BeBrainee: Лабораторные работы

Структуры данных, ООП, UNIX

(Для студентов 1-3 месяцев обучения)

6 October 2021

Краткое содержание

Предисловие 3
Требования к оформлению 4

Глава 1. Базовые структуры данных и алгоритмы 5
Списки 6
Очередь 8
Стек 10
Словари и множества 12
Графы и рекурсия..... 14

Глава 2. ООП и паттерны программирования 16
Экспорт данных 18
Авторизация пользователей 20
Система контроля версий 23

Глава 3. UNIX, LINUX, POSIX 26
TCP/IP мессенджер 27
LZ77 архиватор текста 29
Менеджер пользователей в UNIX системах 31

Список рекомендуемой литературы 32

Предисловие

Данный документ предназначен в первую очередь для людей, которые имеют представление о том, что такое программирование, и знакомы как минимум с одним языком программирования на элементарном уровне.

Для того, чтобы сделать все задания успешно, обязательно нужно читать параллельно с этим книги или изучать материал из любого другого удобного источника, который вам подходит больше, для увеличения теоретических знаний. Теория - это неотъемлемая часть, которая позволит вам писать более эффективный код, применяя наилучшие методы решения для конкретных задач. В конце этого документа есть список используемой литературы, если прочитать ее всю, вы будете точно знать как решать нужные задачи. Книги подобраны таким образом, чтобы разделы в них можно читать выборочно, если на весь материал у вас не хватает времени. Лучше всего сначала прочитать и ознакомиться с теоретической базой, а потом приступать к решению задачи.

Требования к оформлению

- Все решения должны быть написаны на языке C. Разрешается использовать Makefile для сборки проектов
- Разработка ведется в UNIX-подобных ОС (Linux, MacOS, FreeBSD и другие)
- Публикация кода в GitHub, репозиторий должен быть приватный
- Название репозитория должно быть HASH строкой (например, MD5)
- README.md документ в корне каждого проекта, описывающий требования к запуску проекта и методы взаимодействия с ним: инструкции по запуску, настройке и т.д. Приветствуется написание вставок bash-кода для примеров выполнения команд
- Весь текст в коде, включая инструкции по запуску, должен быть на английском языке
- Каждая функция, методы и атрибуты класса должны иметь комментарии
- Приложение должно быть покрыто тестами на столько, на сколько это возможно для проверки правильного выполнения приложения

Глава 1. Базовые структуры данных и алгоритмы

Эта вступительная часть лабораторных работ предназначена для изучения базовых структур данных, которые повсеместно используются во многих современных языках программирования, таких как C/C++, Java, Python, JavaScript и других. Правильно подбирая нужные структуры и алгоритмы для работы, вы как из кирпичиков сможете построить приложение в виде прочного, непоколебимого, бетонного дома.

Эффективное использование структур и алгоритмов - залог высокой производительности любого приложения. Правильно рассчитав затраты на память и использование вычислительных ресурсов центрального процессора, вы сможете сделать задержку для пользователя минимальной. Неэффективное приложение чревато огромной стоимостью поддержания, а также огромным неудобством для пользователя в виде увеличенного времени отклика.

В обычной жизни мы постоянно имеем дело с каким-то хранилищем. Любое приложение, даже самая простая игра - змейка или тетрис, должно хранить информацию о положении объектов на экране, их свойства, параметры игрока, общую информацию о результатах игры. Для того, чтобы правильно организовать доступ к нужным данным и сделать задержки минимальными, мы как раз и будем изучать работу и устройство главных структур данных и алгоритмов.

Списки

Описание:

Списки - наверное, самая часто используемая структура данных, которую можно встретить как в коде, так и в реальной жизни. Обычно мы имеем дело не с одним элементом, а с целым набором и очень часто мы даже не можем сказать точно, сколько их может быть, потому что элементы могут добавляться в процессе работы. Самый простой пример - обслуживание клиентов на кассе. Покупателей много и их количество всегда пополняется. Другой пример - книги на вашей полке. У них есть определенный порядок и их количество также может быть динамическим.

Задача:

А) Написать базу данных “List DataBase”, основанную на списках. База данных должна предоставлять следующее API для работы:

- Добавление элемента по индексу (если индекс больше индекса последнего элемента, можно считать это добавлением элемента в конец списка. Если индекс -1, тогда добавляем в начало списка)
- Чтение элемента по индексу
- Удаление элемента по индексу
- Поиск элемента по значению
- Получение количества элементов списка

Б) Оптимизировать получение элемента по индексу.

В) Сделать возможность сохранения базы на диск (в файл) с последующей загрузкой в память. Сгенерировать от 500 млн до 1 млрд (при генерации каждый раз значение выбирать рандомно) записей и сохранить их в базу. Свести время чтения базы до минимального.

Пример использования:

```
db = database_create()
database_insert(0, ‘Big’)
database_insert(1, ‘Flame’)
database_insert(-1, ‘Great’)

value = database_read(1)
print(value)                    # выведет ‘Big’

index = database_search(‘Big’)
print(index)                    # выведет 1

database_delete(0)              # удалит ‘Hello’ элемент
value = database_read(2)        # должно вызвать ошибку

value = database_read(0)
print(value)                    # выведет ‘Big’
```

Очередь

Описание:

Очередь - это структура данных, которая обеспечивает возможность вставки элемента в конец списка и получения с начала (первый пришел - первый вышел).

Представим себе большой магазин, который выдает заказанный товар через интернет. Людей, ожидающих свою покупку много, например, 1000. Чтобы решить проблему с длинными очередями, магазин каждому покупателю выдает талон с его номерком (такое можно встретить также в почтовых отделениях, банках и во многих других местах). Это его порядковое число в очереди. В большом зале магазина висит огромный экран, который показывает какому номеру нужно сейчас подойти на кассу и забрать свой товар. Когда кассир отдал очередному покупателю его товар, он нажимает на кнопку у себя на рабочем столе и, таким образом, номер на экране увеличивается на единицу, уведомляя следующего покупателя, что ему пора забрать товар. Покупатели же теперь могут не стоять в очереди, а ходить по магазину и смотреть товар, проверяя время от времени следующий номер на экране.

Задача:

Нужно написать приложение используя, базу данных “List DataBase” из первой лабораторной работы. Это приложение должно обеспечивать 3 функции:

- добавление нового пользователя в очередь

- получение из очереди следующего пользователя
- получение размера очереди

Сгенерировать 1 млн записей, при этом добавление и вынимание из очереди сделать в рандомном порядке.

Пример использования:

```
Queue = queue_create()
queue_push(queue, 0)
queue_push(queue, 1)
queue_push(queue, 2)
queue_push(queue, 3)
value = queue_pop(queue)

print(value)                    # выведет 0
queue_push(queue, 4)

value = queue_pop(queue)
print(value)                    # выведет 1

value = queue_pop(queue)
print(value)                    # выведет 2

value = queue_len(queue)
print(value)                    # выведет 2
```

Стек

Описание:

Стек - это структура данных, обеспечивающая вставку элемента в конец и получение элемента также с конца (последний пришел - первый вышел).

Представим себе склад из трех секций, на котором роботы сортируют коробки с различным товаром: еду, бытовую технику, гаджеты. Первый робот разгружает автомобили с товаром, которые приезжают на склад и развозит их по соответствующим секциям. Еще три робота по отдельности контролируют каждую секцию, распаковывают товар из коробок и отправляют их в другие автомобили, которые доставят товар в мелкие магазины для розничной торговли. Робот 2 смотрит за едой, Робот 3 смотрит за бытовой техникой, Робот 4 смотрит за гаджетами. Так как первый робот из автомобиля завозит коробки в секцию - небольшую комнату с одной дверью, все коробки стоят друг за другом. Существует ситуация, когда товар полностью заполнит секцию. Чтобы разгрузить секцию, нужно доставать те коробки, что ближе к выходу, иначе к задним рядам никак не добраться.

Задача:

Нужно написать приложение, которое могло бы реализовать управление такими роботами, а также контролировать товар, который отправляется на розничную торговлю. Нужно реализовать следующие 3 метода для работы со стеком:

- Добавление элемента в стек
- Получение элемента из стека
- Получение длины стека

Пример использования:

```
section_food = stack_create()
section_appliances = stack_create()
section_gadgets = stack_create()

robot_put(section_food, ‘Apples’)
robot_put(section_food, ‘Oranges’)
robot_put(section_food, ‘Meat’)

robot_put(section_appliances, ‘Microwave Owen’)
robot_put(section_appliances, ‘Refrigerator’)

robot_put(section_gadgets, ‘Smartphone’)
robot_put(section_gadgets, ‘Watches’)
robot_put(section_gadgets, ‘Headphones’)
robot_put(section_gadgets, ‘PowerBank’)

robot_pop(section_food)                 # выведет ‘Meat’
robot_pop(section_gadgets)              # выведет ‘PowerBank’
robot_pop(section_gadgets)              # выведет ‘Headphones’
robot_pop(section_appliances)           # выведет ‘Refrigerator’
robot_len(section_gadgets)              # выведет ‘2’
robot_len(section_appliances)           # выведет ‘2’
```

Словари и множества

Описание:

Словарь - это структура данных, обеспечивающая доступ к элементу по ключу за константное время.

Множество - структура, которая хранит только уникальные элементы.

Обе эти структуры можно создать с помощью специальной hash-функции, которая будет конвертировать ключ в адрес памяти.

Организовать хранилище с доступом по ключу за константное время.

Хранилище должно обеспечивать следующую функциональность:

- сохранение элемента по ключу
- получение элемента по ключу
- удаление элемента по ключу
- функцию хэширования.

- Сгенерировать 100 тысяч записей с рандомными ключами и проверить сколько коллизий было обнаружено, попробовать изменить функцию хэширования и уменьшить объем выделенной памяти под хранилище.

Задача:

А) Номерные знаки для автомобилей состоят из уникальной комбинации символов, обычно это указание страны, региона и уникальный код. В сумме это может достигать 8 символов алфавита и цифр. Мы упростим задачу: у нас будет маркировка вида XX, где X - число от 0 до 9, что составляет 100 уникальных номерных знаков (десять в квадрате). Используя хранилище из пункта 1, нужно сгенерировать рандомным образом номерные знаки автомобилей так, чтобы они использовали половину всех доступных номеров. Эти автомобильные номера будут считаться уже используемыми. После этого сгенерировать 10 других рандомных автомобильных номер и показать: используются они уже или нет.

Б) В городе Москва есть около 800 разных маршрутов для автобусов\троллейбусов и других наземных видов общественного транспорта (для простоты будем считать, что весь транспорт - автобусы). На каждом из которых ходит, например, от 5 до 20 автобусов. Каждый из автобусов имеет на своем лобовом стекле номер маршрута от 1 до 800. Нужно сгенерировать рандомное кол-во автобусов для всех маршрутов и записать его в файл в виде

750, 750, 751, 752...

что означает 2 автобуса из маршрута 750 и по одному из маршрутов 751 и 752.

Нужно сделать приложение, используя хранилище из пункта 1, которое прочитает файл с автобусами и подсчитает кл-во автобусов на всех маршрутах.

Графы. Разделяй и властвуй

Описание:

Графы - это структура данных, определяющая множество элементов и связей между ними. Каждый из элементов может иметь одну или больше связей с другими элементами множества.

Разделяй и властвуй - метод создания алгоритмов, который заключается в рекурсивном разбивании задачи на более простые части такого же типа до тех пор, пока части не станут элементарными.

Попробуем организовать сложную систему рекомендаций для пользовательской рекламы. Предположим, имеется социальная сеть, в которой пользователи могут создавать посты: это могут быть картинки, видеоролики, просто текст. Каждый из постов имеет отношение к одной или более категорий. Например, фотография спорткара попадает под категории: “автомобили”, “роскошь”, “предпринимательство”. Категории также могут иметь связи с другими категориями, например, категория “роскошь” могла бы иметь связи с “iphone”, “частный дом” и “премиум алкоголь”. Выстраивание таких сложных связей между категориями дает возможность показывать достаточно точную рекламу, либо подбирать ленту таким образом, чтобы она вызывала максимальный интерес у читателя и заставляла его проводить в приложении больше времени. Для того, чтобы показывать рекламу еще более точно, введем стоимость отдаленности связи. Например: Роскошь -> Iphone -> Чехлы для Iphone -> Ювелирные

изделия. Связь между “Роскошь” и “Iphone” будет составлять 10 очков, так как это первая связь от корневой. “Чехлы для Iphone” будет весить 5 очков, так как это вторая связь, “ювелирные изделия” - 2.5 и т.д.

Задача:

А) Сгенерировать 100 разных категорий и установить между ними рандомные связи. Приложение должно принимать два параметра от пользователя: корневая категория и текущая, посчитать релевантность категории попавшей пользователю.

Б) Используя сгенерированную структуру из пункта 1, осуществить поиск нужной выбранной категории относительно корневой с помощью рекурсивного метода “разделяй и властвуй” и вывести кол-во связи между категориями, которые пришлось пройти при поиске (кол-во связей должно быть минимальным). Оценить скорость и сложность данного метода.

Пример использования:

```
CategoriesGraph = generate_categories(100)

# выведет `10`
graph_relative(CategoriesGraph , ‘Роскошь’, ‘Iphone’)

# выведет `5`
graph_relative(CategoriesGraph, ‘Роскошь’, ‘Чехлы для Iphone’)

# выведет `2.5`
```

```
graph_relative(CategoriesGraph, ‘Роскошь’, ‘Ювелирные изделия’)

# выведет `10`
graph_relative(CategoriesGraph, ‘Чехлы для Iphone’, ‘Iphone’)

# выведет `2`
graph_find(CategoriesGraph, ’Роскошь’, `Чехлы для Iphone`)
```

Глава 2. ООП и паттерны программирования

Объектно-ориентированное программирование в современном мире занимает очень важное место, так как позволяет разработчику структурировать свое приложение и сделать его более простым в модификации. Отнесение определенных частей приложения к классам или объектам имеет прямую аналогию с тем, что мы видим в реальном мире - все вокруг состоит из объектов, многие из которых имеют похожие или одинаковые свойства. ООП предоставляет возможность разложить все эти объекты по полочкам, структурировать и влиять определенным образом на целую группу объектов, если они между собой связанны, как если бы мы модифицировали геном животного и повлияли на следующее поколение.

Но принципы ООП можно применять не только к коду, их проявления можно увидеть буквально везде, например, в автомобилях. Наследование автомобилей - общий набор обязательных параметров, прописанные международным консорциумом: наличие боковых зеркал, клаксон, колес, руля, при этом форма и дополнительные возможности различаются. Полиморфизм же можно применять сразу на все транспортные средства: автомобили, мотоциклы, поезда, судна, самолеты: все они имеют одно общее свойство - возможность передвигаться быстро на дальние расстояния, при этом реализация передвижения у каждого из них разная.

Экспорт данных

Описание:

На сегодняшний день социальные сети занимают очень большое место в жизни среднестатистического человека. Но иногда по каким-то причинам пользователи со временем переходят из одной сети в другую. Особенно это хорошо видно на примере разных мессенджеров, которых в мобильном телефоне может быть установлено иногда более пяти. В виду конкуренции между всеми этими компаниями, каждая из них идет на определенные уступки и при переходе из какой-то соц.сети в их, предлагает пользователям импорт данных из старой соц.сети. В случае с мессенджарами - импорт контактов и старых сообщений. Все эти крупные компании договорились, и дали возможность делать экспорт данных из своей соц.сети.

Задача:

А) Сгенерировать десять разных аккаунтов пользователей (имя, фамилия, возраст, адрес проживания). Написать приложение, которое будет делать экспорт данных в разных форматах: xml, json, yaml и сохранять в файл. Для форматирования текста можно использовать сторонние библиотеки. При запуске программы пользователь указывает какой именно формат данных нужно использовать.

Б) Не изменяя функцию экспорта из пункта 1, расширить ее таким образом, чтобы при сохранении в начало файла еще добавлялась

метаинформация о самом контенте: размер контента, название файла, формат контента, кол-во пользовательских записей.

Пример использования:

```
accounts = [User('Vladimir', 16, 'Kiev'), User('Dimitri', 88, 'Moscov')]

for user in accounts:
    for file_type in ['xml', 'json', 'yaml']:
        account.dump(generate_file_path(user.name, file_type), file_type)
```

Авторизация пользователей

Описание:

Почти каждое современное приложение, особенно облачное, имеет авторизацию пользователей и настройку аккаунтов, разграничение прав доступа, чтобы каждый из пользователей мог видеть только определенную часть приложения.

Задача:

А) Написать приложение - хранилище пользовательских аккаунтов. Аккаунт состоит из следующих данных: имя, фамилия, возраст, пароль и логин. Приложение должно уметь создавать новые аккаунты и сохранять их в отдельный файл, читать информацию о конкретном пользователе, проверяя авторизационные данные: логин и пароль. Пароль хранить в открытом виде нельзя. Дополнительно нужно решить проблему подбора ключей перебором (brute force).

Б) Расширить приложение таким образом, чтобы пользователи могли отправлять друг другу сообщения (подобие электронной почты). Сообщения должны храниться на диске также в виде файлов. Приложение должно позволять отправить сообщение другому пользователю на указанный логин, показывать список прочитанных и непрочитанных сообщений, а также показывать текст самого сообщения по выбору пользователя.

В) Усовершенствовать приложение таким образом, чтобы приложение-мессенджер работало в режиме реального времени и в случае поступления нового сообщения, выводило соответствующее уведомление. Если запустить 2 приложения одновременно, чтобы можно было вести через них диалог как между двумя разными пользователями.

Пример использования:

Выведет `Пользователь pavel создан`
./accounts.bin create-user —login=superstar —password=pwd —name=pavel —lastname=noname —age=20

Выведет `Пользователь maks создан`
./accounts.bin create-user —login=moonlight —password=strong —name=maks —lastname=surname —age=22

Выведет `name=pavel, lastname=noname, age=20`
./accounts.bin read-user —login=superstar —password=pwd

Выведет `Сообщение пользователю moonlight отправлено`
./accounts.bin send-message —from=superstar —password=pwd —destination=moonlight —message="Ну, здаствуйте!"

Выведет `Входящих сообщений: 0`
./accounts.bin read-mailbox —login=superstar —password=pwd

Выведет `Входящих сообщений: 1`

Выведет `id=1, from=superstar”`

./accounts.bin read-mailbox —login=superstar —password=pwd

Выведет `Ну, здаствуйте!`

./accounts.bin read-mail —login=superstar —password=pwd —message-id=1

Выведет `Входящих сообщений: 0`

./accounts.bin read-mailbox —login=superstar —password=pwd

Система контоля версий

Описание:

Системы контроля версий помогают нескольким разработчиками вести одновременную разработку одного приложения удобно синхронизируя их и сохраняя историю изменений.

Задача:

А) Элементарная система контроля версий наподобие GIT. Приложение должно зарегистрировать файл в реестре и следить за его изменениями. Каждое изменение должно хранить информацию о том, с какой и по какую строки нужно сделать замену текста. Выводить список изменений, их название и дату создания.

Б) Расширить приложение и добавить возможность создавать изменения от имени пользователя. Если 2 пользователя сделали изменения в файле в одном и том же месте, нужно вывести ошибку о конфликте изменений. Добавить возможность отката изменений данных в файле.

Пример использования:

```
# Выведет `Файл test.txt добавлен в реестр`
./vcs.bin add test.txt —user=pavel

# Выведет `Файл test.txt добавлен в реестр`
./vcs.bin add test2.txt —user=maks
```

```
# Выведет `Новых изменений нету`
./vcs.bin status

# Выведет `Коммитов еще нету`
./vcs.bin log

# Изменим наш файл, добавим текст
echo $'simple\nmultiline\ntext' > test.txt

# Выведет `Файл test.txt был изменен`
./vcs.bin status

# Выведет `Изменения добавлены`
./vcs.bin commit —message=‘Initial commit’ —user=superstar

# Выведет `id=1 user=superstar message=‘Initial commit’`
./vcs.bin log

# Изменим наш файл
echo $'complex\nmultiline\ntext' > test.txt

# Выведет `Изменения добавлены`
./vcs.bin commit —message=‘Changed first line’ —user=moonlight

# Выведет `id=2 user=moonlight message=‘Changed first line’`
# Выведет `id=1 user=superstar message=‘Initial commit’`
./vcs.bin log
```

Выведет `complex\nmultiline\ntext`

cat test.txt

Выведет `Изменения в файле возвращены до состояния 1`

./vcs.bin revert —commit=2

Выведет `simple\nmultiline\ntext`

cat test.txt

Выведет `id=1 user=superstar message='Initial commit'`

./vcs.bin log

Глава 3. UNIX, LINUX, POSIX

На текущий момент многие компании работают со свободными операционными системами, большинство из которых UNIX-совместимые или UNIX-подобные. К числу таких ОС относятся Linux, FreeBSD, MacOS, Solaris и много других. Эти ОС имеют частичную или полную реализацию POSIX стандарта, который описывает различные возможности самой ОС. Имея единственные интерфейсы для создания ПО, разработчики имеют возможность писать переносимые приложения, которые могут работать приблизительно одинаково на разных ОС без значительных изменений в коде, либо без изменений вообще.

Такие условия представляют огромный интерес для сообщества разработчиков по всему миру. Открытость множества этих ОС также играет важную роль, так как комьюнити может само развивать их, вносить изменения и устранять уязвимости, совершенствовать и ускорять код.

TCP/IP мессенджер

Описание:

Наверное уже невозможно представить современный мир без приложений передачи сообщений. В предыдущей главе мы уже написали небольшой мессенджер, который работает локально и хранит всю информацию в файлах. Сейчас же нужно модифицировать приложение по обмену сообщению таким образом, чтобы оно теперь работало через сеть с использованием TCP протокола.

Задача:

Написать серверное приложение, которое будет принимать запросы на порту 9090 и приложение-клиент, которое будет подключатся к серверу и отправлять сообщение другому пользователю.

Пример использования:

```
# Выведет `Server started at 127.0.0.1:9090`
./server.bin start —host=127.0.0.1 —port=9090

# Выведет `Connected to server successfully as superstar`
./client.bin —host=127.0.0.1 —port=9090 —login=superstar —password=pwd

# Выведет `Connected to server successfully as superstar`
./client.bin —host=127.0.0.1 —port=9090 —login=superstar —password=pwd
[superstar]>
```

```
# Дальше нужно открыть еще один терминал и запустить еще один
# клиент
./client.bin —host=127.0.0.1 —port=9090 —login=moonlight —
password=strong
[moonlight]>

# Дальше в открытых терминалах можно отправлять друг другу
сообщения

[moonlight]>Hello
[superstar]>World
```

LZ77 архиватор текста

Описание:

С сжатием файлов наверное встречались все: это и создание различных архивов, и потоковое видео в стриминговых сервисах, jpeg/png и тому подобные картинки также сжаты. Всё, что мы просматриваем в интернете, часто сжато больше одного раза разными методами для того, чтобы сократить обьем передаваемого трафика. Крупные компании в датацентрах архивируют исторические данные, чтобы тратить на хранение меньше денег. Мы же попробуем написать элементарный архиватор текста.

Задача:

Нужно загрузить большой текстовый файл, например 10МБ. С помощью простого алгоритма сжатия LZ77 упаковать файл и посмотреть как сильно изменился его обьем. Дальше нужно усовершенствовать алгоритм и на выбор пользователя использовать упаковку в нескольких процессах или потоках. Результаты работы приложения сравнить и проанализировать.

Пример использования:

```
# Сжатие в одном потоке, в одном процессе
./compressor.bin —file=test.txt —out=test.compressed —method=standart
```

```
# Сжатие в нескольких потоках, в одном процессе
./compressor.bin —file=test.txt —out=test.compressed —
method=multithreaded

# Сжатие в одном потоке, в нескольких процессах
./compressor.bin —file=test.txt —out=test.compressed —
method=multiprocess
```

Менеджер пользователей в UNIX системах

Описание:

Изолирование пользователей - важная функциональность ОС, так как она делает возможной параллельную работу нескольких пользователей. Конечно, изоляция должна быть как по ОЗУ, так и по процессорному времени, места на диске, работке с переферийными устройствами. Большая часть этого функционала предусмотрена стандартом POSIX и уже реализована в ядре самой ОС, мы воспользуемся этим и сделаем утилиту для удобного управления пользователями в ОС.

Задача:

Приложение должно добавлять нового пользователя в операционную систему и устанавливать группу posix_test_user. Вместе с этим создавать домашний каталог, а также файлы в нем .bashrc, .profile. Дополнительно создавать каталог Secure и хранить в нем документ с персональной информацией о пользователе: логин, номер телефона, электронная почта. Метод шифрования файла - шифр Плейфера. Информацию с файла можно прочитать, если запросить ее и ввести верный ключ. Также пользователей можно удалять и чистить все файлы и папки за ним.

Пример использования:

```
# Выведет `Пользователь superstar успешно создан`
./manager.bin add-user —login=superstar —password=pwd —phone=8822777
—email=superstar@example.com

# Выведет `login=superstar phone=8822777 email=superstar@example.com`
./manager.bin read-user —login=superstar —password=pwd

# Выведет `Пользователь superstar успешно удален`
./manager.bin remove-user —login=superstar —password=pwd
```

Список рекомендуемой литературы

1. Структуры данных и алгоритмы. Альфред Ах
2. Язык программирования Си. Брайан Керниган и Деннис Ритчи
3. Объектно-ориентированное мышление. Вайсфельд Мэтт
4. Язык программирования C++. Базовый курс. Жозе Лажуа и Стенли Б. Липпман
5. Linux. Системное программирование. Роберт Лав