|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

Отчет по лабораторной работе № 2

«Сортировка»

Работу выполнил:

студент группы ИУ7-32Б

Лемешев А.П.

Москва, 2021 г.

**Условие задачи**

Имеются описания:

Type жилье = (дом, общежитие);

Данные: Фамилия, имя, группа, пол (м/ж),

возраст, средний балл за сессию, дата поступления,

aдрес :

дом :

(улица, No-дома, No-кв );

общежитие :

(Noобщ.,Noкомн.);

Ввести общий список студентов. Вывести список студентов, живущих в общежитииуказанного года поступления.

**Техническое задание**

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 при использовании различных структур и эффективность использования различных алгоритмов сортировок.

**Входные данные**

Пункт меню (число от 0 до 10), файл с записями студентов, параметры добавляемых / удаляемых  
студентов.

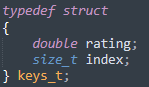
**Выходные данные**

Таблица студентов:  
 1. Вывести все команды.  
 2. Считать данные из файла.  
 3. Вывести таблицу всех студентов.  
 4. Вывести таблицу всех студентов по ключам.  
 5. Поиск студентов по полям.  
 6. Добавить студента.  
 7. Удалить студента.  
 8. Сортировка студентов.  
 9. Сортировка студентов по ключу.  
 10.Измерить время сортировок.  
 0. Выйти из программы.  
, результаты сравнения эффективности сортировок, результаты поиска по заданным полям, таблица студентов обычная и по ключам.

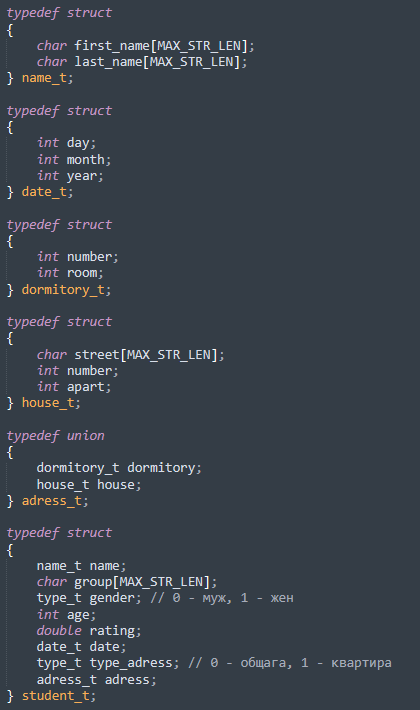
**Возможные аварийные ситуации**

1. Некорректный ввод команды (меньше 0 или больше 10)
2. Некорректный ввод имени файла (файла не существует)
3. В файле некорректные данные, не удалось считать всех студентов (полей меньше, чем в структуре)
4. Ввод команды, требующей студентов, с пустым массивом
5. Превышение количества структур в массиве (больше 500)
6. Неправильный ввод поля при добавлении студента
7. Ввод несуществующего студента при поиске
8. Ввод несуществующего студента при удалении
9. Выбор неправильной сортировки

**Структуры данных**

****

Структура ключей, где rating – поле, по которому происходит сортировка, index – индекс структуры студентов.



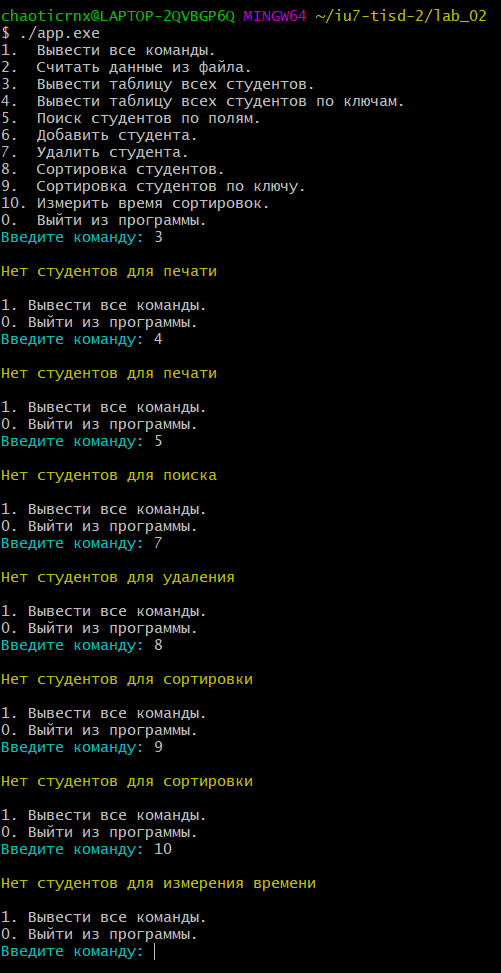
Структура студента, инвариантное поле – тип жилья. Имя объединено в структура с именем и фамилией, дата рождения объединена в структуру с днем, месяцем и годом рождения, Адрес – объединение общежития и квартиры.

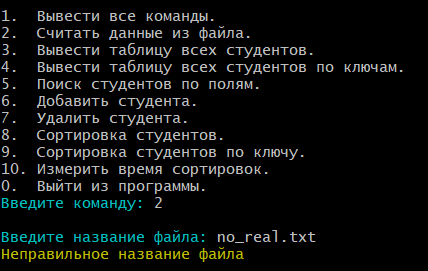
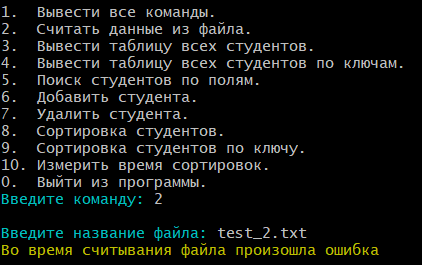
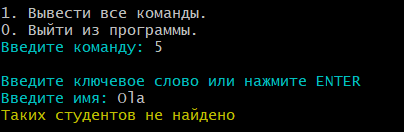
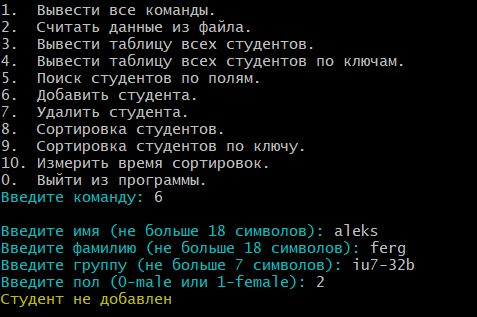
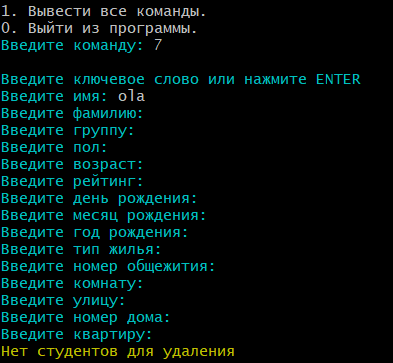
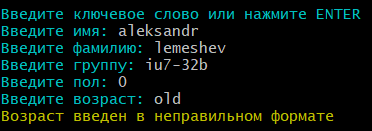
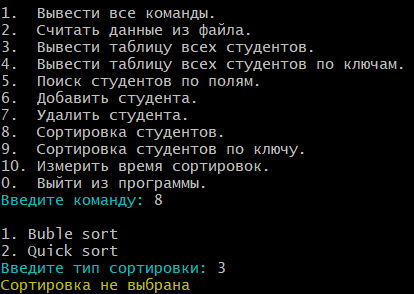
MAX\_STR\_LEN = 20 – максимальная длина строки.  
Размер одной структуры студентов: 40+20+4+4+8+12+4+28=120byte

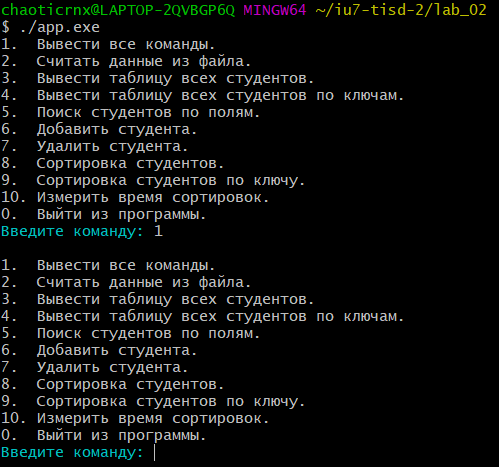
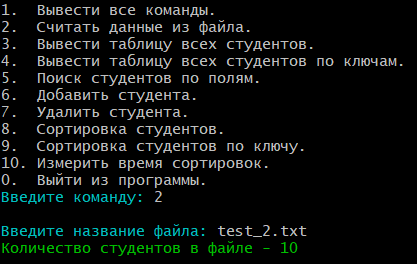
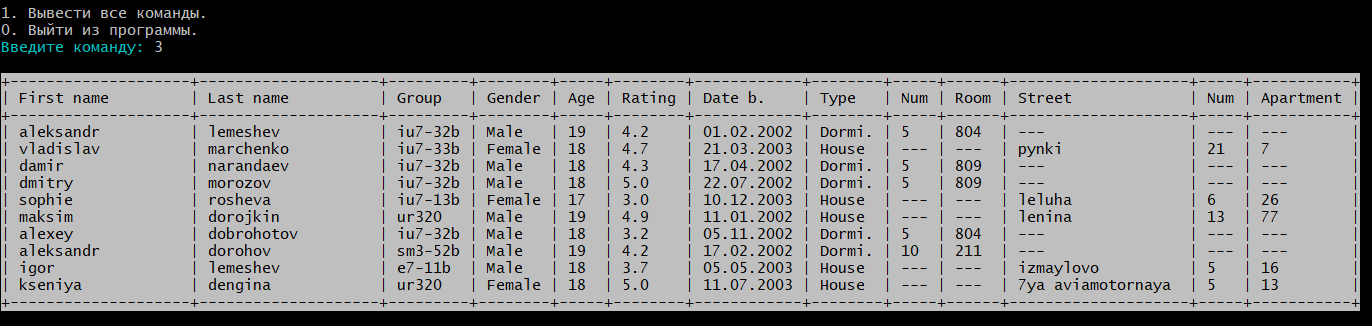
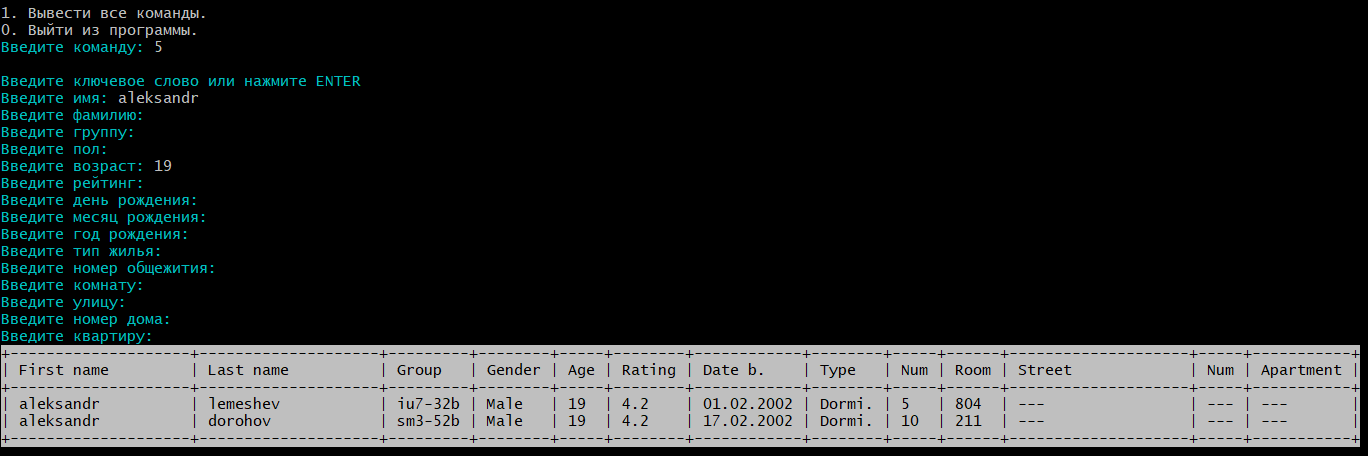
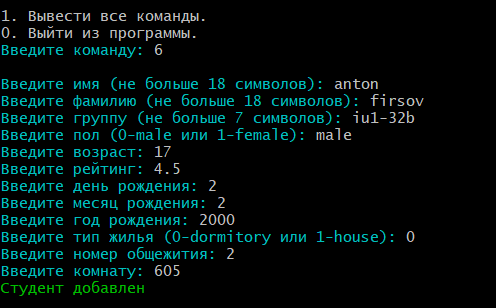
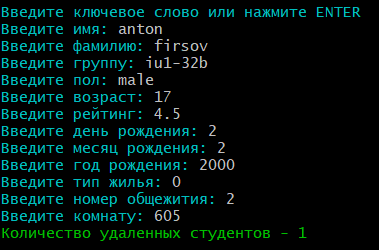
**Описание алгоритма**

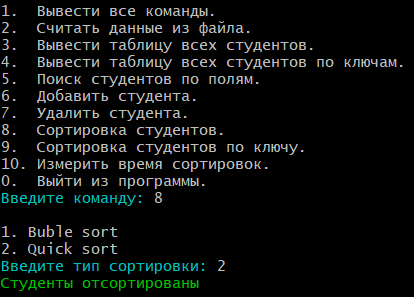
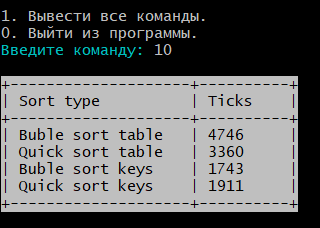
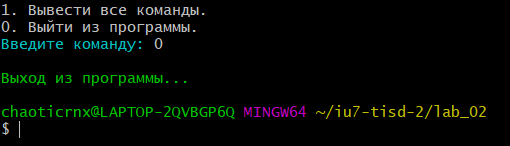
1. Считывание команды
2. Если выбран пункт 1, то печатает все команды и запрашивает ввод
3. Если выбран пункт 2, то запрашивает имя файла и по указанному имени считывает студентов в массив, в конце пишет количество считанных студентов и запрашивает ввод новой команды
4. Если выбран пункт 3, то выводит всех студентов в виде таблице и запрашивает ввод новой команды
5. Если выбран пункт 4, то выводит всех студентов в виде таблице, использую массив ключей, и запрашивает ввод новой команды
6. Если выбран пункт 5, то запрашивает каждое поле студента, далее по непустым полям ищет студентов и выводит в виде таблице, после запрашивает ввод новой команды
7. Если выбран пункт 6, то запрашивает каждое поле студента и формирует по ним нового студента, далее добавляет его в массив всех студентов и запрашивает ввод новой команды
8. Если выбран пункт 7, то запрашивает каждое поле студента, по непустым полям ищет студентов и удаляет их из массива, после запрашивает ввод новой команды
9. Если выбран пункт 8, то запрашивает вид сортировки и сортирует массив студентов выбранной сортировкой, после запрашивает ввод новой команды
10. Если выбран пункт 9, то запрашивает вид сортировки по ключу и сортирует массив студентов выбранной сортировкой, после запрашивает ввод новой команды
11. Если выбран пункт 10, то создает копию массива студентов и сортирует ее каждым способом сортировки, замеряя время выполнения, далее выводит результат замеров в виде таблицы и запрашивает ввод новой команды
12. Если выбран пункт 0, то прекращает работу программы
13. Если выбран пункт не из меню, то выводит список всех команд и запрашивает ввод новой
14. Алгоритм работает, пока пользователь не завершит программу

**Тесты**

**Негативные тесты**Ввод команд, для которых нужны студенты, с пустым массивом ****

Ввод несуществующего файла  
  
Ввод файла с неправильными исходными данными  
  
Поиск несуществующего студента  
  
Ввод некорректный данных при добавлении студента  
  
  
  
  
  
  
  
Удаление несуществующего студента  
  
Ввод некорректных данных при удалении студента  
  
Выбор неправильной сортировки  


**Позитивные тесты**Вывод всех команд  
  
Считывание студентов из файла  
  
Вывод всех студентов в таблице  
  
  
  
  
  
  
  
  
  
Поиск студента  
  
Добавление студента  
  
Удаление студента  


Сортировка студентов  
  
Сравнение времени сортировок  
  
Выход из программы  


**Оценка эффективности**

**Время сортировки**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество записей | BubleSort Таблица | BubleSort Ключи | QuickSort Таблица | QuickSort Ключи |
| 10 | 4746 | 1743 | 3360 | 1191 |
| 50 | 81711 | 24423 | 15435 | 9093 |
| 250 | 1921206 | 353997 | 56154 | 37989 |
| 500 | 9245826 | 1713768 | 184485 | 98364 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество записей | BubbleSort Таблица | BubbleSort Ключи | QuickSort Таблица | QuickSort Ключи |
| 10 | 398% | 146% | 282% | 100% |
| 50 | 899% | 269% | 169% | 100% |
| 250 | 5057% | 932% | 147% | 100% |
| 500 | 9400% | 1742% | 187% | 100% |

**Объем занимаемой памяти**

|  |  |  |
| --- | --- | --- |
| Количество записей | Таблица | Массив ключей |
| 10 | 1200 | 160 |
| 50 | 6000 | 800 |
| 250 | 30000 | 4000 |
| 500 | 60000 | 8000 |

Проигрыш по памяти в 13%.

**Выводы**

Сортировка массива ключей подтвердила свою большую эффективность по сравнению с простой сортировкой массива. Чем больше размер матрицы, тем больший выигрыш по времени сортировки дает этот способ. С малыми данными выигрыш минимален, но даже так чаще всего этот способ окупается по времени, хотя и приводит к дополнительной нагрузке по памяти.

я мог бы еще больше сэкономить памяти в массиве ключей, если бы поле для сравнения выбрал не такое большое (double).

**Контрольные вопросы**

**Как выделяется память под вариантную часть записи?**

В языке C вариативная часть структуры реализована с помощью union. Память выделяется в одном “куске” памяти, имеющий размер бОльшего поля.

**Что будет, если в вариантную часть ввести данные, несоответствующие описанным?**

Тип данных в вариантной части при компиляции не проверяется. Из-за того, что невозможно корректно прочитать данные, поведение будет неопределенным.

**Кто должен следить за правильностью выполнения операций с вариантной частью записи?**

Контроль за правильностью выполнения операций с вариантной частью записи возлагается на программиста.

**Что представляет собой таблица ключей, зачем она нужна?**

Дополнительный массив (структура), содержащий индекс элемента в исходной таблице и выбранный ключ. Она нужна для оптимизации сортировки.

**В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?**

Обрабатывать данные в самой таблице эффективнее, когда время обработки не так важно, как задействованная память. А использование таблицы ключей, наоборот, эффективно, когда нужно быстрое время обработки и не так важна дополнительно задействованная память. Так же, использование таблицы неэффективно, когда сама таблица состоит из маленького количества полей, например, таблица, имеющая два поля: “Ученик” и “Оценка”. В таком случае, таблица ключей будет лишь занимать дополнительное место в памяти и не даст никакой выгоды во времени.

**Какие способы сортировки предпочтительнее для обработки таблиц и почему?**

Если обработка данных производится в таблице, то необходимо использовать алгоритмы сортировки, требующие наименьшее количество операций перестановки. Если сортировка производится по таблице ключей, эффективнее использовать сортировки с наименьшей сложностью работы.