**Отчет по лабораторной работе №2**

**по теме «Записи с вариантами, обработка таблиц»**

Выполнила:  
Лаврова Анастасия

Группа ИУ7-35Б

Вариант 11

Задача №3

**Цель работы:**

Приобрести навыки работы с типом данных «запись» (структура), содержащим вариантную часть (объединение, смесь), и с данными, хранящимися в таблицах, произвести сравнительный анализ реализации алгоритмов сортировки и поиска информации в таблицах, при использовании записей с большим числом полей, и тех же алгоритмов, при использовании таблицы ключей; оценить эффективность программы по времени и по используемому объему памяти; оценить эффективность использования различных алгоритмов сортировок.

**Условие задачи:**

Создать таблицу, содержащую не менее 40 записей с вариантной частью. Произвести поиск информации по вариантному полю. Упорядочить таблицу, по возрастанию ключей (где ключ – любое невариантное поле по выбору программиста), используя: а) исходную таблицу; б) массив ключей, используя 2 разных алгоритма сортировки. Оценить эффективность этих алгоритмов (по времени и по используемому объему памяти) при различной реализации программы, то есть в случаях а) и б). Обосновать выбор алгоритма сортировки. Оценка эффективности должна быть относительной (в %).

Имеются описания:

Type жилье = (дом, общежитие);

Данные:

имя, пол (м, ж), возраст, средний балл за сессию, дата поступления

адрес:

дом : (улица, №дома, №кв );

общежитие : (№общ., №комн.);

Ввести общий список студентов группы. Вывести список студентов, живущих в общежитии указанного года поступления.

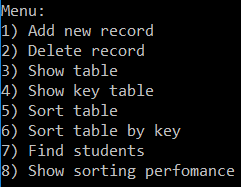
**Входные данные:**

Программа получает на вход файл с данными

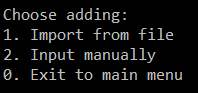
**Выходные данные:**

Программа может вывести: неотсортированную таблицу с данными из файла, отсортированную таблицу с данными из файла, таблицу ключей, отсортированную таблицу ключей, отобразить интересующих студентов (живущих в общежитии указанного года поступления)

Техническое задание:



1. Ввод данных в таблицу

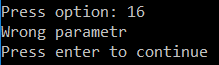


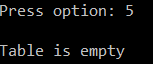
* Импортировать данные из файла
* Ввести данные вручную
* Вернуться в главное меню

1. Удалить запись по индексу
2. Отобразить таблицу
3. Отобразить таблицу ключей
4. Отсортировать таблицу данных
5. Отсортировать таблицу ключей
6. Найти интересующих студентов
7. Показать эффективность всех сортировок по памяти и по времени

**Аварийные ситуации**:

Ввод неправильного параметра меню:



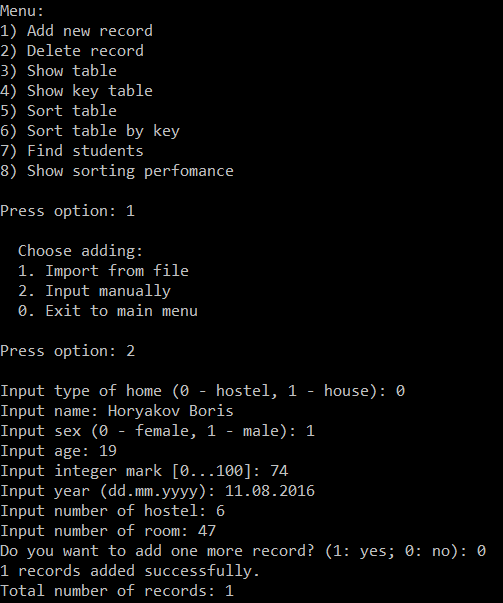
Действия с пустой таблицей:  


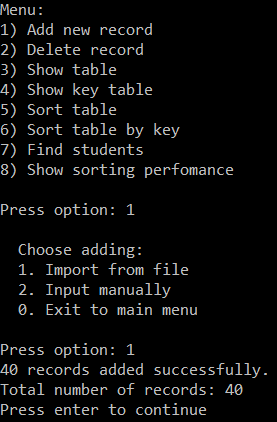
Удаление несуществующего элемента:  


Выход за границы допустимого диапазона:  

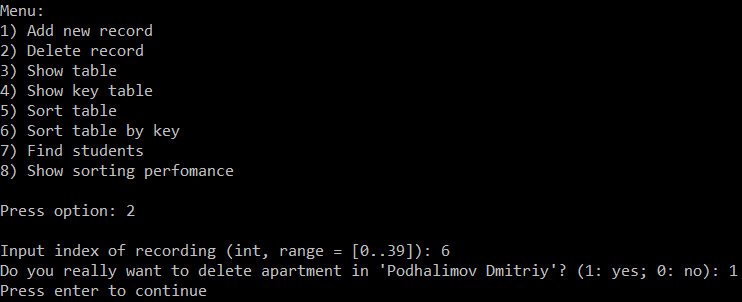

Пустой ввод:  


Ввод недопустимого типа в значение типа Integer:  

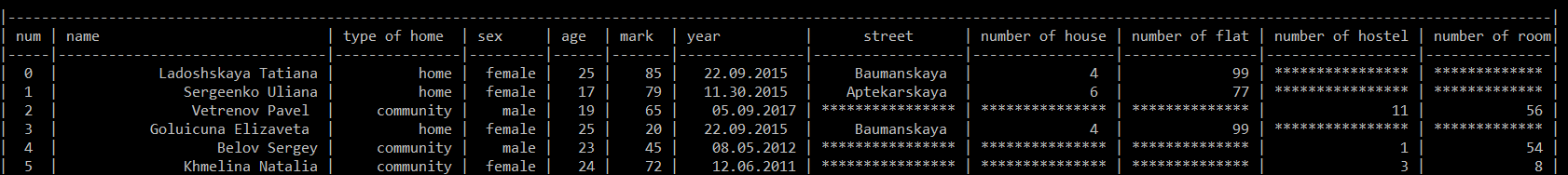

**Тесты:**  
Ввод значений  




Удаление записи:



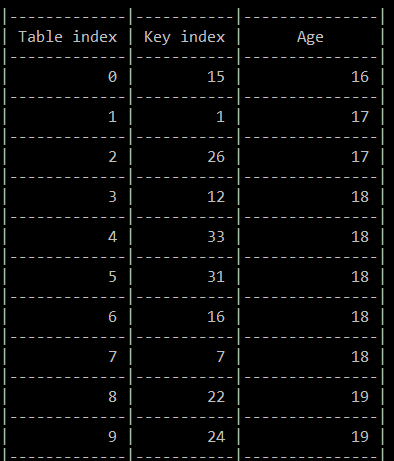
3) Отображение таблицы



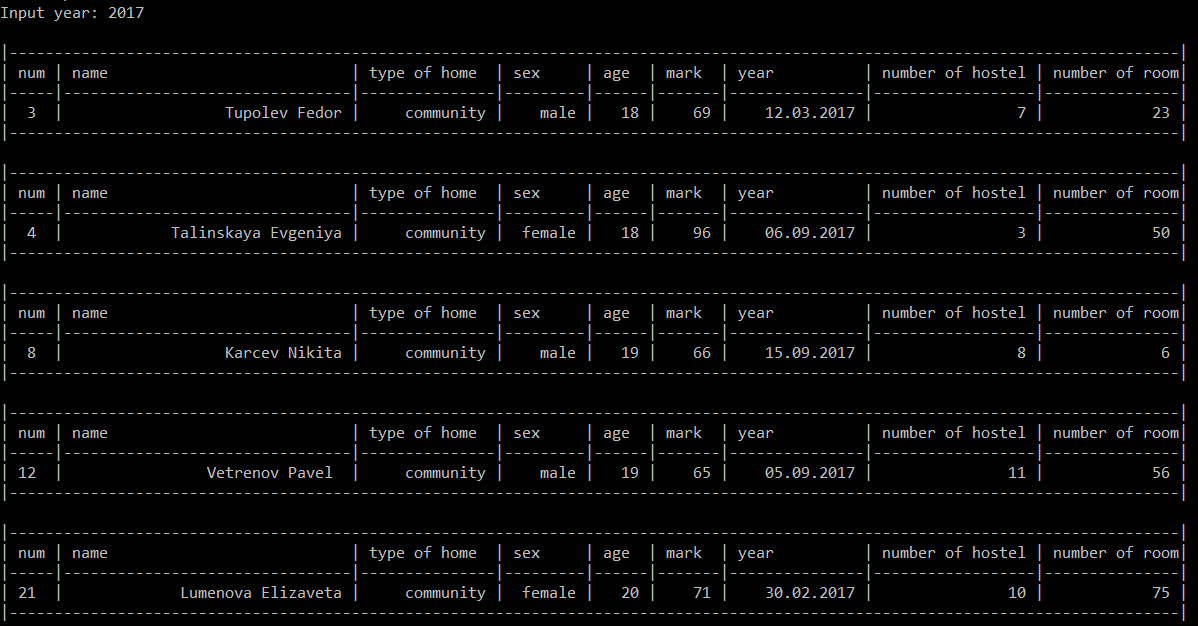
Вывод таблицы ключей на экран:



Вывод отсортированной таблицы ключей:



Поиск по фильтру:



Используемые структуры данных:

typedef struct //структура проживающих дома

{

char street[CHAR\_SIZE]; // улица

int num\_house; //номер дома

int num\_flat; // номер квартиры

}house;

typedef struct //структура проживающих в общежитии

{

int num\_comm; //номер общежития

int num\_room; // номер комнаты

}comm;

typedef union // объединение

{

house home; // проживающие дома

comm hostel; // проживающие в общежитии

}type;

typedef struct // структура каждого студента

{

int type\_home; // тип проживания

char name[CHAR\_SIZE]; //имя студента

int sex; // пол

int age; // возраст

int mark; // средний балл

char year[CHAR\_SIZE]; // дата поступления

int index; // индекс

type type;

}student;

typedef struct // структура таблицы ключей

{

int index;

int age;

} key\_table;

**Функции:**

int checkFile(FILE \*f);

Функция проверяет существования файла и его наполненности

int readChar(char \*s, FILE \*f, int len)

Чтение строки из файла или из потока

int readInt(int \*n, FILE \*f, int min\_range, int max\_range);

Чтение числа из файла или из потока, проверка на вхождение в допустимый диапазон

void recFromFile(student \*houses, key\_table \*key, int \*rec\_num, FILE \*f, int size);

Считывание данных из файла в таблицу (структуру)

void recTable(student \*stud, key\_table \*key, int \*rec\_num);

Добавление данных о студенте из потока «вручную»

void findApartment(student \*stud, key\_table \*key, int rec\_num);

Функция нахождения интересующих студентов (живущих в общежитии указанного года поступления)

void printTable(student \*stud, key\_table \*key, int \*rec\_num, int \*flag\_sort);

Функция вывода в консоль общей таблицы данных

void printKeyTable(key\_table \*key, int \*rec\_num);

Функция вывода таблицы ключей

void b\_sort(student \*stud, int rec\_num);

Сортировка «пузырьком» общей таблицы данных

void quick\_sort(student \*stud, int first, int last)

«Быстрая» сортировка общей таблицы данных

void key\_sort(key\_table \*key, int rec\_num)

Сортировка «пузрьком» таблицы ключей

void key\_quick\_sort(key\_table \*key, int first, int last)

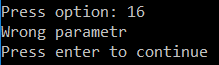
«Быстрая» сортировка таблицы ключей

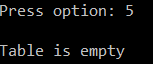
void sorting\_test();

Функция вывода данных с замером времени сортировок, размера структур, анализ эффективности по времени и по памяти

**Аварийные ситуации**:

Ввод неправильного параметра меню:



Действия с пустой таблицей:  


Удаление несуществующего элемента:  

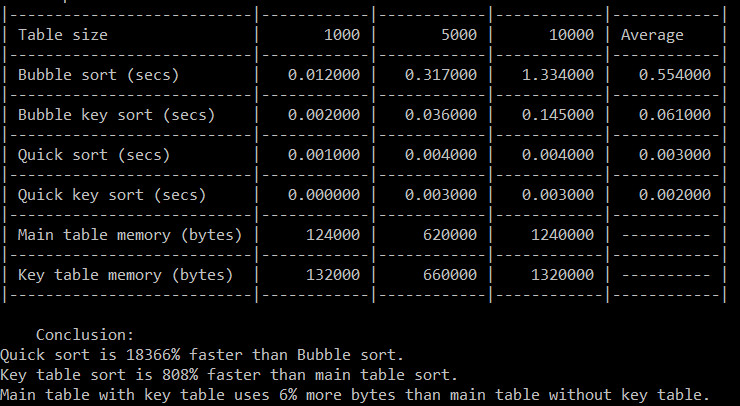

Выход за границы допустимого диапазона:  


Пустой ввод:  


Ввод недопустимого типа в значение типа Integer:  


**Оценка эффективности:**

Проведены тесты с таблицами, состоящими из 1000, 5000 и 10000 записей. Память выделяется и освобождается под каждую таблицу до и после сортировки. Каждый раз таблица заполняется заново для того, чтобы избежать сортировки уже отсортированной таблицы.



Эффективность по времени:

quick sort быстрее bubble sort в 18.36 раз

Сортировка таблицы ключей быстрее сортировки таблицы в среднем в 8.1 раз

Эффективность по памяти:

Работа с таблицей ключей занимает на 6% (8 байт) больше памяти, чем работа с исходной таблицей.

**Вывод:**

Подходящим вариантом для использования данных с вариативной частью является структура с объединением. Для примера были выбраны bubble sort и quick sort, так как это самые известные и простые в реализации типы сортировок. В ходе тестирования было выявлено, что использование таблицы ключей намного эффективнее по времени, чем сортировка целой таблицы. Использование таблицы ключей не эффективно по памяти, но в данной ситуации выбор данного метода оправдан, так как сортировка производиться по числовым данным, а не по текстовым.

**Ответы на вопросы:**

1. Как выделяется память под вариантную часть записи?

Размер памяти равен памяти, занимаемой максимальным по длине полем. Эта память является общей для всех полей вариантной части.

1. Что будет, если в вариантную часть ввести данные, не соответствующие описанным?  
   Невозможно корректно прочитать данные
2. Кто должен следить за правильностью выполнения операций с вариантной частью?  
   Разработчик.
3. Что представляет собой таблица ключей? Зачем она нужна?

Таблица ключей представляет из себя таблицу, где записаны индекс в исходной таблице и сортируемое значение (ключ). После сортировки таблицы ключей, мы можем обратится к основной таблице с данными по индексу элемента.  
При больших размерах таблиц поиск данных и их сортировка может потребовать больших затрат времени. Можно уменьшить время обработки (сортировки) таблицы за счет введения таблицы ключей.

1. В каких случаях эффективнее обрабатывать данные в самой таблице , а когда – использовать таблицу ключей?  
   При использовании таблицы ключей экономится время , так как отсутствует перестановка больших полей. Однако таблица ключей занимает дополнительную память, поэтому мы не можем достичь эффективности по памяти. Поэтому не стоит использовать таблицу ключей, если ключом являются текстовые данные. Выбор данных из основной таблицы в порядке, определенной таблицей ключей, также замедляет программу.
2. Какие способы сортировки предпочтительнее для обработки таблиц и почему?

При сортировке таблиц эффективнее использовать таблицу ключей. При этом быстрая сортировка работает быстрее, чем сортировка пузырьком.