Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №5

Структуры
Тема

Преподаватель _____ А. С. Черниговский
подпись, дата — инициалы, фамилия

Студент КИ19-17/1б 031939175 _____ А. Д. Непомнящий
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2020

1 Цель

Цель настоящей работы состоит в том, чтобы продолжить ознакомление с синтаксисом языка СИ, стандартом кодирования, изучить принципы работы структур.

2 Задачи

Для выполнения лабораторной работы необходимо выполнить следующие задачи:

На оценку 3 балла:

1) написать программу в соответствии с заданием;
2) разбить программу на функции;
3) предусмотреть операции: добавление, удаление и вывод;
4) экземпляры функций хранить в динамическом массиве;
5) реализовать меню пользователя.

На оценку 4 балла:

1) выполнить требования предыдущих пунктов;
2) добавить возможность хранения в файле;
3) не допустить использование глобальных переменных;
4) добавить сортировку по выбранному пользователем полю.

На оценку 5 баллов:

1) выполнить требования предыдущих пунктов;
2) добавить фильтрацию и поиск по одному или нескольким полям на выбор пользователя.

3 Описание задания

Реализовать хранение информации о номерах гостиницы. Создать следующие поля: номер, вместимость, комфортабельность. Добавить вычисляемые поля: цена (зависит от вместимости и комфортабельности).

4 Ход выполнения

Ниже представлен листинг программы по заданию.

Листинг 1 – Код программы, решающей задачу

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>


enum Sizes
{
    // Перечисление наибольших размеров массивов.
    INPUT_SIZE = 100,
};


enum SymbolCodes
{
    // Перечесление кодов символов для функции ввода строки
    START_CHAR_RANGE = 32,
    END_CHAR_RANGE = 126
};


enum OperationsCodes
{
    // Перечисление кодов операций для организации всех меню.
    // Команды для главного меню.
    LOAD_FROM_FILE = 1,
    SAVE_TO_FILE = 2,
    ADD_ROOM = 3,
    DELETE_ROOM = 4,
    PRINT = 5,
    SORT = 6,
    FILTERS = 7,
    FIND_BY_NUMBER = 8,
    QUIT = 9,
    // Команды для меню сортировки.
    SORT_BY_PRICE = 1,
```

Продолжение листинга 1

```
    SORT_BY_CAPACITY = 2,
    SORT_BY_TYPE = 3,
    SORT_BY_NUMBER = 4,
    SORT_CANCEL = 5,
    // Команды для меню настройки фильтров.
    FILTERS_SET = 1,
    FILTERS_RESET = 2,
    FILTERS_CANCEL = 3

};


enum RoomTypes
{
    // Перечисление типов комнат по комфортабельности,
    // вместе с этим - множителей в формуле цены номера.
    JUNISUITE = 1,
    STUDIO = 2,
    LUX = 3,
    APARTMENT = 4,
    SUITE = 5
};


enum PriceFactors
{
    // Перечисление множителей для вычисления стоимости комнат.
    // Эти числа и формулы выбраны произвольно, задание их
    // пользоватлем не предусмотрено заданием.
    BASE_PRICE = 625,
    BASE_CAPACITY_PRICE = 500
};


int CycleInputInt(char* stringToOutput, bool(* pChecker)(int))
{
    // Функция для ввода целого числа с проверкой ввода.
    //
    // char* stringToOutput - строка, которую нужно выводить
    // ... в запросе ввода;
    // bool(* pChecker)(int) - указатель на функцию, проверяющую
```

Продолжение листинга 1

```
    // ... дополнительные условия.

    int number;   // Необходимое число
    int position;   // Позиция числа в введенной строке
    char input[INPUT_SIZE];   // Строка для ввода

    // Считывает и проверяет ввод по нескольким условиям, до тех пор,
    // пока не будет введено корректно.
    while (true)
    {
        printf("%s\n", stringToOutput);
        fflush(stdout);
        char* fgetsRet = fgets(input, INPUT_SIZE, stdin);
        if (fgetsRet == NULL)
        {
            printf("Wrong format!\n");
            continue;
        }
        int inputLength = strlen(input) - 1;
        input[inputLength] = '\0';
        int sscanfRet = sscanf(input, "%d%n", &number, &position);
        if (position != inputLength)
        {
            printf("Wrong format!\n");
            continue;
        }
        if (pChecker && !pChecker(number))
        {
            printf("Wrong format!\n");
            continue;
        }
        if (sscanfRet == 1) break;
        printf("Wrong format!\n");
    }
    return number;
}


bool MainMenuInputChecker(int operationCode)
{
    // Функция для вызова в функциях ввода с проверкой.
```

Продолжение листинга 1

```
    // ... Возвращает true, если введенное значение может быть
    // значением кода операции в главном меню.
    //
    // int operationCode - число, которое нужно проверить.

    return operationCode >= LOAD_FROM_FILE && operationCode <= QUIT;
}


bool PositiveIntInputChecker(int intToCheck)
{
    // Функция для вызова в функциях ввода с проверкой.
    // ... Возвращает true, если введенное значение является положительным
    // ... числом. В настоящей программе такие значения должны принимать
    // значения вместимости и цены номера.
    //
    // int intToCheck - число, которое нужно проверить.
    return intToCheck > 0;
}


bool RoomTypeInputChecker(int roomType)
{
    // Функция для вызова в функциях ввода с проверкой.
    // ... Возвращает true, если введенное значение может быть
    // значением комфортабельности номера.
    //
    // int roomType - число, которое нужно проверить.
    return roomType >= JUNISUITE && roomType <= SUITE;
}


bool SortMenuInputChecker(int operationCode)
{
    // Функция для вызова в функциях ввода с проверкой.
    // ... Возвращает true, если введенное значение может быть
    // значением кода операции в меню сортировки.
    //
    // int operationCode - число, которое нужно проверить.
    return operationCode >= SORT_BY_PRICE &&
           operationCode <= SORT_CANCEL;
```

Продолжения листинга 1

```
}


bool FilterMenuInputChecker(int operationCode)
{
    // Функция для вызова в функциях ввода с проверкой.
    // ... Возвращает true, если введенное значение может быть
    // значением кода операции в меню настройки фильтров.
    //
    // int operationCode - число, которое нужно проверить.
    return operationCode >= FILTERS_SET && operationCode <= FILTERS_CANCEL;
}


typedef struct
{
    // Структура для хранения информации о номерах.
    int number;    // Номер комнаты
    int capacity;    // Вместимость номера
    int roomType;    // Комфортабельность номера
    int price;    // Цена номера
} Room;


typedef struct
{
    // Структура для хранения информации о фитрации номеров.
    bool isFiltered;    // Включен ли фильтр
    int minCapacity;    // Минимальная вместимость номера
    int roomType;    // Комфортабельность номера
    int maxPrice;    // Макисмальная цена номера
    int minPrice;    // Минимальная цена номера
} Filter;


typedef struct
{
    // Структура для хранения набора номеров.
    int basePrice;    // Множитель в формуле цены номера
    int baseCapacityPrice;    // Множитель в формуле цены номера
    int size;    // Количество номеров
```

Продолжение листинга 1

```
    Room* rooms;    // Массив номеров
    bool* isFilled;    // Была ли выделена память
} Hotel;


void CalculateRoomPrice(Room* objectRoom, Hotel* objectHotel)
{
    // Функция для назначения цены на номер.
    // ... Присвивает полю price указанного номера значение,
    // ... вычисленное по формуле, заданной автором произвольно
    //  с использованием множителей указанного набора номеров.
    //
    // Room* objectRoom - указатель на номер, цену которого
    // ... назначем
    // Hotel* objectHotel - указатель на набор номеров, в котором
    // ... хранятся нужные множители
    objectRoom->price = objectHotel->basePrice * objectRoom->roomType +
                        objectHotel->baseCapacityPrice * objectRoom->capacity;
}


void FreeHotel(Hotel* objectHotel)
{
    // Функция для очистки памяти, выделенной под objectHotel
    if (objectHotel->isFilled)
    {
        free(objectHotel->rooms);
        objectHotel->isFilled = false;
    }
}


void LoadHotel(FILE* saveFile, Hotel* objectHotel)
{
    // Функция для загрузки набора номеров из документа.
    // ... Работа функции в случае "повреждения" данных из-за
    // ... редактирования нештатным образом не определена.
    // ... Гарантируется, что файл, созданный функцией SaveHotel()
    // будет обработан корректно.
    //
    // FILE* saveFile - документ с данными
```

Продолжение листинга 1

```
    // Hotel* objectHotel - указатель на набор комнат, который будет заполнен
    // ... данными из документа
    FreeHotel(objectHotel);
    fscanf(saveFile, "%d", &objectHotel->basePrice);
    fscanf(saveFile, "%d", &objectHotel->baseCapacityPrice);
    fscanf(saveFile, "%d", &objectHotel->size);
    objectHotel->rooms = (Room*) malloc(objectHotel->size * sizeof(Room));
    for (int i = 0; i < objectHotel->size; i++)
    {
        Room newRoom;
        fscanf(saveFile, "%d", &newRoom.number);
        fscanf(saveFile, "%d", &newRoom.capacity);
        fscanf(saveFile, "%d", &newRoom.roomType);
        CalculateRoomPrice(&newRoom, objectHotel);
        objectHotel->rooms[i] = newRoom;
    }
    objectHotel->isFilled = true;
}


void SaveHotel(FILE* saveFile, Hotel* objectHotel)
{
    // Функция для создания файла сохранения набора номеров.
    //
    // FILE* saveFile - документ с данными (был создан или перезаписан)
    // Hotel* objectHotel - указатель на набор комнат, который будет сохранен
    // ... в документе
    fprintf(saveFile, "%d\n", objectHotel->basePrice);
    fprintf(saveFile, "%d\n", objectHotel->baseCapacityPrice);
    fprintf(saveFile, "%d\n", objectHotel->size);
    for (int i = 0; i < objectHotel->size; i++)
    {
        fprintf(saveFile, "%d\n", objectHotel->rooms[i].number);
        fprintf(saveFile, "%d\n", objectHotel->rooms[i].capacity);
        fprintf(saveFile, "%d\n", objectHotel->rooms[i].roomType);
    }
}


void PrintRoom(Room* objectRoom)
{
```

Продолжение листинга 1

```
    // Функция для форматированного вывода на экран информации о конкретной
    // комнате.
    //
    // Room* objectRoom - указатель на комнату, информацию о которой
    // ... нужно вывести.
    char roomTypeName[20];
    if (objectRoom->roomType == SUITE)
    {
        strcpy(roomTypeName, "Suite");
    }
    if (objectRoom->roomType == APARTMENT)
    {
        strcpy(roomTypeName, "Apartment");
    }
    if (objectRoom->roomType == LUX)
    {
        strcpy(roomTypeName, "Lux");
    }
    if (objectRoom->roomType == STUDIO)
    {
        strcpy(roomTypeName, "Studio");
    }
    if (objectRoom->roomType == JUNISUITE)
    {
        strcpy(roomTypeName, "Junior studio");
    }
    printf("Room #%d:\nCapacity: %d\nType: %s\nPrice: %d\n\n",
           objectRoom->number,
           objectRoom->capacity,
           roomTypeName,
           objectRoom->price);
}


void PrintHotel(Hotel* objectHotel, Filter filter)
{
    // Функция для форматированного вывода на экран количества комнат в наборе
    // и информации о каждой комнаты, подхожящей под фильтр.
    //
    // Hotel* objectHotel - указатель на набор комнат, который нужно вывести
    // Filter filter - набор фильтров для избирательного вывода информации
```

Продолжение листинга 1

```
    // ... о комнатах
    printf("Rooms amount: %d\n\n", objectHotel->size);
    for (int i = 0; i < objectHotel->size; i++)
    {
        if (!filter.isFiltered ||
            (filter.minCapacity <= objectHotel->rooms[i].capacity ||
             filter.minCapacity <= 0) &&
            (filter.roomType == objectHotel->rooms[i].roomType ||
             filter.roomType <= JUNISUITE || filter.roomType >= SUITE) &&
            (filter.maxPrice >= objectHotel->rooms[i].price ||
             filter.maxPrice <= 0) &&
            filter.minPrice <= objectHotel->rooms[i].price)
        {
            PrintRoom(&objectHotel->rooms[i]);
        }
    }
}


Room* FindRoom(Hotel* objectHotel, int roomNumber)
{
    // Функция для нахождения комнаты по номеру. Возвращает указатель на
    // найденную комнату, или NULL, если комнаты с таким номером нет.
    //
    // Hotel* objectHotel - указатель на набор комнат, в котором выполняется
    // .. поиск
    // int roomNumber - номер комнаты, которую нужно найти
    for (int i = 0; i < objectHotel->size; i++)
    {
        if (objectHotel->rooms[i].number == roomNumber)
        {
            return &objectHotel->rooms[i];
        }
    }
    return NULL;
}


void AddRoom(Hotel* objectHotel, int number, int capacity, int roomType)
{
    // Функция для добавление комнаты в набор.
```

Продолжение листинга 1

```
    // Hotel* objectHotel - указатель на набор комнат, в который добавляется
    // ... комната
    // int number - номер комнаты
    // int capacity - вместимость номера
    // int roomType - комфортабельность номера
    if (!objectHotel->isFilled)
    {
        objectHotel->size = 1;
        objectHotel->rooms = (Room*) malloc(sizeof(Room));
        objectHotel->isFilled = true;
    }
    else
    {
        if (FindRoom(objectHotel, number) != NULL)
        {
            printf("Room with such number already exists!\n");
            return;
        }
        objectHotel->size++;
        objectHotel->rooms = (Room*) realloc(objectHotel->rooms,
                                             objectHotel->size *
                                             sizeof(Room));

    }
    objectHotel->rooms[objectHotel->size - 1].number = number;
    objectHotel->rooms[objectHotel->size - 1].capacity = capacity;
    objectHotel->rooms[objectHotel->size - 1].roomType = roomType;
    CalculateRoomPrice(&objectHotel->rooms[objectHotel->size - 1],
                       objectHotel);
    printf("Room added! Price is %d\n",
           objectHotel->rooms[objectHotel->size - 1].price);
}


void DeleteRoom(Hotel* objectHotel, int number)
{
    // Функция для удаления комнаты из набора по номеру.
    //
    // Hotel* objectHotel - указатель на набор, из которого нужно
    // ... удалить комнату
    // int number - номер комнаты, которую нужно удалить
```

Продолжение листинга 1

```
    if (FindRoom(objectHotel, number) == NULL)
    {
        printf("Room with such number doesn't exist!\n\n");
        return;
    }
    Room* boxRooms;
    boxRooms = (Room*) malloc((objectHotel->size - 1) * sizeof(Room));
    int roomIsSkipped = 0;
    for (int i = 0; i < objectHotel->size; i++)
    {
        if (objectHotel->rooms[i].number != number)
        {
            boxRooms[i - roomIsSkipped] = objectHotel->rooms[i];
        }
        else
        {
            roomIsSkipped++;
        }
    }
    free(objectHotel->rooms);
    objectHotel->rooms = boxRooms;
    objectHotel->size--;
    printf("Deleted room #%d\n\n", number);
}


int PriceComparator(Room* room1, Room* room2)
{
    // Компаратор для сортировки комнат по
    // цене.
    return room1->price > room2->price;
}


int CapacityComparator(Room* room1, Room* room2)
{
    // Компаратор для сортировки комнат по
    // вместимости.
    return room1->capacity > room2->capacity;
}
```

Продолжение листинга 1

```
int TypeComparator(Room* room1, Room* room2)
{
    // Компаратор для сортировки комнат по
    // комфортабельности.
    return room1->roomType > room2->roomType;
}


int NumberComparator(Room* room1, Room* room2)
{
    // Компаратор для сортировки комнат по
    // номеру.
    return room1->number > room2->number;
}


int main()
{
    Hotel object;
    object.basePrice = BASE_PRICE;
    object.baseCapacityPrice = BASE_CAPACITY_PRICE;
    object.isFilled = false;

    Filter filter;
    filter.isFiltered = false;

    int operationCode;
    while (true)
    {
        int subOperationCode;
        if (!object.isFilled)
        {
            printf("\nNO DATABASE\nLoad it "
                   "or start creating new rooms.");
        }
        if (filter.isFiltered)
        {
            printf("\nFILTERED\n");
        }
        printf("\n1. Load data from savefile.\n"
```

Продолжение листинга 1

```
            "2. Save current data to savefile.\n"
            "3. Add new room.\n"
            "4. Delete existing room.\n"
            "5. Print data with filters.\n"
            "6. Sort database.\n"
            "7. Change filter settings.\n"
            "8. Get room info.\n"
            "9. Quit without saving.\n\n");
    operationCode = CycleInputInt(
            "Choose the command and enter its number",
            MainMenuInputChecker);

    // Загрузка базы из документа "savefile.txt"
    if (operationCode == LOAD_FROM_FILE)
    {
        FILE* savefile;
        savefile = fopen("savefile.txt", "r");
        if (savefile == NULL)
        {
            printf("Couldn't open savefile!\n");
        }
        else
        {
            LoadHotel(savefile, &object);
            printf("Loaded!\n");
        }
        fclose(savefile);
    }

    // Сохранение текущей базы данных в документ "savefile.txt"
    if (operationCode == SAVE_TO_FILE)
    {
        if (object.isFilled)
        {
            FILE* savefile;
            savefile = fopen("savefile.txt", "w");
            if (savefile == NULL)
            {
                printf("Couldn't write new data!\n");
            }
            else
```

Продолжение листинга 1

```
            {
                SaveHotel(savefile, &object);
            }
            fclose(savefile);
        }
        else
        {
            printf("NO DATABASE\n\n");
        }
    }

    // Добавление комнаты
    if (operationCode == ADD_ROOM)
    {
        int number;
        number = CycleInputInt("Enter room's number",
                               PositiveIntInputChecker);
        int capacity;
        capacity = CycleInputInt("Enter room's capacity",
                                 PositiveIntInputChecker);
        int roomType;
        roomType = CycleInputInt("Choose room's type:\n"
                                 "1. Junior suite.\n"
                                 "2. Studio.\n"
                                 "3. Lux.\n"
                                 "4. Apartment.\n"
                                 "5. Suite.\n",
                                 RoomTypeInputChecker);
        AddRoom(&object, number, capacity, roomType);
    }

    // Удаление комнаты
    if (operationCode == DELETE_ROOM)
    {
        if (object.isFilled)
        {
            int room;
            room = CycleInputInt("Enter room's number",
                                 PositiveIntInputChecker);
            DeleteRoom(&object, room);
        }
```

Продолжение листинга 1

```
        else
        {
            printf("NO DATABASE\n\n");
        }
    }

    // Вывод набора комнат с учётом фильтров
    if (operationCode == PRINT)
    {
        if (object.isFilled)
        {
            PrintHotel(&object, filter);
        }
        else
        {
            printf("NO DATABASE\n\n");
        }
    }

    // Вход в меню сортировки
    if (operationCode == SORT)
    {
        subOperationCode = CycleInputInt("\n1. Sort by price."
                                         "\n2. Sort by capacity."
                                         "\n3. Sort by type."
                                         "\n4. Sort by room's number."
                                         "\n5. Back.\n",
                                         SortMenuInputChecker);

        // Сортировка по цене
        if (subOperationCode == SORT_BY_PRICE)
        {
            qsort(object.rooms, object.size, sizeof(Room),
                  (int (*)(const void*, const void*)) PriceComparator);
        }

        // Сортировка по вместимости
        if (subOperationCode == SORT_BY_CAPACITY)
        {
            qsort(object.rooms, object.size, sizeof(Room),
                  (int (*)(const void*, const void*)) CapacityComparator);
```

Продолжение листинга 1

```
        }

        // Сортирока по комфортабельности
        if (subOperationCode == SORT_BY_TYPE)
        {
            qsort(object.rooms, object.size, sizeof(Room),
                  (int (*)(const void*, const void*)) TypeComparator);
        }

        // Сортировка по номеру
        if (subOperationCode == SORT_BY_NUMBER)
        {
            qsort(object.rooms, object.size, sizeof(Room),
                  (int (*)(const void*, const void*)) NumberComparator);
        }

        // Выход из меню сортировки без действий
        if (subOperationCode == SORT_CANCEL)
        {
            printf("Sorting cancelled!\n\n");
            continue;
        }

        // Вывод отсортированной базы данных
        printf("Sorted!\n");
        PrintHotel(&object, filter);
    }

    // Вход в меню настройки фильтров
    if (operationCode == FILTERS)
    {
        subOperationCode = CycleInputInt("1. Set new filters."
                                         "\n2. Reset all filters."
                                         "\n3. Back.",
                                         FilterMenuInputChecker);

        // Установка нового набора фильтра
        if (subOperationCode == FILTERS_SET)
        {
            filter.minCapacity = CycleInputInt(
                    "Enter room's minimal capacity"
```

Продолжение листинга 1

```
                    "\nEnter number < 1 to skip",
                    NULL);
            filter.roomType = CycleInputInt("\nChoose room's type:"
                                            "\n1. Junior suite."
                                            "\n2. Studio."
                                            "\n3. Lux."
                                            "\n4. Apartment."
                                            "\n5. Suite."
                                            "\n Different number to"
                                            " skip",
                                            NULL);

            filter.maxPrice = CycleInputInt(
                    "Enter room's maximum price"
                    "\nEnter number < 1 to skip",
                    NULL);
            filter.minPrice = CycleInputInt(
                    "Enter room's minimum price"
                    "\nEnter number < 1 to skip",
                    NULL);

            // Фиксация факта включения фильтров, функция вывода теперь
            // будет проходить этап фильтрации
            filter.isFiltered = true;
        }

        // Отключение фильтрации
        if (subOperationCode == FILTERS_RESET)
        {
            // Для отключения фильтрации достаточно просто задать
            // этой переменной false. Т.к. при этом этап фильтрации будет
            // пропускаться функцией вывода. Если фильтры понадобится
            // включить - значения остальныйх полей Filter filter в любом
            // случае будут заданы заново.
            filter.isFiltered = false;
            printf("\nFilters are reset!\n");
        }
    }

    // Вывод информации о комнате по её номеру
    if (operationCode == FIND_BY_NUMBER)
```

Окончание листинга 1

```
        {
            int number;
            number = CycleInputInt("Enter room's number",
                                   PositiveIntInputChecker);
            Room* result;
            result = FindRoom(&object, number);
            if (result == NULL)
            {
                printf("\nNo such room!\n\n");
            }
            else
            {
                PrintRoom(result);
            }

        }

        // Выход из программы
        if (operationCode == QUIT)
        {
            break;
        }
    }
    // Очистка памяти
    FreeHotel(&object);
    return 0;
```

5 Результат

Ниже представлены скриншоты с консольным выводом.

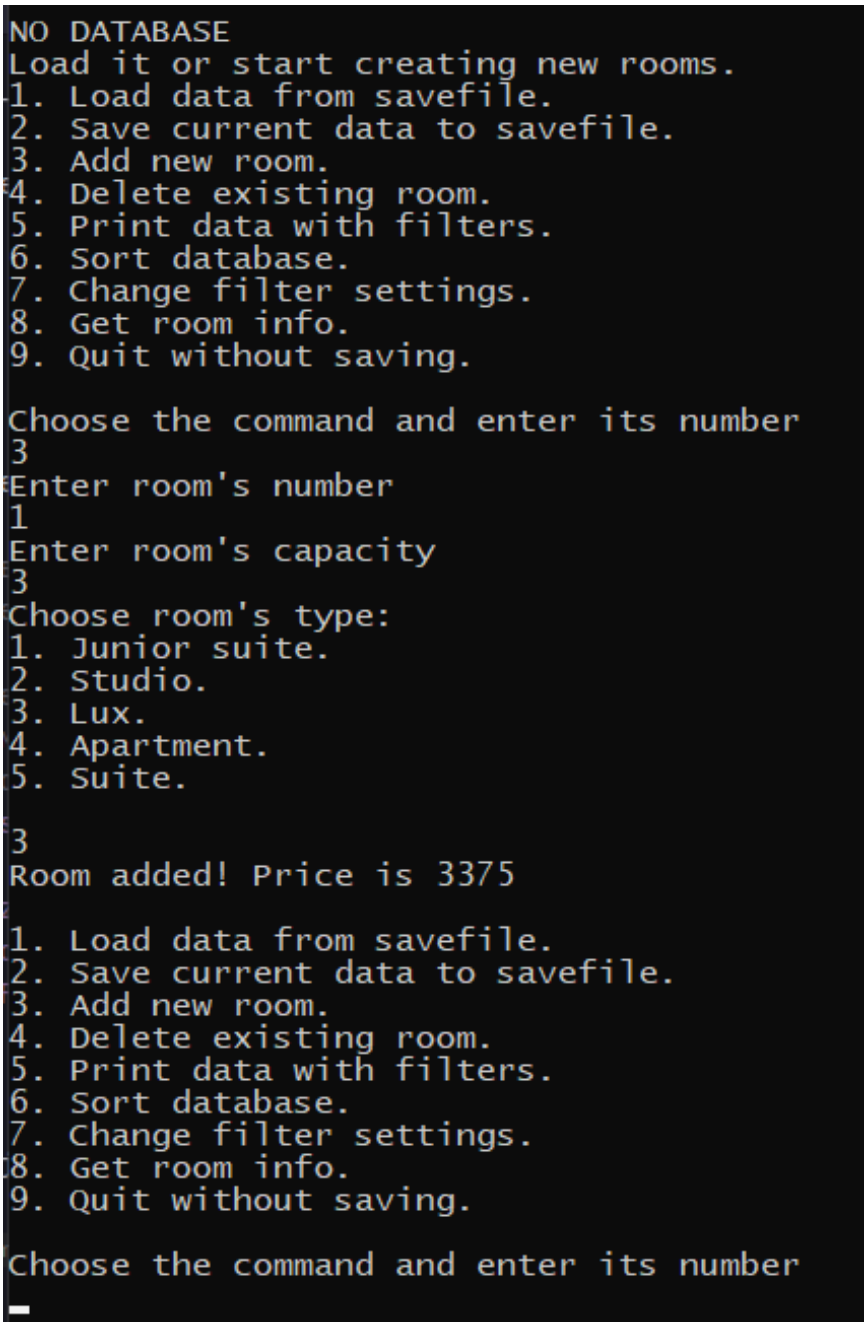

Рисунок 1 – Создание новой комнаты

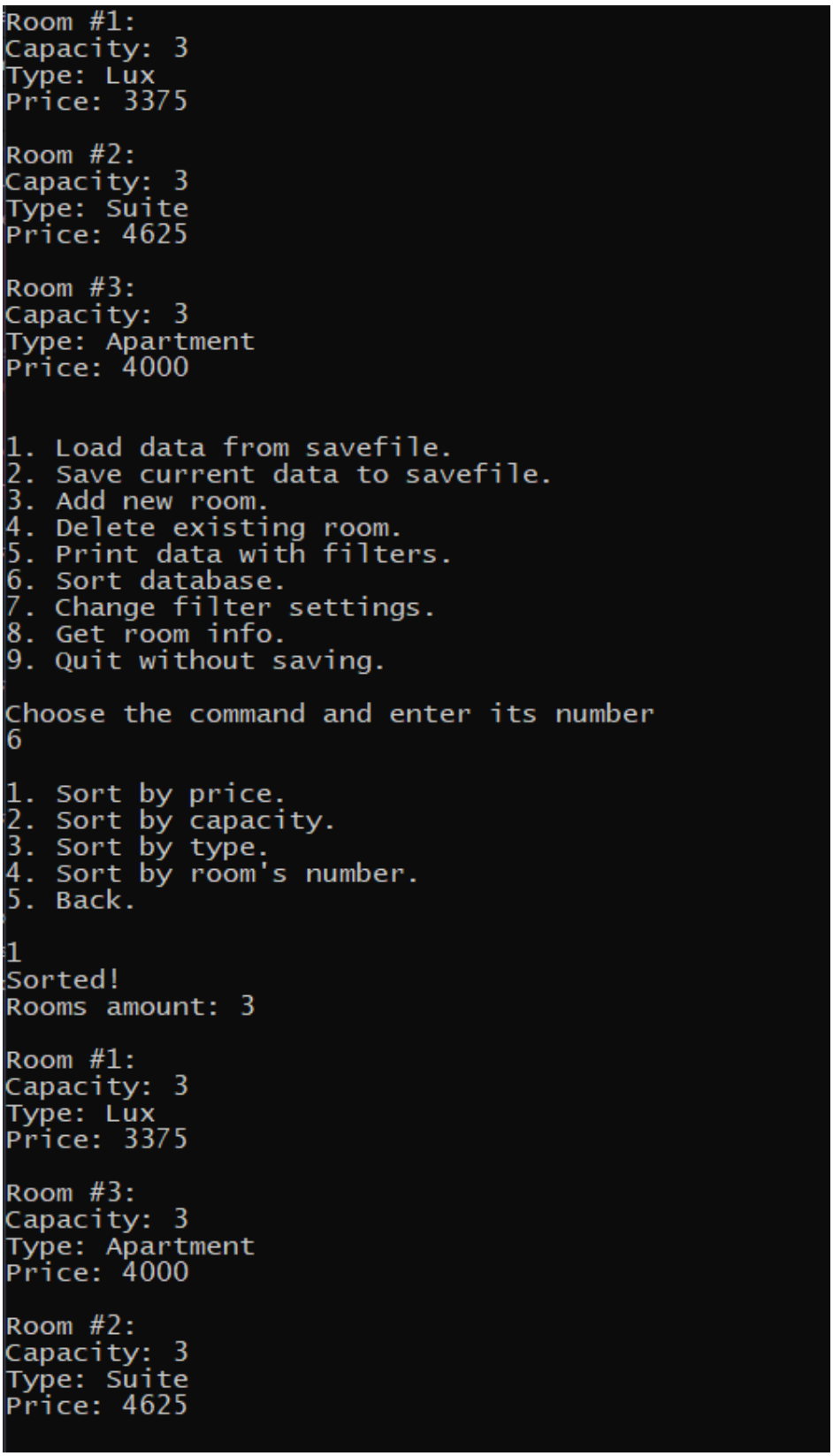

Рисунок 2 – Сортировка комнат по возрастанию цены

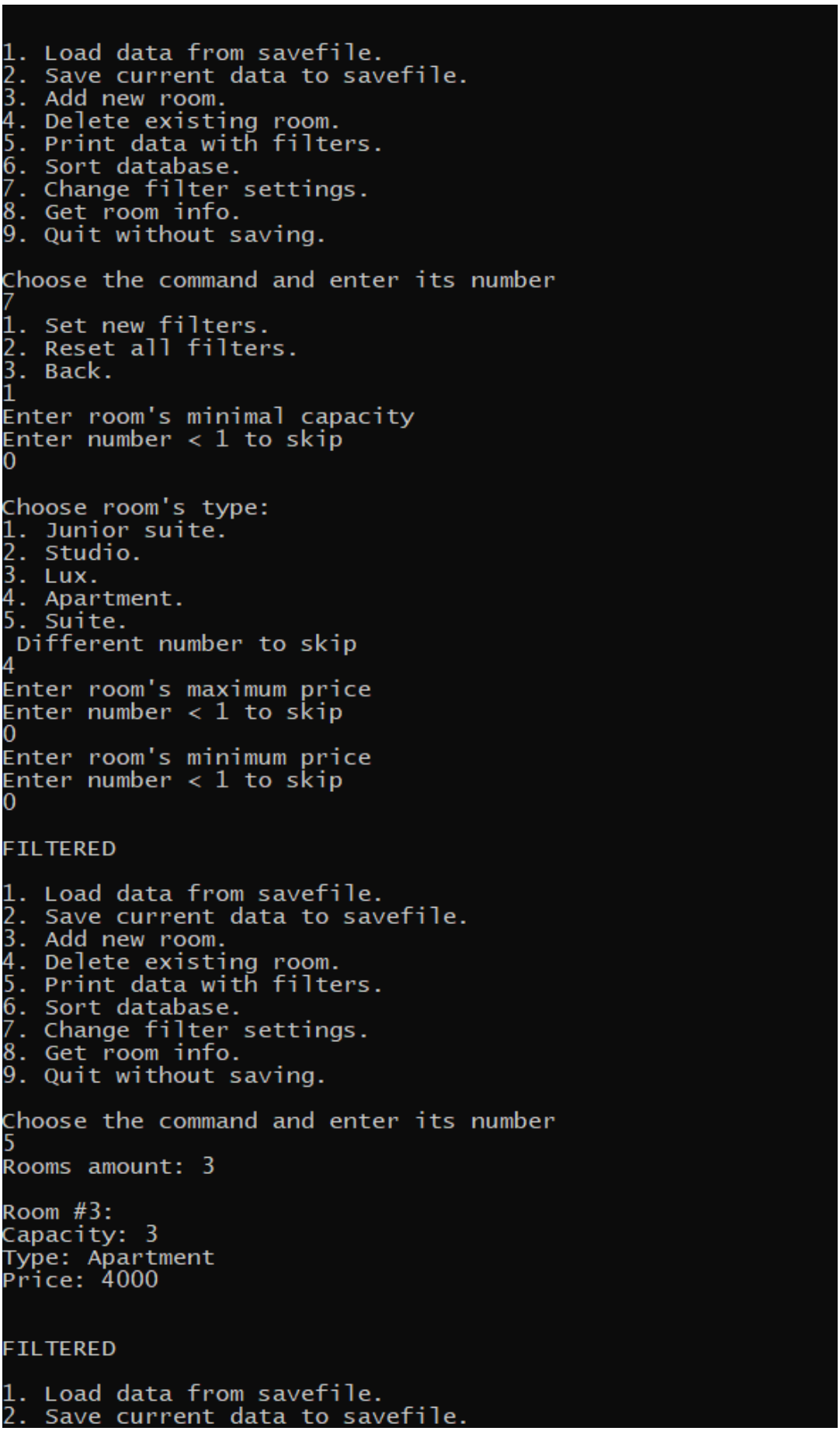

Рисунок 3 – Вывод отфильтрованного списка комнат

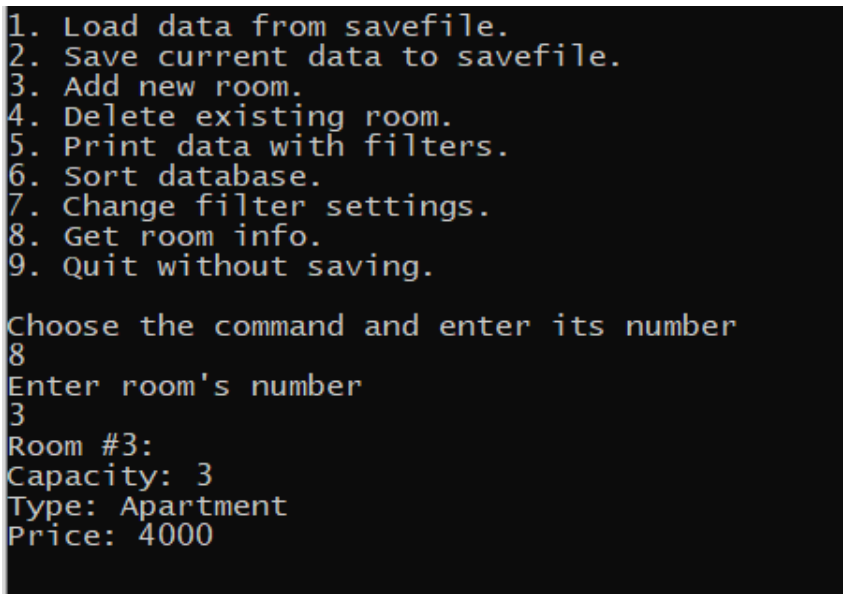

Рисунок 4 – Вывод информации об одной комнате по номеру

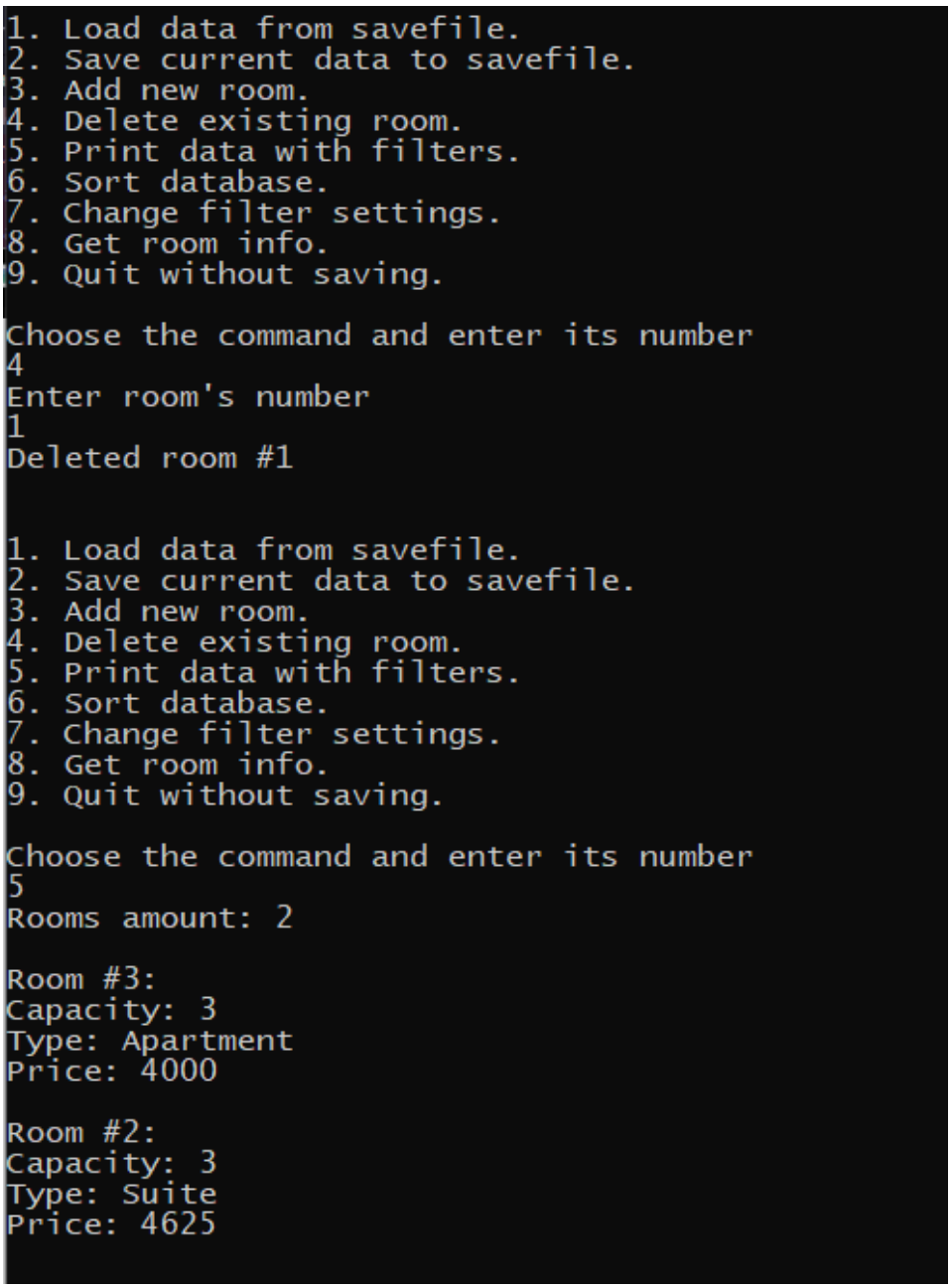

Рисунок 5 – Удаление комнаты по номеру

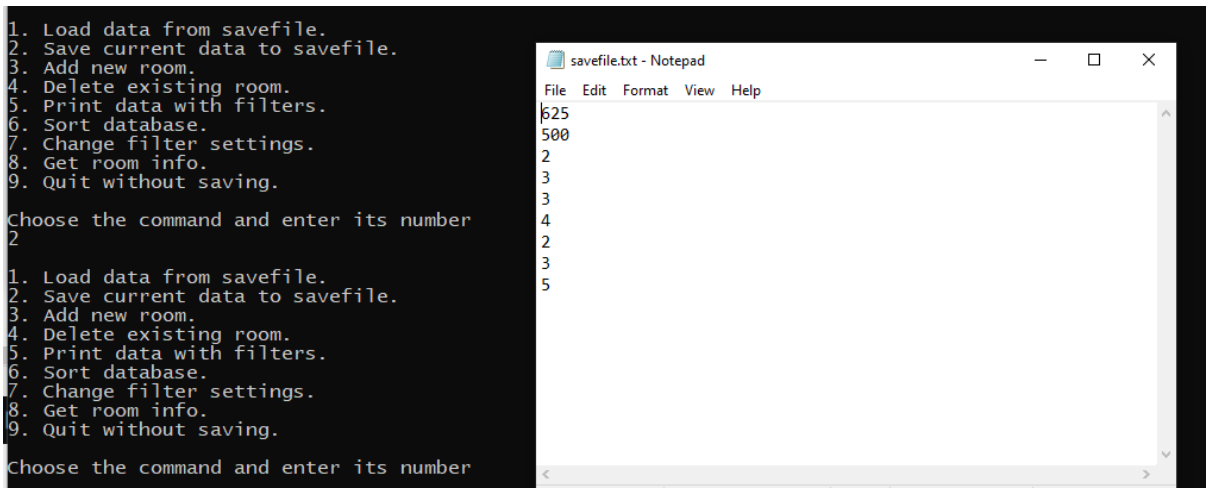

Рисунок 6 – Сохранение в файл

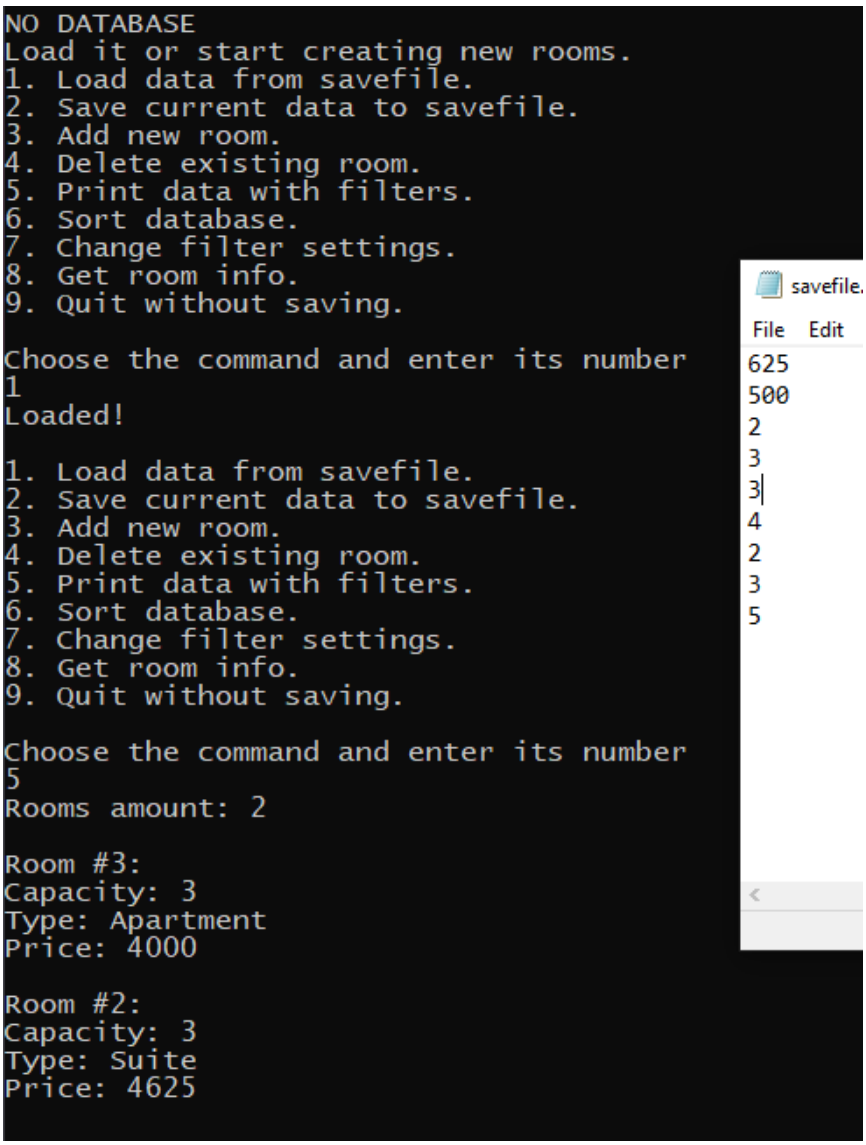

Рисунок 7 – Загрузка из файла

6 Выводы

Была написана программа, соответствующая поставленным задачам. В ходе работы были изучены некоторые аспекты синтаксиса языка программирования Си, в частности – принципы работы со структурами. Код был отформатирован в соответствии со стандартом, принятым в учебном заведении.