Файл My\_love.cpp содержащий main()

#pragma once

#include <iostream>

#include <windows.h>

#include <locale.h>

#include "Interface.h"

using namespace std;

int main() //функция main

{

setlocale(LC\_ALL, "ru"); //русификатор

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Interface qwerty;

qwerty.start(); //вызов меню работы с препаратами

return 0;

}

Файл Medication.h содержащий объявления класса Medication

#pragma once

#include <string>

#include<iostream>

#include<fstream>

#include "Exception\_input.h"

using namespace std;

class Medication //класс лекарственный препарат

{

string name; //название

float price; //стоимость

string manufacturer; //производитель

public:

Medication(); //конструктор

Medication(const Medication&); //конструктор копирования

Medication(Medication&&) noexcept; //конструктор перемещения

~Medication(); //деструктор

string get\_name(); //получить название

float get\_price(); //получить цену

string get\_manufacturer(); //получить производителя

void set\_name(string); //установить название

void set\_price(float); //установить цену

void set\_manufacturer(string); //установить производителя

friend bool equ(Medication&, Medication&); //эквиволентвы ли объекты

friend bool operator==(Medication&, Medication&); //перегрузка оператора ==

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

friend ostream& operator<<(ostream& os, Medication&); //перегрузка оператор вывода

friend istream& operator>>(istream& is, Medication&); //перегрузка оператор ввода

friend ofstream& operator<<(ofstream& fs, Medication&); //перегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream& fs, Medication&); //перегрузка оператор ввода из файла

};

Файл Medication.cpp содержащий реализацию класса Medication

#pragma once

#include "Medication.h"

using namespace std;

Medication::Medication()

{

name = "";

price = 0;

manufacturer = "";

}

Medication::Medication(const Medication& copy)

{

name = copy.name;

price = copy.price;

manufacturer = copy.manufacturer;

}

Medication::Medication(Medication&& copy) noexcept

{

name = copy.name;

price = copy.price;

manufacturer = copy.manufacturer;

}

Medication::~Medication()

{

}

string Medication::get\_name()

{

return name;

}

float Medication::get\_price()

{

return price;

}

string Medication::get\_manufacturer()

{

return manufacturer;

}

void Medication::set\_name(string inf)

{

name = inf;

}

void Medication::set\_price(float inf)

{

price = inf;

}

void Medication::set\_manufacturer(string inf)

{

manufacturer = inf;

}

bool equ(Medication& first\_object, Medication& second\_object)

{

if (first\_object.get\_name() == second\_object.get\_name() &&

first\_object.get\_price() == second\_object.get\_price() &&

first\_object.get\_manufacturer() == second\_object.get\_manufacturer())

return 1;

else

return 0;

}

bool operator==(Medication& first\_object, Medication& second\_object)

{

if (second\_object.get\_name().length())

if (first\_object.get\_name() != second\_object.get\_name())

return false;

if (second\_object.get\_price())

if (first\_object.get\_price() != second\_object.get\_price())

return false;

if (second\_object.get\_manufacturer().length())

if (first\_object.get\_manufacturer() != second\_object.get\_manufacturer())

return false;

return true;

}

Medication& Medication::operator=(const Medication& equ\_object)

{

if (this != &equ\_object)

{

if (equ\_object.name.length())

this->name = equ\_object.name;

if (equ\_object.price)

this->price = equ\_object.price;

if (equ\_object.manufacturer.length())

this->manufacturer = equ\_object.manufacturer;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Medication& obj)

{

os.width(20);

os << obj.name;

os.width(10);

os << obj.price;

os.width(20);

os << obj.manufacturer;

return os;

}

std::istream& operator>>(std::istream& is, Medication& obj)

{

char BUFF[2048];

std::cout << "Введите название препарата\n";

obj.name = input\_ru\_string(is);

std::cout << "Введите цену\n";

obj.price = input\_price(is);

std::cout << "Введите производителя\n";

obj.manufacturer = input\_ru\_string(is);

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Medication& obj)

{

fs << obj.name << " " << obj.price << " " << obj.manufacturer;

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Medication& obj)

{

fs >> obj.name;

fs >> obj.price;

fs >> obj.manufacturer;

return fs;

}

Файл Aerosols.h содержащий объявления класса Aerosols

#pragma once

#include "Medication.h"

using namespace std;

class Aerosols : //класс аэрозоли

public Medication

{

float concetration; //концентрация

std::string time; //время действия препарата

public:

Aerosols(); //конструктор

Aerosols(const Aerosols&); //конструктор копирования

Aerosols(Aerosols&&) noexcept; //конструктор перемещения

~Aerosols(); //деструктор

std::string get\_time(); //получить время действия препарата

float get\_concentration(); //получить концентрацию

std::string get\_something(char); //получить выбранное поле

void set\_time(std::string); //установить время действия препарата

void set\_concentration(float); //установить концентрацию

bool set\_something(); //установить выбранное поле

friend bool equ(Aerosols&, Aerosols&); //эквиволентвы ли объекты

friend bool compare(Aerosols&, Aerosols&, int&); //сравнивает объекты по выбранному полю

friend bool operator==(Aerosols&, Aerosols&); //перегрузка оператора ==

Aerosols& operator=(const Aerosols&); //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

};

Файл Aerosols.cpp содержащий реализацию класса Aerosols

#pragma once

#include "Aerosols.h"

Aerosols::Aerosols()

{

concetration = 0;

time = "";

}

Aerosols::Aerosols(const Aerosols& copy)

{

(Medication&)\*this = (Medication&)copy;

concetration = copy.concetration;

time = copy.time;

}

Aerosols::Aerosols(Aerosols&& copy) noexcept

{

(Medication&)\*this = (Medication&)copy;

concetration = copy.concetration;

time = copy.time;

}

Aerosols::~Aerosols()

{

}

std::string Aerosols::get\_time()

{

return time;

}

float Aerosols::get\_concentration()

{

return concetration;

}

std::string Aerosols::get\_something(char menu)

{

bool iteration = true;

do

{

switch (menu)

{

case '1':

{

return get\_name();

break;

}

case '2':

{

return get\_manufacturer();

break;

}

case '3':

{

std::string temp\_string;

temp\_string = std::to\_string(get\_price());

return temp\_string;

break;

}

case '4':

{

std::string temp\_string;

temp\_string = std::to\_string(get\_concentration());

return temp\_string;

break;

}

case '5':

{

return get\_time();

break;

}

default:

break;

}

} while (iteration);

}

void Aerosols::set\_time(std::string inf)

{

time = inf;

}

void Aerosols::set\_concentration(float inf)

{

concetration = inf;

}

bool Aerosols::set\_something()

{

std::cout << "1 - выбрать по названию\n" << "2 - выбрать по цене\n" << "3 - выбрать по производителю\n"

<< "4 - выбрать по концентрации\n" << "5 - выбрать по времени действия\n" << "0 - закончить\n";

rewind(stdin);

do

{

switch (input\_int(std::cin))

{

case 1:

{

std::cout << "Введите название\n";

set\_name(input\_ru\_string(std::cin));

return 1;

break;

}

case 2:

{

std::cout << "Введите цену\n";

set\_price(input\_price(std::cin));

return 1;

break;

}

case 3:

{

std::cout << "Введите производителя\n";

set\_manufacturer(input\_ru\_string(std::cin));

return 1;

break;

}

case 4:

{

std::cout << "Введите концентрацию препарата\n";

set\_concentration(input\_int(std::cin));

return 1;

break;

}

case 5:

{

std::cout << "Введите время работы препарата\n";

set\_time(input\_dimension(std::cin, "время"));

return 1;

break;

}

case 0:

{

return 0;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

break;

}

}

} while (true);

}

bool equ(Aerosols& first\_object, Aerosols& second\_object)

{

if ((Medication&)first\_object == (Medication&)second\_object &&

first\_object.get\_time() == second\_object.get\_time() &&

first\_object.get\_concentration() == second\_object.get\_concentration())

return 1;

else

return 0;

}

bool compare(Aerosols& first\_obj, Aerosols& second\_obj, int& menu)

{

do

{

if (menu == 0)

{

std::cout << "1 - сравнивать по названию\n" << "2 - сравнивать по цене\n" << "3 - сравнивать по производителю\n"

<< "4 - сравнивать по концентрации\n" << "5 - сравнивать по времени действия\n";

menu = input\_int(std::cin);

}

switch (menu)

{

case 1:

{

if (first\_obj.get\_name() > second\_obj.get\_name())

return 1;

break;

}

case 2:

{

if (first\_obj.get\_price() > second\_obj.get\_price())

return 1;

break;

}

case 3:

{

if (first\_obj.get\_manufacturer() > second\_obj.get\_manufacturer())

return 1;

break;

}

case 4:

{

if (first\_obj.get\_concentration() > second\_obj.get\_concentration())

return 1;

break;

}

case 5:

{

if (first\_obj.get\_time()[first\_obj.get\_time().length() - 1] == 'ч'

&& (second\_obj.get\_time()[second\_obj.get\_time().length() - 1] == 'н'

|| second\_obj.get\_time()[second\_obj.get\_time().length() - 1] == 'с'))

return 1;

if (first\_obj.get\_time()[first\_obj.get\_time().length() - 1] == 'н'

&& second\_obj.get\_time()[second\_obj.get\_time().length() - 1] == 'с')

return 1;

if (first\_obj.get\_time()[first\_obj.get\_time().length() - 1] == 'с'

&& (second\_obj.get\_time()[second\_obj.get\_time().length() - 1] == 'н'

|| second\_obj.get\_time()[second\_obj.get\_time().length() - 1] == 'ч'))

return 0;

if (first\_obj.get\_time()[first\_obj.get\_time().length() - 1] == 'н'

&& second\_obj.get\_time()[second\_obj.get\_time().length()] == 'ч')

return 0;

if (first\_obj.get\_time() > second\_obj.get\_time())

return 1;

break;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

menu = 0;

break;

}

}

return 0;

} while (true);

}

bool operator==(Aerosols& first\_object, Aerosols& second\_object)

{

if (!((Medication&)first\_object == (Medication&)second\_object))

return false;

if (second\_object.get\_time().length())

if (first\_object.get\_time() != second\_object.get\_time())

return false;

if (second\_object.get\_concentration())

if (first\_object.get\_concentration() != second\_object.get\_concentration())

return false;

return true;

}

Aerosols& Aerosols::operator=(const Aerosols& equ\_object)

{

if (this != &equ\_object)

{

(Medication&)\*this = (Medication&)equ\_object;

if (equ\_object.concetration)

this->concetration = equ\_object.concetration;

if(equ\_object.time.length())

this->time = equ\_object.time;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Aerosols& obj)

{

os << (Medication&)obj;

os.width(20);

os << obj.concetration;

os.width(20);

os << obj.time;

os << '\n';

os.fill('-');

os.width(93);

os << '\n';

os.fill(' ');

return os;

}

std::istream& operator>>(std::istream& is, Aerosols& obj)

{

char BUFF[2048];

is >> (Medication&)obj;

std::cout << "Введите концентрацию препарата\n";

obj.concetration = input\_int(is);

std::cout << "Введите время работы препарата\n";

rewind(stdin);

obj.time = input\_dimension(is, "время");

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Aerosols& obj)

{

fs << (Medication&)obj << " " << obj.concetration << " " << obj.time << '\n';

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Aerosols& obj)

{

fs >> (Medication&)obj;

fs >> obj.concetration;

fs >> obj.time;

return fs;

}

void Aerosols::print\_table()

{

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout << " |";

std::cout.fill(' ');

std::cout.width(19);

std::cout << "Название препарата";

std::cout << '|';

std::cout.width(9);

std::cout << "Цена";

std::cout << '|';

std::cout.width(19);

std::cout << "Производитель";

std::cout << '|';

std::cout.width(19);

std::cout << "Концентрация(%)";

std::cout << '|';

std::cout.width(19);

std::cout << "Время действия";

std::cout << "|\n";

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout.fill(' ');

}

Файл Solid.h содержащий объявления класса Solid

#pragma once

#include "Medication.h"

class Solid : //класс твёрдые

public Medication

{

string weight; //масса

public:

Solid(); //конструктор

Solid(const Solid&); //конструктор копирования

Solid(Solid&&) 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

};

Файл Solid.cpp содержащий реализацию класса Solid

#pragma once

#include "Solid.h"

Solid::Solid()

{

weight = '\0';

}

Solid::Solid(const Solid& copy)

{

(Medication&)\*this = (Medication&)copy;

weight = copy.weight;

}

Solid::Solid(Solid&& copy) noexcept

{

(Medication&)\*this = (Medication&)copy;

weight = copy.weight;

}

Solid::~Solid()

{

}

std::string Solid::get\_weight()

{

return weight;

}

void Solid::set\_weight(std::string inf)

{

weight = inf;

}

bool equ(Solid& first\_object, Solid& second\_object)

{

if ((Medication&)first\_object == (Medication&)second\_object &&

first\_object.get\_weight() == second\_object.get\_weight())

return 1;

else

return 0;

}

bool operator==(Solid& first\_object, Solid& second\_object)

{

if (!((Medication&)first\_object == (Medication&)second\_object))

return false;

if (second\_object.get\_weight().length() != 1)

if (first\_object.get\_weight() != second\_object.get\_weight())

return false;

return true;

}

Solid& Solid::operator=(const Solid& equ\_object)

{

if (this != &equ\_object)

{

(Medication&)\*this = (Medication&)equ\_object;

if(equ\_object.weight.length() != 1)

this->weight = equ\_object.weight;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Solid& obj)

{

os << (Medication&)obj;

os.width(20);

os << obj.weight;

return os;

}

std::istream& operator>>(std::istream& is, Solid& obj)

{

char BUFF[2048];

is >> (Medication&)obj;

std::cout << "Введите массу препарата\n";

obj.weight = input\_dimension(is, "масса");

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Solid& obj)

{

fs << (Medication&)obj << " " << obj.weight;

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Solid& obj)

{

fs >> (Medication&)obj;

fs >> obj.weight;

return fs;

}

Файл Liquid.h содержащий объявления класса Liquid

#pragma once

#include "Medication.h"

class Liquid : //класс жидкие

public Medication

{

string volume; //объём

float concentration; //концентрация

public:

Liquid(); //конструктор

Liquid(const Liquid&); //конструктор копирования

Liquid(Liquid&&) 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

};

Файл Liquid.cpp содержащий реализацию класса Liquid

#pragma once

#include "Liquid.h"

Liquid::Liquid()

{

concentration = 0;

volume = "";

}

Liquid::Liquid(const Liquid& copy)

{

(Medication&)\*this = (Medication&)copy;

concentration = copy.concentration;

volume = copy.volume;

}

Liquid::Liquid(Liquid&& copy) noexcept

{

(Medication&)\*this = (Medication&)copy;

concentration = copy.concentration;

volume = copy.volume;

}

Liquid::~Liquid()

{

}

std::string Liquid::get\_volume()

{

return volume;

}

float Liquid::get\_concentration()

{

return concentration;

}

void Liquid::set\_volume(std::string inf)

{

volume = inf;

}

void Liquid::set\_concentration(float inf)

{

concentration = inf;

}

bool equ(Liquid& first\_object, Liquid& second\_object)

{

if ((Medication&)first\_object == (Medication&)second\_object &&

first\_object.get\_volume() == second\_object.get\_volume() &&

first\_object.get\_concentration() == second\_object.get\_concentration())

return 1;

else

return 0;

}

bool operator==(Liquid& first\_object, Liquid& second\_object)

{

if ((Medication&)first\_object == (Medication&)second\_object)

return true;

if (second\_object.get\_volume().length())

if (first\_object.get\_volume() != second\_object.get\_volume())

return false;

if (second\_object.get\_concentration())

if (first\_object.get\_concentration() != second\_object.get\_concentration())

return false;

return true;

}

Liquid& Liquid::operator=(const Liquid& equ\_object)

{

if (this != &equ\_object)

{

(Medication&)\*this = (Medication&)equ\_object;

if (equ\_object.concentration)

this->concentration = equ\_object.concentration;

if(equ\_object.volume.length())

this->volume = equ\_object.volume;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Liquid& obj)

{

os << (Medication&)obj;

os.width(20);

os << obj.concentration;

os.width(20);

os << obj.volume;

return os;

}

std::istream& operator>>(std::istream& is, Liquid& obj)

{

char BUFF[2048];

is >> (Medication&)obj;

std::cout << "Введите концентрацию препарата\n";

obj.concentration = input\_int(is);

std::cout << "Введите объём препарата\n";

obj.volume = input\_dimension(is, "объём");

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Liquid& obj)

{

fs << (Medication&)obj << " " << obj.concentration << " " << obj.volume;

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Liquid& obj)

{

fs >> (Medication&)obj;

fs >> obj.concentration;

fs >> obj.volume;

return fs;

}

Файл Powder.h содержащий объявления класса Powder

#pragma once

#include "Solid.h"

class Powder : //класс порошок

public Solid

{

string environment; //жидкость для растворения

public:

Powder(); //конструктор

Powder(const Powder&); //конструктор копирования

Powder(Powder&&) 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&, Powder&);//ерегрузка оператор вывода в файл

friend ifstream& operator>>(ifstream&, Powder&);//регрузка оператор ввода в файл

void print\_table(); //вывод шапки таблицы

};

Файл Powder.cpp содержащий реализацию класса Powder

#pragma once

#include "Powder.h"

Powder::Powder()

{

environment = "";

}

Powder::Powder(const Powder& copy)

{

(Solid&)\*this = (Solid&)copy;

environment = copy.environment;

}

Powder::Powder(Powder&& copy) noexcept

{

(Solid&)\*this = (Solid&)copy;

environment = copy.environment;

}

Powder::~Powder()

{

}

std::string Powder::get\_environment()

{

return environment;

}

void Powder::set\_environment(std::string inf)

{

environment = inf;

}

bool Powder::set\_something()

{

std::cout << "1 - выбрать по названию\n" << "2 - выбрать по цене\n" << "3 - выбрать по производителю\n"

<< "4 - выбрать по массе\n" << "5 - выбрать по среде для растворения\n" << "0 - закончить\n";

rewind(stdin);

do

{

switch (input\_int(std::cin))

{

case 1:

{

std::cout << "Введите название\n";

set\_name(input\_ru\_string(std::cin));

return 1;

break;

}

case 2:

{

std::cout << "Введите цену\n";

set\_price(input\_price(std::cin));

return 1;

break;

}

case 3:

{

std::cout << "Введите производителя\n";

set\_manufacturer(input\_ru\_string(std::cin));

return 1;

break;

}

case 4:

{

std::cout << "Введите массу\n";

set\_weight(input\_dimension(std::cin, "масса"));

return 1;

break;

}

case 5:

{

std::cout << "Введите жидкость для растворения\n";

set\_environment(input\_ru\_string(std::cin));

return 1;

break;

}

case 0:

{

return 0;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

break;

}

}

} while (true);

}

bool equ(Powder& first\_object, Powder& second\_object)

{

if ((Solid&)first\_object == (Solid&)second\_object &&

first\_object.get\_environment() == second\_object.get\_environment())

return 1;

else

return 0;

}

bool compare(Powder& first\_obj, Powder& second\_obj, int& menu)

{

do

{

if (menu == 0)

{

std::cout << "1 - сравнивать по названию\n" << "2 - сравнивать по цене\n" << "3 - сравнивать по производителю\n"

<< "4 - сравнивать по массе\n" << "5 - сравнивать по жидкость для растворения\n";

menu = input\_int(std::cin);

}

switch (menu)

{

case 1:

{

if (first\_obj.get\_name() > second\_obj.get\_name())

return 1;

break;

}

case 2:

{

if (first\_obj.get\_price() > second\_obj.get\_price())

return 1;

break;

}

case 3:

{

if (first\_obj.get\_manufacturer() > second\_obj.get\_manufacturer())

return 1;

break;

}

case 4:

{

if (first\_obj.get\_weight() > second\_obj.get\_weight())

return 1;

break;

}

case 5:

{

if (first\_obj.get\_environment() > second\_obj.get\_environment())

return 1;

break;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

menu = 0;

break;

}

}

return 0;

} while (true);

}

bool operator==(Powder& first\_object, Powder& second\_object)

{

if (!((Solid&)first\_object == (Solid&)second\_object))

return false;

if (second\_object.get\_environment().length())

if (first\_object.get\_environment() != second\_object.get\_environment())

return false;

return true;

}

Powder& Powder::operator=(const Powder& equ\_object)

{

if (this != &equ\_object)

{

(Solid&)\*this = (Solid&)equ\_object;

if (equ\_object.environment.length())

this->environment = equ\_object.environment;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Powder& obj)

{

os << (Solid&)obj;

os.width(30);

os << obj.environment;

os << '\n';

os.fill('-');

os.width(103);

os << '\n';

os.fill(' ');

return os;

}

std::istream& operator>>(std::istream& is, Powder& obj)

{

char BUFF[2048];

is >> (Solid&)obj;

std::cout << "Введите жидкость для растворения\n";

obj.environment = input\_ru\_string(is);

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Powder& obj)

{

fs << (Solid&)obj << " " << obj.environment;

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Powder& obj)

{

fs >> (Solid&)obj;

fs >> obj.environment;

return fs;

}

void Powder::print\_table()

{

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(30);

std::cout << '|';

std::cout << '\n';

std::cout << " |";

std::cout.fill(' ');

std::cout.width(19);

std::cout << "Название препарата";

std::cout << '|';

std::cout.width(9);

std::cout << "Цена";

std::cout << '|';

std::cout.width(19);

std::cout << "Производитель";

std::cout << '|';

std::cout.width(19);

std::cout << "Масса";

std::cout << '|';

std::cout.width(29);

std::cout << "Жидкость для растворения";

std::cout << "|\n";

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(30);

std::cout << '|';

std::cout << '\n';

std::cout.fill(' ');

}

Файл Tablets.h содержащий объявления класса Tablets

#pragma once

#include "Solid.h"

class Tablets : //класс таблетки

public Solid

{

int kol\_package; //количество в упаковке

public:

Tablets(); //конструктор

Tablets(const Tablets&); //конструктор копирования

Tablets(Tablets&&) 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

};

Файл Tablets.cpp содержащий реализациию класса Tablets

#pragma once

#include "Tablets.h"

Tablets::Tablets()

{

kol\_package = 0;

}

Tablets::Tablets(const Tablets& copy)

{

(Solid&)\*this = (Solid&)copy;

kol\_package = copy.kol\_package;

}

Tablets::Tablets(Tablets&& copy) noexcept

{

(Solid&)\*this = (Solid&)copy;

kol\_package = copy.kol\_package;

}

Tablets::~Tablets()

{

}

int Tablets::get\_kol\_package()

{

return kol\_package;

}

void Tablets::set\_kol\_package(int inf)

{

kol\_package = inf;

}

bool Tablets::set\_something()

{

std::cout << "1 - выбрать по названию\n" << "2 - выбрать по цене\n" << "3 - выбрать по производителю\n"

<< "4 - выбрать по массе\n" << "5 - выбрать по количеству в упаковке\n" << "0 - закончить\n";

rewind(stdin);

do

{

switch (input\_int(std::cin))

{

case 1:

{

std::cout << "Введите название\n";

set\_name(input\_ru\_string(std::cin));

return 1;

break;

}

case 2:

{

std::cout << "Введите цену\n";

set\_price(input\_price(std::cin));

return 1;

break;

}

case 3:

{

std::cout << "Введите производителя\n";

set\_manufacturer(input\_ru\_string(std::cin));

return 1;

break;

}

case 4:

{

std::cout << "Введите массу\n";

set\_weight(input\_dimension(std::cin, "масса"));

return 1;

break;

}

case 5:

{

std::cout << "Введите количество таблеток в упаковке\n";

set\_kol\_package(input\_int(std::cin));

return 1;

break;

}

case 0:

{

return 0;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

break;

}

}

} while (true);

}

bool equ(Tablets& first\_object, Tablets& second\_object)

{

if ((Solid&)first\_object == (Solid&)second\_object &&

first\_object.get\_kol\_package() == second\_object.get\_kol\_package())

return 1;

else

return 0;

}

bool compare(Tablets& first\_obj, Tablets& second\_obj, int& menu)

{

do

{

if (menu == 0)

{

std::cout << "1 - сравнивать по названию\n" << "2 - сравнивать по цене\n" << "3 - сравнивать по производителю\n"

<< "4 - сравнивать по массе\n" << "5 - сравнивать по количеству таблеток в упаковке\n";

int menu = input\_int(std::cin);

}

switch (menu)

{

case 1:

{

if (first\_obj.get\_name() > second\_obj.get\_name())

return 1;

break;

}

case 2:

{

if (first\_obj.get\_price() > second\_obj.get\_price())

return 1;

break;

}

case 3:

{

if (first\_obj.get\_manufacturer() > second\_obj.get\_manufacturer())

return 1;

break;

}

case 4:

{

if (first\_obj.get\_weight() > second\_obj.get\_weight())

return 1;

break;

}

case 5:

{

if (first\_obj.get\_kol\_package() > second\_obj.get\_kol\_package())

return 1;

break;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

menu = 0;

break;

}

}

return 0;

} while (true);

}

bool operator==(Tablets& first\_object, Tablets& second\_object)

{

if ((Solid&)first\_object == (Solid&)second\_object)

return true;

if (second\_object.get\_kol\_package())

if (first\_object.get\_kol\_package() != second\_object.get\_kol\_package())

return false;

return true;

}

Tablets& Tablets::operator=(const Tablets& equ\_object)

{

if (this != &equ\_object)

{

(Solid&)\*this = (Solid&)equ\_object;

if (equ\_object.kol\_package)

this->kol\_package = equ\_object.kol\_package;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Tablets& obj)

{

os << (Solid&)obj;

os.width(20);

os << obj.kol\_package;

os << '\n';

os.fill('-');

os.width(93);

os << '\n';

os.fill(' ');

return os;

}

std::istream& operator>>(std::istream& is, Tablets& obj)

{

char BUFF[2048];

is >> (Solid&)obj;

std::cout << "Введите количество таблеток в упаковке\n";

obj.kol\_package = input\_int(is);

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Tablets& obj)

{

fs << (Solid&)obj << " " << obj.kol\_package << '\n';

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Tablets& obj)

{

fs >> (Solid&)obj;

fs >> obj.kol\_package;

return fs;

}

void Tablets::print\_table()

{

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout << " |";

std::cout.fill(' ');

std::cout.width(19);

std::cout << "Название препарата";

std::cout << '|';

std::cout.width(9);

std::cout << "Цена";

std::cout << '|';

std::cout.width(19);

std::cout << "Производитель";

std::cout << '|';

std::cout.width(19);

std::cout << "Масса";

std::cout << '|';

std::cout.width(19);

std::cout << "Количество таблеток";

std::cout << "|\n";

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout.fill(' ');

}

Файл Solutions.h содержащий объявления класса Solutions

#pragma once

#include "Liquid.h"

class Solutions : //класс мазь

public Liquid

{

string type\_using; //способ нанесения

public:

Solutions(); //конструктор

Solutions(const Solutions&); //конструктор копирования

Solutions(Solutions&&) 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

};

Файл Solutions.cpp содержащий реализацию класса Solutions

#pragma once

#include "Solutions.h"

Solutions::Solutions()

{

type\_using = "";

}

Solutions::Solutions(const Solutions& copy)

{

(Liquid&)\*this = (Liquid&)copy;

type\_using = copy.type\_using;

}

Solutions::Solutions(Solutions&& copy) noexcept

{

(Liquid&)\*this = (Liquid&)copy;

type\_using = copy.type\_using;

}

Solutions::~Solutions()

{

}

std::string Solutions::get\_type\_using()

{

return type\_using;

}

void Solutions::set\_type\_using(std::string inf)

{

type\_using = inf;

}

bool Solutions::set\_something()

{

std::cout << "1 - выбрать по названию\n" << "2 - выбрать по цене\n" << "3 - выбрать по производителю\n"

<< "4 - выбрать по концентрации\n" << "5 - выбрать по объёму\n" << "6 - выбрать по типу использования\n" << "0 - закончить\n";

rewind(stdin);

do

{

switch (input\_int(std::cin))

{

case 1:

{

std::cout << "Введите название\n";

set\_name(input\_ru\_string(std::cin));

return 1;

break;

}

case 2:

{

std::cout << "Введите цену\n";

set\_price(input\_price(std::cin));

return 1;

break;

}

case 3:

{

std::cout << "Введите производителя\n";

set\_manufacturer(input\_ru\_string(std::cin));

return 1;

break;

}

case 4:

{

std::cout << "Введите концентрацию\n";

set\_concentration(input\_int(std::cin));

return 1;

break;

}

case 5:

{

std::cout << "Введите объём\n";

set\_volume(input\_dimension(std::cin, "объём"));

return 1;

break;

}

case 6:

{

std::cout << "Введите способ нанесения\n";

set\_type\_using(input\_ru\_string(std::cin));

return 1;

break;

}

case 0:

{

return 0;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 6\n";

system("pause");

break;

}

}

} while (true);

}

bool equ(Solutions& first\_object, Solutions& second\_object)

{

if ((Liquid&)first\_object == (Liquid&)second\_object &&

first\_object.get\_type\_using() == second\_object.get\_type\_using())

return 1;

else

return 0;

}

bool compare(Solutions& first\_obj, Solutions& second\_obj, int& menu)

{

do

{

if (menu == 0)

{

std::cout << "1 - сравнивать по названию\n" << "2 - сравнивать по цене\n" << "3 - сравнивать по производителю\n"

<< "4 - сравнивать по концентрации\n" << "5 - сравнивать по объёму\n" << "6 - сравнивать по способу нанесения\n";

int menu = input\_int(std::cin);

}

switch (menu)

{

case 1:

{

if (first\_obj.get\_name() > second\_obj.get\_name())

return 1;

break;

}

case 2:

{

if (first\_obj.get\_price() > second\_obj.get\_price())

return 1;

break;

}

case 3:

{

if (first\_obj.get\_manufacturer() > second\_obj.get\_manufacturer())

return 1;

break;

}

case 4:

{

if (first\_obj.get\_concentration() > second\_obj.get\_concentration())

return 1;

break;

}

case 5:

{

if (first\_obj.get\_volume() > second\_obj.get\_volume())

return 1;

break;

}

case 6:

{

if (first\_obj.get\_type\_using() > second\_obj.get\_type\_using())

return 1;

break;

}

{

std::cout << "такого пункта нет, введите число от 1 до 6\n";

system("pause");

menu = 0;

break;

}

}

return 0;

} while (true);

}

bool operator==(Solutions& first\_object, Solutions& second\_object)

{

if ((Liquid&)first\_object == (Liquid&)second\_object)

return true;

if (second\_object.get\_type\_using().length())

if (first\_object.get\_type\_using() != second\_object.get\_type\_using())

return false;

return true;

}

Solutions& Solutions::operator=(const Solutions& equ\_object)

{

if (this != &equ\_object)

{

(Liquid&)\*this = (Liquid&)equ\_object;

if (equ\_object.type\_using.length())

this->type\_using = equ\_object.type\_using;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Solutions& obj)

{

os << (Liquid&)obj;

os.width(20);

os << obj.type\_using;

os << '\n';

os.fill('-');

os.width(113);

os << '\n';

os.fill(' ');

return os;

}

std::istream& operator>>(std::istream& is, Solutions& obj)

{

char BUFF[2048];

is >> (Liquid&)obj;

std::cout << "Введите способ нанесения\n";

rewind(stdin);

obj.type\_using = input\_ru\_string(is);

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Solutions& obj)

{

fs << (Liquid&)obj << " " << obj.type\_using << '\n';

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Solutions& obj)

{

fs >> (Liquid&)obj;

fs >> obj.type\_using;

return fs;

}

void Solutions::print\_table()

{

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout << " |";

std::cout.fill(' ');

std::cout.width(19);

std::cout << "Название препарата";

std::cout << '|';

std::cout.width(9);

std::cout << "Цена";

std::cout << '|';

std::cout.width(19);

std::cout << "Производитель";

std::cout << '|';

std::cout.width(19);

std::cout << "Концентрация(%)";

std::cout << '|';

std::cout.width(19);

std::cout << "Объём";

std::cout << "|";

std::cout.width(19);

std::cout << "Cпособ нанесения";

std::cout << "|\n";

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout.fill(' ');

}

Файл Salve.h содержащий объявления класса Salve

#pragma once

#include "Liquid.h"

class Salve : //класс раствор

public Liquid

{

string application\_method; //способ применения

public:

Salve(); //конструктор

Salve(const Salve&); //конструктор копирования

Salve(Salve&&) 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& equ\_string); //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

};

Файл Salve.cpp содержащий реализацию класса Salve

#pragma once

#include "Salve.h"

Salve::Salve()

{

application\_method = "";

}

Salve::Salve(const Salve& copy)

{

(Liquid&)\*this = (Liquid&)copy;

application\_method = copy.application\_method;

}

Salve::Salve(Salve&& copy) noexcept

{

(Liquid&)\*this = (Liquid&)copy;

application\_method = copy.application\_method;

}

Salve::~Salve()

{

}

std::string Salve::get\_application\_method()

{

return application\_method;

}

void Salve::set\_application\_method(std::string inf)

{

application\_method = inf;

}

bool Salve::set\_something()

{

std::cout << "1 - выбрать по названию\n" << "2 - выбрать по цене\n" << "3 - выбрать по производителю\n"

<< "4 - выбрать по концентрации\n" << "5 - выбрать по объёму\n" << "6 - выбрать по способу применения\n" << "0 - закончить\n";

rewind(stdin);

do

{

switch (input\_int(std::cin))

{

case 1:

{

std::cout << "Введите название\n";

set\_name(input\_ru\_string(std::cin));

return 1;

break;

}

case 2:

{

std::cout << "Введите цену\n";

set\_price(input\_price(std::cin));

return 1;

break;

}

case 3:

{

std::cout << "Введите производителя\n";

set\_manufacturer(input\_ru\_string(std::cin));

return 1;

break;

}

case 4:

{

std::cout << "Введите концентрацию\n";

set\_concentration(input\_int(std::cin));

return 1;

break;

}

case 5:

{

std::cout << "Введите объём\n";

set\_volume(input\_dimension(std::cin, "объём"));

return 1;

break;

}

case 6:

{

std::cout << "Введите способ применения\n";

set\_application\_method(input\_ru\_string(std::cin));

return 1;

break;

}

case 0:

{

return 0;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

break;

}

}

} while (true);

}

bool equ(Salve& first\_object, Salve& second\_object)

{

if ((Liquid&)first\_object == (Liquid&)second\_object &&

first\_object.get\_application\_method() == second\_object.get\_application\_method())

return 1;

else

return 0;

}

bool compare(Salve& first\_obj, Salve& second\_obj, int& menu)

{

do

{

if (menu == 0)

{

std::cout << "1 - сравнивать по названию\n" << "2 - сравнивать по цене\n" << "3 - сравнивать по производителю\n"

<< "4 - сравнивать по концентрации\n" << "5 - сравнивать по объёму\n" << "6 - сравнивать по способу применения\n";

int menu = input\_int(std::cin);

}

switch (menu)

{

case 1:

{

if (first\_obj.get\_name() > second\_obj.get\_name())

return 1;

break;

}

case 2:

{

if (first\_obj.get\_price() > second\_obj.get\_price())

return 1;

break;

}

case 3:

{

if (first\_obj.get\_manufacturer() > second\_obj.get\_manufacturer())

return 1;

break;

}

case 4:

{

if (first\_obj.get\_concentration() > second\_obj.get\_concentration())

return 1;

break;

}

case 5:

{

if (first\_obj.get\_volume() > second\_obj.get\_volume())

return 1;

break;

}

case 6:

{

if (first\_obj.get\_application\_method() > second\_obj.get\_application\_method())

return 1;

break;

}

default:

{

std::cout << "такого пункта нет, введите число от 1 до 5\n";

system("pause");

menu = 0;

break;

}

}

return 0;

} while (true);

}

bool operator==(Salve& first\_object, Salve& second\_object)

{

if ((Liquid&)first\_object == (Liquid&)second\_object)

return true;

if (second\_object.get\_application\_method().length())

if (first\_object.get\_application\_method() != second\_object.get\_application\_method())

return false;

return true;

}

Salve& Salve::operator=(const Salve& equ\_object)

{

if (this != &equ\_object)

{

(Liquid&)\*this = (Liquid&)equ\_object;

if (equ\_object.application\_method.length())

this->application\_method = equ\_object.application\_method;

}

return \*this;

}

std::ostream& operator<<(std::ostream& os, Salve& obj)

{

os << (Liquid&)obj;

os.width(20);

os << obj.application\_method;

os << '\n';

os.fill('-');

os.width(113);

os << '\n';

os.fill(' ');

return os;

}

std::istream& operator>>(std::istream& is, Salve& obj)

{

char BUFF[2048];

is >> (Liquid&)obj;

std::cout << "Введите способ применения\n";

rewind(stdin);

obj.application\_method = input\_ru\_string(is);

return is;

}

std::ofstream& operator<<(std::ofstream& fs, Salve& obj)

{

fs << (Liquid&)obj << " " << obj.application\_method << '\n';

return fs;

}

std::ifstream& operator>>(std::ifstream& fs, Salve& obj)

{

fs >> (Liquid&)obj;

fs >> obj.application\_method;

return fs;

}

void Salve::print\_table()

{

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout << " |";

std::cout.fill(' ');

std::cout.width(19);

std::cout << "Название препарата";

std::cout << '|';

std::cout.width(9);

std::cout << "Цена";

std::cout << '|';

std::cout.width(19);

std::cout << "Производитель";

std::cout << '|';

std::cout.width(19);

std::cout << "Концентрация(%)";

std::cout << '|';

std::cout.width(19);

std::cout << "Объём";

std::cout << "|";

std::cout.width(19);

std::cout << "Способ применения";

std::cout << "|\n";

std::cout << " |";

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(10);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout.fill('-');

std::cout.width(20);

std::cout << '|';

std::cout << '\n';

std::cout.fill(' ');

}

Файл Exception.h содержащий объявления класса Exception

#pragma once

#include <iostream>

class Exception

{

char massage[80];

int code;

public:

Exception(int, const char[80]);

~Exception();

void what();

};

Файл Exception.cpp содержащий реализацию класса Exception

#pragma once

#include "Exception.h"

Exception::Exception(int input\_code, const char input\_massage[80])

{

strcpy\_s(massage, 80, input\_massage);

code = input\_code;

};

Exception::~Exception()

{

};

void Exception::what()

{

std::cout << "код ошибки " << code << ':' << massage << ", ";

};

Файл Exception\_input.h содержащий объявления класса Exception\_input

#pragma once

#include "Exception.h"

class Exception\_input : public Exception

{

char input\_massage[80];

public:

Exception\_input(int, const char[80], const char[80]);

~Exception\_input();

void what\_input();

};

float input\_price(std::istream&);

std::string input\_ru\_string(std::istream&);

std::string input\_en\_string(std::istream&);

std::string input\_ru\_en\_string(std::istream&);

std::string input\_dimension(std::istream&, std::string);

int input\_int(std::istream&);

Файл Exception\_input.cpp содержащий реализацию класса Exception\_input

#pragma once

#include"Exception\_input.h"

Exception\_input::Exception\_input(int input\_code, const char first\_massage[80], const char second\_massage[80])

: Exception(input\_code, first\_massage)

{

strcpy\_s(input\_massage, 80, second\_massage);

};

Exception\_input::~Exception\_input()

{

};

void Exception\_input::what\_input()

{

this->what();

std::cout << input\_massage << '\n';

};

bool is\_dimension(std::string str, char dimension)

{

int temp = 0;

if (str.length() < 2)

return false;

if (str[str.length() - 1] != dimension || (str[str.length() - 2] != 'м' && str[str.length() - 1] != dimension))

return false;

for (int i = 0; i < str.length() - 2; i++)

if (str[i] <= '9' && str[i] >= '0')

temp = temp \* 10 + ((int)str[i] - '0');

else return false;

if (str[str.length() - 2] <= '9' && str[str.length() - 2] >= '0')

temp = temp \* 10 + ((int)str[str.length() - 2] - '0');

else

if (str[str.length() - 2] != 'м')

return false;

if (temp >= 1000)

return false;

return true;

}

bool is\_time(std::string str)

{

int temp = 0;

if (str.length() < 2)

return false;

if (str[str.length() - 1] == 'с' || str[str.length() - 1] == 'ч')

{

for (int i = 0; i < str.length() - 1; i++)

if (str[i] <= '9' && str[i] >= '0')

temp = temp \* 10 + ((int)str[i] - '0');

else return false;

if (str[str.length() - 1] != 'ч' && temp >= 60)

return false;

else

if (temp > 240)

return false;

return true;

}

if (str.length() < 4)

return false;

if (str[str.length() - 3] != 'м' && str[str.length() - 2] != 'и' && str[str.length() - 1] != 'н')

return false;

else

for (int i = 0; i < str.length() - 3; i++)

if (str[i] <= '9' && str[i] >= '0')

temp = temp \* 10 + ((int)str[i] - '0');

else return false;

if (temp >= 60)

return false;

return true;

}

float input\_price(std::istream& is)

{

bool flag = true;

float num;

do

{

flag = true;

try

{

rewind(stdin);

is >> num;

if (is.fail() || std::cin.peek() != '\n')

{

throw Exception\_input(1, "ошибка ввода числа", "вы ввели символ");

}

if ((int)(num \* 1000) % 10 != 0)

{

throw Exception\_input(2, "ошибка ввода числа", "такой цены не существует");

}

}

catch (Exception\_input& obj)

{

std::cin.clear();

obj.what\_input();

flag = false;

}

} while (!flag);

return num;

}

std::string input\_ru\_string(std::istream& is)

{

bool flag = true;

char BUFF[2048];

std::string str;

do

{

flag = true;

try

{

rewind(stdin);

is.getline(BUFF, sizeof BUFF);

str = BUFF;

if (str.length() == 0)

throw Exception\_input(5, "ошибка ввода строки", "вы ничего не ввели");

for (int i = 0; i < str.length(); i++)

if (str[i] > 'я' || str[i] < 'А')

{

throw Exception\_input(3, "ошибка ввода строки", "смените на русскую раскладку");

}

}

catch (Exception\_input& obj)

{

std::cin.clear();

obj.what\_input();

flag = false;

}

} while (!flag);

return str;

}

std::string input\_en\_string(std::istream& is)

{

bool flag = true;

char BUFF[2048];

std::string str;

do

{

flag = true;

try

{

rewind(stdin);

is.getline(BUFF, sizeof BUFF);

str = BUFF;

if (str.length() == 0)

throw Exception\_input(5, "ошибка ввода строки", "вы ничего не ввели");

for (int i = 0; i < str.length(); i++)

if ((str[i] > 'z' || str[i] < 'a') && (str[i] > 'Z' || str[i] < 'A'))

{

throw Exception\_input(3, "ошибка ввода строки", "смените на английскую раскладку");

}

}

catch (Exception\_input& obj)

{

std::cin.clear();

obj.what\_input();

flag = false;

}

} while (!flag);

return str;

}

std::string input\_ru\_en\_string(std::istream& is)

{

bool flag = true;

char BUFF[2048];

std::string str;

do

{

flag = true;

try

{

rewind(stdin);

is.getline(BUFF, sizeof BUFF);

str = BUFF;

if (str.length() == 0)

throw Exception\_input(5, "ошибка ввода строки", "вы ничего не ввели");

for (int i = 0; i < str.length(); i++)

if ((str[i] > 'z' || str[i] < 'a') && (str[i] > 'Z' || str[i] < 'A'))

{

throw Exception\_input(3, "ошибка ввода строки", "смените на английскую раскладку");

}

}

catch (Exception\_input& obj)

{

std::cin.clear();

obj.what\_input();

flag = false;

}

} while (!flag);

return str;

}

std::string input\_dimension(std::istream& is, std::string dimension)

{

bool flag = true;

char BUFF[2048];

std::string str;

do

{

flag = true;

try

{

rewind(stdin);

is.getline(BUFF, sizeof BUFF);

str = BUFF;

if (dimension == "время")

if (!is\_time(str))

throw Exception\_input(7, "ошибка ввода времени", "масса вводится по стандарту ЧИСЛОразмерность(с, мин, ч)");

if (dimension == "объём")

if (!is\_dimension(str, 'л'))

throw Exception\_input(6, "ошибка ввода объёма", "масса вводится по стандарту ЧИСЛОразмерность(л, мл)");

if (dimension == "масса")

if (!is\_dimension(str, 'г'))

throw Exception\_input(5, "ошибка ввода массы", "масса вводится по стандарту ЧИСЛОразмерность(г, мг)");

}

catch (Exception\_input& obj)

{

std::cin.clear();

obj.what\_input();

flag = false;

}

} while (!flag);

return str;

}

int input\_int(std::istream& is)

{

int num;

bool flag = true;

do

{

flag = true;

try

{

rewind(stdin);

is >> num;

if (is.fail() || std::cin.peek() != '\n')

{

throw Exception\_input(1, "ошибка ввода числа", "вы ввели символ, вводите числа");

}

}

catch (Exception\_input& obj)

{

std::cin.clear();

obj.what\_input();

flag = false;

}

} while (!flag);

return num;

}

Файл Exception\_cont.h содержащий объявления класса Exception\_cont

#pragma once

#include "Exception.h"

#include "Ring.h"

class Exception\_cont : public Exception

{

char cont\_massage[80];

public:

Exception\_cont(int, const char[80], const char[80]);

~Exception\_cont();

void what\_cont();

};

template <class T>

int is\_(Ring<T> ring)

{

try

{

if (!ring.isNoEmpty()) // если текущая позиция имеет нулевое значение

throw Exception\_cont(6, "ошибка контейнера", "контейнер пуст");

}

catch (Exception\_cont& obj)

{

obj.what\_cont();

system("pause");

return false;

}

return true;

}

Файл Exception\_cont.cpp содержащий реализацию класса Exception\_cont

#pragma once

#include "Exception\_cont.h"

Exception\_cont::Exception\_cont(int input\_code, const char first\_massage[80], const char second\_massage[80])

: Exception(input\_code, first\_massage)

{

strcpy\_s(cont\_massage, 80, second\_massage);

};

Exception\_cont::~Exception\_cont()

{

};

void Exception\_cont::what\_cont()

{

this->what();

std::cout << cont\_massage << '\n';

};

Файл Exception\_file.h содержащий объявления класса Exception\_file

#pragma once

#include <fstream>

#include "Exception.h"

using namespace std;

class Exception\_file : public Exception

{

char file\_massage[80];

public:

Exception\_file(int, const char[80], const char[80]);

~Exception\_file();

void what\_file();

};

int exist\_file(const char\* name\_file);

Файл Exception\_file.cpp содержащий реализацию класса Exception\_file

#pragma once

#include "Exception\_file.h"

Exception\_file::Exception\_file(int input\_code, const char first\_massage[80], const char second\_massage[80])

: Exception(input\_code, first\_massage)

{

strcpy\_s(file\_massage, 80, second\_massage);

};

Exception\_file::~Exception\_file()

{

};

void Exception\_file::what\_file()

{

this->what();

std::cout << file\_massage << '\n';

};

int exist\_file(const char\* name\_file)

{

ifstream in\_test(name\_file, ios::in);

try

{

if (!in\_test)

throw (Exception\_file(1, "ошибка файла", "файл не существует"));

if (!in\_test.is\_open())

throw (Exception\_file(3, "ошибка файла", "файл не открылся"));

in\_test.seekg(0, ios::end);

if (in\_test.tellg() == 0)

throw (Exception\_file(2, "ошибка файла", "файл пуст"));

}

catch (Exception\_file obj)

{

obj.what\_file();

system("pause");

return 0;

}

return 1;

}

Файл File.h содержащий объявления класса File

#pragma once

#include "Ring.h"

class File //базовый класс работы с файлами

{

protected:

char file\_name[80]; //название файла

ifstream file\_i; //поток для чтения из файла

ofstream file\_o; //поток для записи в файл

public:

File(const char\* name) //конструктор с параметром

{

strcpy\_s(file\_name, 80, name);

};

};

Файл Algorithm.h содержащий объявления класса Algorithm

#pragma once

#include "Ring.h"

#include "Iterator.h"

class Algorithm

{

public:

template <class T>

void sort\_Ring(Ring<T>& ring);

template <class T>

void find(Ring<T>& ring, T inf);

};

Файл Algorithm.cpp содержащий реализацию класса Algorithm

#pragma once

#include "Algorithm.h"

template <class T>

void Algorithm::sort\_Ring(Ring<T>& ring)

{

int menu = 0;

T temp;

Ring<T>::template Iterator it = ring.Begin();

Ring<T>::template Iterator jt = ring.Begin();

it--;

do

{

it++;

do

{

jt++;

if (compare(\*it, \*jt, menu))

{

temp = \*it;

\*it = \*jt;

\*jt = temp;

}

} while (jt != ring.End());

} while (it != ring.End());

}

template <class T> //поиск элеменьов класса

void Algorithm::find(Ring<T>& ring, T inf)

{

Ring<T> temp;

Ring<T>::template Iterator it = ring.Begin();

it--;

do

{

it++;

if (\*it == inf)

temp.push(\*it);

} while (it != ring.End());

ring.clear();

ring.past(temp);

}

Файл Ring.h содержащий объявления класса Ring

#pragma once

#include<string>

#include<iostream>

#include <fstream>

using namespace std;

template <class T>

struct Element // структура представляющая единичный элемент списка

{

T data; // переменная необходимая для хранения данных в элементе списка

Element\* next; // указатель на следующий элемент списка

Element\* prev; // указатель на предыдущий элемент списка

};

template <class T>

class Ring // шаблон класса кольцо

{

private: // данные доступные для класса

Element<T>\* Curr; // указатель на текущий элемент

Element<T>\* Head; // указатель первый добавленный элемент

public:

int length; // количество элементов в списке

Ring(); // конструктор

Ring(T); // конструктор с параметром

Ring(const Ring<T>&);// конструктор копий

~Ring(); // деструктор

void init(); // текущий элемент ссылается на первый

void push(T); // добавляем новый элемент в список

void push(); // реализация добавления элемента

T pop(); // извлекаем текущий элемент

void fiend(Ring<T>, T);//поиск элемента

void del(Ring<T>&); // удаление

void reInf(T); // перезапись

void loop(); // вывести весь список в консоль

void clear(); // очистить весь список

void next(); // перейти к следующему элементу

void prev(); // перейти к предыдущему элементу

int isNoEmpty(); // проверка состояния списка (если список не пуст)

void past(Ring<T>); // вставить в список все элементы другого списка

class Iterator

{

Element<T>\* curr; // указатель на текущий элемент

public:

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 ==

};

Iterator Begin(); // Вернуть начало кольца

Iterator End(); // Вернуть конец кольца

};

// Конструктор итератора

template<typename T>

Ring<T>::Iterator::Iterator()

{

curr = nullptr;

}

// Конструктор для функций Begin() и End()

template<typename T>

Ring<T>::Iterator::Iterator(Element<T>\* current)

{

this->curr = current;

}

// Сдвинуться впереёд по дереву

template<typename T>

T& Ring<T>::Iterator::operator++(int value)

{

this->curr = curr->next;

return curr->data;

}

// Сдануться назад по дереву

template<typename T>

T& Ring<T>::Iterator::operator--(int value)

{

this->curr = curr->prev;

return curr->data;

}

// Получить данные в узле

template<typename T>

T& Ring<T>::Iterator::operator\*()

{

return this->curr->data;

}

//перегрузка оператора !=

template<typename T>

bool Ring<T>::Iterator::operator!=(Iterator b)

{

if (this->curr != b.curr)

return true;

else

return false;

}

//перегрузка оператора ==

template<typename T>

bool Ring<T>::Iterator::operator==(Iterator b)

{

return !(\*this == b);

}

// Вернуть начало кольца

template<typename T>

Ring<T>::template Iterator Ring<T>::Begin()

{

Iterator it(this->Head);

return it;

}

// Вернуть конец кольца

template<typename T>

Ring<T>::template Iterator Ring<T>::End()

{

Iterator it(this->Head->prev);

return it;

}

Файл Ring.cpp содержащий реализацию класса Ring

#include "Ring.h"

template <class T>

Ring<T>::Ring() // реализация конструктора

{

Head = NULL; // обнуляем указатель на первый элемент

Curr = NULL; // обнуляем указатель на текущий элемент

length = 0; // обнуляем количество элементов в списке

}

template <class T>

Ring<T>::Ring(T x) // реализация конструктора с параметром

{

Head = NULL; // обнуляем указатель на первый элемент

Curr = NULL; // обнуляем указатель на текущий элемент

length = 0; // обнуляем количество элементов в списке

push(x); // добавляем в список элемент с данными, переданными в конструктор

}

template <class T>

Ring<T>::Ring(const Ring<T>& C) // реализация конструктора копий

{

this->Head = NULL; // обнуляем указатель на первый элемент

this->Curr = NULL; // обнуляем указалель на текущий элемент

this->length = 0; // обнуляем количество элементов в списке

Element<T>\* p = C.Head; // сохраняем указатель на текущий элемент списка

for (int i = 0; i < C.length; i++) // заводим цикл на количество элементов списка-параметра

{

push(p->data); // добавляем в список текущий элемент списка-параметра

p = p->next; // переходим к следующему элементу списка-параметра

}

}

template <class T>

Ring<T>::~Ring() // реализация деструктора

{

clear(); // очистить список

}

template <class T>

void Ring<T>::init() // приведения списка к состоянию текущий элемент - первый не удаленный

{

Curr = Head; // устанавливаем текущий элемент на первый не удаленный

}

template <class T> void Ring<T>::next() // реализация перехода к следующему элементу

{

if (isNoEmpty()) // если список не пуст

Curr = Curr->next; // установить текущую позицию на следующую

}

template <class T> void Ring<T>::prev() // реализация перехода к предыдущему элементу

{

if (!isNoEmpty()) // если список не пуст

Curr = Curr->prev; // установить текущую позицию на предыдущую

}

template <class T> int Ring<T>::isNoEmpty() // реализация проверки списка на наличие элементов

{

if (Curr == NULL) // если текущая позиция имеет нулевое значение

return 0; // вернуть "ложь"

else // иначе

return 1; // вернуть "истина"

}

template <class T> void Ring<T>::push(T data) // реализация добавления элемента

{

Element<T>\* inserted; // создать новый указатель на элемент

inserted = new Element<T>; // выделить память под элемент

inserted->data = data; // установить входной параметр в поле данных элемента

if (!isNoEmpty()) // если список пуст

{

Head = inserted; // установить указатель первого элемента на новый элемент

Curr = inserted; // установить указатель текущего элемента на новый элемент

Curr->next = inserted; // установить указатель следующего элемента на новый элемент

Curr->prev = inserted; // установить указатель предыдущего элемента на новый элемент

}

else // если список не пуст

{

inserted->next = Curr->next; // перенаправляем указатель следующего элемента в добавляемом

inserted->next->prev = inserted; // перенаправляем указатель следующего элемента на добавляемый

Curr->next = inserted; // перенаправляем следующий указатель на добавляемый

inserted->prev = Curr; // перенаправляем предыдущий добавляемого на текущий

}

length++; // увеличиваем количество элементов в списке

Curr = inserted; // устанавливаем текущий указатель на добавленный

}

template <class T> void Ring<T>::push() // реализация добавления элемента

{

Element<T>\* inserted; // создать новый указатель на элемент

Element<T>\* temp = Head;

inserted = new Element<T>; // выделить память под элемент

cin >> inserted->data; // установить входной параметр в поле данных элемента

if (!isNoEmpty()) // если список пуст

{

Head = inserted; // установить указатель первого элемента на новый элемент

Curr = inserted; // установить указатель текущего элемента на новый элемент

Curr->next = inserted; // установить указатель следующего элемента на новый элемент

Curr->prev = inserted; // установить указатель предыдущего элемента на новый элемент

}

else // если список не пуст

{

for (int i = 0; i < length; i++)

if (equ(inserted->data, temp->data))

{

cout << "Такой объект уже есть\n";

system("pause");

return;

}

else

{

temp = temp->next;

}

inserted->next = Curr->next; // перенаправляем указатель следующего элемента в добавляемом

inserted->next->prev = inserted; // перенаправляем указатель следующего элемента на добавляемый

Curr->next = inserted; // перенаправляем следующий указатель на добавляемый

inserted->prev = Curr; // перенаправляем предыдущий добавляемого на текущий

}

length++; // увеличиваем количество элементов в списке

Curr = inserted; // устанавливаем текущий указатель на добавленный

}

template <class T> T Ring<T>::pop() // реализация извлечения текущего элемента

{

Element<T>\* temp = Curr; // сохраняем указатель на текущий элемент

T tag = temp->data;; // переменная под возвращаемое значение

if (!isNoEmpty()) return tag; // если список пуст вернуть "ложь"

if (length == 1) // если элемент единственный в списке

{

Head = NULL; // обнулить значение первого элемента

Curr = NULL; // обнулить значение текущего элемента

}

else // если элемент не единственный

{

Curr->next->prev = Curr->prev; // связываем следующий и предыдущий

Curr->prev->next = Curr->next; // связываем предыдущий и следующий

Curr = Curr->next; // перенаправляем текущий элемент на следующий

}

if (temp == Head) // если удаляемый элемент - первый добавленный

Head = Head->next; // перенаправить первый на следующий

length--; // уменьшить количество элементов

delete temp; // удалить предыдущий текущий элемент

return tag; // вернуть данные удаленного элемента

}

template <class T> //поиск элеменьов класса

void Ring<T>::fiend(Ring<T> ring, T inf)

{

Element<T>\* tempCar = ring.Head;

Element<T>\* tempFiend = Head;

for (int j = 0; j < ring.length; j++)

{

if (tempCar->data == inf)

push(tempCar->data);

tempCar = tempCar->next;

}

}

template <class T>

void Ring<T>::del(Ring<T>& ring) //удалить элементы класса

{

if (ring.isNoEmpty())

{

Element<T>\* tempCar = Head;

Element<T>\* tempFiend = ring.Head;

int n = this->length;

for (int j = 0; j < n; j++)

{

if (tempCar->data == tempFiend->data)

{

if (Curr == tempCar) // если искомое значение находится в текущем состоянии до поиска

tempCar = tempCar->next;

pop();

tempFiend = tempFiend->next;

}

else

tempCar = tempCar->next;

Curr = tempCar;

}

}

}

template <class T> //перезапись поля класса

void Ring<T>::reInf(T finded)

{

if (isNoEmpty())

{

Element<T>\* tempCar = Head;

int n = this->length;

T new\_inf;

new\_inf.set\_something();

for (int j = 0; j < n; j++)

{

if (tempCar->data == finded)

{

if (Curr == tempCar) // если искомое значение находится в текущем состоянии до поиска

tempCar = tempCar->next;

this->Curr->data = new\_inf;

}

else

tempCar = tempCar->next;

Curr = tempCar;

}

}

}

template <class T> void Ring<T>::loop() // реализация вывода всех элементов списка

{

if (isNoEmpty()) // если список не пуст

{

Element<T>\* tempCar = Head; // сохраняем указатель на текущий элемент

tempCar->data.print\_table();

for (int i = 0; i < length; i++) // заводим цикл на количество элементов списка

{

cout << i + 1 << tempCar->data; // выводим данные текущего элемента

tempCar = tempCar->next; // переходим к следующему элементу

}

cout << "\n"; // вывод терминального символа

}

else // если список пуст

cout << "List is empty\n"; // вывести предупреждение

}

template <class T> void Ring<T>::clear() // реализация очистки всего списка

{

for (int i = 0; i < length;) // заводим цикл на количество элементов списка

pop(); // извлекаем текущий элемент

}

template <class T> void Ring<T>::past(Ring<T> a) // вставка одного списка в другой

{

init(); // устанавливаем текущий элемент на первый

a.init(); // устанавливаем текущий элемент на первый в списке - параметре

for (int i = 0; i < a.length; i++) // заводим цикл на количество элементов списка-параметра

{

push(a.Curr->data); // вставить в список данные текущего элемента списка-параметра

a.next(); // перейти к следующему элементу списка-параметра

}

}

Файл Interface.h содержащий объявления класса Interface

#pragma once

#include <fstream>

#include <deque>

#include "File\_text.h"

#include "File\_text.cpp"

#include "Ring.h"

#include "myRing.cpp"

#include "Aerosols.h"

#include "Liquid.h"

#include "Medication.h"

#include "Powder.h"

#include "Salve.h"

#include "Solid.h"

#include "Solutions.h"

#include "Tablets.h"

#include "Exception.h"

#include "Exception\_input.h"

#include "Exception\_cont.h"

#include "Exception\_file.h"

#include "Algorithm.h"

#include "Algorithm.cpp"

class Interface //класс интерфейс

{

public:

template <class T>

void delete\_ring(Ring<T>&); //меню удаления препаратов

template <class T>

void show\_ring(Ring<T>); //меню вывода препаратов

template <class T>

void reInf\_ring(Ring<T>&); //меню изменения информации о препаратах

template <class T>

void save\_ring(Ring<T>, deque<Ring<T>>&); //сохранение последнего действия

template <class T>

void ctrl\_z(Ring<T>&, deque<Ring<T>>&); //отмена последнего действия

template <class T>

void get\_file(Ring<T>&, const char\*); //достать всю информацию из файла

template <class T>

void geve\_file(Ring<T>&, const char\*); //записать всю информацию в файл

template <class T>

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

int start(); //меню выбора типа препарата

};

Файл Interface.cpp содержащий реализацию класса Interface

#pragma once

#include "Interface.h"

template <class T>

void Interface::show\_ring(Ring<T> test) //меню вывода препаратов

{

T information;

Algorithm alg;

do

{

system("CLS");

test.loop(); //вывод контейнера на экран

if (!information.set\_something()) //параметр поиска

break;

alg.find(test, information); //поиск по контейнеру

} while (true);

};

template <class T>

void Interface::delete\_ring(Ring<T>& test) //меню удаления препаратов

{

Ring<T> temp;

T information;

Algorithm alg;

temp.past(test); //вставка элементов

do

{

system("CLS");

temp.loop(); //вывод контейнера на экран

if (!information.set\_something()) //параметр поиска

break;

alg.find(temp, information); //поиск по контейнеру

cout << "Нажмите ENTER чтобы применить удаление, или любую другую клавишу для продолжения\n";

rewind(stdin);

if (getchar() == '\n')

{

test.del(temp); //удаление элементов

break;

}

} while (true);

}

template <class T>

void Interface::reInf\_ring(Ring<T>& test) //меню изменения информации о препаратах

{

Ring<T> show\_obj;

T information;

show\_obj.past(test); //вставка элементов

do

{

system("CLS");

show\_obj.loop(); //вывод контейнера на экран

if (!information.set\_something()) //параметр поиска

break;

show\_obj.reInf(information); //изменение полей

cout << "Нажмите ENTER чтобы применить изменения, или любую другую клавишу для продолжения\n";

rewind(stdin);

if (getchar() == '\n')

{

test.clear(); //очистка контейнера

test.past(show\_obj);

break;

}

} while (true);

}

template <class T> //сохранение последнего действия

void Interface::save\_ring(Ring<T> ring, deque<Ring<T>>& memorry)

{

Ring<T> temp;

temp.past(ring);

if (memorry.size() > 5) //удаляем самое старое сохранение

memorry.pop\_front(); //если количество сохранений больше 5

memorry.push\_back(temp); //добавляем сохранение

temp.clear();

}

template <class T> //отмена последнего действия

void Interface::ctrl\_z(Ring<T>& ring, deque<Ring<T>>& memorry)

{

if (memorry.size() == 0) //выводис мообщение на экран

{ //если сохранений нет

cout << "Сохранений нет!\n";

system("pause");

return;

}

ring.clear();

ring.past(memorry[memorry.size() - 1]); //дприсваиваем последнее сохранение

memorry.pop\_back(); //удаляем последнее сохранение

}

template <class T> //достать всю информацию из файла

void Interface::get\_file(Ring<T>& test, const char\* name\_file)

{

File\_text<T> fl\_txt(name\_file);

fl\_txt.open\_file\_in(); //открываем файл

while (!fl\_txt.feof()) //пока не конец файла

{

T obj;

fl\_txt.read\_record\_in\_file\_text(obj); //читаем из файла

test.push(obj); //заносим элемент в контейнер

}

test.pop(); //удаляем лишний элемент, который добавился

} //из-за '\n' в конце файла

template <class T> //записать всю информацию в файл

void Interface::geve\_file(Ring<T>& test, const char\* name\_file)

{

File\_text<T> fl\_txt(name\_file);

fl\_txt.open\_file\_out(); //открываем файл

while (test.isNoEmpty()) //пока не пустой контейнер

{

T obj;

obj = test.pop(); //достаём элемент из контейнера

fl\_txt.write\_record\_in\_file\_text(obj); //записать элемент в файл

}

}

template <class T> //меню выбора операции над препаратами

int Interface::chouse(Ring<T>& test, const char\* name\_file)

{

Algorithm alg;

deque<Ring<T>> memorry;

Ring<T> temp;

do

{

system("CLS");

cout << "1 - Добавить\n" << "2 - удалить\n" << "3 - посмотреть\n" << "4 - изменить поле\n"

<< "5 - достать всю информацию из файла\n" << "6 - записать всю информацию в файл\n"

<< "7 - очистить файл\n" << "8 - сортировать контейнер\n" << "9 - отмена последнего действия\n" << "0 - назад\n";

switch (input\_int(cin))

{

system("CLS");

case 1:

{

test.push();

break;

}

case 2:

{

if (is\_(test))

{

save\_ring(test, memorry);

delete\_ring(test);

}

break;

}

case 3:

{

if (is\_(test))

show\_ring(test);

break;

}

case 4:

{

if (is\_(test))

{

save\_ring(test, memorry);

reInf\_ring(test);

}

break;

}

case 5:

{

if (exist\_file(name\_file))

get\_file(test, name\_file);

break;

}

case 6:

{

if (is\_(test))

geve\_file(test, name\_file);

break;

}

case 7:

{

if (exist\_file(name\_file))

{

File\_text<T> fl\_txt(name\_file);

fl\_txt.file\_clear();

}

break;

}

case 8:

{

if (is\_(test))

{

save\_ring(test, memorry);

alg.sort\_Ring(test);

}

break;

}

case 9:

{

ctrl\_z(test, memorry);

break;

}

case 0:

{

return 0;

break;

}

default:

{

std::cout << "такого пункта нет, введите число от 0 до 8\n";

system("pause");

break;

}

}

} while (true);

}

int Interface::start() //меню выбора типа препарата

{

Ring <Aerosols> test\_Aerosols;

Ring <Powder> test\_Powder;

Ring <Tablets> test\_Tablets;

Ring <Solutions> test\_Solutions;

Ring <Salve> test\_Salve;

do

{

system("CLS");

cout << "1 - работать с аэрозолями\n" << "2 - работать с порошками\n" << "3 - работать с таблетками\n"

<< "4 - работать с мазями\n" << "5 - работать с растворами\n" << "0 - закончить выполнение программы\n";

switch (input\_int(std::cin))

{

case 1:

{

chouse(test\_Aerosols, "Aerosols.txt");

break;

}

case 2:

{

chouse(test\_Powder, "Powder.txt");

break;

}

case 3:

{

chouse(test\_Tablets, "Tablets.txt");

break;

}

case 4:

{

chouse(test\_Solutions, "Solutions.txt");

break;

}

case 5:

{

chouse(test\_Salve, "Salve.txt");

break;

}

case 0:

{

return 0;

break;

}

default:

{

std::cout << "такого пункта нет, введите число от 0 до 5\n";

system("pause");

break;

}

}

} while (true);

}