**Лабораторна робота №6. Спадкування**

Тема: Класи. Спадкування.

Мета: Отримати знання про парадигму ООП – спадкування. Навчитися застосовувати отримані знання на практиці.

**ВИМОГИ**

**1.1 Інформація про розробника:**

* Кліщов Б. Р.
* КІТ 102.8а
  1. **Загальне завдання**

Модернізувати попередню лабораторну роботу шляхом:

- додавання класу-спадкоємця, котрий буде поширювати функціонал «базового класу» у відповідності до індивідуального завдання;

- додавання ще одного класу-списку, що буде керувати лише елементами класу-спадкоємця;

- в функціях базового класу та класу-спадкоємця обов’язкове використання ключових слів final та override.

**1.3 Додаткові умови виконання завдання:**

- продемонструвати відсутність витоків пам’яті;

- продемонструвати роботу розроблений методів за допомогою модульних тестів;

- не використовувати конструкцію «using namespace std;» , замість цього слід роботи «using» кожного необхідного класу:using std::string, using std::cout;

- в проекті не повинні використовуватися бібліотеки введення / виведення мови С, а також не повинні використовуватися рядки типу char\*.

- в табл. 6.3 оберіть завдання для створення класу-спадкоємця у відповідності до номера у журналі групи(тип монархії)

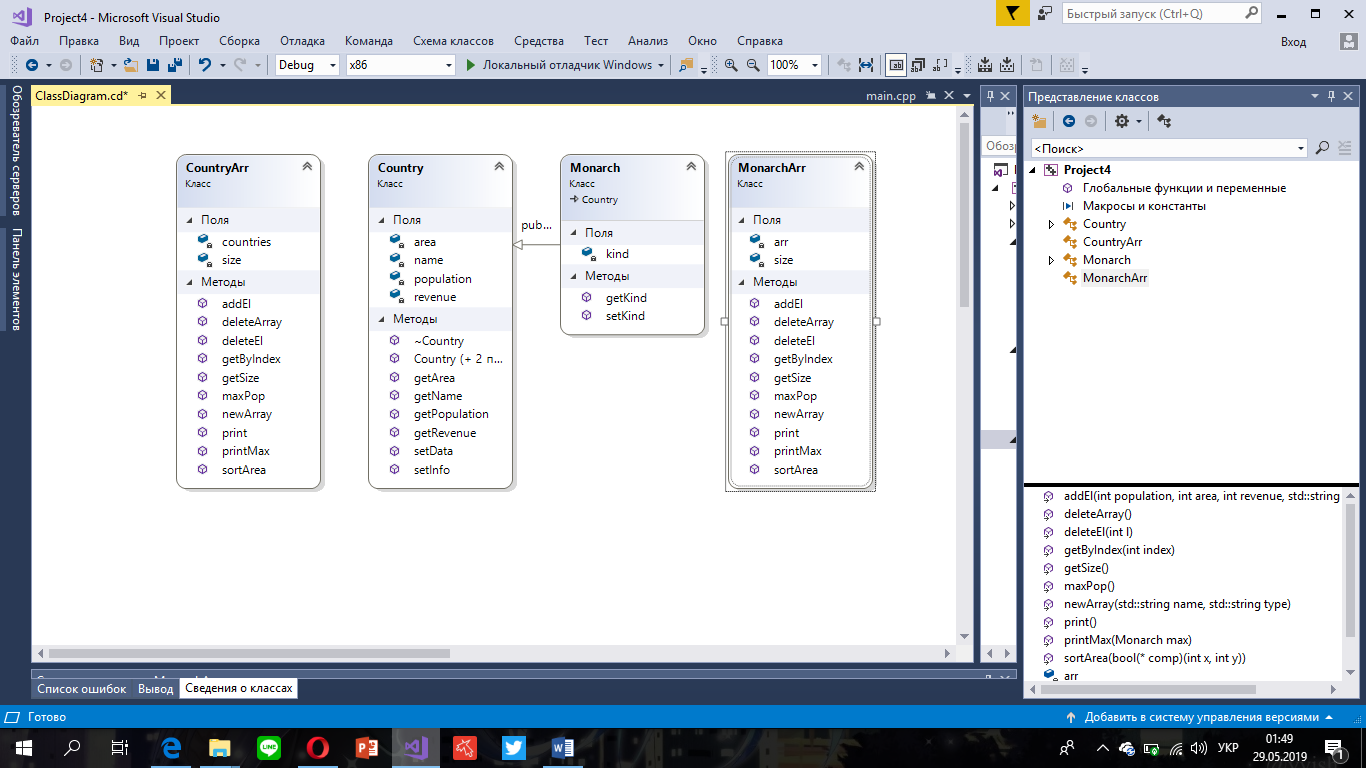
1. **ОПИС ПРОГРАМИ**

**2.1 Функціональне призначення**

Програма призначена щоб отримувати та зберігати інформацію щодо різних країн світу, сортувати масив цих країн та отримувати максимальне значення населення. Інформацію можна зчитувати з файлу та записувати в нього.

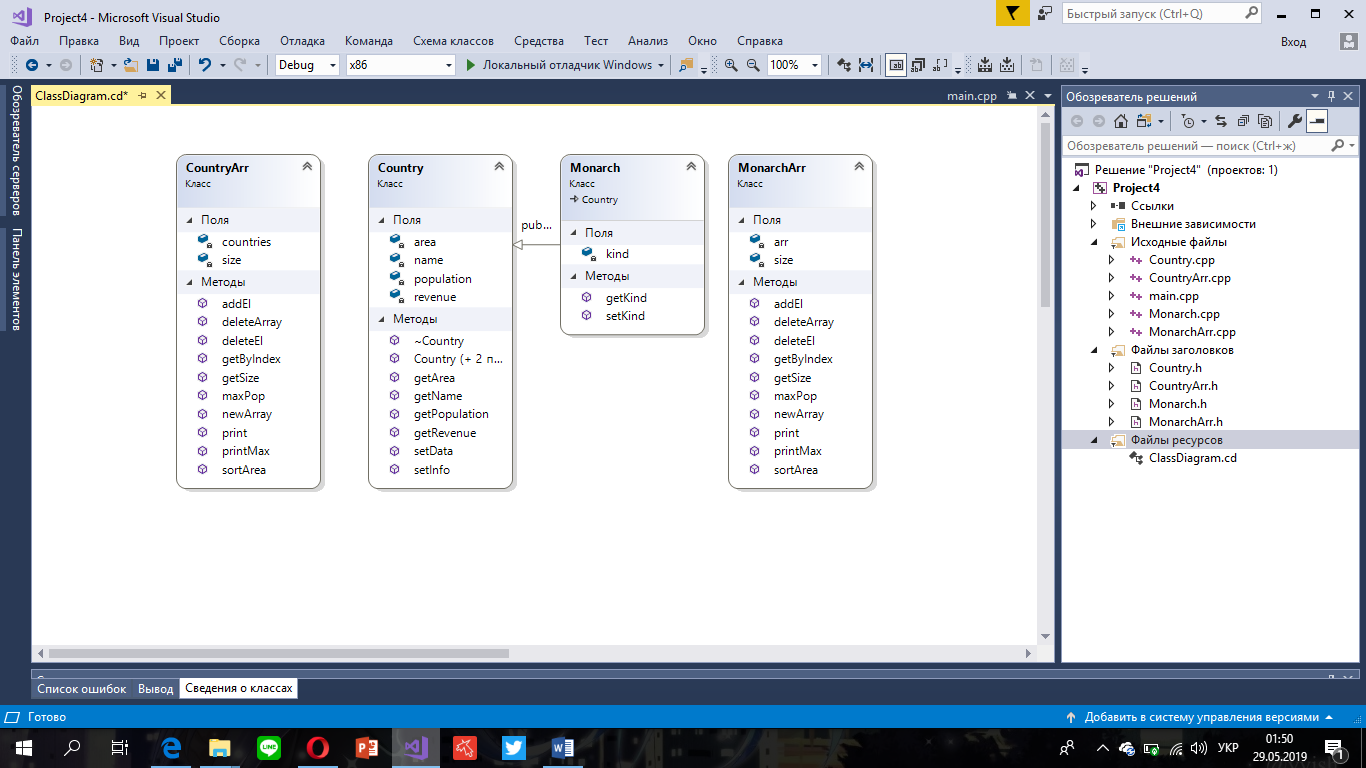
* 1. **Опис логічної структури**

На рисунку № 1 зображена діаграма класу



Малюнок №1. Діаграма класу

На рисунку № 2 зоображена структура програми:



Малюнок №2: Cтруктура програми

**2.3 Важливі фрагменти програми**

Код програми:

int main() {

srand(time(NULL));

MonarchArr Countr;

std::regex regex\_spaces("[\\s]{2,}");

std::regex regex\_firstSymbol("^[A-Z]");

std::string name;

std::cout << "Please, the name of country: ";

getline(std::cin, name);

if (!(regex\_search(name, regex\_firstSymbol)) || regex\_search(name, regex\_spaces)) {

std::cout << "Incorrect entry, writing with large letters(A - Z) and without double spaces : " << std::endl;

std::cout << "Please, the name of country: ";

getline(std::cin, name);

}

std::string \*type;

type = new std::string[3];

type[0] = "Absolute";

type[1] = "Constitutional";

type[2] = "Theocratic";

int k = rand() % 3 + 0;

Countr.newArray(name, type[k]);

Countr.print();

bool(\*p)(int a, int b);

int option = 0;

do {

std::cout << "Choose option:" << std::endl << "0 - Exit " << std::endl << "1 - Add element" << std::endl << "2 - Delete element" << std::endl << "3 - Get by index" << std::endl << "4 - Search by max population" << std::endl << "5 - Sort by area" << std::endl;

std::cout << std::endl;

std::cin >> option;

switch (option) {

case 1: {

int population, area, revenue;

std::string name;

std::ifstream fin("data.txt");

k = rand() % 3 + 0;

fin >> population >> area >> revenue >> name;

Countr.addEl(population, area, revenue, name, type[k]);

k = rand() % 3 + 0;

fin >> population >> area >> revenue >> name;

Countr.addEl(population, area, revenue, name, type[k]);

k = rand() % 3 + 0;

fin >> population >> area >> revenue >> name;

Countr.addEl(population, area, revenue, name, type[k]);

system("cls");

Countr.print();

break;

}

case 2: {

auto id = 0;

std::cout << std::endl << "Enter index: ";

std::cin >> id;

std::cout << std::endl;

Countr.deleteEl(id);

system("cls");

Countr.print();

break;

}

case 3: {

auto index = 0;

std::cout << std::endl << "Enter index : ";

std::cin >> index;

std::cout << std::endl;

system("cls");

Countr.getByIndex(index);

break;

}

case 4: {

Monarch Max = Countr.maxPop();

Countr.printMax(Max);

break;

}

case 5: {

int ch;

std::cout << "Please, enter the type of sort" << std::endl << "1 - up, 0 - down: ";

std::cin >> ch;

std::cout << std::endl;

if (ch == 1) {

p = comp;

}

else if (ch == 0) {

p = comp2;

}

else {

std::cout << "You enter false variant" << std::endl;

break;

}

Countr.sortArea(p);

break;

}

default: {

break;

}

}

} while (option != 0);

Countr.deleteArray();

delete[] type;

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDERR);

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDERR);

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDERR);

\_CrtDumpMemoryLeaks();

return 0;

}

**Регулярні вирази:**

|  |  |
| --- | --- |
| std::regex regex\_spaces("[\\s]{2,}");  std::regex regex\_firstSymbol("^[A-Z]"); | |
|  | |  | |

**Конструктори:**

1. Без параметрів:

Country::Country() :population(0), area(0), revenue(0), name() {

name = new char[24];

};

1. З параметрами:

Country::Country(int a, int b, int c, char\* Name) :population(a), area(b), revenue(c) {

name = new char[24];

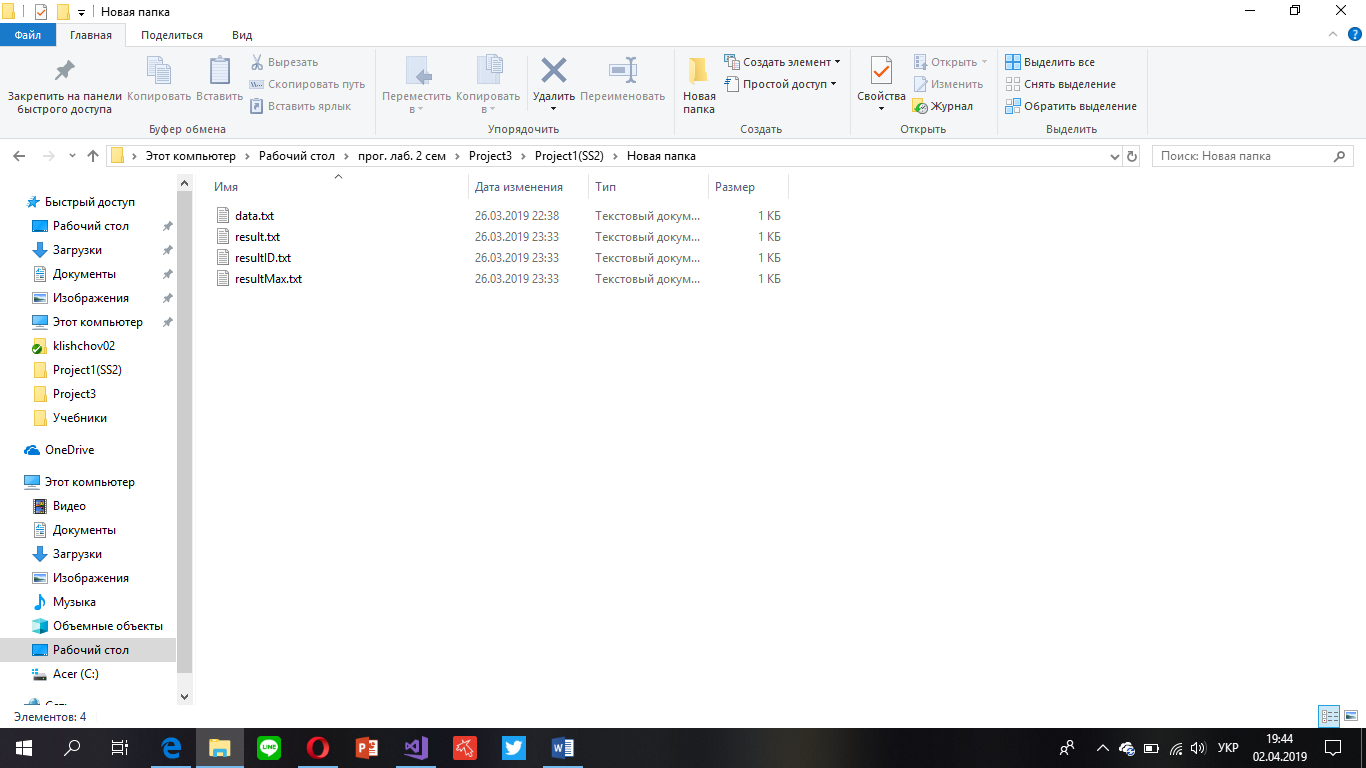
strcpy\_s(name, 24, Name);

};

1. Копіювальний:

Country::Country(const Country &obj) :population(obj.population), area(obj.area), revenue(obj.revenue), name(obj.name) {};

**Файли:**



data.txt – для читання інформації про країни

result.txt – сюда записується результат роботи функції void CountryArr::print()

resultID.txt – для запису роботи функції void getByIndex(int index);

resultMax.txt – для запису результату Country maxPop();

**Клас спадкоємець:**

|  |
| --- |
| class Monarch : public Country { |
| private: |
|  |
| public: |
|  |
|  |
|  |
| }; |

**Реалізація методів масиву класів спадкоємців:**

int MonarchArr::getSize() {

return size;

}

void MonarchArr::sortArea(bool(\*comp)(int x, int y)) {

Monarch temp;

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

for (int j = 0; j < size; j++) {

if (comp(arr[i].getArea(), arr[j].getArea())) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void MonarchArr::newArray(std::string name, std::string type) {

size++;

arr = new Monarch[size];

arr[0].setInfo(name);

arr[0].setKind(type);

}

void MonarchArr::print() {

std::ofstream fout("result.txt");

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

fout << "The name of Monarch: " << arr[i].getName() << std::endl;

fout << "Type of monarch: " << arr[i].getKind() << std::endl;

fout << "Population: " << arr[i].getPopulation() << std::endl;

fout << "Area: " << arr[i].getArea() << std::endl;

fout << "Revenue: " << arr[i].getRevenue() << std::endl << std::endl;

}

fout.close();

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

std::cout << "The name of Monarch: " << arr[i].getName() << std::endl;

std::cout << "Type of monarch: " << arr[i].getKind() << std::endl;

std::cout << "Population: " << arr[i].getPopulation() << std::endl;

std::cout << "Area: " << arr[i].getArea() << std::endl;

std::cout << "Revenue: " << arr[i].getRevenue() << std::endl << std::endl;

}

}

void MonarchArr::addEl(int population, int area, int revenue, std::string name, std::string type) {

Monarch \*temp = new Monarch[size + 1];

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

temp[i] = MonarchArr::arr[i];

}

size++;

temp[size - 1].setData(population, area, revenue, name);

temp[size - 1].setKind(type);

delete[] arr;

arr = temp;

}

void MonarchArr::deleteEl(int index) {

size--;

Monarch\* temp = new Monarch[size];

int j = 0;

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

{

temp[i] = MonarchArr::arr[j];

j++;

}

j++;

for (int i = index - 1; i < size; i++)

{

temp[i] = MonarchArr::arr[i];

j++;

}

delete[] arr;

arr = temp;

}

void MonarchArr::getByIndex(int index) {

std::ofstream fout("resultID.txt");

fout << "The name of Monarch: " << arr[index - 1].getName() << std::endl;

fout << "Type of monarch: " << arr[index - 1].getKind() << std::endl;

fout << "Population: " << arr[index - 1].getPopulation() << std::endl;

fout << "Area: " << arr[index - 1].getArea() << std::endl;

fout << "Revenue: " << arr[index - 1].getRevenue() << std::endl << std::endl;

fout.close();

std::cout << "The name of Monarch: " << arr[index - 1].getName() << std::endl;

std::cout << "Type of monarch: " << arr[index - 1].getKind() << std::endl;

std::cout << "Population: " << arr[index - 1].getPopulation() << std::endl;

std::cout << "Area: " << arr[index - 1].getArea() << std::endl;

std::cout << "Revenue: " << arr[index - 1].getRevenue() << std::endl << std::endl;

}

void MonarchArr::deleteArray() {

delete[] arr;

}

Monarch MonarchArr::maxPop() {

int max = arr[0].getPopulation();

int index = 0;

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

if (max < arr[i].getPopulation()) {

max = arr[i].getPopulation();

index = i;

}

}

return arr[index];

}

void MonarchArr::printMax(Monarch min) {

std::ofstream fout("resultMax.txt");

fout << "The name of Monarch: " << min.getName() << std::endl;

fout << "Type of monarch: " << min.getKind() << std::endl;

fout << "Population: " << min.getPopulation() << std::endl;

fout << "Area: " << min.getArea() << std::endl;

fout << "Revenue: " << min.getRevenue() << std::endl << std::endl;

fout.close();

std::cout << "The name of Monarch: " << min.getName() << std::endl;

std::cout << "Type of monarch: " << min.getKind() << std::endl;

std::cout << "Population: " << min.getPopulation() << std::endl;

std::cout << "Area: " << min.getArea() << std::endl;

std::cout << "Revenue: " << min.getRevenue() << std::endl << std::endl;

1. }
2. **ВАРІАНТИ ВИКОРИСТАННЯ**

**3.1 Результат роботи функцій**

На рисунку № 3 зоображено результат запису даних в файл

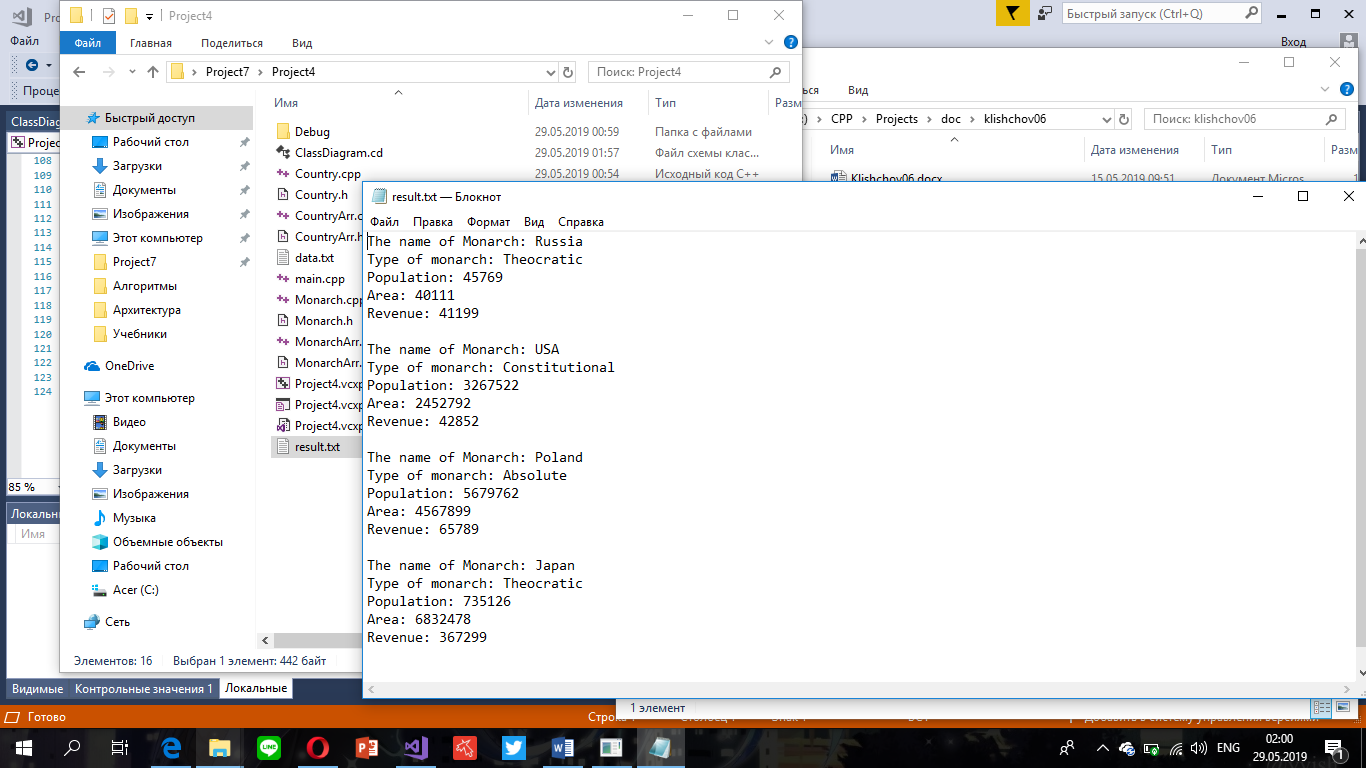


Рисунок № 3. Запис даних в файл

На рисунку № 4 зоображено результат запису в файл країни, отриманої по ID

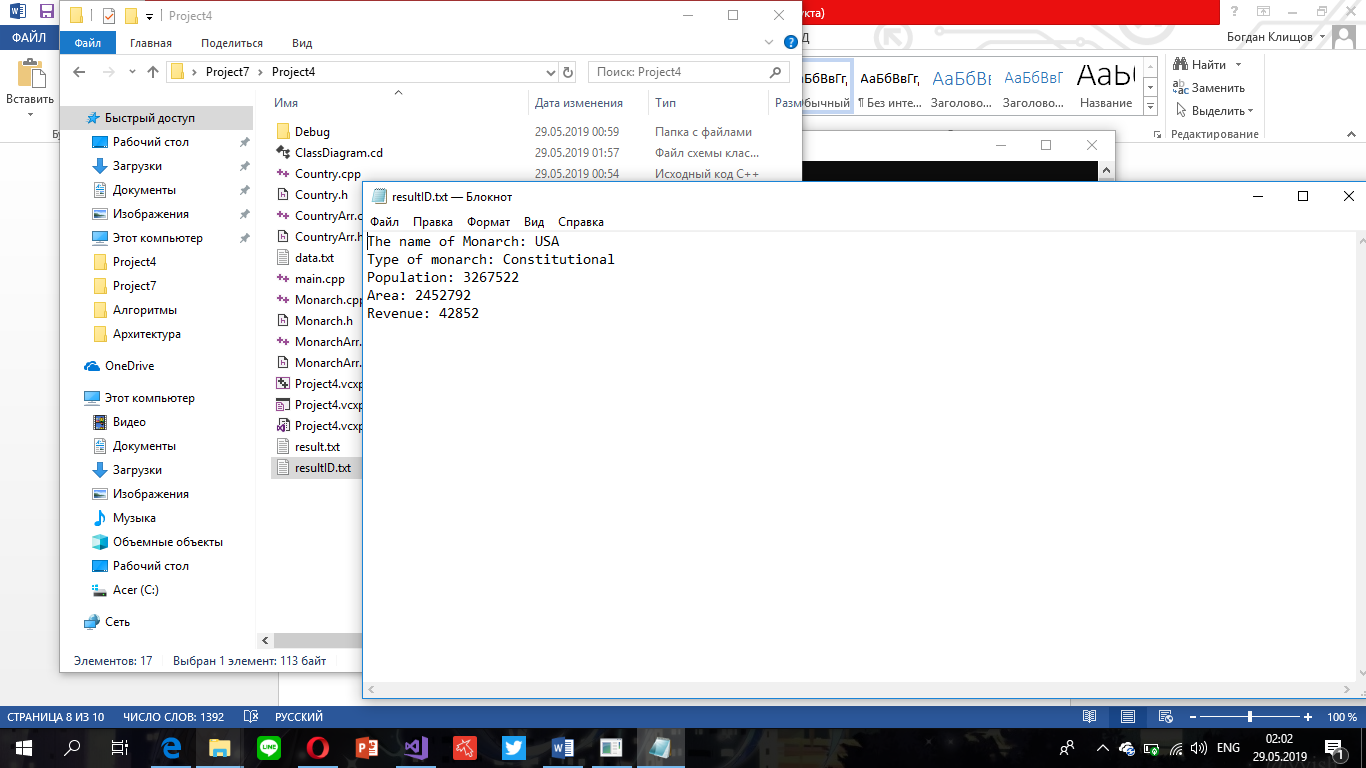


Рисунок № 4. Пошук по ID

На рисунку № 5 зоображено результат запису в файл країни, з максимальною кількістю населення

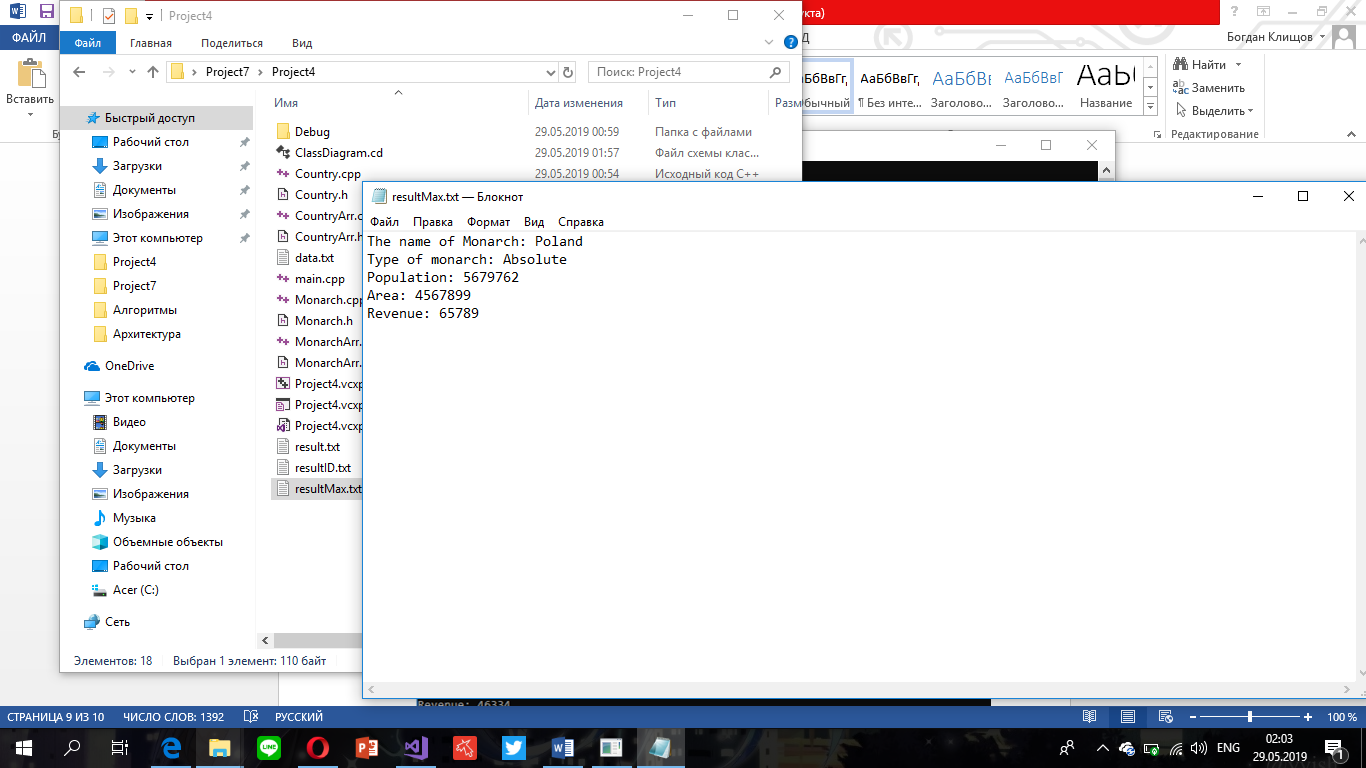


Рисунок № 5. Мінімальна щільність населення

**Програма має декілька функцій:**

1. Додавання елементу
2. Видалення елементу
3. Пошук по індексу
4. Запис в файл
5. Видалення масиву елементів
6. Сортування

**Висновок:** Отримав знання про парадигму ООП – спадкування. Навчився застосовувати отримані знання на практиці.