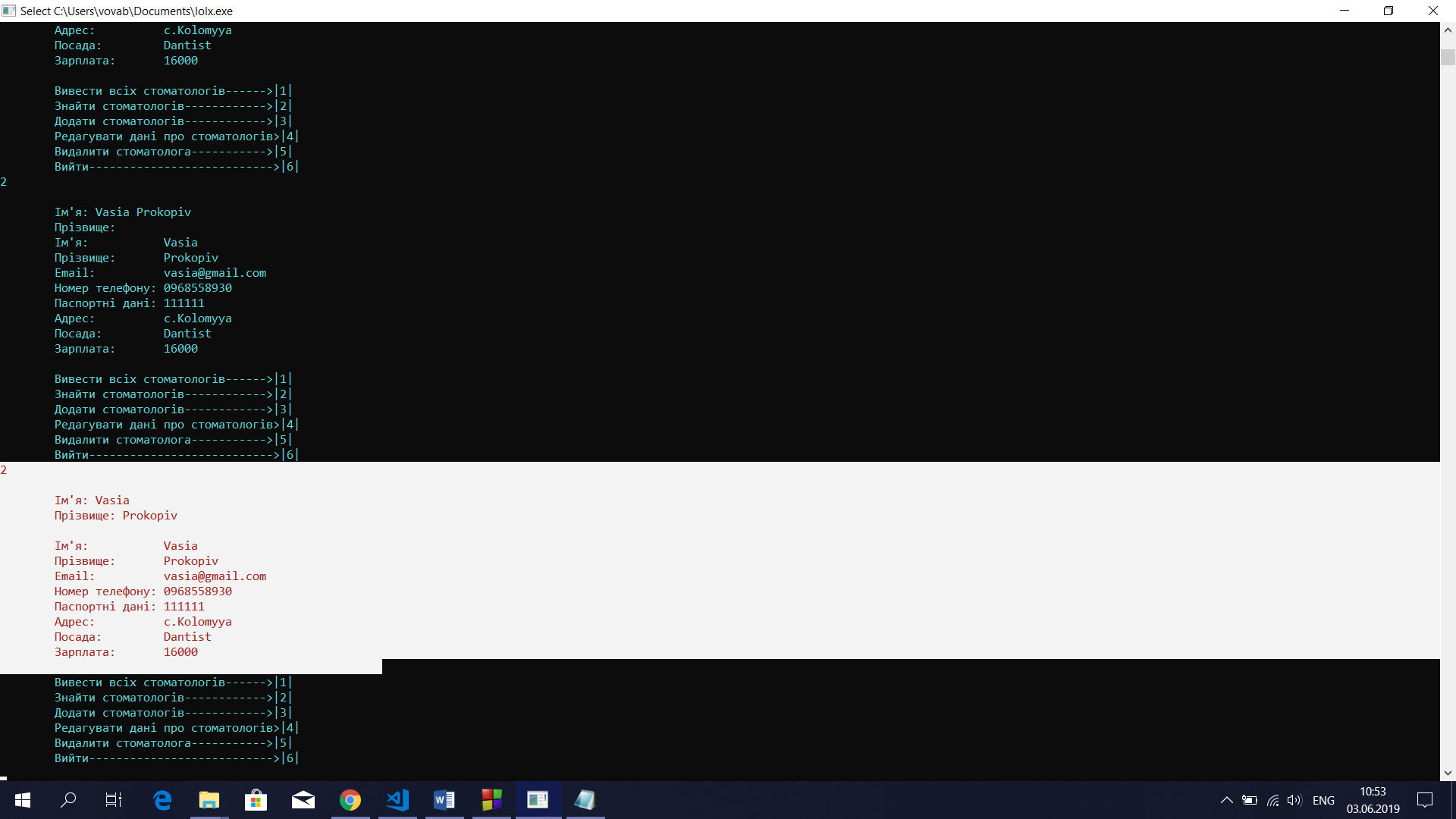
Наприклад “Стоматологи”



Далі необхідно вибрати один з запропонованих варіантів, наприклад “Вивести всіх стоматологів”:

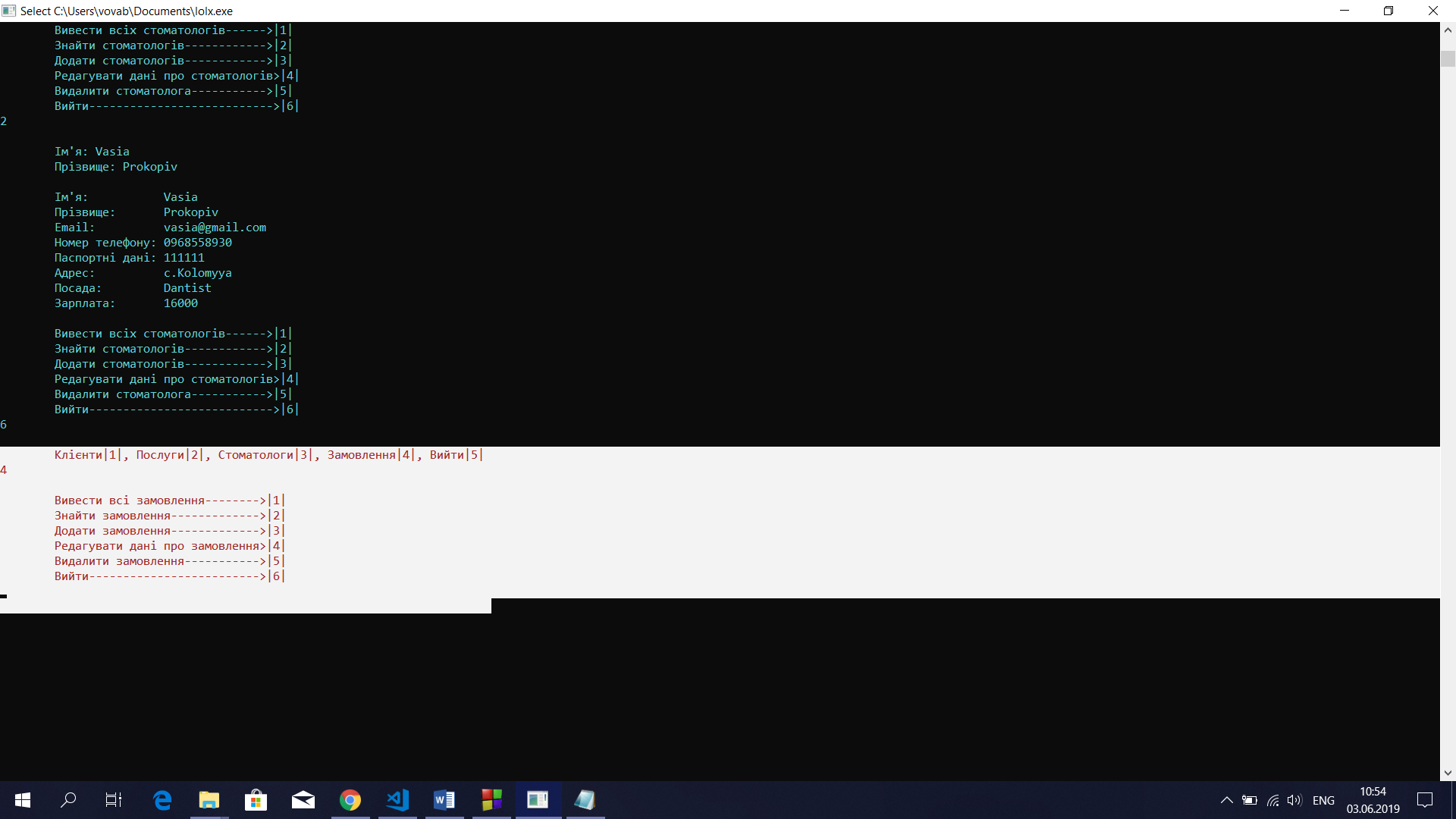


Для повернення в головне меню натискаємо 6 і Enter; в головному меню вибираєм цифру 2 і Enter для знаходження даних про представника фотосалону за іменем та прізвищем:

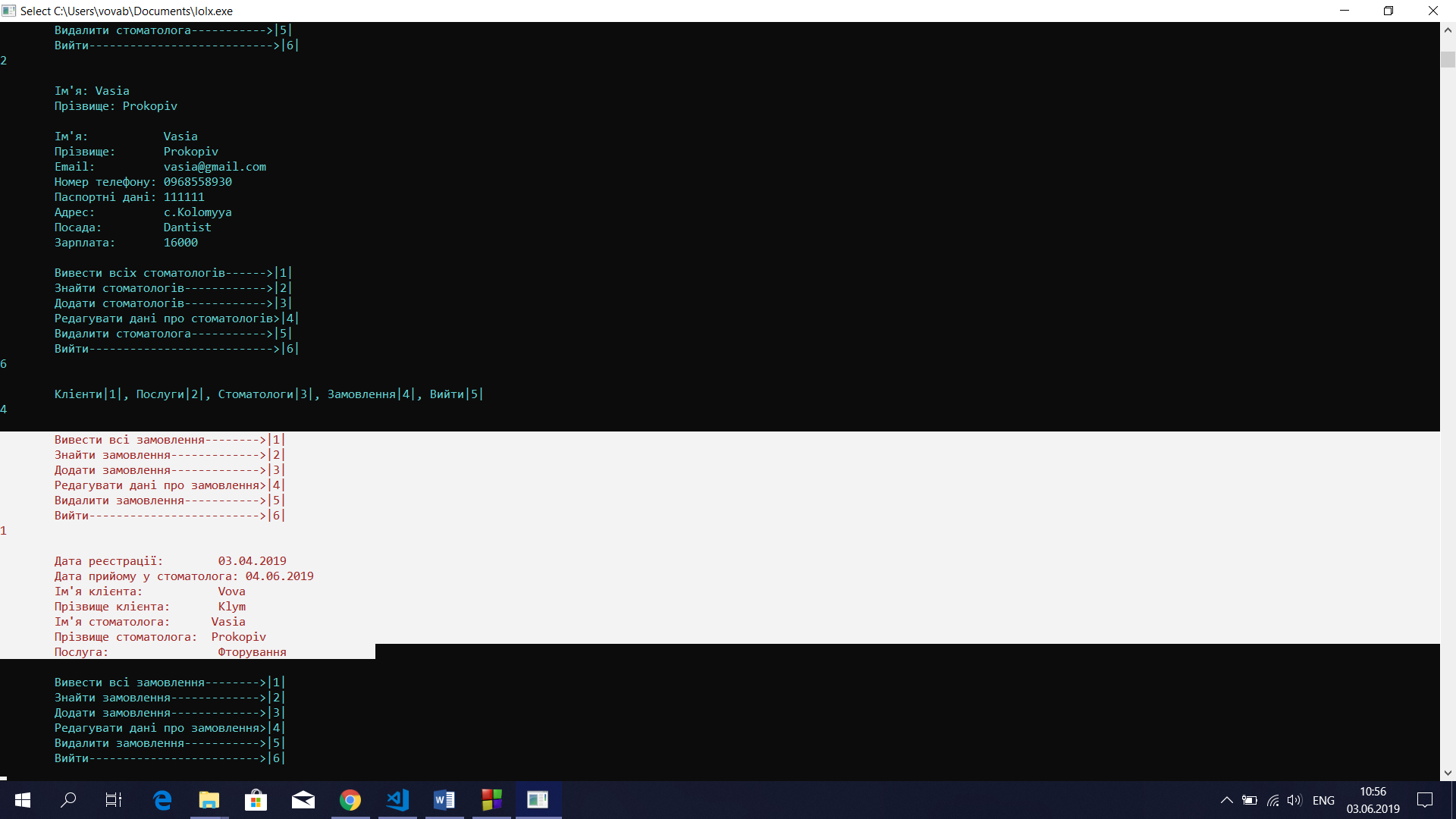


І так далі.

Розглянемо приклад на умовній таблиці “Замовлення”



Вибираємо потрібну цифру, наприклад 1:



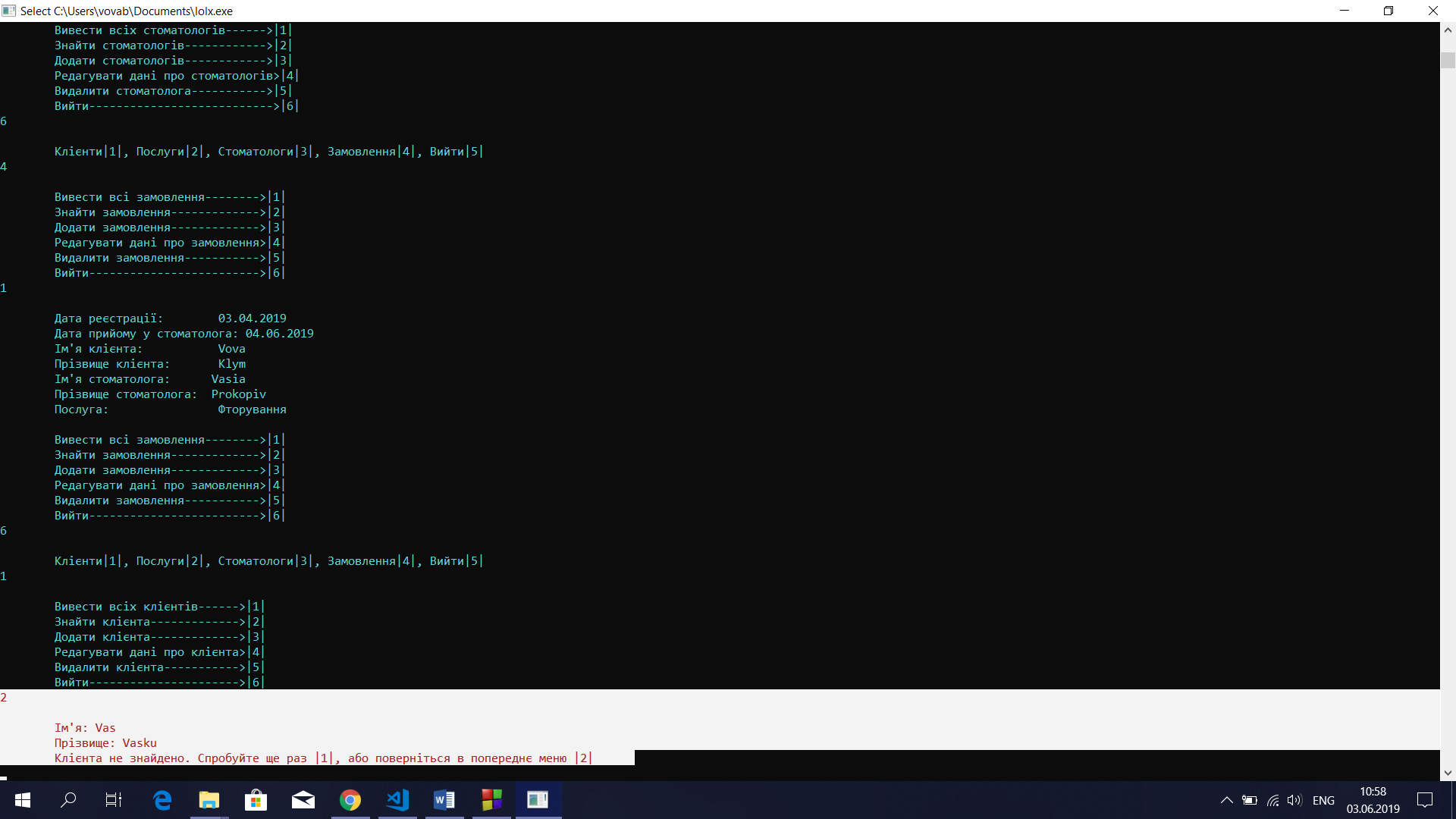
При виборі інших пунктів меню ми зможемо додати замовлення, редагувати дані про замовлення та видалити замовлення.

При тестуванні даної програми було виявлено помилку:

при пошуку клієнтів по імені та прізвищі необхідно ввести точні дані,

в іншому випадку програма зациклюється. Цю проблему було вирішено.

Тобто коли Ви вводите некоректні дані існує ось такий варіант подальших дій



**ВисновОк**

при написанні курсової роботи я отримав важливий досвід роботи у сфері програм обліку записів, зокрема, АРМ адміністратора стоматологічної клініки. Я стикався з проблемами, які вимагали їх вирішення. В результаті створено програму яка працює з даними.

Підвівши підсумки, виділимо переваги використання програмного продукту.

Внесення даних в файлову систему позволяє зручно та швидко знаходити необхідні дані та працювати з ними.

Програма має зрозумілий для кожного користувача інтерфейс. Завдяки цьому користувач може швидко виконувати пошук інформації яка його цікавить. Створена програма позволяє додавати дані, знаходити їх, редагувати та видаляти, а також виводити всі дані в підменю на екран.

**Література**

1. <http://cppstudio.com>
2. Айвор Хортон Microsoft Visual C++ 2005: базовый курс = Beginning Visual C++ 2005. — М.: «Диалектика», 2007. —1152с.
3. Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. Структуры данных и алгоритмы. — Издательский дом «Вильямс», 2000. —384с.
4. <http://cppreference.com>
5. Грегори К. Использование Visual C++ 5. – М.: Вильямс, 1999. – 864с.

# **Додаток А**

## **Код програми**

**Файл header.h**

#include <iostream>

#include<fstream>

#include<cstdlib> //бібліотека для exit

#include <vector>

using namespace std;

class Client //клас

{

private: // специфікатор доступу private

string name,

lastname,

email,

tel;

public: // специфікатор доступу public

string getName()

{

return name;

}

string getLastname()

{

return lastname;

}

void message() // функція (метод класу), яка виводить повідомлення на екран

{

cout << "\tДанi збережено!\n\n";

}

void getCl(); //функція для виведення на екран даних про клієнта

void zdClient(); //функція яка вводитьз клаві

void write\_to\_file(ofstream &file);

void read\_from\_file(ifstream &file);

};

class table\_client

{

public:

vector <Client> vec;

void print\_all\_clients();

void add\_new\_client();

int search\_client(bool mozhna\_povernutus=true);

void poshuk();

void edit\_client();

void delete\_client();

void write\_clients\_to\_file();

void read\_clients\_from\_file();

void menu();

table\_client()

{

read\_clients\_from\_file(); //конструктор

}

~table\_client()

{

write\_clients\_to\_file(); //деструктор

}

};

class Dantist //клас

{

private: // специфікатор доступу private

string NameDantist, //ім'я

SNameDantist, //прізвище

adress, //адреса

posada, //посада

email, //email

pd, //паспортні дані

tel, //номер телефону

salary; //зарплата

public: // специфікатор доступу public

string getfNameR()

{

return NameDantist;

}

string getlNameR()

{

return SNameDantist;

}

void message() // функція (метод класу), яка виводить повідомлення на екран

{

cout << "\tДанi збережено!\n\n";

}

void getRl(); //функція для виведення на екран даних про клієнта

void zkDantist(); //функція яка вводитьз клаві

void write\_to\_file(ofstream &file);

void read\_from\_file(ifstream &file);

};

class table\_dantist

{

public:

vector <Dantist> vec;

void print\_all\_dantists();

void add\_new\_dantists();

int search\_dantists(bool mozhna\_povernutus=true);

void poshuk();

void edit\_dantists();

void delete\_dantists();

void write\_dantists\_to\_file();

void read\_dantists\_from\_file();

void menu\_dantists();

table\_dantist()

{

read\_dantists\_from\_file(); //конструктор

}

~table\_dantist()

{

write\_dantists\_to\_file(); //деструктор

}

};

class Services //клас

{

private: // специфікатор доступу private

string services;

int price;

public: // специфікатор доступу public

string getServices()

{

return services;

}

void message() // функція (метод класу), яка виводить повідомлення на екран

{

cout << "\tДанi збережено!\n\n";

}

void getSer(); //функція для виведення на екран даних про клієнта

void zdServices(); //функція яка вводитьз клаві

void write\_to\_file(ofstream &file);

void read\_from\_file(ifstream &file);

};

class table\_services

{

public:

vector <Services> vec;

void print\_all\_services();

void add\_new\_services();

int search\_services(bool mozhna\_povernutus=true);

void poshuk();

void edit\_services();

void delete\_services();

void write\_services\_to\_file();

void read\_services\_from\_file();

void menuServices();

table\_services()

{

read\_services\_from\_file(); //конструктор

}

~table\_services()

{

write\_services\_to\_file(); //деструктор

}

};

extern table\_client TABLE\_CLIENT;

extern table\_services TABLE\_SERVICES;

extern table\_dantist TABLE\_DANTIST;

class Order: public Client, public Representative, public Order

{

private:

string orderData, //дата замовлення

dataOfIssue; //дата видачі замовлення

int client\_id, dantist\_id, service\_id;

/\*Тут має бути ім я прізвище клієнта та представника

і назва послуги

\*/

public: // специфікатор доступу public

string getOrderData()

{

return orderData;

}

void message() // функція (метод класу), яка виводить повідомлення на екран

{

cout << "\tДанi збережено!\n\n";

}

void getOrder(); //функція для виведення на екран даних про клієнта

void zdOrder(); //функція яка вводитьз клаві

void write\_to\_file(ofstream &file);

void read\_from\_file(ifstream &file);

};

class table\_order

{

public:

vector <Order> vec;

void print\_all\_orders();

void add\_new\_orders();

int search\_orders();

void poshuk();

void edit\_orders();

void delete\_orders();

void write\_orders\_to\_file();

void read\_orders\_from\_file();

void menuOrder();

table\_order()

{

read\_orders\_from\_file(); //конструктор

}

~table\_order()

{

write\_orders\_to\_file(); //деструктор

}

};

#include “header.h”

Class.cpp

void Client::getCl() //функція для виведення на екран даних про клієнта

{

cout<<"\n\tIм'я: "<<name<<"\n\tПрiзвище: "<<lastname<<"\n\tEmail: "<<email<<"\n\tНомер телефону: "<<tel<<endl;

}

void Client::zdClient() //функція яка вводитьз клаві

{

cout << "\tВведiть данi про клiєнта\n";

cout << "\tIм'я: ";

cin >> name;

cout << "\tПрiзвище: ";

cin >> lastname;

cout << "\tEmail: ";

cin >> email;

cout << "\tНомер: ";

cin >> tel;

message();

}

void Client::write\_to\_file(ofstream &file)

{

file<<name<<"\t"<<lastname<<"\t"<<email<<"\t"<<tel<<endl;

}

void Client::read\_from\_file(ifstream &file)

{

file >> name >> lastname >> email >> tel;

}

void table\_client::print\_all\_clients()

{

for(int i=0; i<vec.size(); i++)

{

vec[i].getCl(); /\*cout<<endl;\*/

}

}

void table\_client::add\_new\_client()

{

Client temp;

temp.zdClient();

vec.push\_back(temp);

}

int table\_client::search\_client(bool mozhna\_povernutus)

{

string firstNameC;

string lastNameC;

int id=-1;

while(true)

{

cout << "\n\tIм'я: ";

cin >> firstNameC;

cout << "\tПрiзвище: ";

cin >> lastNameC;

for(int i=0; i<vec.size(); i++)

{

if((vec[i].getName()==firstNameC)&&(vec[i].getLastname()==lastNameC))id=i;

}

if(id==-1)

{

if(!mozhna\_povernutus)

{

cout << "\tКлієнта не знайдено. Спробуйте ще раз." << endl;

continue;

}

cout<<"\tКлієнта не знайдено. Спробуйте ще раз |1|, або поверніться в попереднє меню |2|" << endl;

int in;

cin >> in;

switch(in)

{

case 1:

break;

case 2:

return -1;

break;

}

}

else return id;

}

}

void table\_client::poshuk()

{

int id;

id=search\_client();

if(id!=-1)vec[id].getCl();

}

void table\_client::edit\_client()

{

int id;

id=search\_client();

if(id!=-1)vec[id].zdClient();

}

void table\_client::delete\_client()

{

int id;

id=search\_client();

if(id!=-1)

{

vec.erase(vec.begin()+id);

cout << "\tКлiєнта видалено" <<endl;

}

else cout << "\tКлієнта не видалено" <<endl;

}

void table\_client::write\_clients\_to\_file()

{

ofstream fClient("client1.txt");

for(int i=0; i<vec.size(); i++) vec[i].write\_to\_file(fClient);//для кожного елемента викликати функцію яка записує у файл

fClient.close();

}

void table\_client::read\_clients\_from\_file()

{

ifstream fClient("client1.txt");

if(fClient.is\_open())

{

Client temp;

while(!fClient.eof())

{

temp.read\_from\_file(fClient);

vec.push\_back(temp);

}

vec.erase(vec.end()); //видалення останнього об'єкта, бо його дублює

fClient.close();

}

}

void table\_client::menu()

{

while(1)

{

cout <<"\n\tВивести всiх клiєнтiв------>|1|\n\tЗнайти клiєнта------------->|2|\n\tДодати клiєнта------------->|3|\n\tРедагувати данi про клiєнта>|4|\n\tВидалити клiєнта----------->|5|\n\tВийти---------------------->|6|" << endl;

int inC;

cin >> inC;

switch(inC)

{

case 1:

print\_all\_clients();

break;

case 2:

poshuk();

break;

case 3:

add\_new\_client();

break;

case 4:

edit\_client();

break;

case 5:

delete\_client();

break;

case 6:

return;

}

}

}

void Dantist::getRl() //функція для виведення на екран даних про клієнта

{

cout<<"\n\tIм'я: "<<NameDantist<<"\n\tПрiзвище: "<<SNameDantist<<"\n\tEmail: "<< email<<"\n\tНомер телефону: "<<tel<<"\n\tПаспортнi данi: "<<pd<<"\n\tАдрес: "<<adress<<"\n\tПосада: "<<posada<<"\n\tЗарплата: "<<salary<<endl<<endl;

}

void Dantist::zkDantist() //функція яка вводитьз клаві

{

cout << "\n\tВведiть данi про стоматолога\n";

cout << "\tIм'я: ";

cin >> NameDantist;

cout << "\tПрiзвище: ";

cin >> SNameDantist;

cout << "\tEmail: ";

cin >> email;

cout << "\tНомер: ";

cin >> tel;

cout << "\tПаспортнi данi: ";

cin >> pd;

cout << "\tАдрес: ";

cin >> adress;

cout << "\tПосада: ";

cin >> posada;

cout << "\tЗарплата: ";

cin >> salary;

message();

}

void Dantist::write\_to\_file(ofstream &file)

{

file<<NameDantist<<"\t"<<SNameDantist<<"\t"<<email<<"\t"<<tel<<"\t"<<pd<<"\t"<<adress<<"\t"<<posada<<"\t"<<salary<<endl;

}

void Dantist::read\_from\_file(ifstream &file)

{

file >> NameDantist >> SNameDantist >> email >> tel >> pd >> adress >> posada >> salary;

}

void table\_dantist::print\_all\_dantists()

{

for(int i=0; i<vec.size(); i++)

{

vec[i].getRl(); /\*cout<<endl;\*/

}

}

void table\_dantist::add\_new\_dantists()

{

Dantist temp;

temp.zkDantist();

vec.push\_back(temp);

}

int table\_dantist::search\_dantists(bool mozhna\_povernutus)

{

string NameDantist;

string SNameDantist;

int id=-1;

while(true)

{

cout << "\n\tIм'я: ";

cin >> NameDantist;

cout << "\tПрiзвище: ";

cin >> SNameDantist;

for(int i=0; i<vec.size(); i++)

{

if((vec[i].getfNameR()==NameDantist)&&(vec[i].getlNameR()==SNameDantist))id=i;

}

if(id==-1)

{

if(!mozhna\_povernutus)

{

cout << "\tСтоматолога не знайдено. Спробуйте ще раз." << endl;

continue;

}

cout<<"\tСтоматолога не знайдено. Спробуйте ще раз |1|, або поверніться в попереднє меню |2|" << endl;

int in;

cin >> in;

if(cin.get()!='\n'){cout << "Помилка вводу, спробуйте ще раз." << endl; cin.clear(); cin.ignore();}

switch(in)

{

case 1:

break;

case 2:

return -1;

break;

}

}

else return id;

}

}

void table\_dantist::poshuk()

{

int id;

id=search\_dantists();

if(id!=-1)vec[id].getRl();

}

void table\_dantist::edit\_dantists()

{

int id;

id=search\_dantists();

if(id!=-1)vec[id].zkDantist();

}

void table\_dantist::delete\_dantists()

{

int id;

id=search\_dantists();

if(id!=-1)

{

vec.erase(vec.begin()+id);

cout << "\tСтоматолога видалено" <<endl;

}

else cout << "\tСтоматолога не видалено" <<endl;

}

void table\_dantist::write\_dantists\_to\_file()

{

ofstream fRepresentative("Representatives.txt");

for(int i=0; i<vec.size(); i++) vec[i].write\_to\_file(fRepresentative);//для кожного елемента викликати функцію яка записує у файл

fRepresentative.close();

}

void table\_dantist::read\_dantists\_from\_file()

{

ifstream fRepresentative("Representatives.txt");

if(fRepresentative.is\_open())

{

Dantist temp;

while(!fRepresentative.eof())

{

temp.read\_from\_file(fRepresentative);

vec.push\_back(temp);

}

vec.erase(vec.end()); //видалення останнього об'єкта, бо його дублює

fRepresentative.close();

}

}

void table\_dantist::menu\_dantists()

{

while(1)

{

cout <<"\tВивести всiх стоматологів------>|1|\n\tЗнайти стоматологів------------>|2|\n\tДодати стоматологів------------>|3|\n\tРедагувати данi про стоматологів>|4|\n\tВидалити стоматолога----------->|5|\n\tВийти--------------------------->|6|" << endl;

int inR;

cin >> inR;

switch(inR)

{

case 1:

print\_all\_dantists();

break;

case 2:

poshuk();

break;

case 3:

add\_new\_dantists();

break;

case 4:

edit\_dantists();

break;

case 5:

delete\_dantists();

break;

case 6:

return;

}

}

}

void Services::getSer() //функція для виведення на екран даних про клієнта

{

cout<<"\n\tПослуга: "<<services<<"\n\tЦiна: "<<price<<endl;

}

void Services::zdServices() //функція яка вводитьз клаві

{

cout << "\tВведiть данi про послугу\n";

cout << "\tПослуга: ";

cin >> services;

cout << "\tЦiна: ";

cin >> price;

message();

}

void Services::write\_to\_file(ofstream &file)

{

file<<services<<"\t"<<price<<endl;

}

void Services::read\_from\_file(ifstream &file)

{

file >> services>>price;

}

void table\_services::print\_all\_services()

{

for(int i=0; i<vec.size(); i++)

{

vec[i].getSer(); /\*cout<<endl;\*/

}

}

void table\_services::add\_new\_services()

{

Services temp;

temp.zdServices();

vec.push\_back(temp);

}

int table\_services::search\_services(bool mozhna\_povernutus)

{

string services;

int id=-1;

while(true)

{

cout << "\n\tПослуга: ";

cin >> services;

for(int i=0; i<vec.size(); i++)

{

if((vec[i].getServices()==services))id=i;

}

if(id==-1)

{

if(!mozhna\_povernutus)

{

cout << "\tПослугу не знайдено. Спробуйте ще раз." << endl;

continue;

}

cout<<"\tПослугу не знайдено. Спробуйте ще раз |1|, або поверніться в попереднє меню |2|" << endl;

int in;

cin >> in;

if(cin.get()!='\n'){cout << "Помилка вводу, спробуйте ще раз." << endl; cin.clear(); cin.ignore();}

switch(in)

{

case 1:

break;

case 2:

return -1;

break;

}

}

else return id;

}

}

void table\_services::poshuk()

{

int id;

id=search\_services();

if(id!=-1)vec[id].getSer();

}

void table\_services::edit\_services()

{

int id;

id=search\_services();

if(id!=-1)vec[id].zdServices();

}

void table\_services::delete\_services()

{

int id;

id=search\_services();

if(id!=-1)

{

vec.erase(vec.begin()+id);

cout << "\tПослугу видалено" <<endl;

}

else cout << "\tПослугу не видалено" <<endl;

}

void table\_services::write\_services\_to\_file()

{

ofstream fServices("Services.txt");

for(int i=0; i<vec.size(); i++) vec[i].write\_to\_file(fServices);//для кожного елемента викликати функцію яка записує у файл

fServices.close();

}

void table\_services::read\_services\_from\_file()

{

ifstream fServices("Services.txt");

if(fServices.is\_open())

{

Services temp;

while(!fServices.eof())

{

temp.read\_from\_file(fServices);

vec.push\_back(temp);

}

vec.erase(vec.end()); //видалення останнього об'єкта, бо його дублює

fServices.close();

}

}

void table\_services::menuServices()

{

while(1)

{

cout <<"\n\tВивести всi послуги-------->|1|\n\tЗнайти послугу------------->|2|\n\tДодати послугу------------->|3|\n\tРедагувати данi про послугу>|4|\n\tВидалити послугу----------->|5|\n\tВийти---------------------->|6|" << endl;

int inC;

cin >> inC;

switch(inC)

{

case 1:

print\_all\_services();

break;

case 2:

poshuk();

break;

case 3:

add\_new\_services();

break;

case 4:

edit\_services();

break;

case 5:

delete\_services();

break;

case 6:

return;

}

}

}

void Order::getOrder() //функція для виведення на екран даних про клієнта

{

cout<<"\n\tДата реєстрації: "<<orderData<<"\n\tДата прийому у стоматолога: "<<dataOfIssue<<endl;

cout<<"\tIм'я клiєнта: " << TABLE\_CLIENT.vec[client\_id].getName()

<< "\n\tПрiзвище клiєнта: " << TABLE\_CLIENT.vec[client\_id].getLastname() << endl;

cout<<"\tIм'я стоматолога: " << TABLE\_DANTIST.vec[dantist\_id].getfNameR()

<< "\n\tПрiзвище стоматолога: " << TABLE\_DANTIST.vec[dantist\_id].getlNameR() << endl;

cout<<"\tПослуга: " << TABLE\_SERVICES.vec[service\_id].getServices() << endl;

}

void Order::zdOrder() //функція яка вводитьз клаві

{

cout << "\tВведiть данi про замовлення\n";

cout << "\tДата реєстрації: ";

cin >> orderData;

cout << "\tДата прийому у стоматолога: ";

cin >> dataOfIssue;

cout << "\tВведiть данi про клiєнта";

client\_id=TABLE\_CLIENT.search\_client(false);

cout << "\tВведiть данi про стоматолога";

dantist\_id=TABLE\_DANTIST.search\_dantists(false);

cout << "\tВведiть данi про послугу";

service\_id=TABLE\_SERVICES.search\_services(false);

message();

}

void Order::write\_to\_file(ofstream &file)

{

file<<orderData<<"\t"<<dataOfIssue<<"\t"<<client\_id<<"\t"<<dantist\_id<<"\t"<<service\_id<<endl;

}

void Order::read\_from\_file(ifstream &file)

{

file >> orderData>>dataOfIssue>>client\_id>>dantist\_id>>service\_id;

}

void table\_order::print\_all\_orders()

{

for(int i=0; i<vec.size(); i++)

{

vec[i].getOrder(); /\*cout<<endl;\*/

}

}

void table\_order::add\_new\_orders()

{

Order temp;

temp.zdOrder();

vec.push\_back(temp);

}

int table\_order::search\_orders()

{

string orderData;

int id=-1;

while(true)

{

cout << "\n\tДата реєстрації: ";

cin >> orderData;

for(int i=0; i<vec.size(); i++)

{

if((vec[i].getOrderData()==orderData))id=i;

}

if(id==-1)

{

cout<<"\tЗамовлення не знайдено. Спробуйте ще раз |1|, або поверніться в попереднє меню |2|" << endl;

int in;

cin >> in;

switch(in)

{

case 1:

break;

case 2:

return -1;

break;

}

}

else return id;

}

}

void table\_order::poshuk()

{

int id;

id=search\_orders();

if(id!=-1)vec[id].getOrder();

}

void table\_order::edit\_orders()

{

int id;

id=search\_orders();

if(id!=-1)vec[id].zdOrder();

}

void table\_order::delete\_orders()

{

int id;

id=search\_orders();

if(id!=-1)

{

vec.erase(vec.begin()+id);

cout << "\tЗамовлення видалено" <<endl;

}

else cout << "\tЗамовлення не видалено" <<endl;

}

void table\_order::write\_orders\_to\_file()

{

ofstream fOrder("Order.txt");

for(int i=0; i<vec.size(); i++) vec[i].write\_to\_file(fOrder);//для кожного елемента викликати функцію яка записує у файл

fOrder.close();

}

void table\_order::read\_orders\_from\_file()

{

ifstream fOrder("Order.txt");

if(fOrder.is\_open())

{

Order temp;

while(!fOrder.eof())

{

temp.read\_from\_file(fOrder);

vec.push\_back(temp);

}

vec.erase(vec.end()); //видалення останнього об'єкта, бо його дублює

fOrder.close();

}

}

void table\_order::menuOrder()

{

while(1)

{

cout <<"\n\tВивести всi замовлення-------->|1|\n\tЗнайти замовлення------------->|2|\n\tДодати замовлення------------->|3|\n\tРедагувати данi про замовлення>|4|\n\tВидалити замовлення----------->|5|\n\tВийти------------------------->|6|" << endl;

int inO;

cin >> inO;

if(cin.get()!='\n'){cout << "Помилка вводу, спробуйте ще раз." << endl; cin.clear(); cin.ignore();}

switch(inO)

{

case 1:

print\_all\_orders();

break;

case 2:

poshuk();

break;

case 3:

add\_new\_orders();

break;

case 4:

edit\_orders();

break;

case 5:

delete\_orders();

break;

case 6:

return;

}

}

}

table\_client TABLE\_CLIENT;

table\_services TABLE\_SERVICES;

table\_dantist TABLE\_DANTIST;

table\_order TABLE\_ORDER;

main.cpp

int main()

{

system("color B");

system("chcp 1251 > nul");

setlocale(0,"");

cout<<"\t\t\t\ =====A3 DENTAL=====\n\n";

while(1)

{

cout <<"\n\tКлiєнти|1|, Послуги|2|, Стоматологи|3|, Замовлення|4|, Вийти|5|" << endl;

int choose;

cin >> choose;

switch(choose)

{

case 1:

TABLE\_CLIENT.menu();

break;

case 2:

TABLE\_SERVICES.menuServices();

break;

case 3:

TABLE\_DANTIST.menu\_dantists();

break;

case 4:

TABLE\_ORDER.menuOrder();

break;

case 5:

return 0;

// break;

}

}

return 0;

}