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.

Студент **Батуев А.Г.**

Группа **ИУ7-36Б**

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

Студент _____ **Батуев А.Г.**

Преподаватель _____ **Никульшина Т.А.**

2024 г.

Условие задачи

Цель работы - приобрести навыки работы с типом данных «запись» («структура») содержащим вариантную часть, и с данными, хранящимися в таблицах. Оценить относительную эффективность программы (в процентах) по времени и по используемому объему памяти в зависимости от используемого алгоритма и от объема сортируемой информации.

Вариант: ввести список абонентов, содержащий фамилию, имя, номер телефона, адрес (улица, дом), статус абонента:

1. Друзья:
    a. Дата рождения: день, месяц, год
2. Коллеги:
    a. должность
    b. организация

Вывести список всех друзей, которых необходимо поздравить с днем рождения в ближайшую неделю. (текущая дата вводится пользователем)

Техническое задание

На вход **программа получает**:

- Файл с данными об абонентах
- Номер действия (от пользователя)
- В зависимости от выбранного действия может запрашивать доп. информацию, необходимую для выполнения действия.

Программа должна реализовывать получение, хранение, вывод и различные операции над структурным массивом, такие как добавление элемента в конец, удаление по указанному полю, сортировка.

Обращение к программе осуществляется по указанию названия программы.

Внимательно **обработать возможные аварийные ситуации**, которые могут включать: неправильный ввод данных, пустой входной файл, удаление несуществующего элемента, сортировка пустого массива.

Описание внутренних структур данных

Для удобства обращения с данными реализуется структура, которая описывает в том числе вариативную часть данных. Например, в зависимости от типа абонента (друг или коллега) могут предоставляться одни данные, а другие отсутствовать. Так для абонента друг следует указывать день рождения, а для коллеги должность и организацию. Такой тип данных записывается как объединение для экономии памяти программы, т.к. одновременно абонент быть другом и пользователем не может по условию задачи.

У каждого абонента также присутствует общая часть, описываемая структурой user_t, туда входят поля фамилии, имени, номера и адреса.

В итоге абонент описывается структурой sub_t, куда помимо объединения (по другу и коллеге) попадает перечисляемый тип, для понимания какой тип данных содержится в объединении и его id.

- address_t – описывает адрес пользователя. Состоит из 2 полей (2 строк): улицы и дома.
- user_t – описывает общее поле абонента. Состоит из 4 полей (2 строк, длинного целого, адреса): фамилии, имени, номера и адреса.
- date_t – описывает дату в формате: год, день, месяц. Состоит из 3 полей, представленных типом int.
- friend_t – описывает абонента со статусом друг. Состоит из 2 полей: (user_t, date_t)
- college_t - описывает абонента со статусом коллега. Состоит из 3 полей (user_t и 2 строк)
- kind_sub_e – перечисление (enum), определяющее тип субъекта
- status_t - объединение (union), которое может хранить информацию либо о друге, либо о коллеге
- sub_t - основная структура для представления абонента.

```
#define MAX_STRING_LENGTH 256

typedef struct
{
   char street[MAX_STRING_LENGTH];
   char house[MAX_STRING_LENGTH];
}adress_t;

typedef struct
{
   char surname[MAX_STRING_LENGTH];
   char name[MAX_STRING_LENGTH];
   long long int number;
   adress_t adress;
}user_t;

typedef struct
```

```
{
   int year;
   int month;
   int day;
}date_t;

typedef struct
{
   user_t user;
   date_t date_birth;
}friend_t;

typedef struct
{
   user_t user;
   char position[MAX_STRING_LENGTH];
   char organization[MAX_STRING_LENGTH];
}college_t;

typedef enum
{
   FRIEND,
   COLLEGE
}king_sub_e;

typedef union
{
   friend_t friend;
   college_t college;
}status_t;

typedef struct
{
   unsigned int id;
   king_sub_e kind_sub;
   status_t status;
}sub_t;
```

id описывается не только для однозначного определения, но и для формирования массива ключей, который необходим для сравнения скорости сортировок по структурному массиву sub_t и key_t. Id отражает положение элемента в основном массиве sub_t. Ожидается, что сортировка key_t займет меньшее кол-во времени, т.к. содержит меньшее кол-во полей, которое необходимо переставлять.

Структура key_t состоит из 2 полей:

- id - уникальный идентификатор, представляющий ID (индекс в основном массиве) пользователя.

- number – ключевое поле, по которому происходит сортировка.

```
typedef struct
{
   unsigned int id;
   long long int number;
}key_t;
```

Описание алгоритмов

Программа реализует следующие программные реализации:

- read_date(FILE* file, date_t *date, size_t sub_count): Читает дату из файла или стандартного ввода. Проверяет корректность формата даты. Возвращает код ошибки при некорректном вводе.
- read_sub(FILE *file, size_t sub_count, sub_t *temp_sub): Читает информацию о подписчике из файла. Обрабатывает различные поля для друзей и коллег. Возвращает указатель на структуру sub_t или NULL при ошибке.
- read_sub_comms(FILE *file, size_t sub_count, sub_t *temp_sub): Повторяет предыдущую, но с комментариями по вводу.
- read_subs_from_file(const char* filename, sub_t subs[], size_t *subs_count, key_t keys[]): Читает данные о подписчиках из файла. Заполняет массивы subs и keys. Возвращает код ошибки или 0 при успехе.
- write_into_file(const char* filename, sub_t subs[], size_t subs_count): Записывает данные о подписчиках в файл. Сохраняет информацию из массива subs.
- show_entry_message(void): Выводит начальное сообщение о возможностях программы.
- show_menu(void): Отображает меню с доступными действиями пользователя.
- show_table(sub_t subs[], size_t size): Выводит таблицу всех подписчиков в консоль.
- show_key_table(key_t keys[], size_t size): Отображает таблицу ключей в консоль.
- show_table_by_key(sub_t subs[], key_t keys[], size_t size): Показывает таблицу подписчиков, отсортированную по ключам.
- print_friend(sub_t sub) и print_colleague(sub_t sub): Функции для вывода информации о друзьях и коллегах соответственно.
- is_number(char *string_in): Проверяет, состоит ли строка только из цифр и пробелов.
- del_from_table(sub_t subs[], key_t keys[], size_t *size): Удаляет запись из таблицы по заданному ID. Обновляет массивы subs и keys, уменьшая их размер.
- add_to_table(sub_t subs[], key_t keys[], size_t *size): Добавляет новую запись в конец таблицы. Создает новые структуры для подписчика и ключа.
- bubble_sort_full(sub_t subs[], key_t keys[], size_t size): Сортирует обе таблицы (подписчики и ключи) методом пузырька.

- bubble_sort_items(sub_t subs[], size_t size): Сортирует только таблицу подписчиков методом пузырька.
- bubble_sort_keys(key_t keys[], size_t size): Сортирует только таблицу ключей методом пузырька.
- quick_sort_full(sub_t subs[], key_t keys[], size_t size): Сортирует обе таблицы методом быстрой сортировки.
- quick_sort_items(sub_t subs[], size_t size): Сортирует только таблицу подписчиков методом быстрой сортировки.
- quick_sort_keys(key_t keys[], size_t size): Сортирует только таблицу ключей методом быстрой сортировки.
- sort_by_keys(sub_t subs[], key_t keys[], size_t size): Отсортирует подписчиков по их ключам.
- clear_input_buffer(void): Очищает буфер ввода после чтения строки.
- compare_sub(const void *a, const void *b): Функция сравнения для сортировки подписчиков.
- compare_key(const void *a, const void *b): Функция сравнения для сортировки ключей.
- choose(int rc, sub_t subs[], key_t keys[], size_t *size): Выполняет различные действия в зависимости от выбранного режима.
- measure_sort_time_items(void (*sort_func)(sub_t[], size_t), sub_t items[], size_t size): Мерит время выполнения сортировки для массива подписчиков.
- measure_sort_time_keys(void (*sort_func)(key_t[], size_t), key_t keys[], size_t size): Мерит время выполнения сортировки для массива ключей.
- compare_sorts(sub_t subs[], key_t keys[], size_t size): Сравнивает производительность различных алгоритмов сортировки.
- print_upcoming_birthdays(sub_t subs[], size_t size): Выводит дни рождения следующих 7 дней для всех подписчиков.
- increment_date(date_t *date): Увеличивает дату на один день.
- compare_dates(date_t d1, date_t d2): Сравнивает две даты.
- days_in_month(int month, int year): Возвращает количество дней в заданном месяце.
- is_leap_year(int year): Проверяет, является ли год високосным.

Набор тестов

Входные данные:

1. Для добавления указываются все поля структуры sub_t
2. Для удаления указывается id структуры
3. Для вывода пользователей, которых нужно будет поздравить в течении 7 дней, текущая дата

Выходные данные:

- Вывод в терминал, если выбран вариант с выводом
- Построчный вывод всех данных из таблицы в выходной файл

Позитивные тесты:

Описание	Входные данные	Выходные данные
Добавление элемента в пустую таблицу	4 Второй Человек 81111111112 Ул. Один-три 1 FRIEND 2006 8 23	Второй Человек 81111111112 Ул. Один-три 1 FRIEND 2006 8 23
Вывод таблицы по ключам после удаления элемента	5 0 3	*Таблица без удаленного элемента*
Проверка корректности сортировки на отсортированном массиве	8	*Правильно отсортированная таблица*
Проверка корректности сортировки на полностью неотсортированном массиве	8	*Правильно отсортированная таблица*
Добавление нового элемента (друга) в массив и вывод его даты рождения для поздравления	4 Второй Человек 81111111112 Ул. Один-три 1 FRIEND 2006 8 23 10 2024 8 20	День рождения у Второй Человек будет 8 23

Негативный тесты:

Описание	Входные данные	Выходные данные

Удаление элемента по несуществующему id	5 -10	*Нет элемента с таким номером*
Удаление элемента из пустой таблицы	5 0	*Нет элемента с таким номером*
Сортировка пустой таблицы	8	*Таблица пуста*
Сравнение сортировок на пустой таблице	9	*Таблица пуста*
Попытка ввода некорректных данных при добавлении	4 Второй Человек -81 Ул. Один-три 1 FRIEND 2006 8 23	*Некорректные данные при вводе*
Попытка ввода неправильной даты	4 Второй Человек 81111111112 Ул. Один-три 1 FRIEND 2006 13 23	*Неправильная дата*

Аналитическая часть

Применение типа «запись» с вариантной частью позволяет оптимизировать использование памяти и структуру данных, которая поддерживает разные варианты содержимого в зависимости от типа записи. В этом случае используется union для объединения данных разных типов (friend_t и college_t), что экономит память, поскольку для каждой записи выделяется только та часть, которая необходима для хранения конкретных данных.

Преимущества применения типа «запись» с вариантной частью:

- Экономия памяти: без использования вариантной части каждая запись должна хранить одновременно данные для всех возможных типов (friend_t и college_t), что приводит к увеличению объема памяти.

С использованием union (объединения) выделяется только место под наиболее объемную структуру (в данном случае либо friend_t, либо college_t), что сокращает общий объем памяти.
Упрощение управления данными:

Вместо того чтобы хранить дополнительные флаги или разделять структуры для каждого типа данных, мы используем поле kind_sub_e для указания типа данных, хранимого в объединении. Это позволяет явно управлять типами данных, минимизируя сложность программы.

Недостатки:

- Сложность обработки данных: обработка данных становится немного сложнее, так как нужно учитывать, что любое изменение в поле объединения, или случайное обращение не к тому полю объединения может повлечь изменения в данных.
- Увеличение времени обработки: потребность в проверке типа данных и соответствующем приведении типа увеличивает время обработки по сравнению с более простой структурой, где все данные хранятся одновременно, без необходимости проверок. Однако этот недостаток нивелируется с увеличением размерности массива, выделенного под хранение данных.

Сравнительный анализ объема памяти.
Проведем подсчет занимаемой памяти.
Размер структуры user_t = MAX_STRING_LENGTH * 2 + sizeof(long long) + (MAX_STRING_LENGTH * 2) = MAX_STRING_LENGTH * 4 + sizeof(long long), где MAX_STRING_LENGTH = 256 байт
Размер структуры date_t: sizeof(int) * 3
Общий размер записи: sizeof(friend_t) = sizeof(user_t) + sizeof(date_t)

Общий размер записи: sizeof(college_t) = sizeof(user_t) + MAX_STRING_LENGTH * 2 = MAX_STRING_LENGTH * 6 + sizeof(long long)

Размер одной записи friend_t = 256*4 + 16+ 3*4 = 1052 байта.
Размер одной записи college_t = 1552 байта
Итого: без вариантной части на каждую запись нужно 1052 + 1552 = 2604 байт.

С вариантной частью:
Размер записи sub_t (без учета других полей) примет размер большего поля из объединения friend_t и college_t, а именно 1552, что позволяет сэкономить на 1052 байта. Учитывая, что мы работаем с со структурным массивом то экономия составит 1052 * SIZE_OF_ARRAY.

Вывод:
Применение типа «запись» с вариантной частью позволяет значительно сократить объем необходимой памяти за счет использования объединения данных разных типов, что особенно важно при работе с большими таблицами. Недостатком является небольшое увеличение времени обработки за счет необходимости проверки типа данных, но оно компенсируется существенной экономией памяти, особенно в случае большого количества записей.

Использование структуры ключей при сортировке.
При больших размерах таблиц поиск данных, имеющих указанный ключ, может потребовать больших затрат времени. Если же помимо поиска требуется произвести сортировку данных, то временные затраты многократно возрастут, так как потребуется осуществлять их перестановку (перемещение). В этом случае можно уменьшить время обработки за счет создания дополнительного массива – таблицы 5 ключей, содержащей индекс элемента в исходной таблице и выбранный ключ.
Таким образом, если мы сортируем таблицу ключей, то экономится время, поскольку перестановка записей в исходной таблице, которая иногда может содержать достаточно большое число полей, отсутствует. Этот выигрыш во времени особенно заметен при большой размерности таблиц и при правильно подобранных ключах, что видно на следующих примерах:

40 элементов
Быстрая по таблице: 0.000007
Быстрая по ключам: 0.000002
Пузырьковая по таблице: 0.000031
Пузырьковая по ключам: 0.000003

Разница в скорости сортировок (быстрая по таблице к ,быстрой по ключам): 350.00 %
Разница в скорости сортировок (быстрая по таблице к пузырьковой по таблице): 22.58 %
Разница в скорости сортировок (быстрая по ключам к пузырьковой по ключам): 66.67 %
Разница в скорости сортировок (пузырьковая по таблице к пузырьковой по ключам): 1033.33 %

80 элементов
Быстрая по таблице: 0.000011
Быстрая по ключам: 0.000004
Пузырьковая по таблице: 0.000175
Пузырьковая по ключам: 0.000012
Разница в скорости сортировок (быстрая по таблице к ,быстрой по ключам): 275.00 %
Разница в скорости сортировок (быстрая по таблице к пузырьковой по таблице): 6.29 %
Разница в скорости сортировок (быстрая по ключам к пузырьковой по ключам): 33.33 %
Разница в скорости сортировок (пузырьковая по таблице к пузырьковой по ключам): 1458.33 %

160 элементов
Быстрая по таблице: 0.000018
Быстрая по ключам: 0.000008
Пузырьковая по таблице: 0.000811
Пузырьковая по ключам: 0.000044
Разница в скорости сортировок (быстрая по таблице к ,быстрой по ключам): 225.00 %
Разница в скорости сортировок (быстрая по таблице к пузырьковой по таблице): 2.22 %
Разница в скорости сортировок (быстрая по ключам к пузырьковой по ключам): 18.18 %
Разница в скорости сортировок (пузырьковая по таблице к пузырьковой по ключам): 1843.18 %

По полученным данным сразу видна значительная временная разница при сортировке по всем данным и только по ключам. Также важную роль играет сложность выбранной сортировки, где у пузырьковой O(n^2), а у быстрой O(n * log(n)).

Контрольные вопросы

1. Как выделяется память под вариантную часть записи?

Память в вариантной части выделяется под максимальную (по размеру) запись. Таким образом в выделенной памяти хранится одновременно только тип данных.

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

Если в вариантную часть записи ввести данные, несоответствующие описанным типам, это может привести к неопределённому поведению программы, в частности, к изменению уже существующих данных.

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

Ответственность за корректность операций с вариантной частью записи ложится на программиста.

4. Что представляет собой таблица ключей, зачем она нужна?

Таблица ключей — это структура данных, которая содержит пары "ключ-значение", где ключ используется для быстрого поиска значений в таблице. Таблица ключей полезна для индексации данных, что позволяет эффективно выполнять операции поиска, упорядочивания и доступа к данным, особенно при работе с большими объемами данных.

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

Если доступ к элементам осуществляется последовательно, то имеет смысл обрабатывать данные в самой таблице.
В случае если доступ к элементам происходит выборочный (по ключу), а объемы данных достаточно большие, то использование таблицы ключей предпочтительнее.

6. Какие способы сортировки предпочтительнее для обработки таблиц и почему?

Сортировки с логарифмической сложностью всегда являются более предпочтительными, чем те что работают за квадратичную сложность. Так быстрая сортировка работает за O(n * log(n)), а пузырьковая за O(n^2). Исходя из размера таблицы и действий, которые с таблицей нужно осуществлять, имеет смысл обратить внимание на составление таблицы ключей и сортировать уже её, ведь это существенно сокращает время на работу программы.

Заключение

Применение типа «запись» с вариантной частью демонстрирует значительное преимущество в экономии памяти при работе с данными, содержащими разные типы записей, за счет использования объединений (union). Это позволяет выделять память только под необходимую структуру, сокращая общий объем требуемой памяти, что особенно важно при работе с большими массивами данных.

Однако, использование вариантной части требует дополнительных проверок типов данных, и внимательной работы со стороны программиста.

При сортировке данных применение ключевых структур для индексации также предоставляет заметные преимущества в скорости обработки. Сортировка по ключам существенно сокращает временные затраты по сравнению с сортировкой всех данных, особенно при больших таблицах. Пример с разными методами сортировки (быстрая и пузырьковая) показывает значительное ускорение при использовании ключей, особенно для сложных и объемных таблиц.