МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ИНФОРМАЦИОННО-СЕТЕВЫХ ТЕХНОЛОГИЙ

КУРСОВАЯ РАБОТА  
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. техн. наук |  |  |  | Курицын К.А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ «Здания» |
| по дисциплине: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 1542 |  |  |  | М.Н Прудов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

Содержание

1. Задание …………………………………………………………………………...
2. Функциональные требования…………………………………………………...
3. Требование к оборудованию и стороннему ПО……………………………….
4. Технические спецификации……………………………………………………..
5. Взаимодействие с внешними и внутренними системами……………………..
6. Программа и методика испытаний……………………………………………...
7. Литература………………………………………………………………………..

Приложение 1 – Описание формата………………………………………………...

Приложение 2 – Листинг, Ссылка на систему контроля версий………………….

Приложение 3 – Примеры, скриншоты……………………………………………..

1. **Задание**

Консольное приложение «Здания» должно реализовывать управление электронным каталогом недвижимости. Каждое здание должно содержать параметры: тип, описание, этажность, год постройки, число квартир, цвет, общая площадь дома, адрес.

Сам каталог является текстовым файлом, доступным для просмотра. Управление должно осуществляется через консольное приложение, включающее в себя функции добавления нового здания, поиск по типу зданий, учёт количества зданий в каталоге на данный момент

1. **Функциональные требования**

2.1) Выход из приложения

Реализовать выход из приложения.

2.2) Добавление здания

Реализовать добавление новых зданий в каталог. Добавление должно происходить либо из консоли, либо из файла.

2.3) Поиск зданий

Реализовать поиск всех зданий из каталога по заданному типу.

2.4) Изменение цвета зданий

Реализовать возможность изменения цвета здания в библиотеке файла-каталога.

2.5) Запись результата

Результат записывается в txt файл формата указанного в Приложении 1. здания записанные в файл-каталог должны быть разделены по типам. Файл должен быть доступен для просмотра в любое время.

2.6) Удобный интерфейс

Интерфейс должен быть удобным и понятным. Необходимо очищать консоль от лишних элементов.

2.7) Язык С++

Приложение должно быть реализовано при помощи языка С++ и использовании классов.

1. **Требования к оборудованию и стороннему ПО**

Консольное приложение. Операционная система Windows 7 и Visual Studio 2017.

1. **Технические спецификации**

4.1) При запуске приложения перед пользователем отображается название приложения и варианты действий для работы с недвижимостью.

4.2) Нулевой пункт отвечает за выход из приложения.

4.3) Первый пункт, вариант “c” отвечает за добавление нового здания из консоли.

Если выбран первый пункт, вариант “с”, то пользователь может ввести с консоли все данные о новом здании, после чего оно автоматически добавляется в каталог в файле, о чём приходит сообщение на экран. Затем программа вновь спрашивает хотим ли мы добавить здание и каким способом. Далее ожидается нажатие Enter после чего пользователь вновь попадает в основное меню.

4.4) Первый пункт, вариант “f” отвечает за добавление здания из файла.

Если Первый пункт, вариант “f”, то из файла подгрузки автоматически загружается в библиотеку необходимый образец здания, о чём пользователю сообщается на экране.

Затем программа вновь спрашивает хотим ли мы добавить здание и каким способом.

При выборе загрузки из консоли, приложение очищает экран и последовательно запрашивает пользователя характеристики нового здания. Далее ожидается нажатие Enter после чего пользователь вновь попадает в основное меню.

4.5) Второй пункт отвечает за отображения всех зданий в каталоге.

При выборе второго пункта, пользователю выдаётся список всех зданий из каталога и их параметров. Ожидается нажатие Enter после чего пользователь вновь попадает в основное меню.

4.6) Третий пункт отвечает за отображения количества зданий в каталоге.

При выборе третьего пункта, пользователю покажет количество зданий в каталоге. Ожидается нажатие Enter после чего пользователь вновь попадает в основное меню.

4.7) Четвёртый пункт отвечает за сохранение экземпляров зданий в соответствующие файлы. При выборе четвёртого пункта, консоль опрашивает пользователя хочет ли он очистить перед этим файлы зданий, после чего все здания сохраняются в соответствующий файлах, указанных в Приложении 1 . После чего ожидается нажатие Enter после чего пользователь вновь попадает в основное меню.

4.8) Пятый пункт отвечает за поиск всех зданий по указанному типу.

При выборе пятого пункта, запрашивается интересующий нас тип, после чего выводятся все книги данного тип. Ожидается нажатие Enter после чего пользователь вновь попадает в основное меню.

4.9) Шестой пункт отвечает за изменение цвета здания по указанному идентификатору.

При выборе шестого пункта, запрашивается интересующий нас ID, после чего требуется ввести цвет, в который будет перекрашено здание. Ожидается нажатие Enter после чего пользователь вновь попадает в основное меню.

1. **Взаимодействия с внешними и внутренними системами**

Взаимодействие типа консоль-файл. Приложение взаимодействует с четырьмя файлами:

Cottage.txt, panel house.txt, brick house.txt, brick-monolithic house.txt.

1. **Программа и методика испытаний**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nпп | Сценарий проверки | Ожидаемый результат | Результат  № рисунка (см. в Приложении №3) | №ФТ  Технические спецификации |
| 1 | Запустить ПО | Отображение названия программы и всех пунктов для работы с каталогом:  0)Выход  1)Добавление из консоли/файла  3) Отображение всех зданий  4)Сохранение  5)Поиск по заданному типу  6)Изменение цвета | Получен  Рисунок 1 | 4.1 |
| 2 | В меню выбрать 0 | 1. Выход из приложения |  | 4.2 |
| 3 | В меню выбрать 1 | 1. Добавление здание нужного типа, из консоли 2. При выборе можно добавить здание из файла | Получен рисунок 2  Получен рисунок 3 | 4.3  4.4 |
| 4 | В меню выбрать 2 | 1. Вывод всех зданий из каталога | Получен рисунок 7  Получен рисунок 4 | 4.5 |
| 5 | В меню выбрать 3 | 1. Вывод количества зданий в каталоге |  | 4.6 |
| 6 | В меню выбрать 4 | 1. Предложение очистить файлы перед записью 2. Сохранение зданий в соответствующие типам файлы указанные в Приложении 1 3. Получение сообщения об удачном сохранении | Получен рисунок 5 Получен рисунок 6 | 4.7 |
| 7 | В меню выбрать 5 | 1. Поиск зданий по заданному типу |  | 4.8 |
| 8 | В меню выбрать 6 | Изменение цвета по введённому ID здания |  | 4.9 |

1. **Литература**

* https://support.microsoft.com
* http://www.cyberforum.ru
* https://ru.stackoverflow.com
* http://cppstudio.com
* http://www.c-cpp.ru
* Б. Керниган и Д. Ритчи – Язык программирования Си
* Шилдт Г. Самоучитель С++
* Бьерн Страуструп - Язык программирования С++

**Приложение №1 Описание формата**

Формат файла должен отвечать следующим требованиям:

1. При добавлении здания из файла расширение файла должно быть “.txt” а имя файла не должно содержать пробелов.
2. Файл должен содержать обычный текст без форматирования и без кодирования, состоящий из записей (строк) в кодировки ASCII, русские символы не поддерживаются. Все поля разделены символом переноса ‘\n’.
3. При сохранении здания сохраняются в соответствующие типам здания: cottage.txt, panelhouse.txt, brickhouse.txt, brick-monolithichouse.txt.
4. В один файл может быть записано несколько зданий одного типа,

последующие здания записываются без разделения, со следующей строки.

1. Структура файла:
   * Описание
   * Цвет
   * Адрес
   * Этажность
   * Размер
   * Количество квартир
   * Дата постройки

Структура файла в зависимости от типа здания не отличается.

**Приложение №2 Листинг, система контроля версий**

Ссылка на систему контроля версий https://github.com/Copypas/cursework

Main.cpp

#include "stdafx.h"

#include <iostream>

#include "HouseManager.h"

using namespace std;

void showMainMenu()

{

cout << "--- House Manager ---" << endl

<< "0 - Exit" << endl

<< "1 - Add houses to list" << endl

<< "2 - Show houses list" << endl

<< "3 - Show number of houses in list" << endl

<< "4 - Save list into separate files" << endl

<< "5 - Find by type" << endl

<< "6 - Change house color" << endl

<< "-> ";

}

int main()

{

setlocale(LC\_ALL, "Russian");

HouseManager Houses;

showMainMenu();

int c;

do

{

cin >> c;

switch (c)

{

case 1: { Houses.Add(); cout << "house added." << endl; break; }

case 2: { cout << "Top of list." << endl; Houses.Show(); cout << "End of list." << endl; break; }

case 3: { cout << "Number of houses in list: " << Houses.Number() << endl; break; }

case 4: { Houses.SaveToFile("cottage.txt", "panel house.txt", "brick house.txt", "brick-monolithic house.txt"); cout << "Saved." << endl; break; }

case 5: { Houses.ShowType(); break; }

case 6: { Houses.ChangeColor(); break; }

case 0: { cout << "Exiting program." << endl; return 0; }

}

system("pause");

system("cls");

showMainMenu();

} while (1);

return 0;

}

HouseManager.h

#pragma once

#include <iostream>

#include <list>

#include <fstream>

#include "Factory.h"

//using namespace std;

class HouseManager

{

list <House\*> hous;

public:

void Add();

void SaveToFile(string fcottage, string fpannel, string fbrick,

string fbm);

void Show();

void ShowType();

void ChangeColor();

int Number() { return hous.size(); }

};

void HouseManager::Add()

{

Cottage\_Factory cottage;

Panel\_house\_Factory panelHouse;

Brick\_house\_Factory brickHouse;

Brick\_monolithic\_house\_Factory bmHouse;

bool add = 1;

do

{

bool del = false;

Factory \*factory;

cout << "Input type of house" << endl

<< "0-cottage, 1-panel house, 2-brick house, 3-brick-monolithic house" << endl

<< "-> ";

int choice\_house;

cin >> choice\_house;

switch (choice\_house)

{

case house\_type\_cottage: { factory = &cottage; break; }

case house\_type\_panelHouse: { factory = &panelHouse; break; }

case house\_type\_brickHouse: { factory = &brickHouse; break; }

case house\_type\_bmHouse: { factory = &bmHouse; break; }

default: { throw ("Incorrect type of house"); break; }

}

House \*ObjectHouse = CreateHouse(factory);

cout << "Input type of adding data" << endl

<< "c - console, f - file" << endl

<< "-> ";

char choice\_input;

cin >> choice\_input;

switch (choice\_input)

{

case 'c':

{

ObjectHouse->Read();

break;

}

case 'f':

{

string fname;

cout << "Input file name -> ";

cin >> fname;

ifstream fpin(fname + ".txt");

try

{

if (!fpin) throw ("File not found");

ObjectHouse->Read(fpin);

}

catch (char \* err)

{

cout << err << endl << "Closing file." << endl;

del = true;

}

fpin.close();

break;

}

default:

{

cout << "Incorrect symbol." << endl;

del = true;

}

}

hous.push\_back(ObjectHouse);

if (del)

{

cout << "Deleting this house." << endl;

hous.pop\_back();

del = false;

}

cout << "Do you want to add another house? (0/1) -> ";

cin >> add;

} while (add);

}

void HouseManager::Show()

{

for (auto v : hous)

{

cout << "--------------------" << endl;

cout << "ID: ";

cout << v->getID() << endl;

v->Print();

cout << "--------------------" << endl;

}

}

void HouseManager::ShowType()

{

cout << "Input type of house" << endl

<< "0-cottage, 1-panel house, 2-brick house, 3-brick-monolithic house" << endl

<< "-> ";

int choice\_type;

cin >> choice\_type;

cout << "Top of list." << endl;

for (auto v : hous)

{

if (v->getType() == choice\_type)

{

cout << "--------------------" << endl;

cout << "ID: ";

cout << v->getID() << endl;

v->Print();

cout << "--------------------" << endl;

}

}

cout << "End of list." << endl;

}

void HouseManager::ChangeColor()

{

cout << "Input id" << endl << "-> ";

int id;

cin >> id;

cout << "Input color" << endl << "-> ";

string color;

cin >> color;

for (auto v : hous)

{

if (v->getID() == id)

{

v->setColor(color);

}

}

}

void HouseManager::SaveToFile(string fcottage, string fpannel, string fbrick,

string fbm)

{

bool clr;

cout << "Do you want to clear files first? (0/1) -> ";

cin >> clr;

if (clr)

{

clrFile(fcottage);

clrFile(fpannel);

clrFile(fbrick);

clrFile(fbm);

}

for (auto v : hous)

{

if (v->getType() == house\_type\_cottage)

{

ofstream fpout;

fpout.open(fcottage, ios::app);

v->Print(fpout);

fpout.close();

}

else if (v->getType() == house\_type\_panelHouse)

{

ofstream fpout;

fpout.open(fpannel, ios::app);

v->Print(fpout);

fpout.close();

}

else if (v->getType() == house\_type\_brickHouse)

{

ofstream fpout;

fpout.open(fbrick, ios::app);

v->Print(fpout);

fpout.close();

}

else if (v->getType() == house\_type\_bmHouse)

{

ofstream fpout;

fpout.open(fbm, ios::app);

v->Print(fpout);

fpout.close();

}

}

}

Factory.h

#include "House.h"

class Factory

{

public:

virtual House\* Create() = 0;

virtual ~Factory(){}

};

class Cottage\_Factory : public Factory

{

public:

House \* Create(){ return new Cottage; }

};

class Panel\_house\_Factory : public Factory

{

public:

House \* Create(){ return new Panel\_house; }

};

class Brick\_house\_Factory : public Factory

{

public:

House \* Create(){ return new Brick\_house; }

};

class Brick\_monolithic\_house\_Factory : public Factory

{

public:

House \* Create(){ return new Brick\_monolithic\_house; }

};

House\* CreateHouse(Factory \*value)

{

return value->Create();

}

House.h

#pragma once

#include <iostream>

#include <string>

#include "Add.h"

using namespace std;

#define gen\_fem 0

#define gen\_male 1

#define house\_type\_cottage 0

#define house\_type\_panelHouse 1

#define house\_type\_brickHouse 2

#define house\_type\_bmHouse 3

static int indentificator=0;

class House

{

protected:

int type;

private:

int id = indentificator++;

string Description;

string colour;

string address;

int level;

Date builtDate;

int size;

int apartments;

string type2str();

public:

virtual ~House() {}

virtual void Read(istream &is = cin);

virtual void Print(ostream &os = cout);

int getType() { return type; }

int getID() { return id; }

void setColor(string color) { colour = color; }

};

string House::type2str()

{

switch (type)

{

case house\_type\_cottage: return "cottage";

case house\_type\_panelHouse: return "panel house";

case house\_type\_brickHouse: return "brick house";

case house\_type\_bmHouse: return "Brick-monolithic house";

}

}

void House::Read(istream &is)

{

if (&is == &cin) is.ignore(std::numeric\_limits<size\_t>::max(), '\n');

is.clear();

if (&is == &cin) cout << "Input Description -> ";

getline(is, Description);

is.clear();

if (&is == &cin) cout << "Input colour -> ";

getline(is, colour);

is.clear();

if (&is == &cin) cout << "Input address -> ";

getline(is, address);

is.clear();

for (int i = 0; i<1; ++i)

try

{

if (&is == &cin) cout << "Input level's count -> ";

string c;

getline(is, c);

is.clear();

level = stoi(c);

}

catch (char \* err)

{

if (&is == &cin)

{

system("cls");

cout << err << " Try again." << endl;

i--;

}

else throw ("Error reading file. Level's count.");

}

for (int i = 0; i<1; ++i)

try

{

is.clear();

if (&is == &cin) cout << "Input house size -> ";

string c;

getline(is, c);

if (c != "")

size = stoi(c);

is.clear();

}

catch (char \* err)

{

if (&is == &cin)

{

system("cls");

cout << err << " Try again." << endl;

i--;

}

else throw ("Error reading file. House size.");

}

for (int i = 0; i<1; ++i)

try

{

if (&is == &cin) cout << "Input apartments number -> ";

string c;

getline(is, c);

is.clear();

apartments = stoi(c);

}

catch (char \* err)

{

if (&is == &cin)

{

system("cls");

cout << err << " Try again." << endl;

i--;

}

else throw ("Error reading file. Apartments number.");

}

for (int i = 0; i<1; ++i)

try

{

if (&is == &cin) cout << "Input built date (dd.mm.yyyy) -> ";

is >> builtDate;

is.clear();

}

catch (char \* err)

{

if (&is == &cin)

{

system("cls");

cout << err << " Try again." << endl;

i--;

}

else throw ("Error reading file. Incorrect date.");

}

}

void House::Print(ostream &os)

{

if (&os == &cout) { cout << "Type: "; cout << type2str() << endl; }

if (&os == &cout) cout << "Description: ";

os << Description << endl;

if (&os == &cout) cout << "Colour: ";

os << colour << endl;

if (&os == &cout) cout << "Adress: ";

os << address << endl;

if (&os == &cout) cout << "Levels: ";

os << level << endl;

if (&os == &cout) cout << "size: ";

os << size << endl;

if (&os == &cout) cout << "apartments number: ";

os << apartments << endl;

if (&os == &cout) cout << "Built date: ";

os << builtDate << endl;

}

//-----------------------------------------------------------------------------

class Cottage : public House

{

public:

void Read(istream &is = cin) { type = house\_type\_cottage; House::Read(is); }

};

class Panel\_house : public House

{

public:

void Read(istream &is = cin) { type = house\_type\_panelHouse; House::Read(is); }

};

class Brick\_house : public House

{

public:

void Read(istream &is = cin) { House::Read(is); type = house\_type\_brickHouse; }

};

class Brick\_monolithic\_house : public House

{

public:

void Read(istream &is = cin) { House::Read(is); type = house\_type\_bmHouse; }

};

Add.h

#pragma once

#include <fstream>

#include <iomanip>

#include <iostream>

#include <cstring>

struct Date {

int Day, Month, Year;

Date() : Day(1), Month(1), Year(1700) {}

Date(int d, int m, int y) : Day(d), Month(m), Year(y)

{

if ((d<1) || (d>31)) throw ("Incorrect day");

if ((m<1) || (m>12)) throw ("Incorrect month");

if ((y<1700) || (y>2200)) throw ("Incorrect year");

}

};

std::istream & operator>>(std::istream &is, Date &d)

{

char dot;

is >> d.Day >> dot >> d.Month >> dot >> d.Year;

if ((d.Day<1) || (d.Day>31))

throw ("Incorrect day");

if ((d.Month<1) || (d.Month>12))

throw ("Incorrect month");

if ((d.Year<1700) || (d.Year>2200))

throw ("Incorrect year");

return is;

}

std::ostream & operator<<(std::ostream &os, const Date &d)

{

return os << std::setfill('0') << std::setw(2) << d.Day

<< '.' << std::setfill('0') << std::setw(2) << d.Month

<< '.' << d.Year;

}

//-----------------------------------------------------------------------------

struct Razmery {

float length;

float height;

float weight;

Razmery() {}

Razmery(float l, float h, float w) : length(l), height(h), weight(w) {}

};

std::istream & operator>>(std::istream &is, Razmery &r)

{

is >> r.length >> r.height >> r.weight;

return is;

}

std::ostream & operator<<(std::ostream &os, const Razmery &r)

{

return os << r.length << ' ' << r.height << ' ' << r.weight;

}

//-----------------------------------------------------------------------------

void clrFile(std::string fname) {

std::ofstream ofs;

ofs.open(fname, std::ofstream::out | std::ofstream::trunc);

ofs.close();

}

**Приложение №3 Примеры**

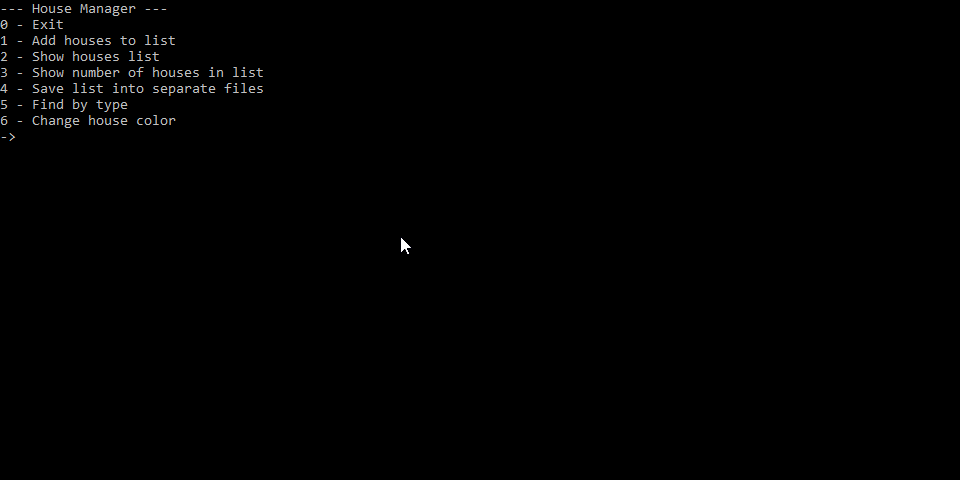


Рисунок 1.



Рисунок 2.

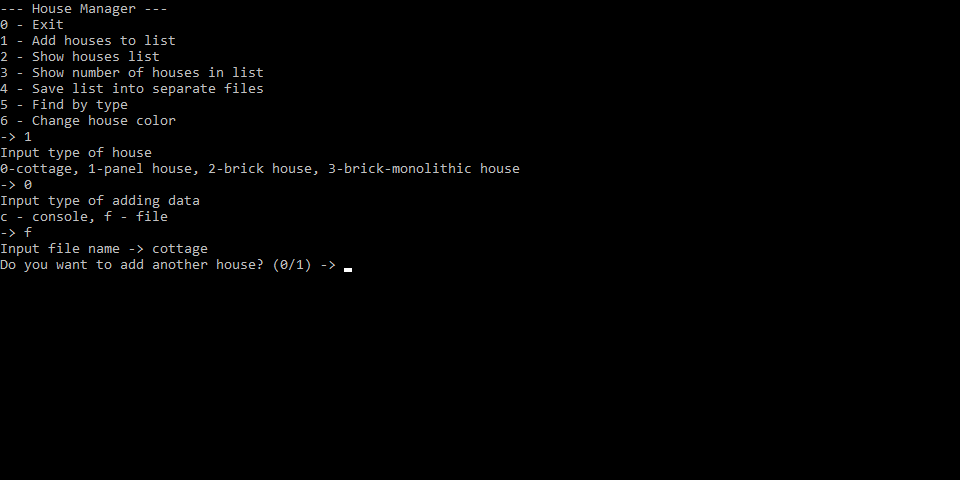


Рисунок 3.

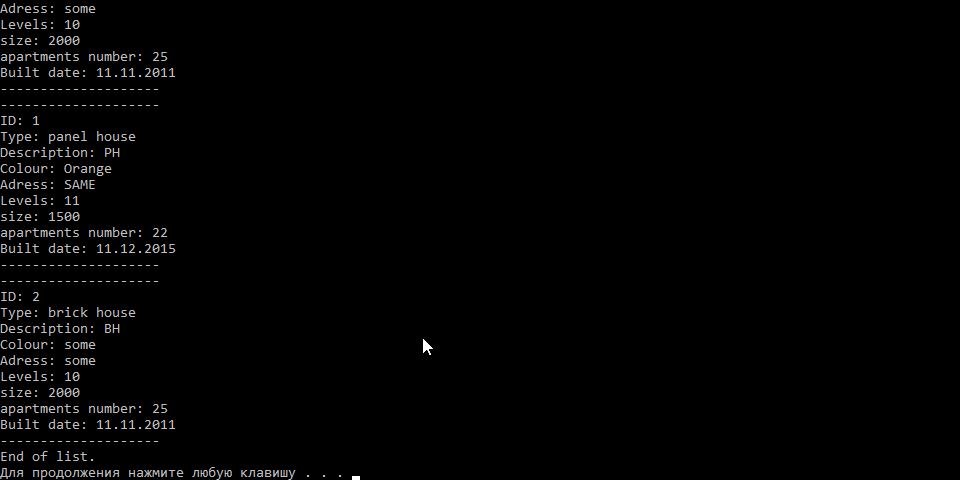


Рисунок 4.

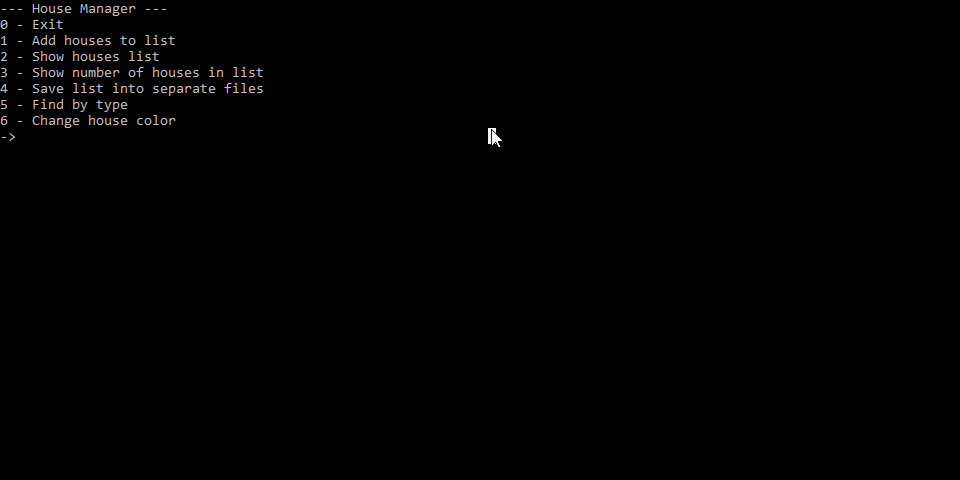


Рисунок 5.

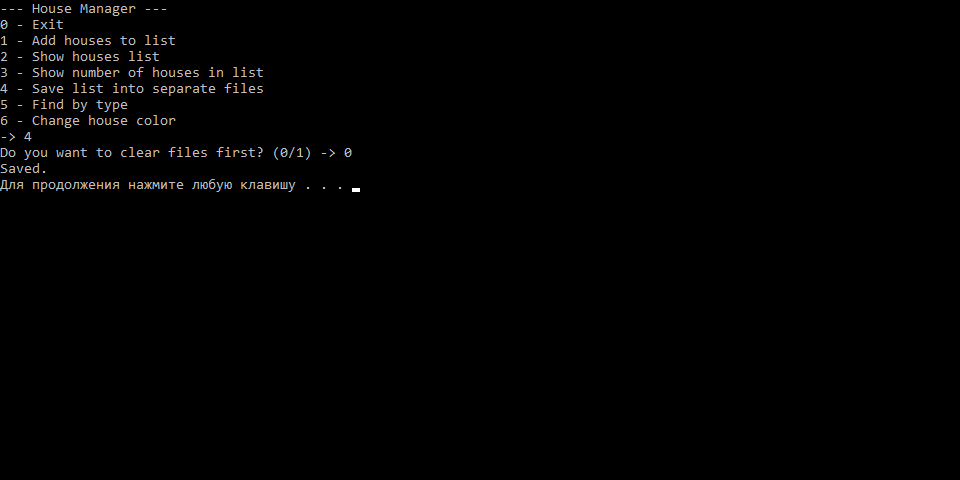


Рисунок 6.

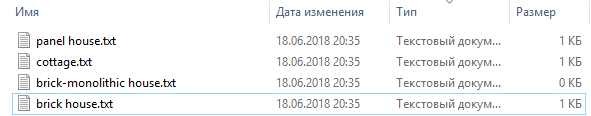


Рисунок 7.

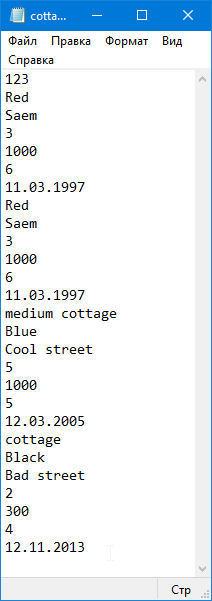


Рисунок 8.

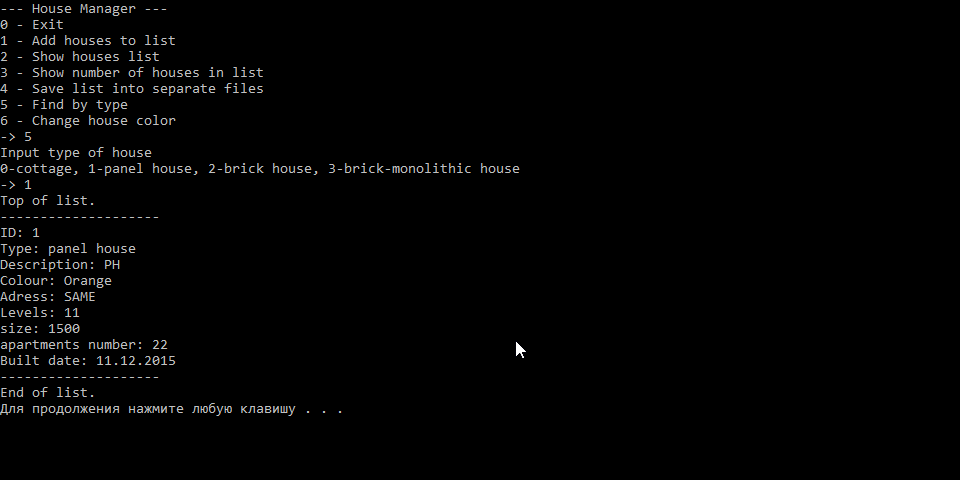


Рисунок 9.

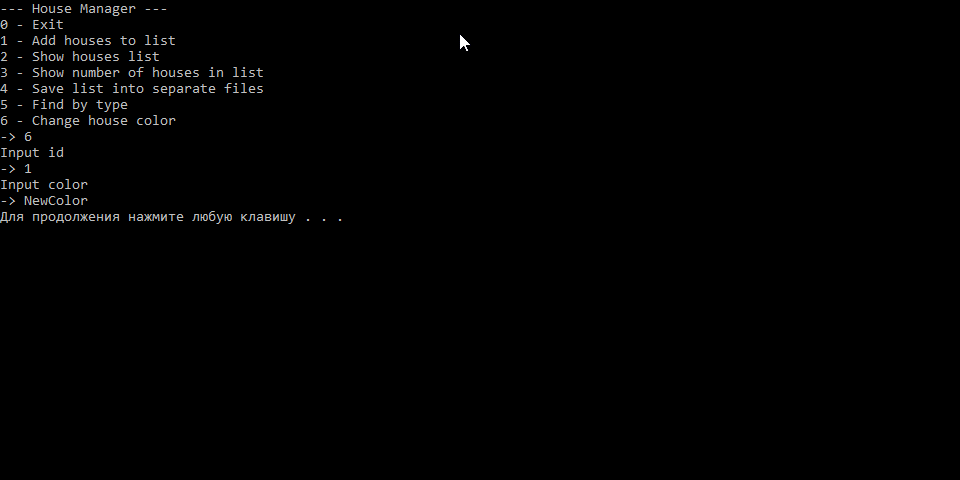


Рисунок 10.

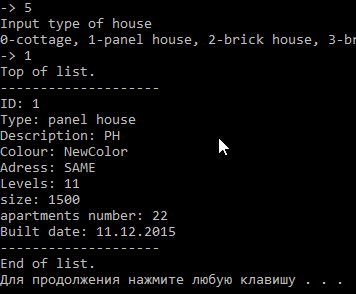


Рисунок 11.