

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА
по дисциплине: «Программирование»
на тему: «Класс – структура данных в двоичном файле»

Факультет: АВТФ
Группа: АВТ-143
Студент: Васютин А. М.
Преподаватель: Новицкая Ю. В.
Представлено к защите: 24.12.2022

Новосибирск, 2022 г.

ЗАДАНИЕ

Создать класс, производный от `fstream`. Файл содержит массив указателей на упорядоченные по алфавиту строки, представленные записями переменной длины:

- целый счетчик
- и последовательность символов

Формат файла:

- размер массива указателей
- текущее количество указателей
- адрес массива указателей в файле.

Обеспечить полную функциональность класса. Протестировать его. Программа тестирования должна содержать меню, обеспечивающее выбор операций.

ОГЛАВЛЕНИЕ

Введение	3
1 Структура	4
1.1 Структура класса	4
1.2 Структура итератора	4
2 Функционал	5
2.1 Конструкторы	5
2.2 Деструктор	6
2.3 Открытие файла	6
2.4 Заккрытие файла	7
2.5 Получение размера массива	7
2.6 Бинарный поиск	7
2.7 Поиск	8
2.8 Добавление записи	8
2.9 Получение записи по индексу	9
2.10 Удаление записи	10
2.11 Дефрагментация	10
2.12 Исключения	11
2.13 Итератор	12
3 Описание пользовательского интерфейса	15
3.1 quit	15
3.2 open [filename]	16
3.3 close	16
3.4 save	17
3.5 defrag	17
3.6 show	18
3.7 find [string]	18
3.8 add [string]	19
3.9 remove [index]	19
3.10 at [index]	20
3.11 help	20
4 Скорость работы структуры данных	22
Заключение	25
Приложения	26
Листинг программы	26

ВВЕДЕНИЕ

В данной работе требуется спроектировать и реализовать структуру данных, представляющую собой

- Заголовок, в котором хранятся сведения о дальнейшем расположении объектов

- Записи определенного формата и массив, хранящий их в лексикографическом порядке.

Так же нужно реализовать класс-итератор, в силу неизменяемой семантики элемента массива который будет являться константным.

1. СТРУКТУРА

1.1. Структура класса

```
class SortedStringArray : public std::fstream {
    /* ... */
private:
    size_t capacity_;
    size_t size_;
    pos_type strings_;
    pos_type offset_;
    /* ... */
    class FileNotOpen : public std::exception {
        /* ... */
    };
    struct ConstIterator;
    /* ... */
};
```

Файл с точки зрения организации данных представляет собой заголовок, в котором имеются 4 поля (*capacity_*, *size_*, *strings_*, *offset_*)

- *capacity_* максимальный размер массива записей.
- *size_* текущий размер массива записей (количество записей).
- *strings_* позиция первого байта массива записей, можно интерпретировать как адрес в файле.
- *offset_* адрес следующего за последним байта файла, может не быть равным размеру файла в силу дефрагментации (данные сдвигаются влево).

1.2. Структура итератора

```
struct SortedStringArray::ConstIterator {
    /* ... */
private:
    size_type position_;
    SortedStringArray *ssa_;
    /* ... */
};
```

2. ФУНКЦИОНАЛ

Полный набор методов, реализованных в классе, подробное их описание находится далее.

```
private:
    void throw_if_not_open();
    size_t _get_file_size();
    double _get_used_memory_ratio();
    void _write(pos_type pos, const char *val, size_t size);
    void _read(pos_type pos, char *buf, size_t size);
    void _read_header();
    void _write_header();
    void _alloc();
    void _resize();
public:
    SortedStringArray() = default;
    SortedStringArray(const char *filename);
    SortedStringArray(const std::string &filename);
    ~SortedStringArray();
    void open(const char *filename);
    void open(const std::string &filename);
    void close();
    size_type size() const;
    size_type bisect_left(const_reference str);
    size_type bisect_right(const_reference str);
    size_type bisect(const_reference str);
    int find(const_reference str);
    int rfind(const_reference str);
    void add(const_reference str);
    value_type at(size_type index);
    void remove(size_type index);
    void defragment();
#ifdef NDEBUG
    void print_debug_info();
#endif
    ConstIterator begin() const;
    ConstIterator end() const;
```

2.1. Конструкторы

```
SortedStringArray() = default;
```

```
SortedStringArray::SortedStringArray(const char *filename) {
    open(filename);
}
```

```
SortedStringArray::SortedStringArray(const std::string &filename)
: SortedStringArray(filename.c_str()) {}
```

Конструктор, принимающий константный референс на строку, вызывает другой конструктор, принимающий указатель на константный массив символов, ограниченных нулем. Он же в свою очередь вызывает метод *open(const char*)* для полученного через параметры имени файла.

2.2. Деструктор

```
SortedStringArray::~~SortedStringArray() {
    close();
}
```

Деструктор вызывает метод *close()*.

2.3. Открытие файла

Имеются две перегрузки метода открытия файла

```
void SortedStringArray::open(const char *filename) {
    bool is_empty = false;
    std::fstream::open(filename, in | out | binary);
    if (!is_open()) {
        is_empty = true;
        std::ofstream ofs(filename);
        ofs.close();
        std::fstream::open(filename, in | out | binary);
    }
    throw_if_not_open();

    capacity_ = MINIMAL_CAPACITY;
    size_ = 0;
    strings_ = FILE_HEADER_SIZE;
    offset_ = strings_ + capacity_ * sizeof(pos_type);

    if (is_empty || _get_file_size() < FILE_HEADER_SIZE) {
        _write_header();
        _alloc();
    } else {
        _read_header();
    }
}

void SortedStringArray::open(const std::string &filename) {
    open(filename.c_str());
}
```

Вторая перегрузка метода вызывает первую. Первая перегрузка пытается открыть файл и если он не открыт, то создает его, и открывает вновь (данное поведение вызывается если файла нет, следовательно, его нельзя открыть в режиме для чтения) после идет проверка на открытие файла (вызывающая исключение в обратном случае). Далее предварительно инициализируются поля и если файл был создан этим методом или размер файла меньше размера заголовка, то в файл пишется заголовок и пишется пустой массив начального размера, иначе заголовок читается из файла.

2.4. Заккрытие файла

```
void SortedStringArray::close() {  
    _write_header();  
    std::fstream::close();  
}
```

Метод *close()* записывает заголовок и вызывает метод (*std::fstream::close()*).

2.5. Получение размера массива

```
SortedStringArray::size_type SortedStringArray::size() const {  
    return size_;  
}
```

2.6. Бинарный поиск

```
SortedStringArray::size_type  
SortedStringArray::bisect_left(const_reference str) {  
    auto lo = 0;  
    auto hi = size_;  
  
    while (lo < hi) {  
        auto mid = (lo + hi) / 2;  
        if (str <= at(mid)) {  
            hi = mid;  
        } else {  
            lo = mid + 1;  
        }  
    }  
    return lo;  
}  
  
SortedStringArray::size_type  
SortedStringArray::bisect_right(const_reference str) {  
    auto lo = 0;
```



```

    auto hi = size_;

    while (lo < hi) {
        auto mid = (lo + hi) / 2;
        if (str < at(mid)) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
}

SortedStringArray::size_type SortedStringArray::bisect(const_reference str) {
    return bisect_right(str);
}

```

Реализация алгоритма бинарного поиска ($O(\log n)$). *bisect_left* ищет индекс первого элемента (если они повторяются), когда как *bisect_right* и его синоним *bisect* ищет индекс элемента, следующего за самым правым (если они повторяются).

2.7. Поиск

```

int SortedStringArray::find(const_reference str) {
    auto pos = bisect_left(str);
    if (pos < size_ && at(pos) == str)
        return pos;
    return -1;
}

int SortedStringArray::rfind(const_reference str) {
    auto pos = bisect_right(str);
    if (pos < size_ && at(pos - 1) == str)
        return pos;
    return -1;
}

```

Функции поиска основаны на бинарном поиске и если элемент не был найден, то возвращается -1 .

2.8. Добавление записи

```

void SortedStringArray::add(const_reference str) {
    throw_if_not_open();
    const auto size = str.size();

    if (size_ + 1 > capacity_) {

```

```

    _resize();
}

pos_type pos = offset_;
_write(pos, reinterpret_cast<const char*>(&size), sizeof(size));
_write(pos + sizeof(size), str.c_str(), sizeof(char) * size);
offset_ = pos + sizeof(pos) + size;

long long position = bisect(str);

if (size_)
    for (long i = (long)size_ - 1; i >= position; --i) {
        pos_type buf = 0;
        _read(strings_ + i * sizeof(buf), reinterpret_cast<char*>(&buf),
            sizeof(buf));
        _write(strings_ + (i + 1) * sizeof(buf),
            reinterpret_cast<const char*>(&buf), sizeof(buf));
    }
_write(strings_ + position * sizeof(pos),
    reinterpret_cast<const char*>(&pos), sizeof(pos));
++size_;
}

```

Добавление новой записи происходит в 2 этапа, в конец файла (по позиции *offset_*) записывается строка в формате – длина строки (беззнаковое целое число, длиной 4 байта), после бинарным поиском ищется и записывается позиция в массиве для записи адреса начала записи, предварительно сдвинув следующие индексы (при необходимости расширяет массив).

2.9. Получение записи по индексу

```

SortedStringArray::value_type
SortedStringArray::at(size_type index) {
    throw_if_not_open();
    if (index >= size_)
        throw std::out_of_range("array index out of range");

    pos_type pos = 0;
    _read(strings_ + index * sizeof(pos_type),
        reinterpret_cast<char*>(&pos), sizeof(pos));
    size_t size = 0;
    _read(pos, reinterpret_cast<char*>(&size), sizeof(size));

    seekg(pos + sizeof(size));
    std::string result(size, 0);
    for (pos_type i = 0; i < size; ++i)
        result[i] = get();
    return result;
}

```

Индекс валидируется, читается позиция записи, после длина строки, создается строка нужной длины и в неё читается запись.

2.10. Удаление записи

```
void SortedStringArray::remove(size_type index) {
    throw_if_not_open();
    if (index >= size_)
        throw std::out_of_range("array index out of range");

    pos_type next_pos = 0;
    for (pos_type pos = index; pos + 1 < size_; ++pos) {
        _read((pos + 1) * sizeof(pos) + strings_,
              reinterpret_cast<char*>(&next_pos),
              sizeof(next_pos));
        _write(pos * sizeof(pos) + strings_,
               reinterpret_cast<char*>(&next_pos),
               sizeof(next_pos));
    }
    --size_;
}
```

Индекс валидируется, элементы сдвигаются на одну влево, затирая элемент по нужному индексу.

2.11. Дефрагментация

```
void SortedStringArray::defragment() {
    throw_if_not_open();
    typedef struct {
        pos_type pos;
        pos_type index;
    } defrag_item;
    std::vector<defrag_item> strings(size_);

    for (pos_type i = 0; i < size_; ++i) {
        strings[i].index = i;
        _read(strings_ + i * sizeof(i), reinterpret_cast<char*>(&strings[i].pos),
              sizeof(strings[i].pos));
    }

    std::sort(strings.begin(), strings.end(),
              [](defrag_item &di1, defrag_item &di2) -> bool {
                  return di1.pos < di2.pos;
              });

    bool shifted = false;
    pos_type new_offset = FILE_HEADER_SIZE;
    for (auto di : strings) {
        if (di.pos > strings_ && !shifted) {
```

```

    for (pos_type i = 0; i < capacity_; ++i) {
        pos_type p = 0;
        seekg(strings_ + i * sizeof(i));
        read(reinterpret_cast<char*>(&p), sizeof(i));
        seekp(new_offset + i * sizeof(i));
        write(reinterpret_cast<const char*>(&p), sizeof(i));
    }

    strings_ = new_offset;
    new_offset += capacity_ * sizeof(pos_type);
    shifted = true;
}

size_t size = 0;
_read(di.pos, reinterpret_cast<char*>(&size), sizeof(size));
_write(new_offset, reinterpret_cast<const char*>(&size), sizeof(size));

di.pos += sizeof(size);
new_offset += sizeof(size);
for (size_t i = 0; i < size; ++i) {
    seekg(di.pos + i);
    char_type c = get();
    seekp(new_offset + i);
    put(c);
}
pos_type new_pos = new_offset - sizeof(size);
_write(strings_ + di.index * sizeof(di.index),
        reinterpret_cast<const char*>(&new_pos), sizeof(new_pos));
new_offset += size;
}
offset_ = new_offset;
_write_header();
}

```

Создается вектор структур, хранящих позицию строки в файле и её индекс в массиве, этот вектор сортируется по возрастанию позиции строки в файле. Проходя по этому вектору строки сдвигаются влево (до заголовка в случае 1 сдвига и до конца предыдущей строки в общем случае) и если позиция строки в файле больше позиции массива и массив не был сдвинут, то сначала сдвигается массив, а потом и строка, после обновляется заголовок (т.к. изменилось поле *offset_*)

2.12. Исключения

```

/* sorted_string_array.hpp */
class FileNotOpen : public std::exception {
    const char *what() const noexcept override;
};

/* sorted_string_array.cpp */

```

```
const char *SortedStringArray::FileNotOpen::what() const noexcept {
    return "File not open";
}
```

2.13. Итератор

```
ConstIterator SortedStringArray::begin() const {
    return ConstIterator(const_cast<SortedStringArray *>(this), 0);
}

ConstIterator SortedStringArray::end() const {
    return ConstIterator(const_cast<SortedStringArray *>(this), size_);
}
```

Функции *begin()* и *end()* возвращают константный итератор произвольного доступа (согласно стандарту *c++20* соответствует концепту *std::random_access_iterator*).

2.13.1. Конструктор

```
using ConstIterator = SortedStringArray::ConstIterator;

ConstIterator::ConstIterator(SortedStringArray *ssa)
    : ConstIterator::ConstIterator(ssa, 0) {}

ConstIterator::ConstIterator(SortedStringArray *ssa, size_type position)
    : ssa_{ssa}, position_{position} {}

ConstIterator::ConstIterator(const ConstIterator &other)
    : ssa_{other.ssa_}, position_{other.position_} {}
```

Конструкторы инициализируют поля.

2.13.2. Деструктор

```
~ConstIterator() = default;
```

Динамических полей нет, деструктор создается компилятором.

2.13.3. Copy-assignment

```

ConstIterator &ConstIterator::operator=(const ConstIterator &other) {
    if (this == &other)
        return *this;
    ssa_ = other.ssa_;
    position_ = other.position_;
    return *this;
}

```

Копирующий оператор присваивания проверяет равенство указателей и копирует поля одного объекта в другой.

2.13.4. Операторы сравнения

```

operator<=>(const ConstIterator &) const noexcept = default;

```

Стандарт языка *c++20* вводит оператор трехстороннего сравнения и если поручить его реализацию компилятору, то он реализует все методы сравнения для класса. Равенство определяется равенством всех полей класса, другие сравнения определяются сравнениями полей в порядке их следования в классе, но т.к. итераторы разных объектов не имеет смысла сравнивать кроме как на равенство, то поведение сравнения итераторов разных объектов не имеет смысла.

2.13.5. Инкремент и декремент

```

ConstIterator &ConstIterator::operator++() {
    position_++;
    return *this;
}

ConstIterator ConstIterator::operator++(int) {
    ConstIterator res(*this);
    operator++();
    return res;
}

ConstIterator &ConstIterator::operator--() {
    position_--;
    return *this;
}

ConstIterator ConstIterator::operator--(int) {
    ConstIterator res(*this);
    operator--();
    return *this;
}

```

2.13.6. Основные арифметические операции

```
ConstIterator &ConstIterator::operator+=(size_type value) {
    position_ += value;
    return *this;
}

ConstIterator ConstIterator::operator+(size_type value) const {
    return ConstIterator{ssa_, position_ + value};
}

ConstIterator operator+(SortedStringArray::size_type value,
                        const ConstIterator &ci) {
    return ConstIterator{ci.ssa_, ci.position_ + value};
}

ConstIterator &ConstIterator::operator-=(size_type value) {
    position_ -= value;
    return *this;
}

ConstIterator ConstIterator::operator-(size_type value) const {
    return ConstIterator{ssa_, position_ - value};
}

ConstIterator::difference_type
ConstIterator::operator-(ConstIterator ci) const {
    return position_ - ci.position_;
}
```

2.13.7. Получение значения итератора

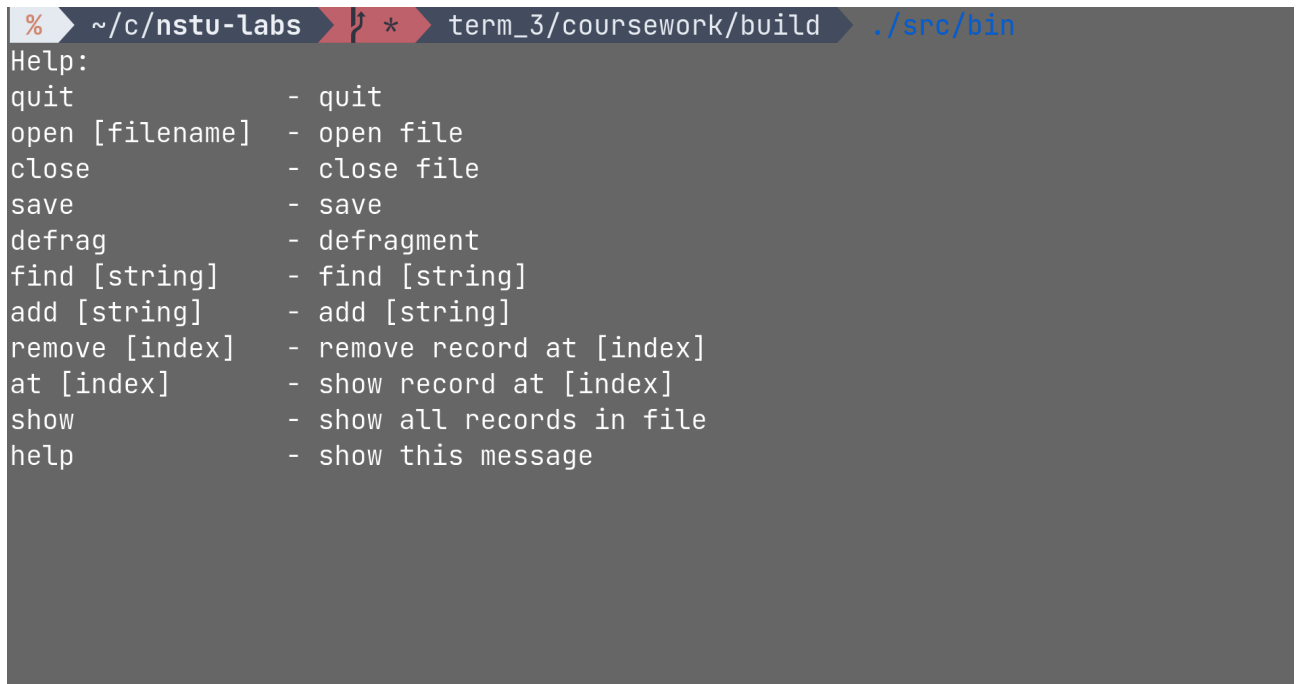
```
ConstIterator::value_type ConstIterator::operator*() const {
    return ssa_>at(position_);
}

ConstIterator::value_type ConstIterator::operator[](size_type offset) const {
    return ssa_>at(position_ + offset);
}
```

При разыменовании итератора возвращается строка по позиции поля *position_*. При вызове оператора *[]* возвращается строка по позиции *position_ + offset*.

3. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Для демонстрации работы класса был создан консольный пользовательский интерфейс.



```
% ~/c/nstu-labs * term_3/coursework/build ./src/bin
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
```

Рисунок 1 – Меню программы.

3.1. quit

Сохраняет файл, если он был открыт и выходит.


```
% ~ /c/nstu-labs * term_3/coursework/build ./src/bin
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
quit
```

Рисунок 2 – Выход из программы.

3.2. open [filename]

Открывает файл, предварительно закрыв тот, что был открыт ранее (замена файла на другой).

```
% ~ /c/nstu-labs * term_3/coursework/build ./src/bin
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
open 1.ssa
```

Рисунок 3 – Открытие файла

3.3. close

Закрывает файл, если он был открыт.

```
% ~ /c/nstu-labs * term_3/coursework/build ./src/bin
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
open 1.ssa
close
```

Рисунок 4 – Заккрытие файла.

3.4. save

Сохраняет файл, если он был открыт.

```
% ~ /c/nstu-labs * term_3/coursework/build ./src/bin
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
open 1.ssa
save
```

Рисунок 5 – Сохранение файла.

3.5. defrag

Производит дефрагментацию файла (2.11).

```
% ~/c/nstu-labs * term_3/coursework/build ./src/bin
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
open 1.ssa
defrag
```

Рисунок 6 – Дефрагментация файла.

3.6. show

Выводит массив строк.

```
% ~/c/nstu-labs * term_3/coursework/build ./src/bin
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
open 1.ssa
show
[0] - "0"
[1] - "0"
[2] - "1"
[3] - "131331"
[4] - "2"
[5] - "3"
[6] - "3"
[7] - "3"
[8] - "4"
[9] - "abc"
[10] - "abcd"
[11] - "qwerty"
[12] - "some string"
```

Рисунок 7 – Дефрагментация файла.

3.7. find [string]

Ищет строку в структуре данных, в случае, если строка не найдена выводит —1.

```

quit          - quit
open [filename] - open file
close         - close file
save         - save
defrag       - defragment
find [string] - find [string]
add [string]  - add [string]
remove [index] - remove record at [index]
at [index]   - show record at [index]
show         - show all records in file
help         - show this message
open 1.ssa
show
[0] - "0"
[1] - "0"
[2] - "1"
[3] - "131331"
[4] - "2"
[5] - "3"
[6] - "3"
[7] - "3"
[8] - "4"
[9] - "abc"
[10] - "abcd"
[11] - "qwerty"
[12] - "some string"
find abc
9
find abcde
-1

```

Рисунок 8 – Дефрагментация файла.

3.8. add [string]

Добавляет строку в структуру данных.

```

show
[0] - "0"
[1] - "0"
[2] - "1"
[3] - "131331"
[4] - "2"
[5] - "3"
[6] - "3"
[7] - "3"
[8] - "4"
[9] - "abc"
[10] - "abcd"
[11] - "qwerty"
[12] - "some string"
add ssa
show
[0] - "0"
[1] - "0"
[2] - "1"
[3] - "131331"
[4] - "2"
[5] - "3"
[6] - "3"
[7] - "3"
[8] - "4"
[9] - "abc"
[10] - "abcd"
[11] - "qwerty"
[12] - "some string"
[13] - "ssa"

```

Рисунок 9 – Добавление записи.

3.9. remove [index]

Удаляет запись из структуры данных.

```

show
[0] - "0"
[1] - "0"
[2] - "1"
[3] - "131331"
[4] - "2"
[5] - "3"
[6] - "3"
[7] - "3"
[8] - "4"
[9] - "abc"
[10] - "abcd"
[11] - "qwerty"
[12] - "some string"
[13] - "ssa"
remove 1
show
[0] - "0"
[1] - "1"
[2] - "131331"
[3] - "2"
[4] - "3"
[5] - "3"
[6] - "3"
[7] - "4"
[8] - "abc"
[9] - "abcd"
[10] - "qwerty"
[11] - "some string"
[12] - "ssa"

```

Рисунок 10 – Удаление записи.

3.10. at [index]

Получение значения по индексу.

```

show
[0] - "0"
[1] - "0"
[2] - "1"
[3] - "131331"
[4] - "2"
[5] - "3"
[6] - "3"
[7] - "3"
[8] - "4"
[9] - "abc"
[10] - "abcd"
[11] - "qwerty"
[12] - "some string"
[13] - "ssa"
remove 1
show
[0] - "0"
[1] - "1"
[2] - "131331"
[3] - "2"
[4] - "3"
[5] - "3"
[6] - "3"
[7] - "4"
[8] - "abc"
[9] - "abcd"
[10] - "qwerty"
[11] - "some string"
[12] - "ssa"

```

Рисунок 11 – Получение записи по индексу

3.11. help

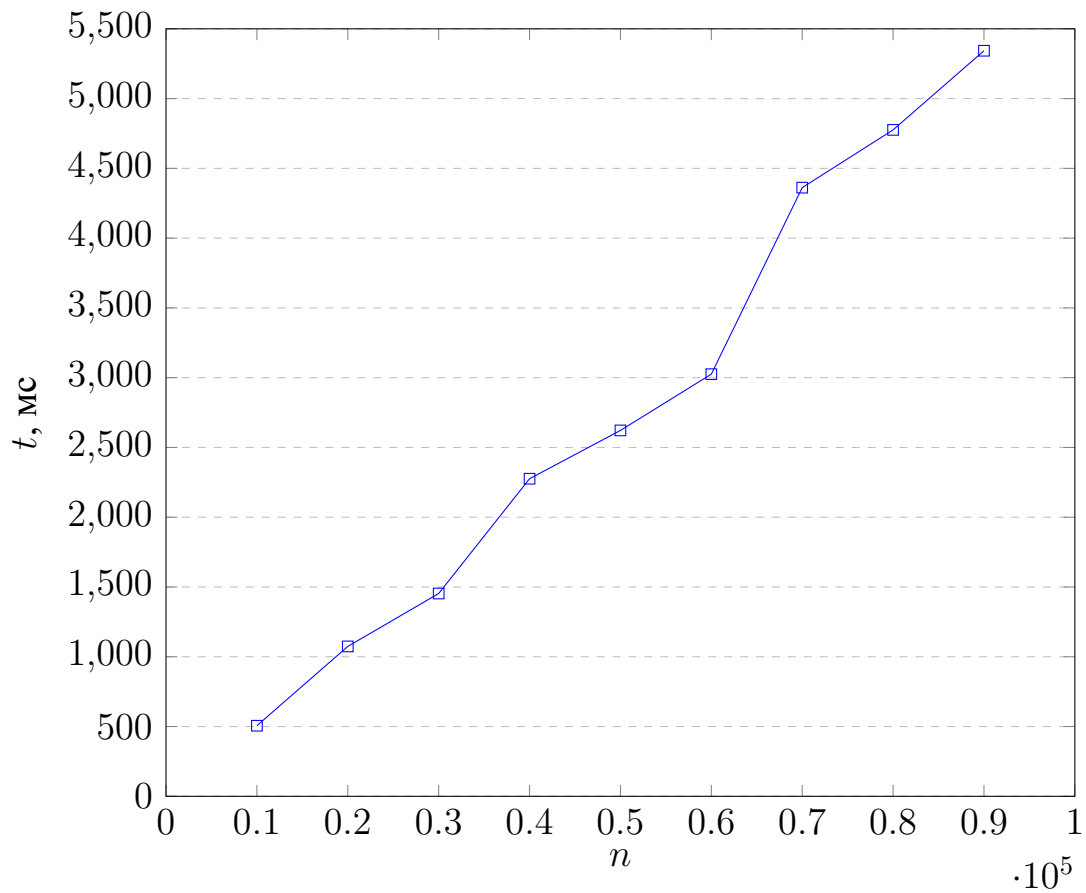
Выводит список команд и их краткое описание.

```
help
Help:
quit          - quit
open [filename] - open file
close         - close file
save          - save
defrag        - defragment
find [string]  - find [string]
add [string]   - add [string]
remove [index] - remove record at [index]
at [index]     - show record at [index]
show          - show all records in file
help          - show this message
█
```

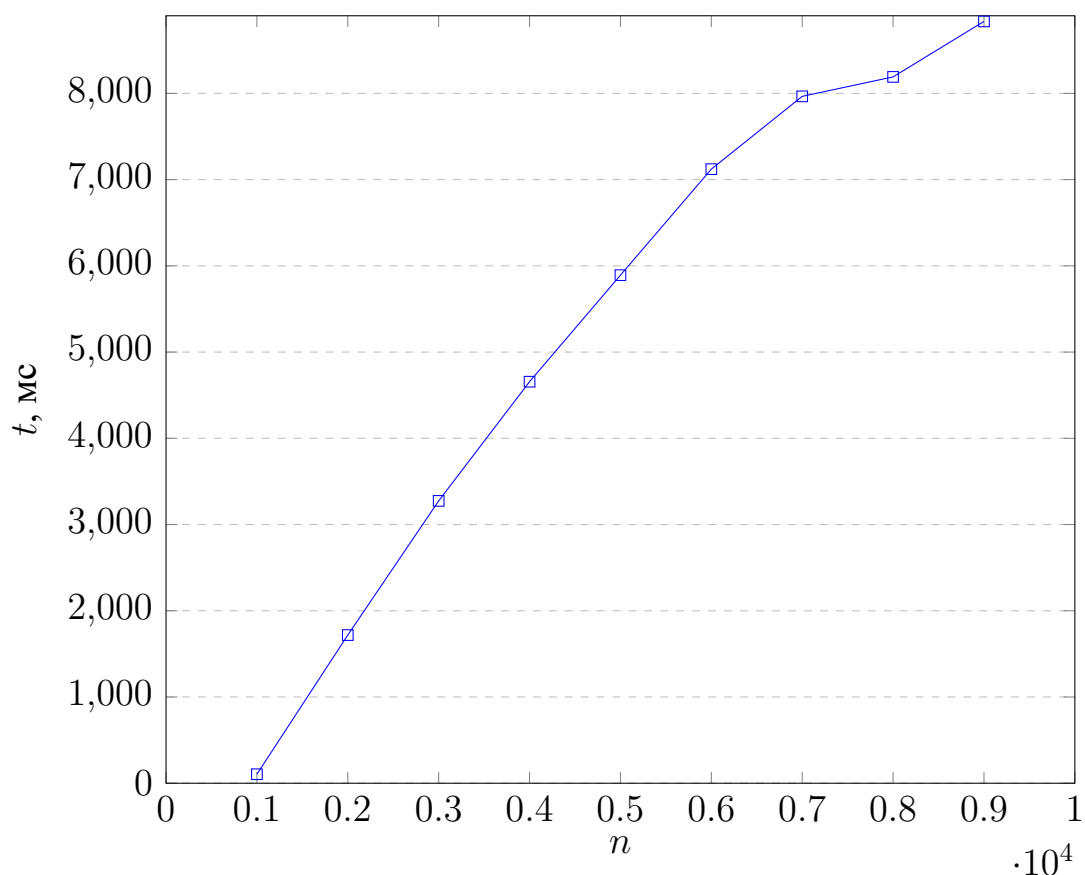
Рисунок 12 – Список команд.

4. СКОРОСТЬ РАБОТЫ СТРУКТУРЫ ДАННЫХ

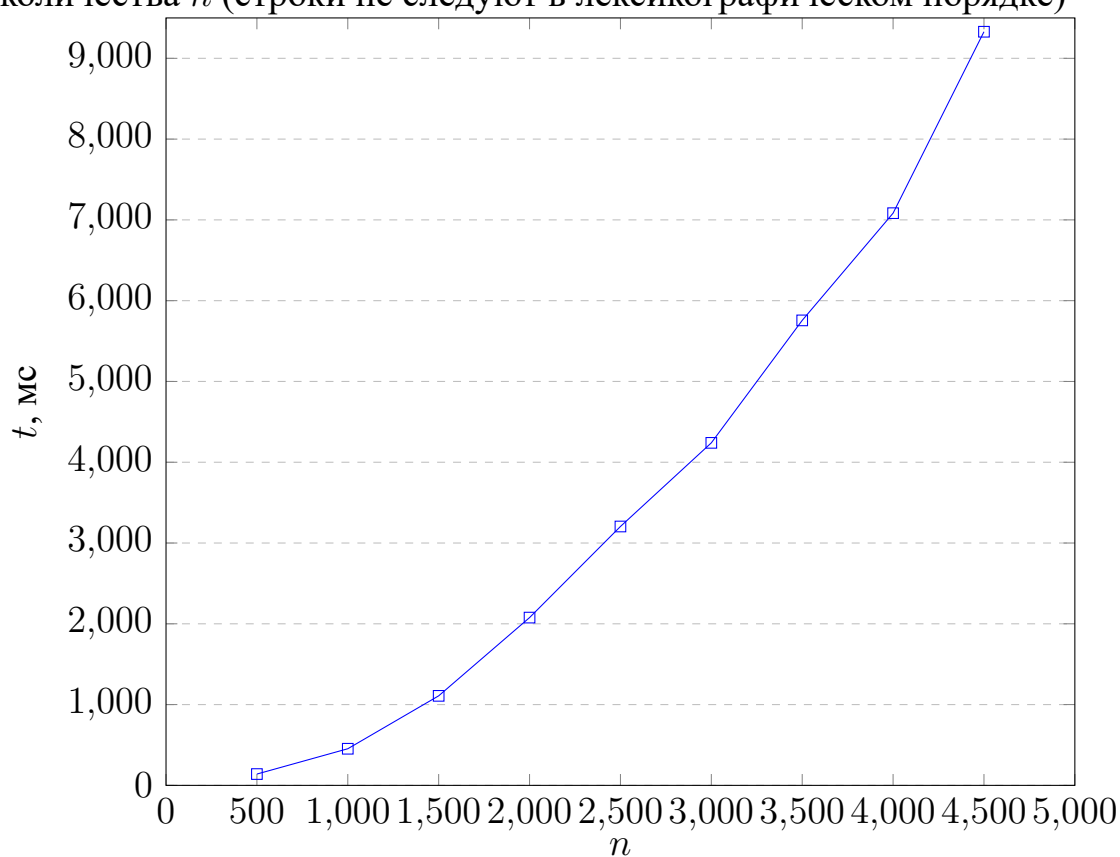
Зависимость времени заполнения структуры одинаковыми строками от их количества n



Зависимость времени заполнения структуры различными строками от их количества n (строки следуют друг за другом в лексикографическом порядке)



Зависимость времени заполнения структуры различными строками от их количества n (строки не следуют в лексикографическом порядке)



Проанализировав графики можно сделать выводы, что на одинаковых данных и данных, идущих в лексикографическом порядке, структура данных

ведет себя как прямая. Хотя и асимптотически зависимость должна быть логарифмическая, т.к. в таком случае вставка занимает $O(1)$, но из-за расширений массива оно сводится к линейной в полученных частных случаях. В случае случайных данных следует еще и за $O(n)$ вставлять элементы в массив, что придает зависимости экспоненциальный вид, но довольно близкий к прямой линии, т.к. расширения массива происходят все реже.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была спроектирована и реализована структура данных, представляющая собой массив строк в лексикографическом порядке, хранящимся, используя парадигму объектно-ориентированного программирования.

В частности, для структуры данных был разработан пользовательский интерфейс, с помощью которого она была протестирована. Задание предполагает наследование от класса *std::fstream*, что позволяет расширять функционал класса и что несомненно является преимуществом объектно-ориентированной парадигмы программирования.

ПРИЛОЖЕНИЯ

Листинг программы

main.cpp

```
#include <iostream>
#include <sstream>
#include <string>
#include <unordered_map>

#include "commands.hpp"
#include "sorted_string_array.hpp"

int main() {
    coursework::editor::Editor editor;
    std::unordered_map<std::string, coursework::editor::Command *> commands = {
        {"quit", new coursework::editor::QuitCommand(&editor)},
        {"open", new coursework::editor::OpenCommand(&editor)},
        {"close", new coursework::editor::CloseCommand(&editor)},
        {"save", new coursework::editor::SaveCommand(&editor)},
        {"defrag", new coursework::editor::DefragmentCommand(&editor)},
        {"find", new coursework::editor::FindCommand(&editor)},
        {"add", new coursework::editor::AddCommand(&editor)},
        {"remove", new coursework::editor::RemoveCommand(&editor)},
        {"at", new coursework::editor::AtCommand(&editor)},
        {"show", new coursework::editor::ShowCommand(&editor)},
        {"help", new coursework::editor::HelpCommand(&editor)},
    };
    bool quit = false;
    std::string query;
    coursework::editor::Command::Status status =
        coursework::editor::Command::Status::ok;

    std::string q = "";
    commands["help"]->execute(q);

    while (!quit) {
        std::getline(std::cin, query);

        std::stringstream ss(query);
        std::string command_name, args;

        ss >> command_name;
        ss.get();
        std::getline(ss, args);

        try {
            auto command = commands.at(command_name);
            try {
                status = command->execute(args);
            } catch (std::out_of_range) {
```

```

        std::cout << "this index out of range" << std::endl;
    } catch (std::invalid_argument) {
        std::cout << "invalid argument for command" << std::endl;
    } catch (coursework::SortedStringArray::FileNotOpen) {
        std::cout << "file not open" << std::endl;
    }
} catch (std::out_of_range) {}
quit = quit || (status == coursework::editor::Command::Status::exit);
}

for (const auto &[key, command] : commands) {
    delete command;
}
return 0;
}

```

sorted_string_array.hpp

```

#pragma once

#include <compare>
#include <cstdint>
#include <exception>
#include <fstream>
#include <ios>
#include <iterator>
#include <streambuf>
#include <string>

namespace coursework {

    class SortedStringArray : public std::fstream {
        using pos_type = std::size_t;

    private:
        std::size_t capacity_;
        std::size_t size_;
        pos_type strings_;
        pos_type offset_;

        void throw_if_not_open();
        size_t _get_file_size();
        double _get_used_memory_ratio();
        void _write(pos_type pos, const char *val, size_t size);
        void _read(pos_type pos, char *buf, size_t size);

        void _read_header();
        void _write_header();

        void _alloc();
        void _resize();

    public:

```

```

using size_type = std::size_t;
using value_type = std::string;
using reference = std::string &;
using const_reference = const std::string &;

SortedStringArray() = default;
SortedStringArray(const char *filename);
SortedStringArray(const std::string &filename);
~SortedStringArray();

void open(const char *filename);
void open(const std::string &filename);
void close();
size_type size() const;
size_type bisect_left(const_reference str);
size_type bisect_right(const_reference str);
size_type bisect(const_reference str);
int find(const_reference str);
int rfind(const_reference str);
void add(const_reference str);
value_type at(size_type index);
void remove(size_type index);
void defragment();
#ifdef NDEBUB
void print_debug_info();
#endif

class FileNotOpen : public std::exception {
    const char *what() const noexcept override;
};

struct ConstIterator;
ConstIterator begin() const;
ConstIterator end() const;
};

struct SortedStringArray::ConstIterator {
    using size_type = SortedStringArray::size_type;
    using iterator_category = std::random_access_iterator_tag;
    using difference_type = std::size_t;
    using value_type = std::string;
    using pointer = const std::string *;
    using reference = const std::string &;

private:
    size_type position_;
    SortedStringArray *ssa_;

public:
    ConstIterator(SortedStringArray *);
    ConstIterator(SortedStringArray *, size_type);
    ConstIterator(const ConstIterator &);
    ~ConstIterator() = default;

    ConstIterator &operator=(const ConstIterator &);
    std::strong_ordering

```

```

operator<=>(const ConstIterator &) const noexcept = default;

ConstIterator &operator++();
ConstIterator operator++(int);
ConstIterator &operator--();
ConstIterator operator--(int);
ConstIterator &operator+=(size_type);
ConstIterator operator+(size_type) const;
friend ConstIterator operator+(size_type, const ConstIterator &);
ConstIterator &operator--=(size_type);
ConstIterator operator-(size_type) const;
difference_type operator-(ConstIterator) const;

value_type operator*() const;
value_type operator[](size_type) const;
};
} // namespace coursework

```

sorted_string_array.cpp

```

#include "sorted_string_array.hpp"

#include <algorithm>
#include <cmath>
#include <compare>
#include <fstream>
#include <stdexcept>
#include <vector>

#ifdef NDEBUG
#include <cassert>
#include <iostream>
#endif

#define FILE_HEADER_SIZE \
    (sizeof(capacity_) + sizeof(size_) + sizeof(strings_) + sizeof(offset_))
#define MINIMAL_CAPACITY (1 << 4)
#define RESIZE_FACTOR (2)

namespace coursework {

void SortedStringArray::throw_if_not_open() {
    if (!is_open())
        throw FileNotOpen();
}

size_t SortedStringArray::_get_file_size() {
    seekg(0, std::fstream::beg);
    const auto beginp = tellg();
    seekg(0, std::fstream::end);
    const auto endp = tellg();
    return endp - beginp;
}
}

```

```

void SortedStringArray::_write(pos_type pos, const char *value, size_t size) {
    seekp(pos);
    write(value, size);
}

void SortedStringArray::_read(pos_type pos, char *buf, size_t size) {
    seekg(pos);
    read(buf, size);
}

void SortedStringArray::_read_header() {
    seekg(0);
    read(reinterpret_cast<char *>(&capacity_), sizeof(capacity_));
    read(reinterpret_cast<char *>(&size_), sizeof(size_));
    read(reinterpret_cast<char *>(&strings_), sizeof(strings_));
    read(reinterpret_cast<char *>(&offset_), sizeof(offset_));
}

void SortedStringArray::_write_header() {
    seekp(0);
    write(reinterpret_cast<const char *>(&capacity_), sizeof(capacity_));
    write(reinterpret_cast<const char *>(&size_), sizeof(size_));
    write(reinterpret_cast<const char *>(&strings_), sizeof(strings_));
    write(reinterpret_cast<const char *>(&offset_), sizeof(offset_));
}

void SortedStringArray::_alloc() {
    const auto offset = strings_ + capacity_ * sizeof(pos_type);
    seekp(offset);
    for (; offset_ < offset; ++offset_)
        put(0);
}

void SortedStringArray::_resize() {
    pos_type old_strings = strings_;
    size_t old_size = size_;
    strings_ = offset_;
    capacity_ *= RESIZE_FACTOR;
    _alloc();

    for (size_t i = 0; i < old_size * sizeof(pos_type); ++i) {
        seekg(old_strings + i);
        char_type c = get();
        seekp(strings_ + i);
        put(c);
    }
}

double SortedStringArray::_get_used_memory_ratio() {
    size_t used_bytes = FILE_HEADER_SIZE + capacity_ * sizeof(pos_type);
    for (pos_type i = 0; i < size_; ++i)
        used_bytes += at(i).size() + sizeof(size_t);
    return static_cast<double>(used_bytes) / offset_;
}

```

```

SortedStringArray::SortedStringArray(const char *filename) {
    open(filename);
}

SortedStringArray::SortedStringArray(const std::string &filename)
    : SortedStringArray(filename.c_str()) {}

SortedStringArray::~SortedStringArray() {
    close();
}

void SortedStringArray::open(const char *filename) {
    bool is_empty = false;
    std::fstream::open(filename, in | out | binary);
    if (!is_open()) {
        is_empty = true;
        std::ofstream ofs(filename);
        ofs.close();
        std::fstream::open(filename, in | out | binary);
    }
    throw_if_not_open();

    capacity_ = MINIMAL_CAPACITY;
    size_ = 0;
    strings_ = FILE_HEADER_SIZE;
    offset_ = strings_ + capacity_ * sizeof(pos_type);

    if (is_empty || _get_file_size() < FILE_HEADER_SIZE) {
        _write_header();
        _alloc();
    } else {
        _read_header();
    }
}

void SortedStringArray::open(const std::string &filename) {
    open(filename.c_str());
}

void SortedStringArray::close() {
    _write_header();
    std::fstream::close();
}

SortedStringArray::size_type SortedStringArray::size() const {
    return size_;
}

SortedStringArray::size_type
SortedStringArray::bisect_left(const_reference str) {
    auto lo = 0;
    auto hi = size_;

```



```

    while (lo < hi) {
        auto mid = (lo + hi) / 2;
        if (str <= at(mid)) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
}

SortedStringArray::size_type
SortedStringArray::bisect_right(const_reference str) {
    auto lo = 0;
    auto hi = size_;

    while (lo < hi) {
        auto mid = (lo + hi) / 2;
        if (str < at(mid)) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
}

SortedStringArray::size_type SortedStringArray::bisect(const_reference str) {
    return bisect_right(str);
}

int SortedStringArray::find(const_reference str) {
    auto pos = bisect_left(str);
    if (pos < size_ && at(pos) == str)
        return pos;
    return -1;
}

int SortedStringArray::rfind(const_reference str) {
    auto pos = bisect_left(str);
    if (pos < size_ && at(pos - 1) == str)
        return pos;
    return -1;
}

void SortedStringArray::add(const_reference str) {
    throw_if_not_open();
    const auto size = str.size();

    if (size_ + 1 > capacity_) {
        _resize();
    }

    pos_type pos = offset_;

```

```

_write(pos, reinterpret_cast<const char *>(&size), sizeof(size));
_write(pos + sizeof(size), str.c_str(), sizeof(char) * size);
offset_ = pos + sizeof(pos) + size;

long long position = bisect(str);

if (size_)
    for (long i = (long)size_ - 1; i >= position; --i) {
        pos_type buf = 0;
        _read(strings_ + i * sizeof(buf), reinterpret_cast<char *>(&buf),
            sizeof(buf));
        _write(strings_ + (i + 1) * sizeof(buf),
            reinterpret_cast<const char *>(&buf), sizeof(buf));
    }
_write(strings_ + position * sizeof(pos),
    reinterpret_cast<const char *>(&pos), sizeof(pos));
++size_;
}

SortedStringArray::value_type SortedStringArray::at(size_type index) {
    throw_if_not_open();
    if (index >= size_)
        throw std::out_of_range("array index out of range");

    pos_type pos = 0;
    _read(strings_ + index * sizeof(pos_type), reinterpret_cast<char *>(&pos),
        sizeof(pos));
    size_t size = 0;
    _read(pos, reinterpret_cast<char *>(&size), sizeof(size));

    seekg(pos + sizeof(size));
    std::string result(size, 0);
    for (pos_type i = 0; i < size; ++i)
        result[i] = get();
    return result;
}

void SortedStringArray::remove(size_type index) {
    throw_if_not_open();
    if (index >= size_)
        throw std::out_of_range("array index out of range");

    pos_type next_pos = 0;
    for (pos_type pos = index; pos + 1 < size_; ++pos) {
        _read((pos + 1) * sizeof(pos) + strings_,
            reinterpret_cast<char *>(&next_pos), sizeof(next_pos));
        _write(pos * sizeof(pos) + strings_, reinterpret_cast<char *>(&next_pos),
            sizeof(next_pos));
    }
    --size_;
}

void SortedStringArray::defragment() {
    throw_if_not_open();

```

```

typedef struct {
    pos_type pos;
    pos_type index;
} defrag_item;
std::vector<defrag_item> strings(size_);

for (pos_type i = 0; i < size_; ++i) {
    strings[i].index = i;
    _read(strings_ + i * sizeof(i), reinterpret_cast<char*>(&strings[i].pos),
          sizeof(strings[i].pos));
}

std::sort(strings.begin(), strings.end(),
          [](defrag_item &di1, defrag_item &di2) -> bool {
              return di1.pos < di2.pos;
          });

bool shifted = false;
pos_type new_offset = FILE_HEADER_SIZE;
for (auto di : strings) {
    if (di.pos > strings_ && !shifted) {
        for (pos_type i = 0; i < capacity_; ++i) {
            pos_type p = 0;
            seekg(strings_ + i * sizeof(i));
            read(reinterpret_cast<char*>(&p), sizeof(i));
            seekp(new_offset + i * sizeof(i));
            write(reinterpret_cast<const char*>(&p), sizeof(i));
        }

        strings_ = new_offset;
        new_offset += capacity_ * sizeof(pos_type);
        shifted = true;
    }

    size_t size = 0;
    _read(di.pos, reinterpret_cast<char*>(&size), sizeof(size));
    _write(new_offset, reinterpret_cast<const char*>(&size), sizeof(size));

    di.pos += sizeof(size);
    new_offset += sizeof(size);
    for (size_t i = 0; i < size; ++i) {
        seekg(di.pos + i);
        char_type c = get();
        seekp(new_offset + i);
        put(c);
    }
    pos_type new_pos = new_offset - sizeof(size);
    _write(strings_ + di.index * sizeof(di.index),
          reinterpret_cast<const char*>(&new_pos), sizeof(new_pos));
    new_offset += size;
}
offset_ = new_offset;
_write_header();
}

```

```

#ifdef NDEBUG
void SortedStringArray::print_debug_info() {
    std::cout << "[DEBUG INFO]" << std::endl;
    if (!is_open()) {
        std::cout << "File not open" << std::endl;
        return;
    }

    std::cout << "_get_file_size: " << _get_file_size() << '\n'
        << "capacity_: " << capacity_ << '\n'
        << "size_: " << size_ << '\n'
        << "strings_: " << strings_ << '\n'
        << "offset_: " << offset_ << '\n'
        << "used memory ratio: " << _get_used_memory_ratio() << std::endl;

    pos_type pos = 0;
    auto it = begin();
    for (pos_type i = 0; i < size_; ++i, ++it) {
        _read(strings_ + i * sizeof(pos_type), reinterpret_cast<char*>(&pos),
            sizeof(pos));
        auto str = *it;
        std::cout << "[" << i << "]: "
            << "(" << str.size() << ") "
            << "\"" << str << "\"" << std::endl;
    }
}
#endif

const char *SortedStringArray::FileNotOpen::what() const noexcept {
    return "File not open";
}

using ConstIterator = SortedStringArray::ConstIterator;

ConstIterator::ConstIterator(SortedStringArray *ssa)
    : ConstIterator::ConstIterator(ssa, 0) {}

ConstIterator::ConstIterator(SortedStringArray *ssa, size_type position)
    : ssa_{ssa}, position_{position} {}

ConstIterator::ConstIterator(const ConstIterator &other)
    : ssa_{other.ssa_}, position_{other.position_} {}

ConstIterator &ConstIterator::operator=(const ConstIterator &other) {
    if (this == &other)
        return *this;
    ssa_ = other.ssa_;
    position_ = other.position_;
    return *this;
}

ConstIterator &ConstIterator::operator++() {
    position_++;
}

```

```

    return *this;
}

ConstIterator ConstIterator::operator++(int) {
    ConstIterator res(*this);
    operator++();
    return res;
}

ConstIterator &ConstIterator::operator--() {
    position_--;
    return *this;
}

ConstIterator ConstIterator::operator--(int) {
    ConstIterator res(*this);
    operator--();
    return *this;
}

ConstIterator &ConstIterator::operator+=(size_type value) {
    position_ += value;
    return *this;
}

ConstIterator ConstIterator::operator+(size_type value) const {
    return ConstIterator{ssa_, position_ + value};
}

ConstIterator operator+(SortedStringArray::size_type value,
                        const ConstIterator &ci) {
    return ConstIterator{ci.ssa_, ci.position_ + value};
}

ConstIterator &ConstIterator::operator+=(size_type value) {
    position_ += value;
    return *this;
}

ConstIterator ConstIterator::operator-(size_type value) const {
    return ConstIterator{ssa_, position_ - value};
}

ConstIterator::difference_type
ConstIterator::operator-(ConstIterator ci) const {
    return position_ - ci.position_;
}

ConstIterator::value_type ConstIterator::operator*() const {
    return ssa_>at(position_);
}

ConstIterator::value_type ConstIterator::operator[](size_type offset) const {
    return ssa_>at(position_ + offset);
}

```

```

    }

    ConstIterator SortedStringArray::begin() const {
        return ConstIterator(const_cast<SortedStringArray *>(this), 0);
    }

    ConstIterator SortedStringArray::end() const {
        return ConstIterator(const_cast<SortedStringArray *>(this), size_);
    }
} // namespace coursework

```

commands.hpp

```

#pragma once

#include <string>

#include "sorted_string_array.hpp"

namespace coursework {
    namespace editor {
        class Editor {
        private:
            std::string filename_;
            SortedStringArray *ssa_;

        public:
            Editor(SortedStringArray *ssa = nullptr);
            ~Editor();
            void open(std::string &filename);
            void close();
            void save();
            int find(std::string &str);
            void add(std::string &str);
            void remove(size_t pos);
            void defragment();
            std::string at(size_t pos);
            size_t size();
            SortedStringArray::ConstIterator itbegin();
            SortedStringArray::ConstIterator itend();
        };

        class Command {
        protected:
            Editor *editor_;
            Command(Editor *editor);

        public:
            virtual ~Command() = default;
            enum class Status { exit, ok };
            virtual Status execute(std::string &queue) const = 0;
        };

        class QuitCommand : public Command {

```

```

public:
    QuitCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class OpenCommand : public Command {
public:
    OpenCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class CloseCommand : public Command {
public:
    CloseCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class SaveCommand : public Command {
public:
    SaveCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class FindCommand : public Command {
public:
    FindCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class AddCommand : public Command {
public:
    AddCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class RemoveCommand : public Command {
public:
    RemoveCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class DefragmentCommand : public Command {
public:
    DefragmentCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class AtCommand : public Command {
public:
    AtCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};

class ShowCommand : public Command {

```

```

public:
    ShowCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};
class HelpCommand : public Command {
public:
    HelpCommand(Editor *editor);
    Status execute(std::string &queue) const override;
};
} // namespace editor
} // namespace coursework

```

commands.cpp

```

#include <iostream>

#include "commands.hpp"
#include "sorted_string_array.hpp"

namespace coursework {
    namespace editor {
        Editor::Editor(SortedStringArray *ssa) : ssa_(ssa) {}

        Editor::~Editor() {
            if (ssa_)
                delete ssa_;
        }

        void Editor::open(std::string &filename) {
            close();
            filename_ = filename;
            ssa_ = new SortedStringArray(filename_);
        }

        void Editor::close() {
            if (ssa_)
                delete ssa_;
            ssa_ = nullptr;
        }

        void Editor::save() {
            close();
            open(filename_);
        }

        int Editor::find(std::string &str) {
            if (ssa_)
                return ssa_>find(str);
            else
                throw SortedStringArray::FileNotOpen();
        }

        void Editor::add(std::string &str) {
            if (ssa_)

```



```

        ssa_>add(str);
    else
        throw SortedStringArray::FileNotOpen();
}

void Editor::remove(size_t pos) {
    if (ssa_)
        ssa_>remove(pos);
    else
        throw SortedStringArray::FileNotOpen();
}

void Editor::defragment() {
    if (ssa_)
        ssa_>defragment();
    else
        throw SortedStringArray::FileNotOpen();
}

std::string Editor::at(size_t pos) {
    if (ssa_)
        return ssa_>at(pos);
    else
        throw SortedStringArray::FileNotOpen();
}

size_t Editor::size() {
    if (ssa_)
        return ssa_>size();
    else
        throw SortedStringArray::FileNotOpen();
}

SortedStringArray::ConstIterator Editor::itbegin() {
    return ssa_>begin();
}

SortedStringArray::ConstIterator Editor::itend() {
    return ssa_>end();
}

Command::Command(Editor *editor) : editor_(editor) {}

QuitCommand::QuitCommand(Editor *editor) : Command(editor) {}

Command::Status QuitCommand::execute(std::string &queue) const {
    editor_>close();
    return Status::exit;
}

OpenCommand::OpenCommand(Editor *editor) : Command(editor) {}

```

```

Command::Status OpenCommand::execute(std::string &queue) const {
    editor_>open(queue);
    return Status::ok;
}

CloseCommand::CloseCommand(Editor *editor) : Command(editor) {}

Command::Status CloseCommand::execute(std::string &queue) const {
    editor_>close();
    return Status::ok;
}

SaveCommand::SaveCommand(Editor *editor) : Command(editor) {}

Command::Status SaveCommand::execute(std::string &queue) const {
    editor_>save();
    return Status::ok;
}

FindCommand::FindCommand(Editor *editor) : Command(editor) {}

Command::Status FindCommand::execute(std::string &queue) const {
    std::cout << editor_>find(queue) << std::endl;
    return Status::ok;
}

AddCommand::AddCommand(Editor *editor) : Command(editor) {}

Command::Status AddCommand::execute(std::string &queue) const {
    editor_>add(queue);
    return Status::ok;
}

RemoveCommand::RemoveCommand(Editor *editor) : Command(editor) {}

Command::Status RemoveCommand::execute(std::string &queue) const {
    editor_>remove(std::stoi(queue));
    return Status::ok;
}

DefragmentCommand::DefragmentCommand(Editor *editor) : Command(editor) {}

Command::Status DefragmentCommand::execute(std::string &queue) const {
    editor_>defragment();
    return Status::ok;
}

AtCommand::AtCommand(Editor *editor) : Command(editor) {}

Command::Status AtCommand::execute(std::string &queue) const {
    std::cout << editor_>at(std::stoi(queue)) << std::endl;
    return Status::ok;
}

```

```

ShowCommand::ShowCommand(Editor *editor) : Command(editor) {}

Command::Status ShowCommand::execute(std::string &queue) const {
    auto end = editor_>itend();
    auto index = 0;
    for (auto it = editor_>itbegin(); it < end; ++it, ++index) {
        std::cout << "[" << index << " ] - \"" << *it << "\"\n"
            << std::endl;
    }

    return Status::ok;
}

HelpCommand::HelpCommand(Editor *editor) : Command(editor) {}
Command::Status HelpCommand::execute(std::string &queue) const {
    std::cout << "Help:\n"
        << "quit                - quit\n"
        << "open [filename]         - open file\n"
        << "close                   - close file\n"
        << "save                    - save\n"
        << "defrag                  - defragment\n"
        << "find [string]           - find [string]\n"
        << "add [string]            - add [string]\n"
        << "remove [index]          - remove record at [index]\n"
        << "at [index]              - show record at [index]\n"
        << "show                   - show all records in file\n"
        << "help                   - show this message" << std::endl;
    return Status::ok;
}

} // namespace editor
} // namespace coursework

```