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)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ
№2
Вариант 3

Студент **Шавиш Тарек**

Группа **ИУ7И-31Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Руководитель **_____СилантьеваА.В.**

2023 г.

Условие задачи:

Создать таблицу, содержащую не менее 40-ка записей (тип – запись с вариантами (объединениями)). Упорядочить данные в ней по возрастанию ключей, двумя алгоритмами сортировки, где ключ – любое невариантное поле (по выбору программиста), используя:
а) саму таблицу,
б) массив ключей. (Возможность добавления и удаления записей в ручном режиме обязательна).
Осуществить поиск информации по варианту.

По варианту:
Имеются описания:
Тип жилье = (дом, общежитие, съемное);
Данные : Фамилия, имя, группа, пол (м, ж), возраст, средний балл за сессию, дата поступления адрес:
Дом : (улица, №дома, №кв);
общежитие : (№общ., №комн.)
съемное : (улица, №дома, №кв, стоимость);
Ввести общий список студентов.
Вывести список студентов, указанного года поступления, живущих в съемном жилье.

1. Ввод данных: При запуске программы пользователь увидит интерфейс, который предоставляет два варианта для ввода входных данных:
   1) Вручную ввести информацию о студенте (добавление в конец таблицы). (Структура, указанная в условии, должна быть введена вручную, каждое поле на новой строке.)
   2) Загрузить информацию о студентах из файла (не более 1000 записей). (Структуры, указанные в условии, каждое поле которых находится на новой строке, будут считаны из файла до тех пор, пока это возможно.)
2. Выходные данные: В зависимости от выполненной операции, программа может предоставлять следующую информацию:
   - Таблица данных.
   - Таблица ключей.
   - Таблицы замеров времени и памяти.
3. Задачи, реализуемые программой: В зависимости от выполненной операции, программа выполняет следующие задачи:
   - Считывает информацию из файла.
   - Добавляет студента в конец таблицы.
   - Выводит таблицу.
   - Сортирует таблицу студентов по среднему значению.
   - Создает и сортирует таблицу ключей.

- Удаляет студентов с указанным значением среднего балла.
- Находит и выводит студентов указанного года поступления, живущих в съемных квартирах.
- Сравнивает эффективность сортировки в зависимости от метода реализации и типа таблицы, которую мы сортируем.

4. Способ обращения: Для взаимодействия с программой необходимо Запустить исполняемый файл из командной строки, например, с помощью команды `./main`.

5. Возможные аварийные ситуации и ошибки пользователя:
- Некорректный ввод данных студента с клавиатуры.
- Несоответствие типа данных.
- Введенная строка превышает допустимую длину.
- Число не попадает в допустимый диапазон.
- Некорректное считывание данных из файла.
- Несуществующее имя файла.
- В файле отсутствуют валидные данные.
- Ошибка при открытии файла.
- Попытка обработать пустую таблицу.
- В таблице отсутствуют студенты, соответствующие заданному условию.

Внутренние структуры данных:

Для описания длинных чисел используется структура.

С помощью директивы `define` задаются следующие параметры:
- Максимальное количество студентов в таблице.
- Максимальная длина имени студента.
- Максимальная длина фамилии студента.
- Максимальная длина названия улицы, на которой студент живет.

```
#define STREET_MAX 20
#define SURNAME_MAX 20
#define NAME_MAX 7
#define STUDENT_MAX 1000
#define FILE_MAX 40
#define EXIT -1

//enumerated type that determines the gender of the student
enum gender
{
female,
male
};

//enumerated type that defines the type of dwelling
enum type_house
{
house,
dorm,
rent
};

//structure containing the date of receipt
typedef struct
{
int day;
int month;
int year;
}data;

//structure containing the address of the house
typedef struct
{
char street[STREET_MAX + 1];
int build_number;
int flat_number;
}building;

//structure containing the address of the hostel
typedef struct
```

```
{
int host_number;
int room_number;
}hostel;

//structure containing the address of a rented dwelling
typedef struct
{
char street[STREET_MAX + 1];
int build_number;
int flat_number;
int cost;
}rental;

//structure containing student data
typedef struct
{
char surname[SURNAME_MAX + 1];
char name[NAME_MAX + 1];
int group;
enum gender gnd;
int age;
double average_mark;
data data;
enum type_house type;
union
{
building bld;
hostel hst;
rental rnt;
}choice;
}student;

//structure- table by keys
typedef struct
{
size_t index;
double average;
}key;
```

Структура `student` содержит информацию о конкретном студенте.

Поля структуры:

Структура `student` содержит следующие данные о студенте:

- `surname` – массив, содержащий фамилию студента (максимальный размер - 20 символов).
- `name` – массив, содержащий имя студента (максимальный размер - 7 символов).
- `group` – номер группы (целое число).
- `gender` – пол студента (перечисляемый тип, содержащий поля:
- `female` – женский пол и `male` – мужской пол).
- `age` – возраст студента (целое число).
- `average_mark` – средний балл студента (вещественное число).
- `date` – структура, содержащая дату поступления (поля структуры:
- `day` – день,
- `month` – месяц и `year` – год, все в целочисленном типе).
- `type` – тип жилья студента (перечисляемый тип, содержащий поля:
- `house` – дом,
- `dorm` – общежитие и `rent` – съемное жилье).
- `choice` – объединение, содержащее информацию о жилье студента, включая структуру:
- `bld` (поля: `street` – массив с названием улицы,
- `build_number` – номер дома, и
- `flat_number` – номер квартиры), структуру
- `hst` (поля: `host_number` – номер дома и
- `room_number` – номер квартиры общежития), и
- структуру `rnt` (поля: `street` – массив с названием улицы,
- `build_number` – номер дома,
- `flat_number` – номер квартиры и
- `cost` – цена жилья).

Эти данные описывают структуру и поля, содержащие информацию о студенте и его месте жительства.

Структура `key` содержит данные о ключе.

Описание функций:
1. **int scan_student(student *st)**

   - Объяснение: Данная функция считывает информацию о студенте с клавиатуры.
   - Параметры:
     - `student *st`: Указатель на структуру, где будет храниться информация.
   - Возвращает:
     - `1`, если произошли ошибки при вводе, `0` для успешного чтения.

2. **int scan_student_file(student *st, FILE *f)**

   - Объяснение: Эта функция считывает информацию о студенте из файла в структуру.
   - Параметры:
     - `student *st`: Указатель на структуру, где будет храниться информация.
     - `FILE *f`: Дескриптор файла для чтения.
   - Возвращает:
     - `1`, если произошли ошибки при чтении, `0` для успешного чтения.

3. **int read_array_of_st(student array[STUDENT_MAX], size_t *size, FILE *f)**

   - Объяснение: Эта функция считывает данные о студентах из файла в массив структур.
   - Параметры:
     - `student array[STUDENT_MAX]`: Массив структур для хранения данных.
     - `size_t *size`: Указатель на длину массива.
     - `FILE *f`: Дескриптор файла для чтения.
   - Возвращает:
     - `1`, если превышен лимит структур или не было считано данных, `0` для успешного чтения.

4. **void print_head(void)**

   - Объяснение: Эта функция выводит заголовок таблицы на экран.
   - Параметры: Нет.
   - Возвращает: Нет.

5. **void print_student(student *st, size_t index)**

   - Объяснение: Эта функция выводит информацию о студенте в определенном формате.
   - Параметры:
     - `student *st`: Указатель на структуру, содержащую данные студента.
     - `size_t index`: Индекс студента в массиве.
- Возвращает: Нет.

6. **void table_bubble_sort(student array[STUDENT_MAX], size_t size)**

   - Объяснение: Эта функция сортирует массив структур студентов методом пузырьковой сортировки по средним оценкам.
   - Параметры:
     - `student array[STUDENT_MAX]`: Массив структур студентов для сортировки.
     - `size_t size`: Длина массива.
   - Возвращает: Нет.

7. **int compare_student(const void *p, const void *q)**

   - Объяснение: Эта функция сравнивает два значения для целей сортировки.
   - Параметры:
     - `const void *p`: Указатель на первое значение.
     - `const void *q`: Указатель на второе значение.
   - Возвращает:
     - `1`, если первое значение больше второго.
     - `0`, если они равны.
     - `-1`, если первое значение меньше второго.

8. **void quick_sort_main(student array[STUDENT_MAX], size_t size)**

   - Объяснение: Это стандартная функция быстрой сортировки для сортировки массива структур студентов.
   - Параметры:
     - `student array[STUDENT_MAX]`: Массив структур студентов для сортировки.
     - `size_t size`: Длина массива.
   - Возвращает: Нет.

9. **void delete_st_by_aver(student array[STUDENT_MAX], size_t *size, double aver)**
   - Объяснение: Эта функция удаляет студентов с указанным минимальным баллом из массива.
   - Параметры:
     - `student array[STUDENT_MAX]`: Массив структур студентов.
     - `size_t *size`: Указатель на длину массива.
     - `double aver`: Указанное среднее значение.
   - Возвращает: Нет.

10. **void create_key_table(student st[STUDENT_MAX], size_t size, key k[STUDENT_MAX])**

    - Объяснение: Эта функция создает массив ключей на основе структур студентов с наивысшими средними баллами.
    - Параметры:

- `student st[STUDENT_MAX]`: Массив структур студентов.
- `size_t size`: Длина массива.
- `key k[STUDENT_MAX]`: Массив структур ключей для хранения ключей.
- Возвращает: Нет.

11. **void key_bubble_sort(key array[STUDENT_MAX], size_t size)**

- Объяснение: Эта функция сортирует массив структур ключей методом пузырьковой сортировки по средним оценкам.
- Параметры:
- `key array[STUDENT_MAX]`: Массив структур ключей для сортировки.
- `size_t size`: Длина массива.
- Возвращает: Нет.

12. **int compare_key(const void *p, const void *q)**

- Объяснение: Эта функция сравнивает два значения по ключу для целей сортировки.
- Параметры:
- `const void *p`: Указатель на первое значение.
- `const void *q`: Указатель на второе значение.
- Возвращает:
- `1`, если первое значение больше второго.
- `0`, если они равны.
- `-1`, если первое значение меньше второго.

13. **void quick_sort_key(key array[STUDENT_MAX], size_t size)**

- Объяснение: Это стандартная функция быстрой сортировки для сортиров

ки массива структур ключей.
- Параметры:
- `key array[STUDENT_MAX]`: Массив структур ключей для сортировки.
- `size_t size`: Длина массива.
- Возвращает: Нет.

14. **int print_my_students(student st[STUDENT_MAX], size_t size, int year)**

- Объяснение: Эта функция выводит информацию о студентах указанного года поступления, проживающих в арендуемом жилье, на экран.
- Параметры:
- `student st[STUDENT_MAX]`: Массив структур студентов.
- `size_t size`: Длина массива.
- `int year`: Указанный год.
- Возвращает:
- `1`, если необходимые студенты не найдены, `0`, если хотя бы один такой

студент найден.

15. **void print_key_table(key k[STUDENT_MAX], size_t size)**

- Объяснение: Эта функция выводит таблицу ключей на экран.
- Параметры:
  - `key k[STUDENT_MAX]`: Массив структур ключей.
  - `size_t size`: Длина массива.
- Возвращает: Нет.

16. **void copy_students(student src[STUDENT_MAX], student

dst[STUDENT_MAX], size_t size)**
- Объяснение: Эта функция создает копию массива структур студентов.
- Параметры:
  - `student src[STUDENT_MAX]`: Исходный массив структур студентов.
  - `student dst[STUDENT_MAX]`: Массив, в который будет скопирована информация.
  - `size_t size`: Длина массивов.
- Возвращает: Нет.

17. **void copy_keys(key src[STUDENT_MAX], key dst[STUDENT_MAX], size_t size)**

- Объяснение: Эта функция создает копию массива структур ключей.
- Параметры:
  - `key src[STUDENT_MAX]`: Исходный массив структур ключей.
  - `key dst[STUDENT_MAX]`: Массив, в который будет скопирована информация.
  - `size_t size`: Длина массивов.
- Возвращает: Нет.

18. **void time_calculations(student st[STUDENT_MAX], key k[STUDENT_MAX], size_t *size)**

- Объяснение: Эта функция измеряет время сортировки пузырьковым методом и сортировки на основе ключей, вычисляет использование памяти и эффективность.
- Параметры:
  - `student st[STUDENT_MAX]`: Массив структур студентов.
  - `key k[STUDENT_MAX]`: Массив структур ключей.
  - `size_t *size`: Указатель на длину массивов.
- Возвращает: Нет.

19. **void print_menu(void)**

- Объяснение: Эта функция выводит меню программы.
- Параметры: Нет.
- Возвращает: Нет.

20. **int process(student st[STUDENT_MAX], size_t *size)**

- Объяснение: Эта функция обрабатывает запросы пользователей.
- Параметры:
  - `student st[STUDENT_MAX]`: Массив структур студентов.
  - `size_t *size`: Указатель на длину массива.
- Возвращает:
  - `-1`, если пользователь хочет выйти из программы.
  - `1`, если произошла ошибка при обработке данных и необходимо повторно отобразить меню.
  - `0`, если ошибок не было и необходимо повторно отобразить меню.

Поля структуры:

- index – студентский идентификатор (тип данных size_t)
- average – средний рейтинг студента (число с плавающей запятой)

Альтернативное описание алгоритма:

1) Ввод данных студента с клавиатуры и добавление его в конец массива.
Используя функции scanf и fgets, последовательно считываем информацию в структуру.

2) Загрузка данных студентов из файла.
Используя функции fscanf и fgets, последовательно считываем информацию из файла и добавляем её в массив студентов.

3) Создание ключевой таблицы.
Проходим по циклу и создаём дополнительную таблицу с двумя столбцами, куда записываем индексы и средние баллы студентов.

4) Сортировка таблицы методом пузырька.
Меняем элементы местами до тех пор, пока они не будут упорядочены. Этот процесс выполняется с использованием двух вложенных циклов.

5) Сортировка таблицы с использованием стандартной функции qsort.
Используем библиотечную функцию qsort для сортировки таблицы.

6) Удаление студентов с определённым средним баллом.
Проходим по циклу, находим студентов с заданным средним баллом и сдвигаем остальных студентов вверх для удаления.

7) Поиск и вывод студентов, поступивших в указанном году и проживающих в арендованных квартирах.
Проходим по циклу и осуществляем поиск среди студентов по указанным критериям.

8) Сравнение производительности программы в зависимости от типа сортировки и таблицы.
Реализуем функции для копирования основной таблицы и ключевой таблицы во временные массивы и используем функцию clock из библиотеки time для измерения времени сортировки. Для измерения памяти используем функцию sizeof.

Информация о эффективности программы:

Для оценки производительности программы мы провели измерения времени сортировки таблиц различных размеров с использованием библиотечной функции clock. Для каждого размера таблицы мы провели 500 измерений, после чего усреднили результаты. Также, с помощью функции sizeof, мы оценили объем памяти, выделенный программой. Ниже приведены результаты работы программы в виде таблицы:

Размер массива	**Время пузырьковой сортировки основной таблицы, мс**	**Время сортировки qsort основной таблицы, мс**	**Время пузырьковой сортировки таблицы ключей, мс**	**Время сортировки qsort таблицы ключей, мс**	**Память для хранения основной таблицы, б**	**Память для хранения таблицы ключей, б**
50	0.015	0.002	0.012	0.004	4650	630
400	1.654	0.031	0.622	0.037	32400	4720
600	4.834	0.071	1.770	0.048	67200	9600
800	8.296	0.069	2.995	0.047	89600	12800
1000	12.383	0.081	3.747	0.054	112000	16000

Вывод

Информация о производительности программы:

Для представления таблицы данных с вариативной частью, мы использовали структуру, включающую объединения. Для наглядности, были применены два метода сортировки: сортировка пузырьком и быстрая сортировка.

Для небольшого массива данных (40 элементов):
- Сортировка пузырьком по таблице ключей оказалась более эффективной, чем сортировка по обычной таблице, на 114.29%.
- Быстрая сортировка по таблице ключей оказалась более эффективной, чем сортировка по основной таблице, на 133.33%.
- Выигрыш по использованию памяти для основной таблицы составил 14%.

Для большого массива данных (1000 элементов):
- Сортировка пузырьком по таблице ключей оказалась более эффективной, чем сортировка по обычной таблице, на 170.83%.
- Быстрая сортировка по таблице ключей оказалась более эффективной, чем сортировка по основной таблице, на 8023.91%.
- Выигрыш по использованию памяти для основной таблицы составил 14%.

Из проведенных экспериментов видно, что с увеличением размера сортируемой таблицы объем памяти, используемый для хранения таблицы ключей, растет пропорционально размеру, в то время как время сортировки растет намного быстрее. Это позволяет сделать вывод о том, что при больших объемах данных предпочтительнее использовать таблицу ключей. Однако, для небольших объемов данных, программист может самостоятельно выбрать, на что он хочет сэкономить - время или память.

Также стоит отметить, что при неэффективной сортировке выигрыш по времени при сортировке по таблице ключей не сильно увеличивается при увеличении количества элементов (40 элементов - 114.29%; 1000 элементов - 170.83%), в то время как быстрая сортировка становится гораздо более эффективной (40 элементов - 133.33%; 1000 элементов - 8023.91%). Следовательно, для больших объемов данных наиболее эффективным вариантом является использование быстрой сортировки и сортировки по таблице ключей.

В ходе выполнения задания были приобретены навыки работы с типом данных

"запись" (или "структура"), содержащим вариативную часть, а также работа с данными, хранящимися в таблицах. Мы сравнили относительную эффективность программы как по времени, так и по объему используемой памяти, в зависимости от выбранного алгоритма и объема данных для сортировки. На основе выполненной работы была создана программа и составлен отчет, и цель работы была достигнута.

Ответы на контрольные вопросы:

1.Как выделяется память под вариантную часть записи?
Относительно максимального размера поля.

2.Что будет, если в вариантную часть ввести данные, несоответствующие описанным?
Неопределенное поведение.

3.Кто должен следить за правильностью выполнения операций с вариантной частью записи?
Программист.

4.Что представляет собой таблица ключей, зачем она нужна?
Таблица ключей представляет собой индекс и элементы, по которым производится сортировка. С ее помощью можно восстановить исходную таблицу, где записи следуют в определенном порядке. При работе с большими таблицами, сортировка таблицы ключей приводит к значительному ускорению процесса сортировки по времени и незначительному увеличению использования памяти по сравнению с сортировкой большой таблицы.

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

Из двух выбранных сортировок, таких как сортировка пузырьком и qsort, очевидно, что предпочтительнее использовать более эффективную библиотечную сортировку. При небольшом количестве записей разница во времени обработки незначительна, поэтому стоит отдавать предпочтение варианту, который экономит память, однако при работе с большими объемами данных таблица ключей становится гораздо более эффективной по времени.