Міністерство освіти і науки України

Національний технічний університет "ХПІ"

кафедра "Інформатика та інтелектуальна власність"

**КУРСОВА РОБОТА**

з дисципліни "*Об'єктно-орієнтоване програмування"*

**Тема**

«Інформаційна таблиця»

|  |  |
| --- | --- |
|  | Виконавець:  Миргород В. І. |
|  | Керівник курсової роботи:  Соболь М. О. |
|  |  |

Харків 2021

РЕФЕРАТ

Предмет дослідження – сучасні методи розробки програм таких, як об'єктно-орієнтоване програмування та візуальне проектування, а також структурний і модульне програмування.

Мета курсової роботи – систематизація, поглиблення і активне застосування знань з основ об’єктно-орієнтованого програмування, закріплення знань, отриманих в лекційному курсі, а також на практичних і лабораторних заняттях.

Метод дослідження – вивчення літератури, складання і налагодження програм на комп'ютері.

Програма, що реалізована в цій роботі, призначена для виконання елементарних дій з таблицями користувачем.

Ключові слова: SORT, Visual C++, CLR, функцІя, проект, ПОВІДОМЛЕННЯ, Таблиця, програма, VISUAL STUDIO 19.

# ЗМІСТ

# 

|  |  |
| --- | --- |
| Вступ....……………………………………………………………………….... | 4 |
| 1.Аналіз предметної області ..………………………..………………………. | 5 |
| 1.1 Основні особливості середовища програмування Visual Studio 19 | 5 |
| 1.2 Стандартні компоненти... ............................................................... | 5 |
| 2.Розробка програми.....……………………………………………………...... | 8 |
| 2.1.Опис алгоритму створення форми.....…………………………......... | 8 |
| 2.2.Опис алгоритму створення елементів форми.................................. | 8 |
| 2.3.Опис використовуваних методів і подій.......……........................... | 9 |
| 2.4.Текст програми з коментарями .....…………………………………. | 9 |
| 3.Опис програмної реалізації............………………………………………… | 28 |
| 3.1.Інтерфейс користувача..………………………………………........... | 28 |
| 3.2.Інструкція користувача...…………………………………………...... | 29 |
| Висновок....……………………………………………………………….......... | 32 |
| Список літератури.........................……………………………………............. | 33 |

ВСТУП

У сьогоднішньому суспільстві все більше стає актуальним питання про розробку різних програм, які б допомагали людям зробити їх життя більш комфортним та простим. Програма, яку розроблено, допомагає користувачеві працювати з таблицею і даними в ній. Сьогодні кожне підприємство чи фірма використовує подібні програми для зберігання усієї потрібної інформації.

CLR проект дає змогу розробити програму з простим та зрозумілим інтерфейсом користувача, дозволяю зручно та швидко створювати та змінювати об’єкти форми. Код програми розробляється на мові C++, але для створення та програмування об’єктів форми використовується мова C#, що є досить схожою до C++.

Для вирішення завдання даної курсової роботи було обране середовище розробки Visual Studio 19. Середовище має широкий набір можливостей, C ++ Buider може використовуватися для розробки програм будь-якої складності.

Мета курсової роботи – підтвердження знань з об’єктно-орієнтованого програмування, основних його принципів. Також у програмі використано знання з таких тем, як обробка виключень, тестування, робота з файлами. Важливу роль у проекті займає бібліотека STL, що також входить до курсу ООП. Бібліотека дозволяє працювати з шаблонними класами та їх методами, що досить спрощує розробку програми, робить код та структуру програми більш зрозумілими.

Завдання даної роботи: створити програму, що за функціоналом дозволить обробляти таблицю, редагувати різним чином дані в ній. Таблиці можуть бути введені та виведені з використанням файлів. Також потрібно розробити методи, що дозволяють сортувати таблицю за різними даними. Створити пошукову систему таблиці, що дозволить знайти потрібну інформацію за будь-яким критерієм. Програма має бути написана на класах, їх методах. Під час розробки застосувати інкапсуляцію, поліморфізм та успадкування класів.

Практична значимість курсової роботи: Програма з графічним інтерфейсом, що дозволить користувачеві швидко та зручно обробляти дані та виконивати різноманітні дії над ними.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ
   1. Основні особливості середовища програмування Visual Studio 19.

Система об’єктно-орієнтованого програмування Visual Studio 19 виробництва корпорації **Microsoft** призначена для операційних систем **Windows7 – Windows 10.** Вона функціонує під керуванням ОС типу **Microsoft Windows.**

Інтегроване середовище Visual Studio 19 цілком підтримує стандарт мови **С++,** завдяки чому існує можливість створення за допомогою цієї системи програмування модулів і бібліотек, що застосовуються в інших засобах розробки.

Панель елементів.

Панель елементів дозволяє швидко додати чи змінити елементи форми. Панель елементів має різні типи елементів, що дозволяють розв’язувати різноманітні задачі і використовуються для якоїсь мети. Під час розробки програми найчастіше використовуємо Стандартні елементи управління, але й інші розділи включають не менш важливі та цікаві елементи.

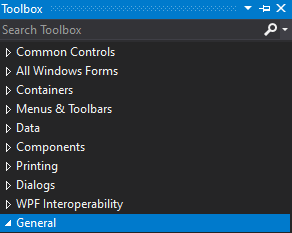


Рисунок 1.1 – Панель елементів VS19

Властивості об’єктів дозволяють змінити будь-які характеристики елементу форми, або ж самої форми. Ця панель пов’язана безпосередньо з кодом програми, тому під час внесення змін у панелі властивостей, потрібні характеристики одразу змінюються в коді.

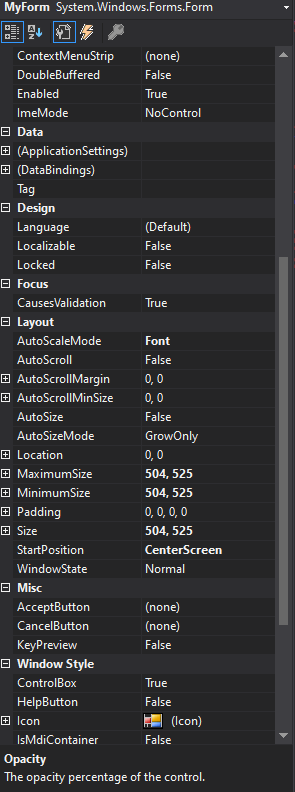


Рисунок 1.2 – Палітра властивостей форм VS19

Вкладка Огляду Рішень дозволяє відкрити потрібні файли програми чи змінити їх. Також з неї можна швидко дізнатися, які елементи включає програма, або створити нові.

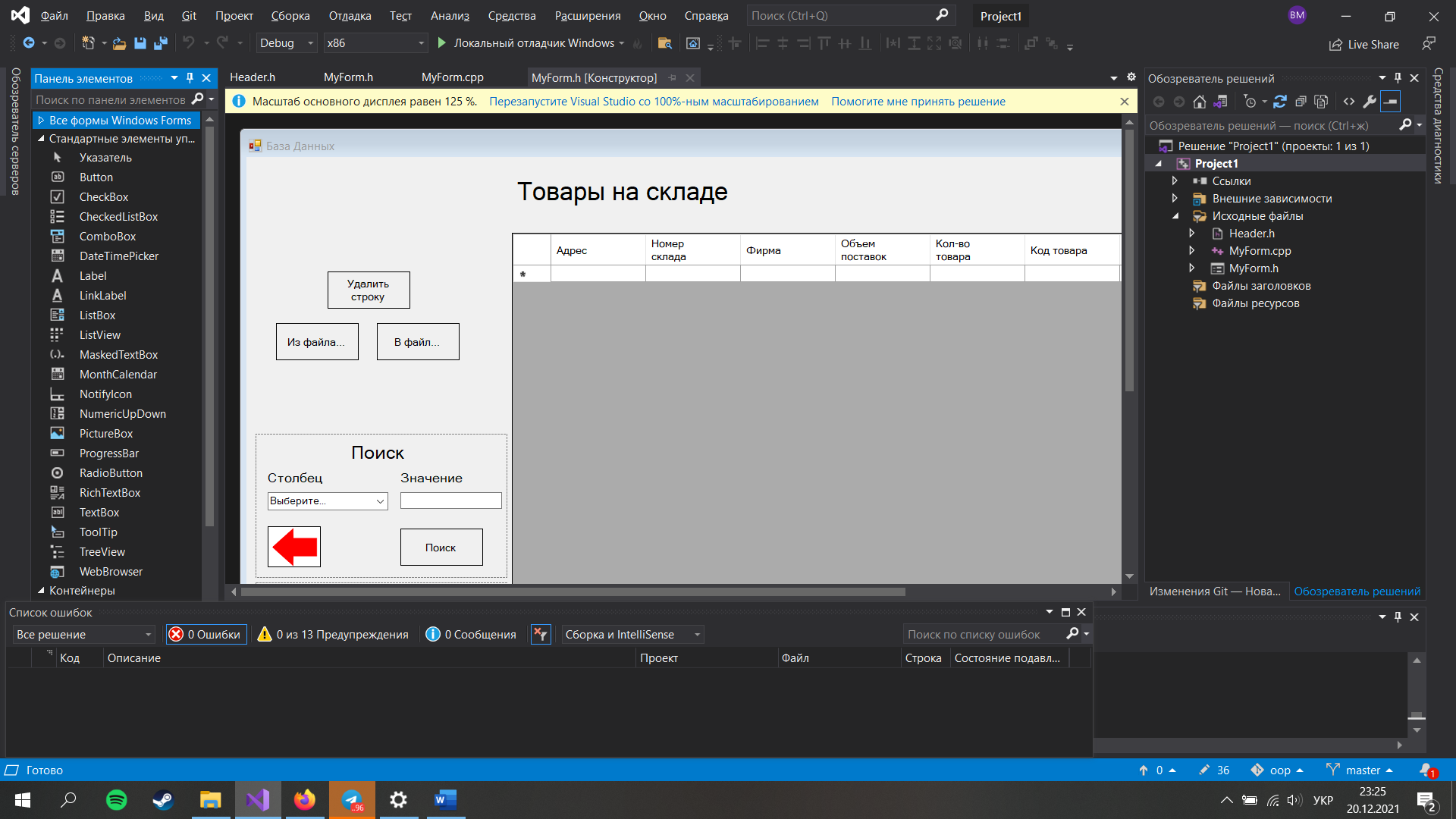


Рисунок 1.3 – Оглядач рішень VS19

2. РОЗРОБКА ПРОГРАМИ

2.1 Опис алгоритму створення форми

Алгоритм створення форми для програми можна представити наступними кроками.

1. Запуск Visual Studio 19. Створення проекту C++ CLR.

Записуємо до файлу форми MyForm.cpp наступний код, який забезпечує запуск форми.

#include "MyForm.h"

#include "Header.h"

#include <Windows.h>

using namespace Project1;

using namespace std;

using namespace System;

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {

setlocale(LC\_ALL, "Russian");

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm);

return 0;

}

2. Створення головної форми додатка.

* Створюєму головну форму.
* Додаємо в неї потрібні елементи з Панелі елементів (рис 1.1)
* Змінюємо їх властивості в Властивостях (рис. 1.2).

2.2 Опис алгоритму створення елементів форми:

Елемент Data grid view – таблиця зі строк та рядків, що дозволяє зручно налаштовувати та змінювати чарунку таблиці.

Елементи Button – кнопки, для яких можна запрограмувати подію і її обробку. Для кнопок зробимо події, що відповідатиме початку сотрування, пошуку чи введення-виведення та інше.

Елементи Label – строка, що дозволяє вивести текстову інформацію на форму.

Елементи Radio button – перемикач, що дозволяє обрати тільки один з кількох компонентів, для кожного з яких створено свої обробки подій. Ці елементи застосуємо для вибору типу сортировки.

Елемент Combo box – строка з можливістю вибору значення з введених. Застосуємо цей елемент для вибору стовпця, для якого буде виконано пошук або сортування.

2.3 Опис використовуваних методів і подій

Для компонента форми подія OnClick викликає певну логіку, задану автором.

Наприклад, для кнопок ці події починають методи сортування або пошуку.

2.4 Текст програми з коментарями

Програма включає:

* Myform.cpp (код описано вище)
* Myform.h
* Header.h

Розглянемо код файлу Header.h. В цьому файлі описано основні класи та методи, які безпосередньо пов’язані з будь-якими змінами на формі і, зокрема, в таблиці. Кожна чарунка таблиці пов’язана із відповідною змінною, а строка – з об’єктом. Об’єкти класу зберігаються у шаблонному класі vector, що дозволяє зручно змінювати та сортувати об’єкти в ньому.

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <msclr\marshal\_cppstd.h>

#include <iomanip>

using namespace std;

Підключаємо потрібні бібліотеки для роботи програми. Ці бібліотеки дозволяють використовувати функції, що не описано в даній програмі.

Далі слідує повна структура класу, його успадкування, а також всі потрібні для роботи методи:

class Objectt {

public:

virtual void Set(string) = 0;

};

class Building : virtual public Objectt {

protected:

string address;

public:

void Set(string s) {

address = s;

}

string GetAddress() {

return address;

}

};

class Warehouse : public Building {

protected:

string number;

public:

void Set(string s) {

number = s;

}

string GetNumber() {

return number;

}

};

class Supplier : public Warehouse {

protected:

string firm;

string count\_del;

public:

void SetFirm(string s) {

firm = s;

}

void SetCount\_del(string s) {

count\_del = s;

}

void Set(string s1, string s2) {

SetFirm(s1);

SetCount\_del(s2);

}

string GetFirm() {

return firm;

}

string GetCount\_del() {

return count\_del;

}

};

class Product : public Supplier {

protected:

string count;

string password;

string name;

public:

void SetCount(string s) {

count = s;

}

void SetPassword(string s) {

password = s;

}

void SetName(string s) {

name = s;

}

void Set(string s1, string s2, string s3) {

SetCount(s1);

SetPassword(s2);

SetName(s3);

}

string GetCount() {

return count;

}

string GetPassword() {

return password;

}

string GetName() {

return name;

}

string\* GetAll() {

string\* s = new string[7]{ address, number, firm, count\_del, count, password, name };

return s;

}

friend ostream& operator<< (ostream& out, const Product& obj);

friend istream& operator>> (istream& out, Product& obj);

friend bool operator== (const Product& obj1, const Product& obj2);

friend class Compare;

};

string tostr(System::String^ managed) {

return msclr::interop::marshal\_as<std::string>(managed);

}

bool compared(string s1, string s2) {

for (int i = 0; i < s2.size(); i++)

if (tolower(s1[i]) != tolower(s2[i]))

return false;

return true;

}

Маємо наступну ієрархію класів:

Object—Building—Warehouse—Supplier—Product

Дані класи відображають структуру успадкування. При чому 1 клас є інтерфейсом, а всі решта класи мають свої змінні та методи. Наслідування проходить за типом public, тому тип змінних та методів не змінюється в класах-нащадках.

Кожний клас має перегружений метод Set(), що дозволяє ввести нове значення у відповідну змінну. Також ций метод є чистим віртуальним методом у класі-інтерфейсі.

Кожний клас має метод Get(), що повертає значення змінної. Змінні класу мають тип private, тому для доступу до них потрібно виеористовувати такий метод.

Також для останнього класу перегружено 3 оператори, що дозволяють виконувати дії з об’єктами цього класу.

ostream& operator<< (ostream& out, const Product& obj) {

out << obj.address << " " << obj.number << " " << obj.firm << " " << obj.count\_del << " " << obj.count

<< " " << obj.password << " " << obj.name << endl;

return out;

}

istream& operator>> (istream& in, Product& obj) {

in >> obj.address;

in >> obj.number;

in >> obj.firm;

in >> obj.count\_del;

in >> obj.count;

in >> obj.password;

in >> obj.name;

return in;

}

bool operator== (const Product& obj1, const Product& obj2) {

if ((obj1.address == obj2.address) && (obj1.number == obj2.number) && (obj1.firm == obj2.firm) && (obj1.count\_del == obj2.count\_del) && (obj1.count == obj2.count) && (obj1.password == obj2.password) && (obj1.name == obj2.name))

return true;

else

return false;

}

Оператори потрібні для введення-виведення у файл, а також для порівняння 2 об’єктів класу між собою.

Також цей файл має в своему коді кілька додаткових функцій, що спрощують та прискорюють багаторазові обчислення.

string tostr(System::String^ managed) {

return msclr::interop::marshal\_as<std::string>(managed);

}

bool compared(string s1, string s2) {

for (int i = 0; i < s2.size(); i++)

if (tolower(s1[i]) != tolower(s2[i]))

return false;

return true;

}

Наприклад, перша функція дозволяє змінити тип строкового тексту з System::String на std::string, бо елементи форми зберігають текст у вигляді системного, коли у класах легше та зручніше працювати саме із звичайним текстовим типом.

Друга функція дозволяє порівняти 2 строки тексту, і поверне логічне значення в залежності від результату. Дана функція буде використана під час пошуку потрібного елемента, а використання окремої функції дозволяє виконувати ці дію для будь-якого стовпця таблиці.

Далі маємо кілька однотипних функцій:

bool CompareAddressUp(Product& left, Product& right) {

return left.GetAddress() < right.GetAddress();

}

bool CompareNumberUp(Product& left, Product& right) {

int l = stoi(left.GetNumber());

int r = stoi(right.GetNumber());

return l < r;

}

bool CompareFirmUp(Product& left, Product& right) {

return left.GetFirm() < right.GetFirm();

}

bool CompareCount\_delUp(Product& left, Product& right) {

int l = stoi(left.GetCount\_del());

int r = stoi(right.GetCount\_del());

return l < r;

}

bool CompareCountUp(Product& left, Product& right) {

int l = stoi(left.GetCount());

int r = stoi(right.GetCount());

return l < r;

}

bool ComparePasswordUp(Product& left, Product& right) {

int l = stoi(left.GetPassword());

int r = stoi(right.GetPassword());

return l < r;

}

bool CompareNameUp(Product& left, Product& right) {

return left.GetName() < right.GetName();

}

bool CompareAddressDown(Product& left, Product& right) {

return left.GetAddress() > right.GetAddress();

}

bool CompareNumberDown(Product& left, Product& right) {

int l = stoi(left.GetNumber());

int r = stoi(right.GetNumber());

return l > r;

}

bool CompareFirmDown(Product& left, Product& right) {

return left.GetFirm() > right.GetFirm();

}

bool CompareCount\_delDown(Product& left, Product& right) {

int l = stoi(left.GetCount\_del());

int r = stoi(right.GetCount\_del());

return l > r;

}

bool CompareCountDown(Product& left, Product& right) {

int l = stoi(left.GetCount());

int r = stoi(right.GetCount());

return l > r;

}

bool ComparePasswordDown(Product& left, Product& right) {

int l = stoi(left.GetPassword());

int r = stoi(right.GetPassword());

return l > r;

}

bool CompareNameDown(Product& left, Product& right) {

return left.GetName() > right.GetName();

}

Ці функції відповідають за сортування. Під час застосування відповідного алгоритму у якості параметра використовується одна з функцій, що дозволяє обрати параметр, за яким буде проведено сортування.

Далі перейдемо до файлу форми. Для початку розглянемо підключені файли програми:

#include "Header.h"

#include <iostream>

#include <fstream>

#include <string>

#include <msclr\marshal\_cppstd.h>

#include <vector>

#include <algorithm>

Після файлів починається текст програми, що відповідає коду об’єктів на формі. Цей код має в собі всі характеристики об’єктів форми, а також обробку відповідних подій. Код для опису характеристик нам не цікавий, бо створюється програмно і досить схожий між собою. Тому розглянемо обробку подій:

private: System::Void dataGridView1\_CellEndEdit(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e) {

if (this->dataGridView1->RowCount > vec.size() + 1) {

Product temp;

vec.push\_back(temp);

}

vector<Product>::iterator it;

it = vec.begin();

advance(it, this->dataGridView1->CurrentRow->Index);

System::String^ managed;

string s;

if (this->dataGridView1->CurrentRow->Cells[0]->Value != nullptr) {

managed = this->dataGridView1->CurrentRow->Cells[0]->Value->ToString();

s = msclr::interop::marshal\_as<std::string>(managed);

it->Building::Set(s);

}

if (this->dataGridView1->CurrentRow->Cells[1]->Value != nullptr) {

managed = this->dataGridView1->CurrentRow->Cells[1]->Value->ToString();

s = tostr(managed);

it->Warehouse::Set(s);

}

if (this->dataGridView1->CurrentRow->Cells[2]->Value != nullptr) {

managed = this->dataGridView1->CurrentRow->Cells[2]->Value->ToString();

s = tostr(managed);

it->SetFirm(s);

}

if (this->dataGridView1->CurrentRow->Cells[3]->Value != nullptr) {

managed = this->dataGridView1->CurrentRow->Cells[3]->Value->ToString();

s = tostr(managed);

it->SetCount\_del(s);

}

if (this->dataGridView1->CurrentRow->Cells[4]->Value != nullptr) {

managed = this->dataGridView1->CurrentRow->Cells[4]->Value->ToString();

s = tostr(managed);

it->SetCount(s);

}

if (this->dataGridView1->CurrentRow->Cells[5]->Value != nullptr) {

managed = this->dataGridView1->CurrentRow->Cells[5]->Value->ToString();

s = tostr(managed);

it->SetPassword(s);

}

if (this->dataGridView1->CurrentRow->Cells[6]->Value != nullptr) {

managed = this->dataGridView1->CurrentRow->Cells[6]->Value->ToString();

s = tostr(managed);

it->SetName(s);

}

Одна з найважливіших подій у програмі, пов’язана із зміною будь-якої чарунки таблиці. Таблиця пов’язана з вектором об’єктів, у яких зберігається вся потрібна інформація щодо таблиці. Тому будь-яка зміна таблиці викликає зміну поточного стану вектора. Дана подія змінює об’єкт повністю, а значення у таблиці змінюється відповідно до коду таблиці.

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

if (vec\_temp.empty()) {

int n = this->dataGridView1->CurrentRow->Index;

vector<Product>::iterator it;

it = vec.begin();

if (vec.size() > n) {

advance(it, n);

vec.erase(it);

this->dataGridView1->Rows->RemoveAt(n);

}

}

else {

int n = this->dataGridView1->CurrentRow->Index;

vector<Product>::iterator it\_temp;

vector<Product>::iterator it = vec.begin();

it\_temp = vec\_temp.begin();

if (vec\_temp.size() > n) {

advance(it\_temp, n);

for (int i = 0; it != vec.end(); i++) {

if (\*it == \*it\_temp) {

vec.erase(it);

break;

}

it++;

}

vec\_temp.erase(it\_temp);

this->dataGridView1->Rows->RemoveAt(n);

}

}

}

Подія обробки видалення строки з таблиці, що також тягне за собою видалення відповідного об’єкта. У коді обробки події використано потрібні перевірки, тому все працює навіть тоді, коли потрібної строки немає, або ж таблиця не має строк зовсім.

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

vec.clear();

ifstream in("in.txt");

while (!in.eof()) {

Product temp;

in >> temp;

vec.push\_back(temp);

}

in.close();

int n = this->dataGridView1->RowCount;

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

this->dataGridView1->Rows->RemoveAt(0);

}

string\* s;

vector<Product>::iterator it = vec.begin();

Product temp;

for (int i = 0; it != vec.end(); i++) {

this->dataGridView1->Rows->Add();

s = it->GetAll();

for (int j = 0; j < 7; j++) {

System::String^ str = gcnew String(s[j].c\_str());

this->dataGridView1->Rows[i]->Cells[j]->Value = str;

}

it++;

delete[] s;

}

Обробка даної події дає змогу зчитати з файлу дані для заповнення таблиці. Звісно, вектор об’єктів також заповнюється відповідними значеннями, а також виконується перевірка на можливість відкриття файлу. Після обробки події в таблицю буде внесено всі дані з файлу, а попередні дані буде видалено.

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

vector<Product>::iterator it;

ofstream out("out.txt");

for (it = vec.begin(); it != vec.end(); it++) {

out << \*it;

}

out.close();

MessageBox::Show("Сохранено!\nФайл: out.txt\n");

}

Ця подія працює протилежно до попередньої. Вона виводить дані таблиці до файлу. При цьому таблиця ніяким чином не змінюється.

private: System::Void button5\_Click(System::Object^ sender, System::EventArgs^ e) {

int x = this->comboBox1->SelectedIndex;

vec\_temp.clear();

vector<Product>::iterator it = vec.begin();

//vector<Product, int>::iterator it\_temp = vec\_temp.begin();

System::String^ q\_sys = this->textBox1->Text;

string q = tostr(q\_sys);

if (x == 0) {

string s;

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

s = it->GetAddress();

if (compared(s, q)) {

vec\_temp.push\_back(\*it);

}

it++;

}

}

else if (x == 1) {

string s;

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

s = it->GetNumber();

if (compared(s, q)) {

vec\_temp.push\_back(\*it);

}

it++;

}

}

else if (x == 2) {

string s;

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

s = it->GetFirm();

if (compared(s, q)) {

vec\_temp.push\_back(\*it);

}

it++;

}

}

else if (x == 3) {

string s;

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

s = it->GetCount\_del();

if (compared(s, q)) {

vec\_temp.push\_back(\*it);

}

it++;

}

}

else if (x == 4) {

string s;

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

s = it->GetCount();

if (compared(s, q)) {

vec\_temp.push\_back(\*it);

}

it++;

}

}

else if (x == 5) {

string s;

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

s = it->GetPassword();

if (compared(s, q)) {

vec\_temp.push\_back(\*it);

}

it++;

}

}

else if (x == 6) {

string s;

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

s = it->GetName();

if (compared(s, q)) {

vec\_temp.push\_back(\*it);

}

it++;

}

}

int n = this->dataGridView1->RowCount;

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

this->dataGridView1->Rows->RemoveAt(0);

}

string\* s;

vector<Product>::iterator it\_temp = vec\_temp.begin();

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

this->dataGridView1->Rows->Add();

s = it\_temp->GetAll();

for (int j = 0; j < 7; j++) {

System::String^ str = gcnew String(s[j].c\_str());

this->dataGridView1->Rows[i]->Cells[j]->Value = str;

}

it\_temp++;

delete[] s;

}

}

Ця велика подія відповідає за функцію пошуку по таблиці. Дозволяє шукати у будь-якому стовпці за потрібним значенням. Під час події викликають функції, що описано вище.

private: System::Void button6\_Click(System::Object^ sender, System::EventArgs^ e) {

vec\_temp.clear();

int n = this->dataGridView1->RowCount;

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

this->dataGridView1->Rows->RemoveAt(0);

}

string\* s;

vector<Product>::iterator it = vec.begin();

for (int i = 0; it != vec.end(); i++) {

this->dataGridView1->Rows->Add();

s = it->GetAll();

for (int j = 0; j < 7; j++) {

System::String^ str = gcnew String(s[j].c\_str());

this->dataGridView1->Rows[i]->Cells[j]->Value = str;

}

it++;

delete[] s;

}

}

Ця подія повертає попередню таблицю після виконання функції пошуку. Звертається до вектора, який зберігає всі об’єкти до пошуку, і виводить інформацію на екран.

private: System::Void button7\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

if ((!this->radioButton1->Checked) && (!this->radioButton2->Checked))

throw 1;

}

catch (int) {

MessageBox::Show("Выберите тип сортировки!\n");

}

int x = this->comboBox2->SelectedIndex;

bool q1 = this->radioButton1->Checked;

bool q2 = this->radioButton2->Checked;

vector<Product>::iterator it = vec.begin();

if (x == 0) {

if (q1)

sort(vec.begin(), vec.end(), CompareAddressUp);

else if (q2)

sort(vec.begin(), vec.end(), CompareAddressDown);

}

else if (x == 1) {

if (q1)

sort(vec.begin(), vec.end(), CompareNumberUp);

else if (q2)

sort(vec.begin(), vec.end(), CompareNumberDown);

}

else if (x == 2) {

if (q1)

sort(vec.begin(), vec.end(), CompareFirmUp);

else if (q2)

sort(vec.begin(), vec.end(), CompareFirmDown);

}

else if (x == 3) {

if (q1)

sort(vec.begin(), vec.end(), CompareCount\_delUp);

else if (q2)

sort(vec.begin(), vec.end(), CompareCount\_delDown);

}

else if (x == 4) {

if (q1)

sort(vec.begin(), vec.end(), CompareCountUp);

else if (q2)

sort(vec.begin(), vec.end(), CompareCountDown);

}

else if (x == 5) {

if (q1)

sort(vec.begin(), vec.end(), ComparePasswordUp);

else if (q2)

sort(vec.begin(), vec.end(), ComparePasswordDown);

}

else if (x == 6) {

if (q1)

sort(vec.begin(), vec.end(), CompareNameUp);

else if (q2)

sort(vec.begin(), vec.end(), CompareNameDown);

}

int n = this->dataGridView1->RowCount;

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

this->dataGridView1->Rows->RemoveAt(0);

}

string\* s;

it = vec.begin();

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

this->dataGridView1->Rows->Add();

s = it->GetAll();

for (int j = 0; j < 7; j++) {

System::String^ str = gcnew String(s[j].c\_str());

this->dataGridView1->Rows[i]->Cells[j]->Value = str;

}

it++;

delete[] s;

}

}

Одна з основних подій, потрібна для виконання алгоритму сортування ветора та таблиці. Сортує значення таблиці за обраним параметром, також є перевірка на правильність вибору даних. Під час сортування використано алгоритми, які описано у відповідній бібліотеці, а також функції, що описано вище в записці.

private: System::Void button4\_Click\_1(System::Object^ sender, System::EventArgs^ e) {

exit(0);

}

Остання подія у програмі, потрібна лише для того, щоб завершити програму. Але програму буде завершено потрібним чином, тобто видалено всі динамічні структури та зупинено виконання всіх методів і функцій.

Отже, всі ці події оброблюються, як правило, при натисканні на об’єкт, і виконують потрібну дію. Всі зміни на форму викликають свою обробки подій, що дозволяю користувачеві взаємодіяти з програмою через використання зручного та простого інтерфейсу.

3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Інтерфейс користувача

Інтерфейс включає в себе:

* Таблицю
* Кнопки, що дозволяють змінювати таблицю
* Інші об’єкти для взаємодії з таблицею

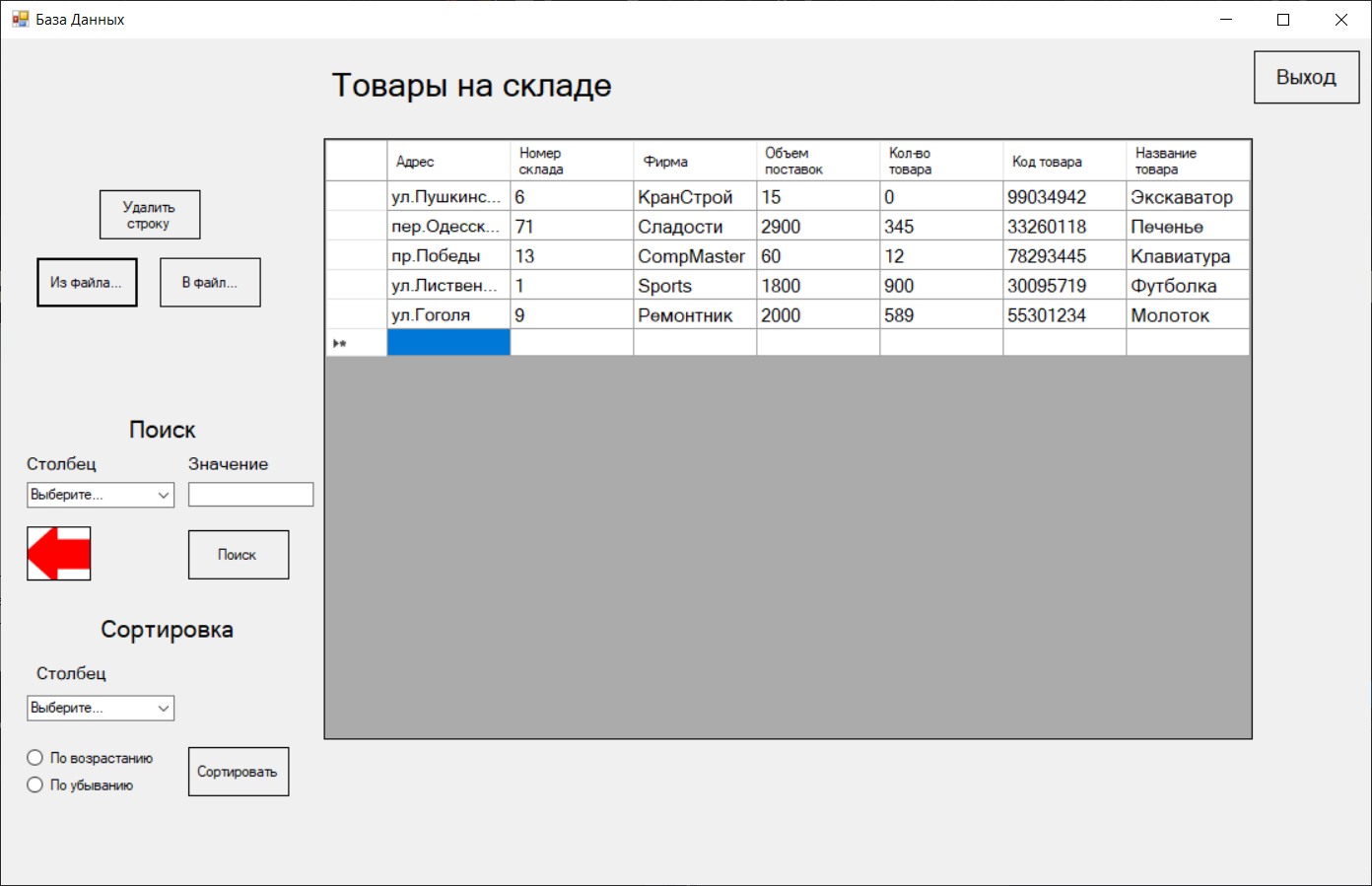


Рисунок 3.1 – Загальний інтерфейс програми

3.2 Інструкція користувача

Щоб скористатися програмою користувач має виконати наступні дії.

1. Відкрити програму.
2. Ввести значення до таблиці з файлу чи клавіатури.
3. Змінити таблицю за потреби.

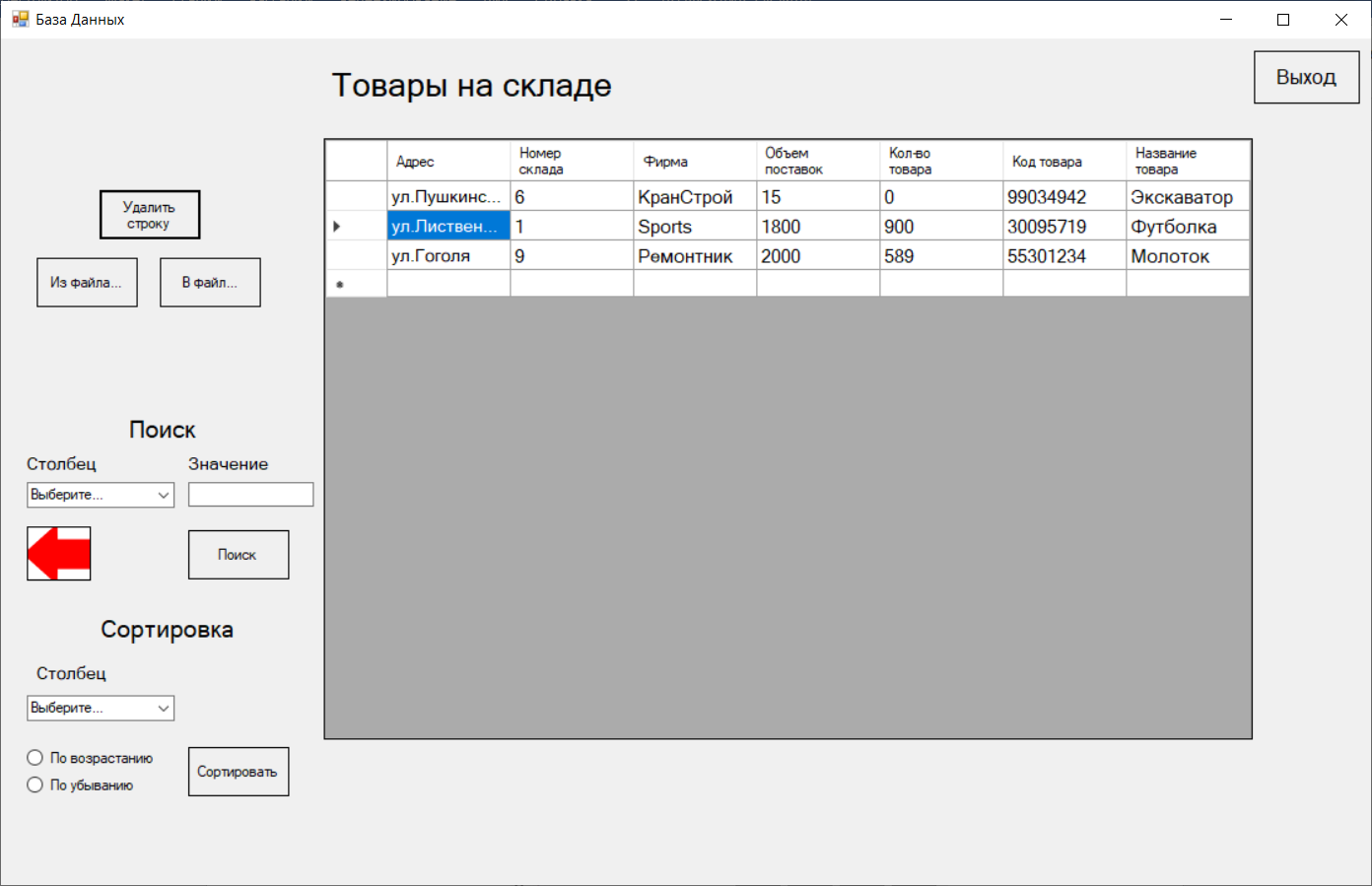


Рисунок 3.2 – Приклад видалення строк з таблиці

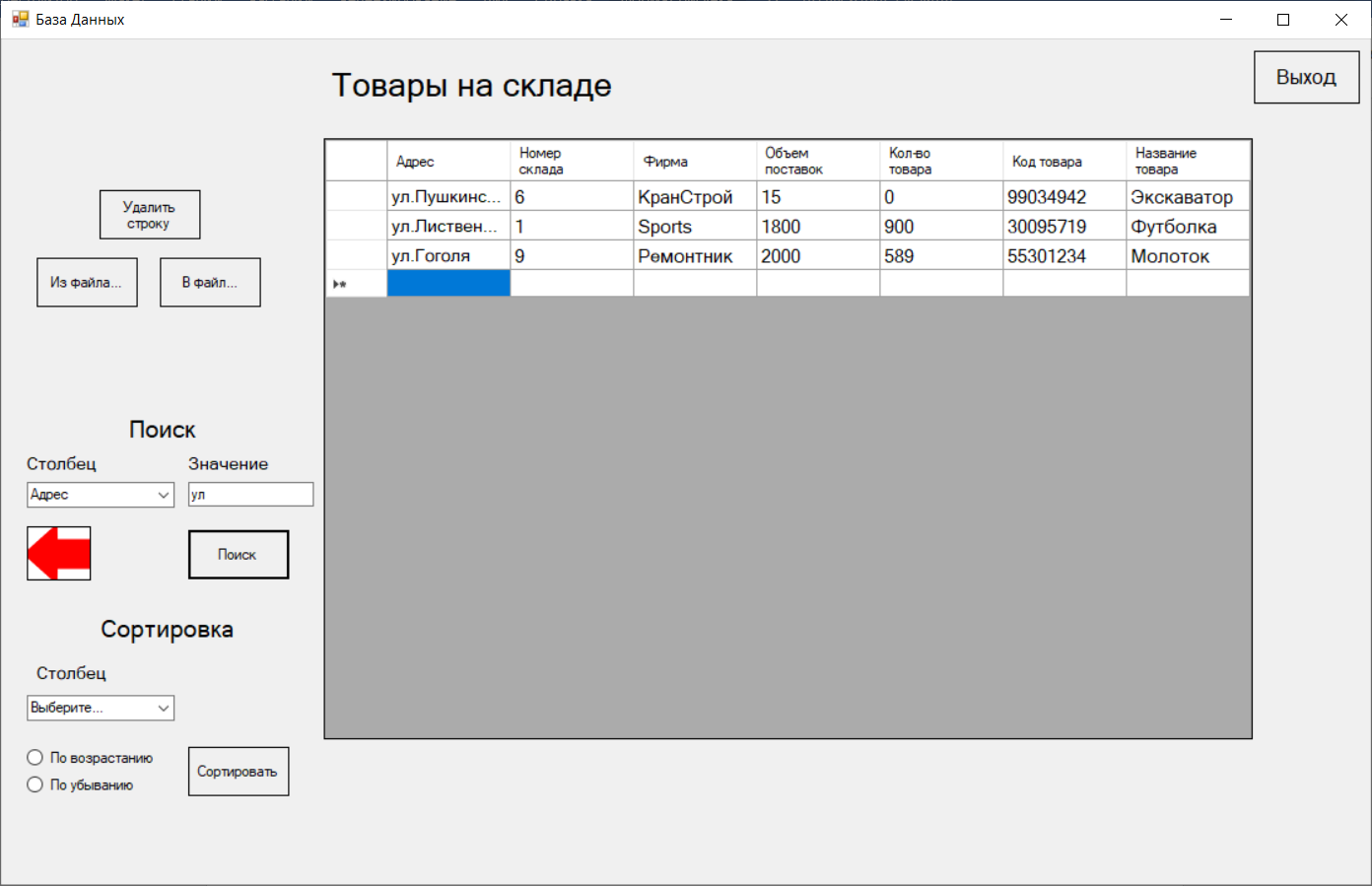


Рисунок 3.3 – Пошук по таблиці

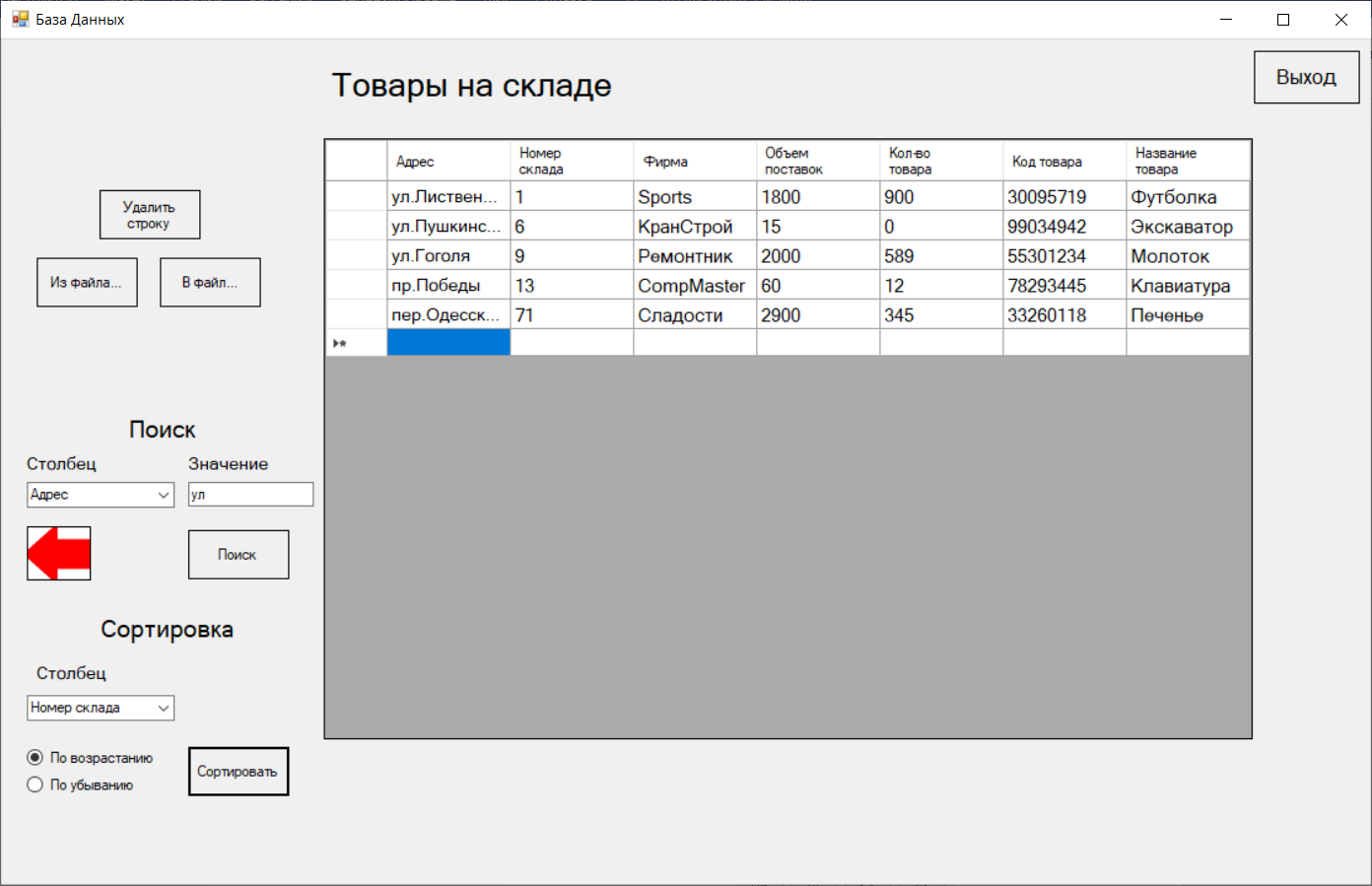


Рисунок 3.4 – Сортування таблиці

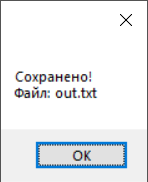


Рисунок 3.5 – Збереження даних таблиці до файлу

ВИСНОВОК

Середовище програмування Microsoft Visual Studio надає безліч можливостей для розробки програмного забезпечення. Visual Studio 19 дає можливість створювати і підключати сторонні додатки (плагіни) для розширення функціоналу на кожному рівні, включаючи підтримку систем контролю версій вихідного коду (Subversion і VisualSourceSafe), додавання нових наборів інструментів (для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування або інструментів для інших аспектів процесу розробки програмного забезпечення). Її плюсом є комплексність(багато інструментів в одному), підтримка багатьох мов програмування, що дозволяє писати код своєю мовою чи будь-якими іншими бажаними мовами, висока швидкість розробки, гнучка можливість налагодження.

В результаті виконання проекту була повністю реалізована програма База Даних, яка дозволяє проводити обробку таблиці. В процесі розробки було використано різноманітні ресурси, а також виконано всі потрібні вимогу щодо застосування тем програмування. Програму протестовано, помилки не знайдено.

СПИСОК ЛІТЕРАТУРИ

1. Campbell Parallel Programming with Microsoft® Visual C++® / Campbell. – Москва: Гостехиздат, 2011. – 784 c.
2. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. – Москва: Высшая школа, 2015. – 882 c.
3. Балена, Франческо Современная практика программирования на Microsoft Visual Basic и Visual C# / Франческо Балена , Джузеппе Димауро. – М.: Русская Редакция, 2015. – 640 c.
4. Боровский, А. C++ и Pascal в Kylix 3. Разработка интернет–приложений и СУБД / А. Боровский. – М.: БХВ–Петербург, 2015. – 544 c.
5. Давыдов, В. Visual C++. Разработка Windows-приложений с помощью MFC и API–функций / В. Давыдов. – М.: БХВ–Петербург, 2014. – 576 c.
6. Довбуш, Галина Visual C++ на примерах / Галина Довбуш , Анатолий Хомоненко. –М.: БХВ-Петербург, 2012. – 528 c.
7. Зиборов, В. MS Visual C++ 2010 в среде .NET / В. Зиборов. – М.: Питер, 2012. – 320 c.
8. Кетков, Юлий Практика программирования: Visual Basic, C++ Builder, Delphi. Самоучитель (+ дискета) / Юлий Кетков , Александр Кетков. – М.: БХВ–Петербург, 2012. – 464 c.
9. Мешков, А. Visual C++ и MFC / А. Мешков, Ю. Тихомиров. – М.: БХВ-Петербург, 2013. – 546 c.
10. Неформальное введение в C++ и Turbo Vision. – Москва: ИЛ, 2010. – 384 c.