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

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6

Связные списки
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 с связными списками.

2 Задачи

Для выполнения лабораторной работы необходимо, взяв за основу задание из лабораторной работы №5, выполнить следующие задачи:

На оценку 3 балла:

1) написать программу в соответствии с заданием;
2) заменить массив структур односвязным списком;

На оценку 4 балла:

1) выполнить требования предыдущих пунктов;
2) заменить массив структур двусвязным списком;

На оценку 5 баллов:

1) выполнить требования предыдущих пунктов;
2) добавить функцию определения длины списка;
3) добавить функцию инвертирования списка.

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,
    REVESE = 9,
    QUIT = 10,
    // Команды для меню сортировки.
```

Продолжение листинга 1

```
    SORT_BY_PRICE = 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;
```

Продолжение листинга 1

```
    // bool(* pChecker)(int) - указатель на функцию, проверяющую
    // ... 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)
{
```

Продолжение листинга 1

```
    // Функция для вызова в функциях ввода с проверкой.
    // 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 &&
```

Продолжение листинга 1

```
           operationCode <= SORT_CANCEL;
}


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 ListLink
{
    // Структура для хранения двусвязного списка комнат.
    struct ListLink* previous;    // Указатель на предыдущий элемент
    struct ListLink* next;    // Указатель на следующий элемент
```

Продолжение листинга 1

```
    Room* roomData;    // Указатель на информацию о комнате
} ListLink;


ListLink* ListLinkHead(ListLink* objectLink)
{
    // Функция для поиска первого элемента списка. Возвращает указатель
    // на первый элемент.
    //
    // ListLink* objectLink - элемент списка, первый элемент которого
    // ... необходимо найти
    ListLink* position;
    position = objectLink;
    if (position == NULL)
    {
        return NULL;
    }
    while (position->previous != NULL)
    {
        position = position->previous;
    }
    return position;
}


ListLink* ListLinkTail(ListLink* objectLink)
{
    // Функция для поиска последнего элемента списка. Возвращает указатель
    // на последний элемент.
    //
    // ListLink* objectLink - элемент списка, последний элемент которого
    // ... необходимо найти
    ListLink* position;
    position = objectLink;
    if (position == NULL)
    {
        return NULL;
    }
    while (position->next != NULL)
    {
        position = position->next;
```

Продолжение листинга 1

```
    }
    return position;
}


void ListLinkSwap(ListLink* firstLink, ListLink* secondLink)
{
    // Функция для замены местами двух элементов списка.
    //
    // ListLink* firstLink - элемент списка, который нужно поменять с другим
    // ListLink* secondLink - элемент списка, который нужно поменять с другим
    Room* roomDataBox;
    roomDataBox = firstLink->roomData;
    firstLink->roomData = secondLink->roomData;
    secondLink->roomData = roomDataBox;
}


int ListLinkSize(ListLink* objectLink)
{
    // Функция для нахождения количества элементов списка.
    // Возвращает количество элементов списка.
    //
    // ListLink* objectLink - элемент списка, размер которого
    // ... необходимо найти
    ListLink* position;
    if (objectLink == NULL)
    {
        return 0;
    }
    position = ListLinkHead(objectLink);
    int size;
    size = 1;
    while (position->next != NULL)
    {
        size++;
        position = position->next;
    }
    return size;
}
```

Продолжение листинга 1

```
void ListLinkBubbleSort(ListLink* objectLink, int(* comparator)(Room*, Room*))
{
    // Функция, сортирующая методом пузырьков двусвязный список.
    //
    // ListLink* objectLink - элемент списка, который нужно отсортировать
    // int(* comparator)(Room*, Room*) - функция-компоратор по аналогии
    // ... с функцией qsort()
    ListLink* position;
    position = NULL;
    ListLink* index;
    index = NULL;
    for (position = ListLinkHead(objectLink);
         position->next != NULL; position = position->next)
    {
        for (index = position->next;
             index != NULL; index = index->next)
        {
            if (comparator(index->roomData, position->roomData))
            {
                ListLinkSwap(position, index);
            }
        }
    }
}


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
    ListLink* rooms;    // Указатель на головной элемент списка номеров
} Hotel;


void ListLinkReverse(Hotel* objectHotel)
{
    // Функция для инвертирования списка.
    //
```

Продолжение листинга 1

```
    // ListLink* objectLink - элемент списка, который необходимо
    // ... инвертировать;
    ListLink* position;
    position = ListLinkHead(objectHotel->rooms);
    ListLink* box;
    while (position != NULL)
    {
        box = position->previous;
        position->previous = position->next;
        position->next = box;
        position = position->previous;
    }
    if (box != NULL)
    {
        objectHotel->rooms = box->previous;
    }
}


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


void PrintRoom(Room* objectRoom)
{
    // Функция для форматированного вывода на экран информации о конкретной
    // комнате.
    //
    // Room* objectRoom - указатель на комнату, информацию о которой
```

Продолжение листинга 1

```
    // ... 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
    // ... о комнатах;
    printf("Rooms amount: %d\n\n", objectHotel->size);
    ListLink* position;
    position = objectHotel->rooms;
```

Продолжение листинга 1

```
    while (position != NULL)
    {
        if (!filter.isFiltered ||
            (filter.minCapacity <= position->roomData->capacity ||
             filter.minCapacity <= 0) &&
            (filter.roomType == position->roomData->roomType ||
             filter.roomType <= JUNISUITE || filter.roomType >= SUITE) &&
            (filter.maxPrice >= position->roomData->price ||
             filter.maxPrice <= 0) &&
            filter.minPrice <= position->roomData->price)
        {
            PrintRoom(position->roomData);
        }
        position = position->next;
    }
}


ListLink* 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;
    ListLink* position;
    position = objectHotel->rooms;
    while (position != NULL)
    {
        if (position->roomData->number == roomNumber)
        {
            return position;
        }
        position = position->next;
    }
    return NULL;
}


int AddRoom(Hotel* objectHotel, int number, int capacity, int roomType)
```

Продолжение листинга 1

```
{
    // Функция для добавление комнаты в набор.
    // Hotel* objectHotel - указатель на набор комнат, в который добавляется
    // ... комната
    //
    // int number - номер комнаты;
    // int capacity - вместимость номера;
    // int roomType - комфортабельность номера;
    if (FindRoom(objectHotel, number) != NULL)
    {
        printf("Attention, there is a room with such # already!\n");
    }
    Room* room;
    room = (Room*) malloc(sizeof(Room));
    room->roomType = roomType;
    room->number = number;
    room->capacity = capacity;
    CalculateRoomPrice(objectHotel, room);

    ListLink* linkToAdd;
    linkToAdd = (ListLink*) malloc(sizeof(ListLink));
    linkToAdd->next = NULL;
    linkToAdd->previous = ListLinkTail(objectHotel->rooms);
    linkToAdd->roomData = room;

    if (linkToAdd->previous == NULL)
    {
        objectHotel->rooms = linkToAdd;
    }
    else
    {
        linkToAdd->previous->next = linkToAdd;
    }
    objectHotel->size = ListLinkSize(objectHotel->rooms);
    return linkToAdd->roomData->price;
}


void DeleteRoom(Hotel* objectHotel, int number)
{
    // Функция для удаления комнаты из набора по номеру.
```

Продолжение листинга 1

```
    //
    // Hotel* objectHotel - указатель на набор, из которого нужно
    // ... удалить комнату;
    // int number - номер комнаты, которую нужно удалить;
    ListLink* roomToDelete;
    roomToDelete = FindRoom(objectHotel, number);
    if (roomToDelete == NULL)
    {
        printf("Room with such number doesn't exist!\n\n");
        return;
    }

    if (roomToDelete->previous != NULL)
    {
        roomToDelete->previous->next = roomToDelete->next;
    }
    else
    {
        objectHotel->rooms = roomToDelete->next;
    }
    if (roomToDelete->next != NULL)
    {
        roomToDelete->next->previous = roomToDelete->previous;
    }
    objectHotel->size = ListLinkSize(objectHotel->rooms);
    free(roomToDelete->roomData);
    free(roomToDelete);
    printf("Deleted room #%d\n\n", number);
}


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;
    // Hotel* objectHotel - указатель на набор комнат, который будет заполнен
```

Продолжение листинга 1

```
    // ... данными из документа;
    int inputRoomNumber;
    fscanf(saveFile, "%d", &objectHotel->basePrice);
    fscanf(saveFile, "%d", &objectHotel->baseCapacityPrice);
    fscanf(saveFile, "%d", &inputRoomNumber);
    for (int i = 0; i < inputRoomNumber; i++)
    {
        int number;
        int capacity;
        int roomType;
        fscanf(saveFile, "%d", &number);
        fscanf(saveFile, "%d", &capacity);
        fscanf(saveFile, "%d", &roomType);
        AddRoom(objectHotel, number, capacity, roomType);
    }
}


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);
    ListLink* position;
    position = objectHotel->rooms;
    while (position != NULL)
    {
        fprintf(saveFile, "%d\n", position->roomData->number);
        fprintf(saveFile, "%d\n", position->roomData->capacity);
        fprintf(saveFile, "%d\n", position->roomData->roomType);
        position = position->next;
    }
}


void ListLinkFree(ListLink* objectLink)
```

Продолжение листинга 1

```
{
    // Функция для очистки памяти, занятой под информацию о комнатах.
    //
    // ListLink* objectLink - элемент списка, памяьб для хранения
    // ... которого необходимо очистить;
    if(objectLink!=NULL)
    {
        free(objectLink->roomData);
        if (objectLink->next != NULL)
        {
            ListLinkFree(objectLink->next);
        }
        if (objectLink->previous != NULL)
        {
            ListLinkFree(objectLink->previous);
        }
        free(objectLink);
    }
}


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;
}


int TypeComparator(Room* room1, Room* room2)
{
    // Компаратор для сортировки комнат по
    // комфортабельности.
```

Продолжение листинга 1

```
    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.rooms = NULL;
    object.size = 0;

    Filter filter;
    filter.isFiltered = false;

    int operationCode;
    while (true)
    {
        int subOperationCode;
        if (object.size == 0)
        {
            printf("\nHotel is empty!\nLoad it "
                   "or start creating new rooms.");
        }
        if (filter.isFiltered)
        {
            printf("\nFILTERED!\n");
        }
        printf("\n1. Load data from savefile.\n"
               "2. Save current data to savefile.\n"
               "3. Add new room.\n"
               "4. Delete existing room.\n"
               "5. Print data with filters.\n"
```

Продолжение листинга 1

```
            "6. Sort database.\n"
            "7. Change filter settings.\n"
            "8. Get room info.\n"
            "9. Reverse list.\n"
            "10. 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)
    {

        FILE* savefile;
        savefile = fopen("savefile.txt", "w");
        if (savefile == NULL)
        {
            printf("Couldn't write new data!\n");
        }
        else
        {
            SaveHotel(savefile, &object);
            printf("Saved!\n");
        }
```

Продолжение листинга 1

```
        fclose(savefile);
    }

    // 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);
        int price;
        price = AddRoom(&object, number, capacity, roomType);
        printf("Room added! Price is %d\n", price);

    }

    // Удаление комнаты
    if (operationCode == DELETE_ROOM)
    {

        int room;
        room = CycleInputInt("Enter room's number",
                             PositiveIntInputChecker);
        DeleteRoom(&object, room);
    }

    // Вывод набора комнат с учётом фильтров
    if (operationCode == PRINT)
    {
        PrintHotel(&object, filter);
    }
```

Продолжение листинга 1

```
// 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)
    {
        ListLinkBubbleSort(object.rooms, PriceComparator);
    }

    // Сортировка по вместимости
    if (subOperationCode == SORT_BY_CAPACITY)
    {
        ListLinkBubbleSort(object.rooms, CapacityComparator);
    }

    // Сортирока по комфортабельности
    if (subOperationCode == SORT_BY_TYPE)
    {
        ListLinkBubbleSort(object.rooms, TypeComparator);
    }

    // Сортировка по номеру
    if (subOperationCode == SORT_BY_NUMBER)
    {
        ListLinkBubbleSort(object.rooms, NumberComparator);
    }

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

Продолжение листинга 1

```
        // 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"
                    "\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
```

Продолжение листинга 1

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

    }

    // Инверсия списка
    if (operationCode == REVESE)
    {
        ListLinkReverse(&object);
        printf("Reversed!\n");
```

Окончание листинга 1

```
            PrintHotel(&object, filter);
        }

        // Выход из программы
        if (operationCode == QUIT)
        {
            break;
        }
    }
    ListLinkFree(object.rooms);
    return 0;
}
```

5 Результат

Ниже представлены скриншоты с консольным выводом.

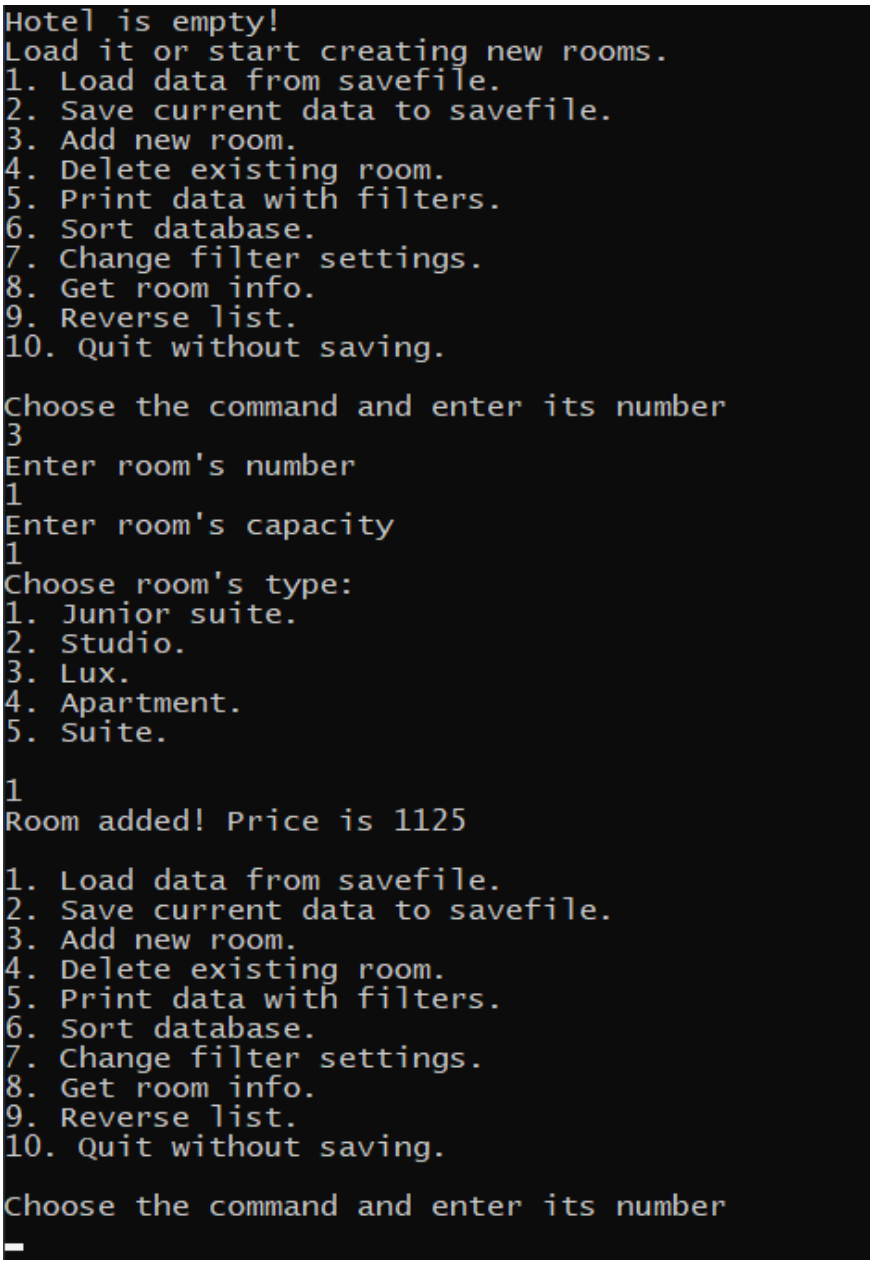

Рисунок 1 – Создание новой комнаты

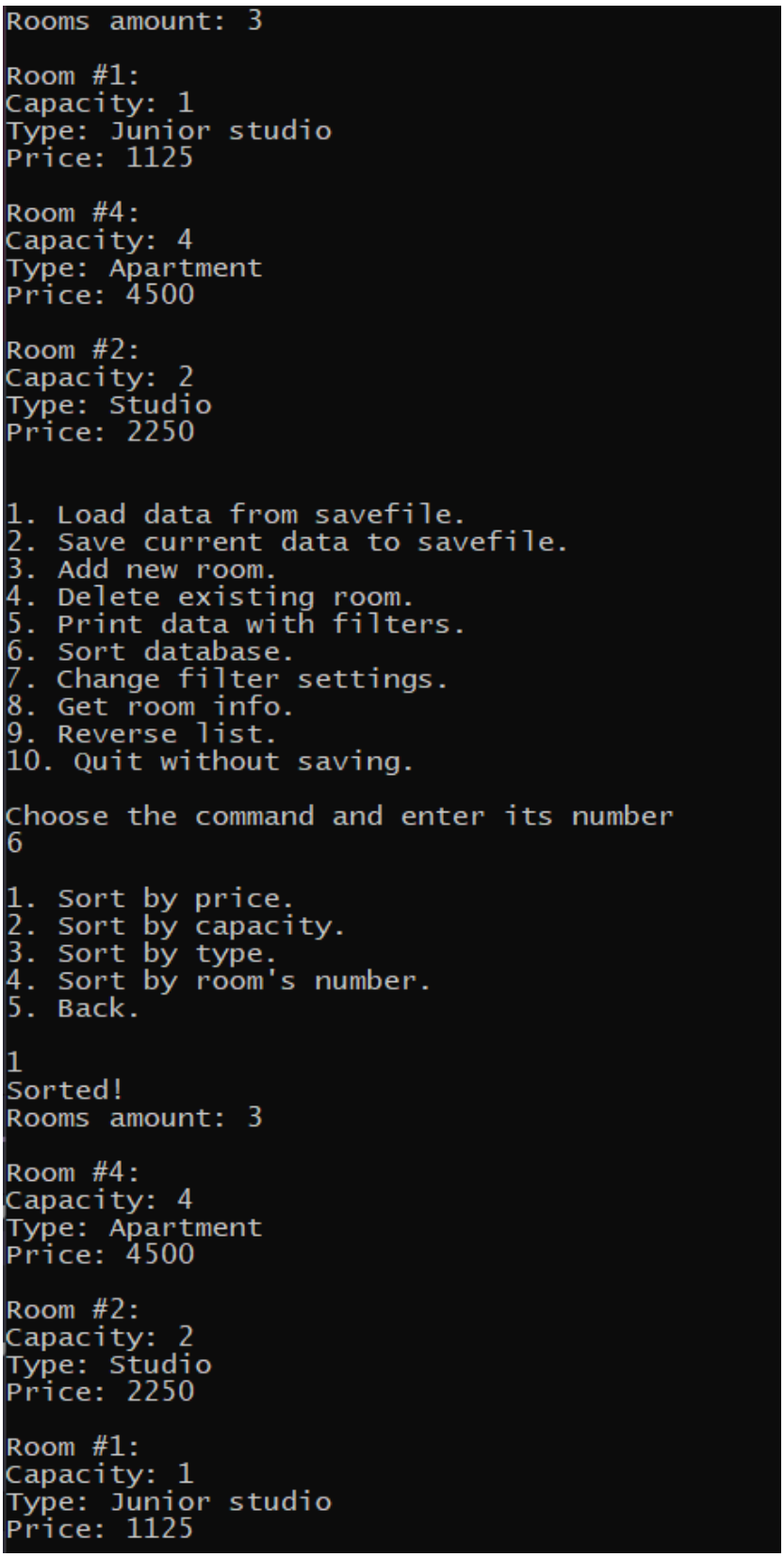

Рисунок 2 – Сортировка комнат по возрастанию цены

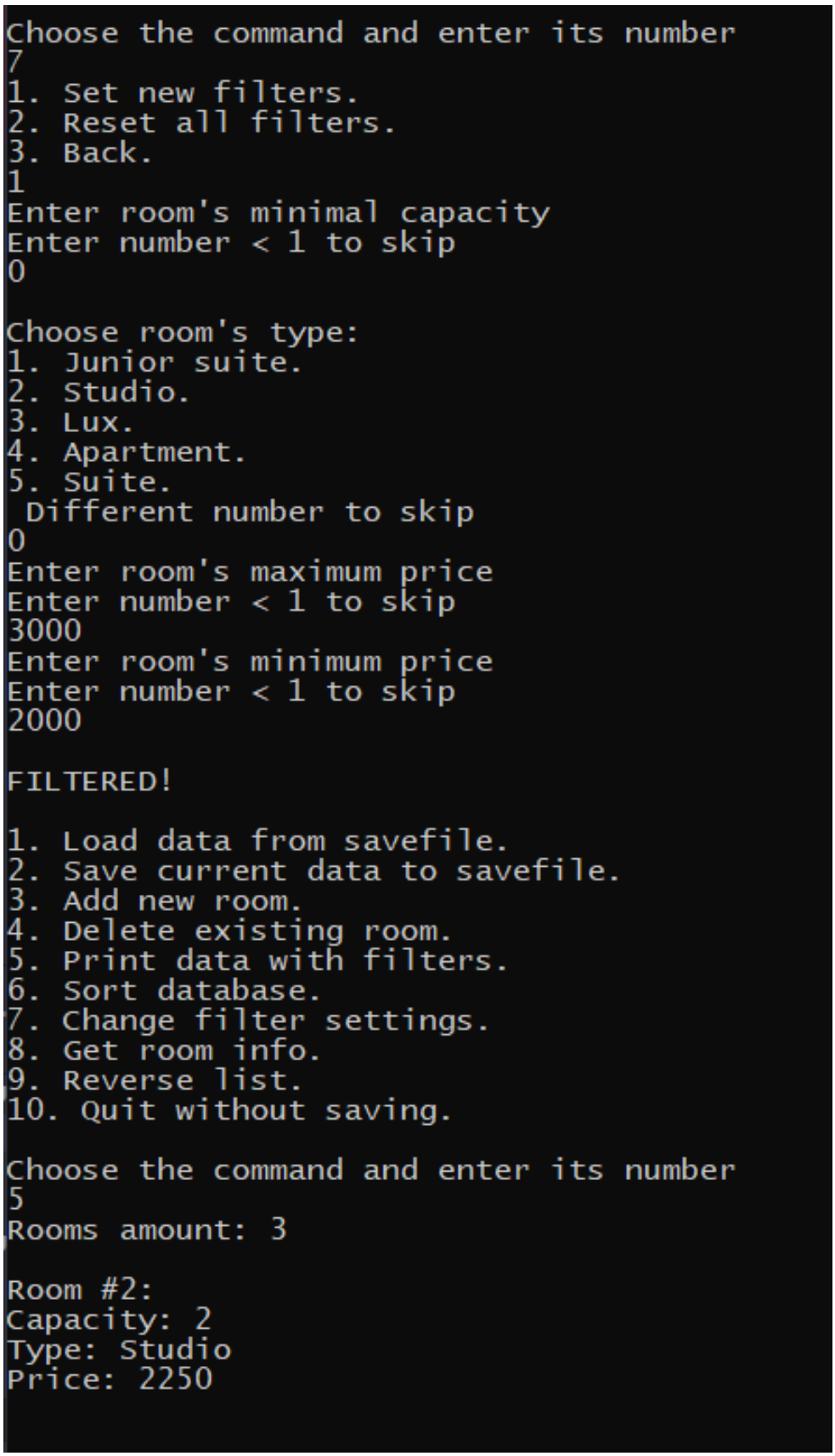

Рисунок 3 – Вывод отфильтрованного списка комнат

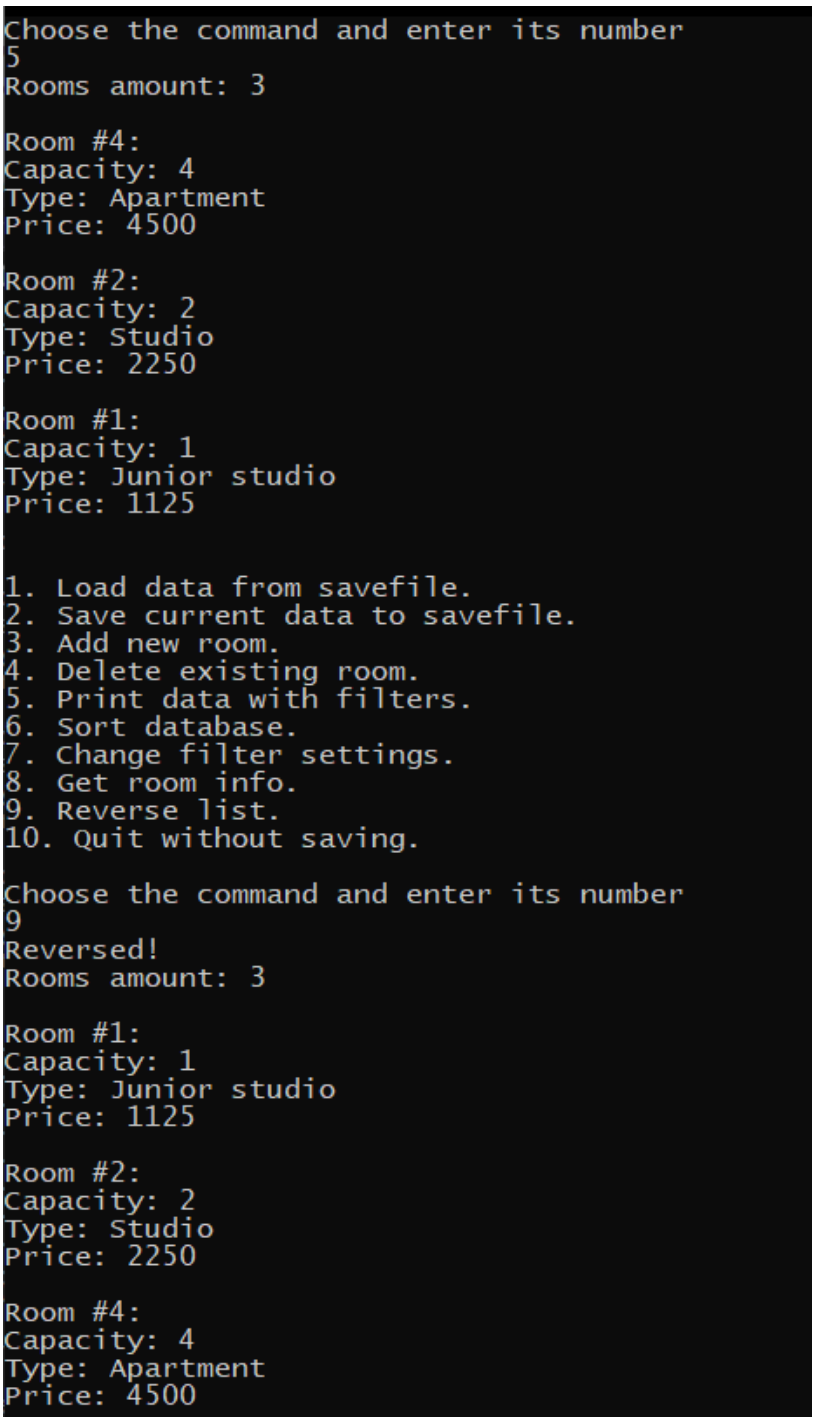

Рисунок 4 – Инвертирование списка

6 Выводы

Была написана программа, соответствующая поставленным задачам. В ходе работы была изучена такая базовая структура данных в информатике, как связный список, а также изучены некоторые аспекты синтаксиса языка программирования Си при работе с такими структурами. Код был отформатирован в соответствии со стандартом, принятым в учебном заведении.