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 РАБОТЕ №2 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

<Записи с вариантами. Обработка таблиц>

Вариант 6

Студент <Ермаков И.Г>

Группа <ИУ7 -32Б>

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

Студент <Ермаков И.Г>

Преподаватель <Фамилия ИО>

2024

Содержание

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, просмотр таблицы, просмотр таблицы в порядке расположения таблицы ключей обязательна.

Вывод списка квартир, содержащий: адрес, общую площадь, количество комнат, стоимость квадратного метра, тип:

1. Первичное
    a. С отделкой / без отделки
2. Вторичное
    a. Год постройки
    b. Количество предыдущих собственников
    c. Были ли животные

Вывод вторичное двухкомнатное жилье в указанном ценовом диапазоне без животных

Описание ТЗ

Входные данные:

Пользователь вводит число(опцию) от 1-10. Программа в зависимости от выбранной опции решает ту или иную задачу. К каждому действию пользователю предоставляется подробное приглашение к выводу.

```
1 - Загрузить таблицу из файла
2 - Вывести текущую таблицу на экран
3 - Добавить запись в конец таблицы
4 - Удалить запись по кол-ву комнат
5 - Вывод вторичного двухкомнатного жилья в указанном ценовом диапазоне без животных
6 - Отсортировать таблицу по кол-ву комнат по возрастнию пузырьковой сортировкой
7 - Отсортировать таблицу по кол-ву комнат по возрастанию qsort`ом
8 - Просмотр отсортированной таблицы ключей при несортированной исходной таблице
9 - Вывод исходной таблицы в упорядоченном виде, используя упорядоченную таблицу ключей
10 - вывод результатов сравнения эффективности работы программы при обработке данных в исходной таблице и в таблице ключей
```

Выходные данные:

Для каждой опции пользователю выводится информация об успешности или ошибке работы программы. В случае успешности программа в зависимости от опции выводит пользователю на экран.

Действие программы:

Программа работает до тех пор пока пользователь не введет 0. В случае некорректного ввода опции программа предлагает набор доступных опций и приглашение к вводу для повторного ввода опции. Так же при ошибке ввода в процессе работы с таблицей (например неправильный формат года) программа вернется в исходное состояние выбора опций, описанное ранее.
Обращение к программе:

Программа запускается из терминала по команде ./app.exe

Аварийные ситуации:

ERROR_READING_TYPE_ROOM (1): Ошибка чтения типа жилья. Происходит при некорректном или неуспешном вводе типа жилья (например, "primary" или "secondary").

ERROR_READING_ADDRESS (2): Ошибка чтения адреса. Возникает, если при вводе или чтении данных адреса произошла ошибка.

ERROR_READING_SQUARE (3): Ошибка чтения площади. Происходит при некорректном вводе значения площади.

ERROR_READING_AMOUNT_ROOMS (4): Ошибка чтения количества комнат. Возникает, если данные о количестве комнат некорректны или не могут быть прочитаны.

ERROR_READING_COST (5): Ошибка чтения стоимости за квадратный метр. Происходит, если введены неверные или недопустимые данные.

ERROR_READING_TRIM (6): Ошибка чтения состояния отделки. Возникает, если данные об отделке некорректны.

ERROR_READING_YEAR (7): Ошибка чтения года постройки. Происходит при некорректном вводе года или при его отсутствии.

ERROR_READING_LAST_OWNERS (8): Ошибка чтения количества предыдущих владельцев. Возникает, если введённые данные о количестве владельцев неверны.

ERROR_READING_PETS (9): Ошибка чтения данных о наличии домашних животных. Происходит, если данные о домашних животных некорректны.

ERROR_VALUE_PETS (10): Некорректное значение для данных о наличии домашних животных. Возникает, если введённое значение не соответствует допустимому диапазону (например, ввод другого символа вместо "1" или "0").

ERROR_MATCHING_REG_EXPR (11): Ошибка сопоставления с регулярным выражением. Происходит, если введённые данные не соответствуют ожидаемому формату, проверяемому с помощью регулярных выражений.
ERROR_READING_FILENAME (12): Ошибка чтения имени файла. Возникает, если имя файла не может быть прочитано (например, файл не существует).

ERROR_OPENING_FILE (13): Ошибка открытия файла. Происходит, если файл не удаётся открыть (например, файл отсутствует или нет прав доступа).

ERROR_READING_AMOUNT_OF_STRUCTS (14): Ошибка чтения количества структур. Возникает, если данные о количестве структур не могут быть прочитаны.

ERROR_INCORRECT_AMOUNT_SCRUCTS (15): Некорректное количество структур. Происходит, если количество введённых структур превышает или не соответствует допустимому значению.

ERROR_EXCESS_STRUCTS (16): Превышение допустимого количества структур. Возникает, если количество структур превышает заранее установленное ограничение.

ERROR_READING_STRUCT (17): Ошибка чтения структуры. Происходит, если структура данных не может быть корректно прочитана или загружена.

Описание структур данных

```
typedef struct keys_t
{ int index;
   int rooms_quantity;
} keys_t;
```

```
typedef struct primary_t
{ short trim;
} primary_t;
```

```
typedef struct secondary_t
{
   char build_year[MAX_LEN_YEAR + 1];
   int quantity_prev_owners;
   short is_pet;
} secondary_t;
```

```
typedef union choose_type_t
{    primary_t    prime;
   secondary_t second;
} choose_type_t;
```

```
typedef struct desc_t
{
   char type_room[MAX_LEN_TYPE_ROOM + 1];
   char address[MAX_LEN_ADDRESS + 1];
   float area;
   int rooms_quantity;
   float square_cost;
   choose_type_t type;
} desc_t;
```

Структура desc_t содержит такие поля как тип жилья (type_room), нужен для первоначального определения вложенного типа основной структуры. Поле address определяет адрес. Поле area определяет жилую площадь помещения. Поле rooms_quantity определяет кол-во комнат в квартире. Поле square_cost определяет стоимость квадратного метра. Поле type предназначено для следующего уровня

вложенности структуры, точнее обьединение для выбора первичного или вторичного типа жилья в зависимости от ввода пользователя.
Переходим к обьединению choose_type_t. Это обьединение содержит два поля : prime и second соответственно отвечающие за первичный или вторичный тип жилья, который описывает следующий уровень вложенности : структуры primary_t и secondary_t

Структура primary_t содержит в себе одно поле trim, которая в программа может принимать только два значения : 0 или 1. 0 – отделки нет, 1 – отделка есть

Структура secondary_t содержит поля build_year, quantity_prev_owners, is_pet
Поле build_year содержит информацию о годе постройки жилья
Поле quantity_prev_owners содержит в себе информацию о кол-ве предыдущих собственников жилья
Поле is_pet говорит о том, жили в квартире какие то животные. 0 – нет, 1 – да

```
#define MAX_LEN_ADDRESS 25
#define MAX_LEN_YEAR 10
#define MAX_LEN_FILENAME 15
#define MAX_LEN_TYPE_ROOM 10
```

Реализация работы программы подразумевает описанные выше константы для взаимодействия с полями структур.

Время сортировки таблиц

Ниже представленны сортировки для размеров массивов (100, 250, 500, 1000)

Сортировка для 100 записей

```
Сортировка файла из 100 записей

|-------------------------------------------------------------------------------|
| Сортировка таблицы пузырьком         |    177023 тактов |    0.0000478441 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей пузырьком  |    151447 тактов |    0.0000409316 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы qsort             |     97552 тактов |    0.0000263654 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей qsort      |      8777 тактов |    0.0000023722 секунд |
|-------------------------------------------------------------------------------|

7200 размер таблицы (в байтах)


800 размер таблицы ключей (в байтах)
```

Сортировка для 250 записей

```
Сортировка файла из 250 записей

|-------------------------------------------------------------------------------|
| Сортировка таблицы пузырьком         |    598418 тактов |    0.0001617346 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей пузырьком  |    401170 тактов |    0.0001084243 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы qsort             |     81164 тактов |    0.0000219362 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей qsort      |      8624 тактов |    0.0000023308 секунд |
|-------------------------------------------------------------------------------|

18000 размер таблицы (в байтах)


2000 размер таблицы ключей (в байтах)
```

Сортировка для 500 записей

```
Сортировка файла из 500 записей

|-------------------------------------------------------------------------------|
| Сортировка таблицы пузырьком         |   1495865 тактов |    0.0004042878 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей пузырьком  |   1265787 тактов |    0.0003421046 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы qsort             |     64322 тактов |    0.0000173843 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей qsort      |     14015 тактов |    0.0000037878 секунд |
|-------------------------------------------------------------------------------|

36000 размер таблицы (в байтах)


4000 размер таблицы ключей (в байтах)
```

Сортировка для 1000 записей

```
Сортировка файла из 1000 записей

|-------------------------------------------------------------------------------|
| Сортировка таблицы пузырьком         |  10779283 тактов |    0.0029133197 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей пузырьком  |   8564563 тактов |    0.0023147468 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы qsort             |    172885 тактов |    0.0000467257 секунд |
|-------------------------------------------------------------------------------|
| Сортировка таблицы ключей qsort      |     42101 тактов |    0.0000113786 секунд |
|-------------------------------------------------------------------------------|

72000 размер таблицы (в байтах)


8000 размер таблицы ключей (в байтах)
```

Описание как работает алгоритм подсчета времени:

```
uint64_t tick_count(void)
{
    uint32_t high, low;
    __asm__ __volatile__ (
        "rdtsc¥n"
        "movl %%edx, %0¥n"
        "movl %%eax, %1¥n"
        : "=r" (high), "=r" (low)
        :: "%rax", "%rbx", "%rcx", "%rdx"
    );

    uint64_t ticks = ((uint64_t)high << 32) | low;
    return ticks;
}
```

Функция tick_count() измеряет количество процессорных тактов с момента запуска системы, используя инструкцию процессора rdtsc. Эта инструкция возвращает 64-битное значение, разделённое на два регистра: младшие 32 бита сохраняются в EAX, старшие — в EDX. Ассемблерный код считывает эти значения и объединяет их в одно 64-битное число. Полученное значение можно использовать для измерения времени выполнения кода с высокой точностью, так как оно отражает количество тактов процессора, прошедших за определённый промежуток времени.

Выводы о подсчетах времени и используемой памяти:

На основании проведённых тестов можно сделать общий вывод: сортировка с использованием алгоритма быстрой сортировки (qsort) существенно превосходит пузырьковую сортировку по времени выполнения, особенно при работе с большими массивами данных. Это подтверждается результатами для различных размеров массивов.

Для массивов из 1000 записей время выполнения qsort на данных оказалось более чем в 62 раза быстрее по сравнению с пузырьковой сортировкой (10779283 тактов против 172885 тактов). Для сортировки ключей разница оказалась ещё более впечатляющей: qsort выполнялся более чем в 200 раз быстрее, чем пузырьковая сортировка ключей (8564563 тактов против 42101 тактов).

Размер таблицы ключей составляет около 11% от размера исходного массива данных, что делает сортировку по ключам более экономичной только по времени, не по памяти (111%) от общего размера. Это особенно важно для больших объемов данных, где работа с полным массивом требует значительно больше ресурсов.

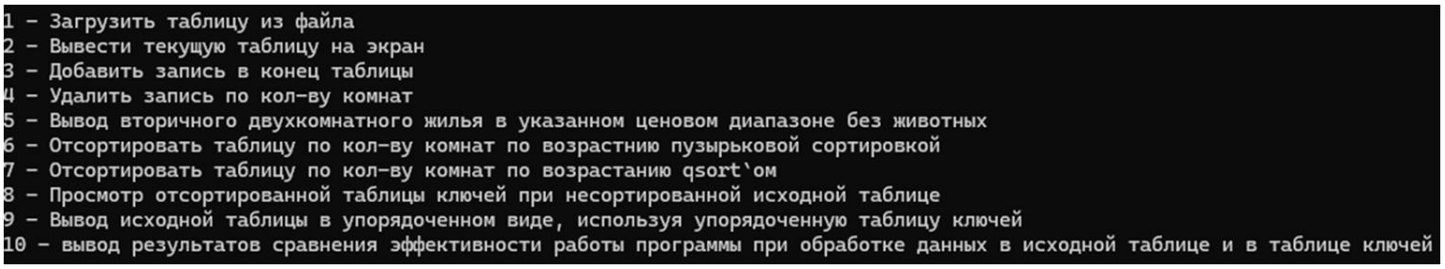
Сортировка qsort универсальна и эффективна, но требует использования динамической памяти (кучи), что может создавать дополнительные накладные расходы на её выделение и управление. Тем не менее, несмотря на эти накладные расходы, qsort остаётся лучшим выбором по сравнению с пузырьковой сортировкой, особенно при больших объемах данных.

В итоге, для любых задач, связанных с сортировкой больших объемов данных, qsort показывает наилучшие результаты по производительности, в то время как пузырьковая сортировка может быть применима лишь для небольших наборов данных, где её простота может компенсировать низкую эффективность.

Сравнение времени работы bubble_sort и qsort для массива и ключей

Кол-во записей	Время работы bubble_sort для массива ключей относительно qsort для ключей	Время работы bubble_sort для таблицы относительно qsort для таблицы
10	114%	27%
100	1725%	181%
250	5883%	684%
500	9031%	2326%
1000	20342%	6234%

Вывод опций для пользователя



Тесты

Описание теста	Входные данные	Вывод
Ввод неверного пункта меню	-1	Ошибка ввода опции! Введите правильное значение от 0 до 10
Ввод неверного пункта меню	11	Ошибка ввода опции! Введите правильное значение от 0 до 10
Ввод неверного пункта меню	egsg	Ошибка ввода опции! Введите правильное значение от 0 до 10.
Загрузка таблицы	Кол-во жилья больше 1000 или меньше 40	Некорретное кол-во структур в файле
Добавление в таблицу	Одно из полей не соответствует стандарту	Ошибка чтения параметра!
Вывод таблицы	Если не была загружена	Пустой шаблон
Вывод жилья в указанном диапазоне со вторичным типом, две комнаты	Жилье не входит в диапазон, отсутствие других параметров	Не найдено ни одного элемента в таком целовом диапазоне и без животных
Загрузка таблицы из файла	Все указано правильно	Таблица из файла успешно считана!
Вывод времени сортировок таблиц	Нет данных	Ошибка, пустой файл
Вывод времени сортировок	Если таблица считана, данные есть	Выводится таблица с измерениями
Удаление записи	Ошибок нет	Запись удалена успешно!
Добавление записи с переполнением	Добавляется 1001 запись	Некорректное кол-во структур

Выводы

Таблица, описывающая сравнение сортировок времени работы двух алгоритмов явно показывает, что с увеличением количества записей разница в производительности между qsort и пузырьковой сортировкой становится все более значительной, особенно на больших массивах данных.

Использование union в данном случае позволяет избежать выделения памяти под обе структуры сразу. Память выделяется только для одного из типов — либо для первичного жилья, либо для вторичного. Это может быть очень полезным, если в твоей программе будет обрабатываться множество записей, и разные записи могут использовать разные типы жилья.

Использование сортировки ключей приводит к сильной прибавке в эффективности, но в то же время мы жертуем памятью (на 11% больше искомой)

Ответы на контрольные вопросы

Как выделяется память под вариантную часть записи?

Память под вариантную часть записи выделяется так же, как и для обычных структур, но используется общий блок памяти, который может содержать разные типы данных, в зависимости от того, какая часть активна в данный момент. Это часто делается с использованием union.

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

Это приведет к некорректной интерпретации данных, что может вызвать непредсказуемое поведение программы или ошибки при доступе к данным.

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

Программист несет ответственность за корректную работу с вариантной частью записи, обеспечивая правильное использование поля, которое активно в данный момент.

Что представляет собой таблица ключей, зачем она нужна?

Таблица ключей — это структура данных, содержащая индексы записей и значения, по которым происходит сортировка. Она позволяет сортировать и искать данные без изменения исходной таблицы.

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

Эффективнее обрабатывать данные в самой таблице, если требуется частый доступ к элементам и не нужна многократная сортировка. Таблицу ключей используют, когда нужно сортировать данные несколько раз или оптимизировать операции поиска.
Какие способы сортировки предпочтительнее для обработки таблиц и почему?

Для больших таблиц предпочтительнее использовать алгоритмы с временной сложностью O(nlogn), такие как qsort или быстрая сортировка. Они быстрее, чем простые алгоритмы вроде пузырьковой сортировки O(n**2) особенно на больших объемах данных.