Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ИНФОРМАЦИОННАЯ СИСТЕМА

ЛЕКАРСТВЕННЫХ ПРЕПАРАТОВ

БГУИР КП 1-40 02 01 109 ПЗ

Студент: гр. 150501 Кипятков В. И.

Руководитель: старший преподаватель

каф. ЭВМ Ковальчук А. М.

МИНСК 2022

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 2

1 ПОСТАНОВКА ЗАДАЧИ 2

2 ОБЗОР ЛИТЕРАТУРЫ 2

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВНИЕ 2

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 2

4.1 Разработка схем алгоритмов 2

4.2 Разработка алгоритмов 2

5 РЕЗУЛЬТАТ РАБОТЫ 2

ЗАКЛЮЧЕНИЕ 2

СПИСОК ЛИТЕРАТОРЫ 2

ПРИЛОЖЕНИЕ А 2

ПРИЛОЖЕНИЕ Б 2

ПРИЛОЖЕНИЕ В 2

ПРИЛОЖЕНИЕ Г 2

ПРИЛОЖЕНИЕ Д 2

# ВВЕДЕНИЕ

С++ - язык программирования, который поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщенное программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником - языком C, - наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

Название «язык программирования С++» происходит от языка программирования С, в котором унарный оператор ++ обозначает инкремент переменной.

Язык программирования широко используется для разработки программного обеспечения. А именно, создание разнообразных прикладных программ, разработка операционных систем, драйверов устройств, а также видео игр и многое другое.

С целью поддержки принципов объектно-ориентированного программирования все объектно-ориентированные языки, включая С++, обеспечивают три характерных принципа: инкапсуляцию, полиморфизм, и наследование.

Инкапсуляция — это программный механизм, который связывает данные с обрабатывающими их кодами и защищает и те, и другие от внешних воздействий и ошибочных действий.

Полиморфизм обозначает средство, позволяющее посредством единого интерфейса получить доступ к целому классу действий.

Наследование является процессом, который позволяет одному объекту приобретать свойства другого объекта.

# 1 ПОСТАНОВКА ЗАДАЧИ

Программный модуль “Информационная система лекарственных препаратов” должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Информация должна храниться в нескольких файлах, связанных определенным образом. Реализовать добавление, удаление, поиск лекарственных препаратов по цене, названию, производителю. Другие методы, в зависимости от специфики задачи.

Разработать иерархию классов с использованием наследования (не меньше 3-х уровней наследования). Разработать и использовать классы контейнеров, итераторов и алгоритмов (свои и STL). Производить обработку исключительных ситуаций. При реализации операции редактирования, добавления, удаления информации необходимо предусмотреть операцию отмены последних действий.

# 2 ОБЗОР ЛИТЕРАТУРЫ

**2.1 Обзор методов и алгоритмов решения поставленной задачи**

Для решения задачи был выбран язык программирования С++ и методология объектно-ориентированного программирования.

В процессе разработки программы были использованы различные возможности языка С++, которые будут описаны ниже.

Для хранения объектов было использовано такое средство языка, как шаблонный класс. Конкретно в данной программе для этой цели был реализован свой контейнер двунаправленное кольцо.

Получение доступа к элементам контейнера осуществляется с помощью итератора – это объект, который может перебирать элементы в контейнере и предоставлять доступ к отдельным элементам. В процессе написания программы был создан свой класс итератор.

Язык С++ предоставляет возможность работы со стандартной библиотекой шаблонов(STL) – набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций. STL-контейнер deque, реализующий динамический массив, и его методы push\_back – добавить элемент в конец контейнера, pop\_back – удаление последнего элемента контейнера, pop\_front – удаление первого элемента контейнера, size – возвращает количество элементов контейнера, оператор [] – возвращает элемент по указанному индексу, были использованы для реализации отмены последнего действия.

В программе осуществляется обработка исключительных ситуаций – механизм, предназначенный для описания реакции программы на ошибки, которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей обработки программой её базового алгоритма, этот механизм реализуется с помощью конструкции try…catch и оператора throw.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

**3.1 Структура входных и выходных данных**

Таблица 3.1 - файл мазей *Solutions.txt*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название препарата | Цена(BYN) | Производитель | Масса(мг, г) | Концентрация(%) | Способ нанесения |
| Вазилин | 2.99 | Здоровье | 80г | 20 | Втирание |

Таблица 3.2 - файл растворов *Salve.txt*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название препарата | Цена(BYN) | Производитель | Объём(мл, л) | Концентрация(%) | способ применения |
| Касторка | 8 | Цветок | 500мл | 96 | Внутренний |

Таблица 3.3 - файл таблеток *Tablets.txt*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название препарата | Цена(BYN) | Производитель | Масса(мг, г) | количество в упаковке(шт) |
| Анальгин | 4.45 | Санофи | 500мг | 20 |

Таблица 3.4 - файл порошков *Powder.txt*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название препарата | Цена(BYN) | Производитель | Масса(мг, г) | основная жидкость |
| Терафлю | 8,9 | БелАвита | 50г | Горячая вода |

Таблица 3.5 - файл аэрозолей *Aerosols.txt*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название препарата | Цена(BYN) | Производитель | Концентрация(%) | время активности(с, мин, ч) |
| Ксилин | 3,00 | Вечность | 2 | 5ч |

**3.2 Разработка диаграммы классов**

Диаграмма классов данной работы показана в приложении А.

**3.3 Описание классов**

Класс **Ring<T> -** шаблонный класс контейнера двунаправленного кольца. Является основным классом программы и служит для хранения объектов прочих классов.

Описание полей класса:

Element<T>\* Curr – указатель на текущий элемент кольца.

Element<T>\* Head – указатель на первый элемент кольца.

int length – количество элементов кольца.

Описание методов:

Ring() – конструктор контейнера.

Ring(T obj) – конструктор с параметром.

Ring(const Ring<T>& obj) – конструктор копирования.

~Ring() – деструктор контейнера.

void init() – текущий элемент ссылается на первый.

void push(T data) – добавляем в кольцо новый элемент T.

void push() – добавляем в кольцо новый элемент введённый с клавиатуры.

T pop() – возвращает значение последнего элемента и удаляет его.

void del(Ring<T>& ring) – удаление элементов из контейнера совпадающих с элементами ring.

void reInf(T finded) – метод изменения поля.

void loop() – выводит содержимое контейнера.

void clear() – очистка контейнера.

int isNoEmpty() – проверяет пуст ли контейнер.

void past(Ring<T> ring) – вставляет элементы из ring в данный контейнер.

Iterator Begin() – вернуть начало кольца

Iterator End() – вернуть конец кольца

Структура **Element<T>** – шаблонная структура элемента контейнера

Описание полей структуры:

Element\* next – указатель на следующий элемент контейнера.

Element\* prev – указатель на предыдущий элемент контейнера.

T data – информация, хранящаяся в элементе.

Класс **Iterator** – класс итератор вложенный в Ring

Описание полей класса:

Element<T>\* curr – указатель на текущий элемент.

Описание методов класса:

Iterator() **–** Конструктор.

Iterator(Element<T>\*) **–** конструктор для функций Begin() и End()

T& operator++(int) **–** двинуться впереёд по кольцу

T& operator--(int) **–** двинуться назад по кольцу

T& operator\*() **–** получить данные по указателю

bool operator!=(Iterator) **–** перегрузка оператора !=

bool operator==(Iterator) **–** перегрузка оператора ==

Класс **Interface** – класс интерфейса, включающий в себя все основные меню работы с программой

Описание методов класса:

int start() – меню выбора типа препарата.

int chouse(Ring<T>&, const char\*) – меню выбора операции над препаратами.

void reInf\_ring(Ring<T>& test) – меню изменения информации о препаратах.

void show\_ring(Ring<T> test) – меню вывода препаратов.

void delete\_ring(Ring<T>& test) – меню удаления препаратов.

void save\_ring(Ring<T> ring, deque<Ring<T>>& memorry) – сохранение последнего действия.

void ctrl\_z(Ring<T>& ring, deque<Ring<T>>& memorry) – отмена последнего действия.

void get\_file(Ring<T>& test, const char\* name\_file) – достать всю информацию из файла.

void geve\_file(Ring<T>& test, const char\* name\_file) – записать всю информацию в файл.

Класс **File** – базовый класс работы с файлами

Описание полей класса:

char file\_name[80] – название файла.

ifstream file\_i – поток для чтения из файла

ofstream file\_o – поток для записи в файл

Описание методов класса:

File(const char\* name) **–** конструктор с параметром.

Класс **File\_text –** класс работы с текстовыми файлами

Описание методов класса:

File\_text(const char\* name) : File(name) – конструктор с параметром.

~File\_text() – деструктор.

bool open\_file\_in() – открыть файл в режиме чтения.

bool open\_file\_out() – открыть файл в режиме записи.

void remote() – очистить файл.

void write\_record\_in\_file\_text(T& obj) – записать объект T в файл.

void read\_record\_in\_file\_text(T& obj) – записать информацию в объект T из файла.

bool feof() – проверяет, конец ли файла.

Класс **Exception** – базовый класс для ошибок разного типа

Описание полей класса:

int code – код ошибки.

char massage[80] – тип ошибки

Описание методов класса:

Exception(int, const char[80])– конструктор класса

~Exception() – деструктор класса

virtual void what() – абсолютно виртуальная функция вывода ошибки на экран

Класс **Exception\_input** - класс обработки ошибок ввода

Описание полей класса:

char input\_massage[80] – сообщение чтобы помочь исправить ошибку

Exception\_input(int, const char[80], const char[80]) – конструктор с параметрами.

~Exception\_input() – деструктор класса

void what() – вывод ошибки

Класс **Exception\_cont** - класс обработки ошибок контейнера

Описание полей класса:

char cont\_massage[80] – сообщение чтобы помочь исправить ошибку

Exception\_cont(int, const char[80], const char[80]) – конструктор с параметрами

~Exception\_cont() – деструктор класса

void what() – вывод ошибки

Класс **Exception\_file** – базовый класс для ошибок с файлами

Описание полей класса:

char file\_massage[80] – сообщение чтобы помочь исправить ошибку

Описание методов класса:

Exception\_file(int, const char[80], const char[80]) – конструктор класса

~ Exception\_file () – деструктор класса

virtual void what() – абсолютно виртуальная функция вывода ошибки на экран

Класс **Algorithm** – класс для манипуляций с контейнером

Описание методов класса:

void sort\_Ring(Ring<T>& ring) – сортирует контейнер.

void find(Ring<T>& ring, T inf) – выделяет элементы из контейнера ring равные элементу inf.

Класс **Medication** – класс лекарственный препарат

Описание полей класса:

string name – название

float price – цена

string manufacturer – производитель

Описание методов класса:

Medication() – конструктор

Medication(const Medication& copy) – конструктор копирования

Medication(Medication&& copy) noexcept – конструктор перемещения

~Medication() – деструктор

string get\_name() – получить название

float get\_price() – получить цену

string get\_manufacturer() – получить производителя

void set\_name(std::string) – установить название

void set\_price(float) – установить цену

void set\_manufacturer(string) – установить производителя

friend bool equ(Medication& first\_object, Medication& second\_object) – эквиволентвы ли объекты

friend bool operator==(Medication&, Medication&) – перегрузка оператора ==

Medication& operator=(const Medication& equ\_string) – перегрузка оператора =

friend ostream& operator<<(ostream&, Medication&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Medication&) – перегрузка оператор ввода

friend ofstream& operator<<(ofstream& fs, Medication& obj) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream& fs, Medication& obj) – перегрузка оператор ввода из файла

Класс **Aerosols –** классаэрозоли,наследуется от Medication

Описание полей класса:

float concetration – концентрация

string time – время действия препарата

Описание методов класса:

Aerosols() – конструктор

Aerosols(const Aerosols& copy) – конструктор копирования

Aerosols(Aerosols&& copy) noexcept – конструктор перемещения

~Aerosols() – деструктор

string get\_time() – получить время действия препарата

float get\_concentration() – получить концентрацию

void set\_time(string) – установить время действия препарата

void set\_concentration(float) – установить концентрацию

bool set\_something()– установить выбранное поле

friend bool equ(Aerosols& first\_object, Aerosols& second\_object) – эквиволентвы ли объекты

friend bool compare(Aerosols& first\_obj, Aerosols& second\_obj, int& menu) – сравнивает объекты по выбранному полю

friend bool operator==(Aerosols&, Aerosols&) – перегрузка оператора ==

Aerosols& operator=(const Aerosols& equ\_string) – перегрузка оператора =

friend ostream& operator<<(ostream&, Aerosols&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Aerosols&) – перегрузка оператор ввода

friend ofstream& operator<<(ofstream&, Aerosols&) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Aerosols&) – перегрузка оператор ввода из файла

void print\_table() – вывод шапки таблицы

Класс **Liquid –** класс жидкие, наследуется от Medication

Описание полей класса:

double volume – объём

float concentration – концентрация

Описание методов класса:

Liquid() – конструктор

Liquid(const Liquid& copy) – конструктор копирования

Liquid(Liquid&& copy) noexcept – конструктор перемещения

~Liquid() – деструктор

string get\_volume() – получить объём

float get\_concentration() – получить концентрацию

void set\_volume(string) – установить объём

void set\_concentration(float) – установить концентрацию

friend bool equ(Liquid&, Liquid&) – эквиволентвы ли объекты

friend bool operator==(Liquid&, Liquid) – перегрузка оператора ==

Liquid& operator=(const Liquid&) – перегрузка оператора =

friend ostream& operator<<(ostream&, Liquid&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Liquid&) – перегрузка оператор ввода

friend ofstream& operator<<(ofstream&, Liquid&) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Liquid&) – перегрузка оператор ввода из файла

Класс **Solid –** класс твёрдые, наследуется от Medication

Описание полей класса:

string weight – масса

Описание методов класса:

Solid() – конструктор

Solid(const Solid& copy) – конструктор копирования

Solid(Solid&& copy) noexcept – конструктор перемещения

~Solid() –деструктор

string get\_weight() – получить масса

void set\_weight(string) – установить масса

friend bool equ(Solid&, Solid&) – эквиволентвы ли объекты

friend bool operator==(Solid&, Solid&) – перегрузка оператора ==

Solid& operator=(const Solid&) – перегрузка оператора =

friend ostream& operator<<(ostream&, Solid&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Solid&) – перегрузка оператор ввода

friend ofstream& operator<<(ofstream&, Solid&) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Solid&) – перегрузка оператор ввода в файл

Класс **Salve –** класс раствор, наследуется от Liquid

Описание полей класса:

string application\_method – способ применения

Описание методов класса:

Salve() – конструктор

Salve(const Salve& copy) – конструктор копирования

Salve(Salve&& copy) noexcept – конструктор перемещения

~Salve() – деструктор

string get\_application\_method() – получить способ применения

void set\_application\_method(string) – установить способ применения

bool set\_something() – установить по выбранному полю

friend bool equ(Salve&, Salve&) – эквиволентвы ли объекты

friend bool compare(Salve&, Salve&, int&) – сравнивает объекты по выбранному полю

friend bool operator==(Salve&, Salve&) – перегрузка оператора ==

Salve& operator=(const Salve&) – перегрузка оператора =

friend ostream& operator<<(ostream&, Salve&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Salve&) – перегрузка оператор ввода

friend ofstream& operator<<(ofstream&, Salve&) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Salve&) – перегрузка оператор ввода в файл

void print\_table() – вывод шапки таблицы

Класс **Solutions –** класс мазь, наследуется от Liquid

Описание полей класса:

string type\_using – способ нанесения

Описание методов класса:

Solutions() – конструктор

Solutions(const Solutions& copy) – конструктор копирования

Solutions(Solutions&& copy) noexcept – конструктор перемещения

~Solutions() – деструктор

string get\_type\_using() – получить способ нанесения

void set\_type\_using(string) – установить способ нанесения

bool set\_something() – установить по выбранному полю

friend bool equ(Solutions&, Solutions&) – эквиволентвы ли объекты

friend bool compare(Solutions&, Solutions&, int&) – сравнивает объекты по выбранному полю

friend bool operator==(Solutions&, Solutions&) – перегрузка оператора ==

Solutions& operator=(const Solutions&) – перегрузка оператора =

friend ostream& operator<<(ostream&, Solutions&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Solutions&) – перегрузка оператор ввода

friend ofstream& operator<<(ofstream&, Solutions&) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Solutions&) – перегрузка оператор ввода в файл

void print\_table() – вывод шапки таблицы

Класс **Powder –** класс порошок, наследуется от Solid

Описание полей класса:

string environment – жидкость для растворения

Описание методов класса:

Powder() – конструктор

Powder(const Powder& copy) – конструктор копирования

Powder(Powder&& copy) noexcept – конструктор перемещения

~Powder() – деструктор

string get\_environment() – получить жидкость для растворения

void set\_environment(string) – установить жидкость для растворения

bool set\_something() – установить по выбранному полю

friend bool equ(Powder&, Powder&) – эквиволентвы ли объекты

friend bool compare(Powder&, Powder&, int&) – сравнивает объекты по выбранному полю

friend bool operator==(Powder&, Powder&) – перегрузка оператора ==

Powder& operator=(const Powder&) – перегрузка оператора =

friend ostream& operator<<(ostream&, Powder&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Powder&) – перегрузка оператор ввода

friend ofstream& operator<<(ofstream& fs, Powder&) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Powder&) – перегрузка оператор ввода в файл

void print\_table() – вывод шапки таблицы

Класс **Tablets –** класс таблетки, наследуется от Solid

Описание полей класса:

int kol\_package – количество в упаковке

Описание методов класса:

Tablets() – конструктор

Tablets(const Tablets& copy) – конструктор копирования

Tablets(Tablets&& copy) noexcept – конструктор перемещения

~Tablets() – деструктор

int get\_kol\_package() – получить количество в упаковке

void set\_kol\_package(int) – установить количество в упаковке

bool set\_something() – установить по выбранному полю

friend bool equ(Tablets&, Tablets&) – эквиволентвы ли объекты

friend bool compare(Tablets&, Tablets&, int&) – сравнивает объекты по выбранному полю

friend bool operator==(Tablets&, Tablets&) – перегрузка оператора ==

Tablets& operator=(const Tablets&) – перегрузка оператора =

friend ostream& operator<<(ostream&, Tablets&) – перегрузка оператор вывода

friend istream& operator>>(istream&, Tablets&) – перегрузка оператор ввода

friend ofstream& operator<<( ofstream&, Tablets&) – перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Tablets&) – перегрузка оператор ввода в файл

void print\_table() – вывод шапки таблицы

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

# 4.1 Разработка схем алгоритмов

Метод geve\_file() сохраняет кольцо из объектов класса T в файл. Схема метода geve\_file() показана в приложении Б.

Метод fiend() осуществляет поиск элементов контейнера.

Схема метода fiend() показана в приложении В.

# 4.2 Разработка алгоритмов

Метод delete\_ring () шаблонного класса Interface

(меню удаления препаратов):

Шаг 1: начало.

Шаг 2: входные данные:

Ring<T>& test – контейнер, для удаления из него элементов.

промежуточные данные:

T information – данные по которым будем искать элементы.

Ring<T> temp – временное кольцо, в котором будем хранить элементы для удаления.

Algorithm alg – алгоритм для вызова поиска элементов

выходные данные:

Ring<T>& test – кольцо с удалёнными элементами.

Шаг 3: вызов метода past(test), который копируем элементы из контейнера test в temp.

Шаг 4: очищает консоль.

Шаг 5: вызов метода loop(), который выводит контейнер temp на экран.

Шаг 6: вызов метода set\_something(), который вводит в information параметр для поиска, если не выбран параметр для поиска, переход на шаг 12, иначе переход на шаг 7.

Шаг 7: вызов метод find(temp, information), который оставит в контейнере temp только те элементы, которые равны information.

Шаг 8: вывод сообщения на экран "Нажмите ENTER чтобы применить удаление, или любую другую клавишу для продолжения" с переходом на следующую строку.

Шаг 9: если был нажат ENTER, переход на шаг 10, иначе переход на шаг 12.

Шаг 10: вызов метода del(temp), который удаляет все элементы из test равные элементам temp.

Шаг 11: переход на шаг 13.

Шаг 12: переход на шаг 4.

Шаг 13: конец.

Метод sort\_Ring() шаблонного класса Algorithm (сортирует кольцо по введённому параметру)

Шаг 1: начало.

Шаг 2: входные данные:

Ring<T>& test – кольцо, для сортировки.

промежуточные данные:

int menu – номер поля для сравнения(1 – название, 2 – цена, …).

T temp – временный объект для смены мест объектов.

Iterator it – первый итератор.

Iterator jt – второй итератор.

выходные данные:

Ring<T>& test – отсортированное кольцо.

Шаг 3: установить оба итератора на начало кольца ring с помощью метода Begin().

Шаг 4: it = it – 1, перемещение инератора it на предыдущий элемент.

Шаг 5: it = it + 1, перемещение инератора it на следующий элемент.

Шаг 6: jt = jt + 1, перемещение итератора jt на следующий элемент.

Шаг 7: вызов функции compare(\*it, \*jt, menu)(возвращает 1 если данные итератора it больше jt по полю menu), если равно 1, переход на шаг 8, иначе переход на шаг.

Шаг 8: temp = \*it(данные итератора it присвоить объекту temp), \*it = \*jt(данные итератора it заменяются данными итератора jt), \*jt = temp(заменить данные итератора jt на данные объекта temp).

Шаг 9: если итератор jt указывает на последний элемент ring(проверить через метод End(), который возвращает последний элемент ring), переход на шаг 10, иначе переход на шаг 6.

Шаг 10: если итератор it указывает на последний элемент ring(проверить через метод End(), который возвращает последний элемент ring), переход на шаг 11, иначе переход на шаг 5.

Шаг 11: конец.

# 5 РЕЗУЛЬТАТ РАБОТЫ

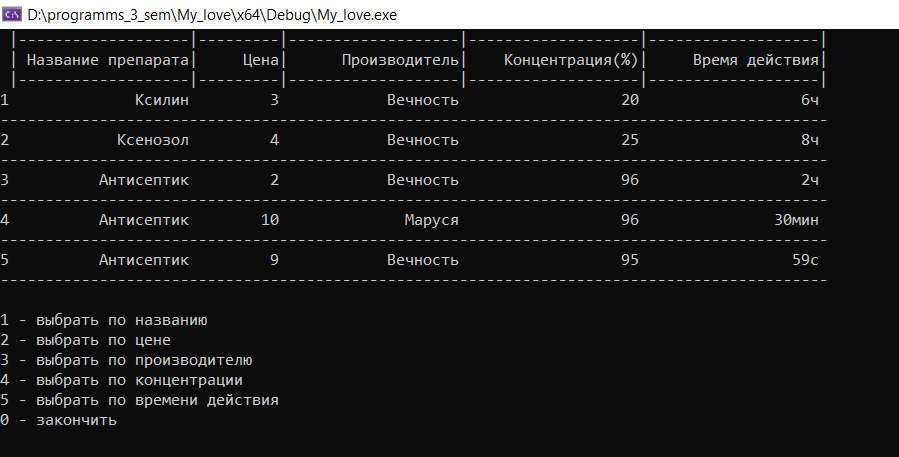


Рисунок 1 – Для примера взяли данные из файла с аэрозолями

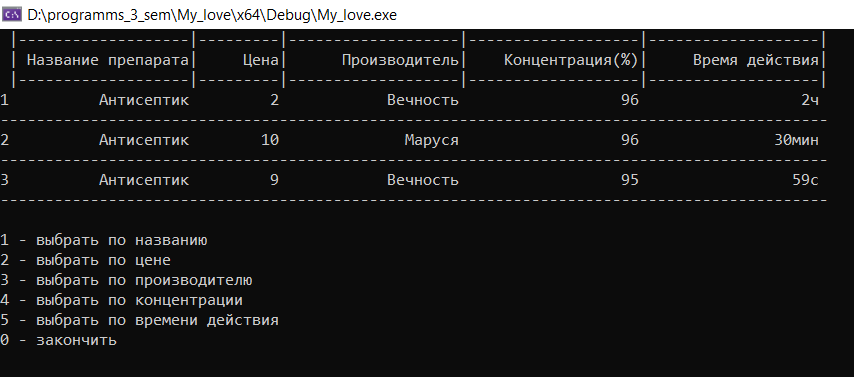


Рисунок 2 – Среди аэрозолей выбрали Антисептики

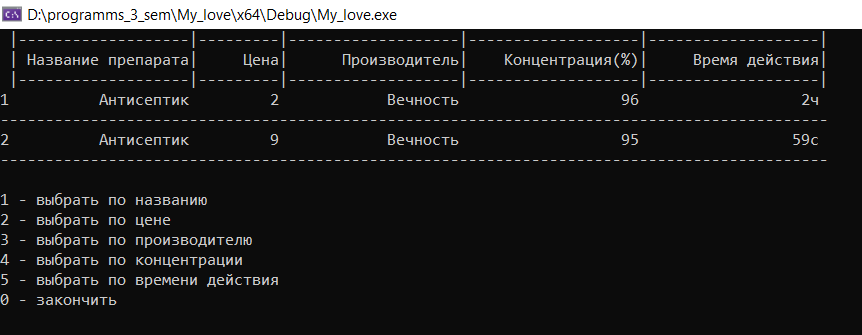


Рисунок 3 – Среди Антисептиков выбрали с производителем Вечность

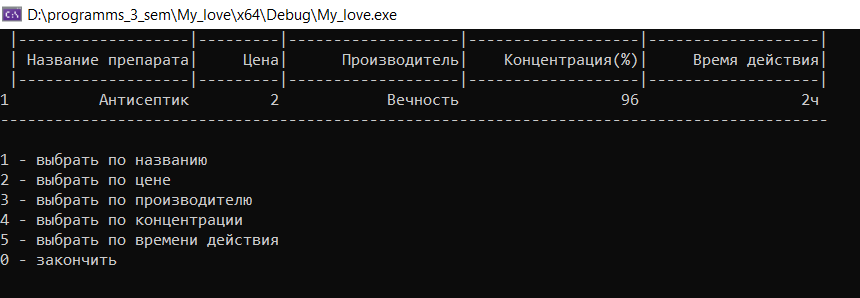


Рисунок 4 – Среди Антисептиков с производителем Вечность выбрали с ценой 2

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта были выполнены первоначально заданные цели, а именно реализована программа авто паркинга с удобным интерфейсом. Так же были реализованы следующие алгоритмы: отмена последнего действия, сортировка и поиск объектов разных классов по определённым полям.

Информационные системы ежедневно встречаются в нашей жизни – дома, на работе, на улице, в транспорте. И сегодня только представить жизнь без таких систем довольно сложно! На предприятиях активно используются информационные системы управления. С помощью таких систем жизнь человечества существенно облегчается, это огромная и неоценимая помощь, ведь один или несколько человек не могут держать в голове или же на бумажных носителях данные, которые в компьютере занимают терабайты оперативной памяти. Однако просто хранить такую информацию – это мало, ее нужно систематизировать и адаптировать для удобного использования.

Системные требования:

– ОС – Windows 10

– Среда разработки – Microsoft Visual Studio 2022

– Минимальные системные требования:

– Занимаемая память процессора – 512 мб

– Время выполнения – 2 c

# СПИСОК ЛИТЕРАТУРЫ

1. Объектно-ориентированное программирование на языке С++: учеб. пособие / Ю. А. Луцик, В. Н. Комличенко. – Минск : БГУИР, 2008.

2. Конструирование программ и языки программирования: метод. указания по курсовому проектированию для студ. спец. I-40 02 01 “Вычислительные машины, системы и сети” для всех форм обуч. / сост. А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. – Минск : БГУИР, 2009.

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Диаграмма классов

# ПРИЛОЖЕНИЕ Б

*(обязательное)*

Схема метода geve\_file()

# ПРИЛОЖЕНИЕ В

*(обязательное)*

Схема метода find()

# ПРИЛОЖЕНИЕ Г

*(обязательное)*

Код программы

# ПРИЛОЖЕНИЕ Д

*(обязательное)*

Ведомость документов