

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Московский Авиационный Институт
(Национальный исследовательский университет)

Институт №8
«Компьютерные науки и прикладная математика»
Кафедра 806
«Вычислительная математика и программирование»

Курсовой проект по дисциплине «Фундаментальные алгоритмы»
Тема: «Разработка алгоритмов системы хранения и управления данными
на основе динамических структур данных»

Студент: Бабенко Александр Юрьевич

Группа: М8О-211Б-21

Преподаватель: Ирбитский И.С.

Оценка:

Дата:

Москва, 2023

Введение

Задание курсовой работы:

На языке программирования C++ (стандарт C++14 и выше) реализуйте приложение, позволяющее выполнять операции над коллекциями данных заданных типов (типы обслуживаемых объектов данных определяются вариантом) и контекстами их хранения (коллекциями данных). Коллекция данных описывается набором строковых параметров (набор параметров однозначно идентифицирует коллекцию данных):

- название пула схем данных, хранящего схемы данных;
- название схемы данных, хранящей коллекции данных;
- название коллекции данных.

Коллекция данных представляет собой ассоциативный контейнер (конкретная реализация определяется вариантом), в котором каждый объект данных соответствует некоторому уникальному ключу. Для ассоциативного контейнера необходимо вынести интерфейсную часть (в виде абстрактного класса C++) и реализовать этот интерфейс. Взаимодействие с коллекцией объектов происходит посредством выполнения одной из операций над ней:

- добавление новой записи по ключу;
- чтение записи по её ключу;
- чтение набора записей с ключами из диапазона [*minbound... maxbound*];
- обновление данных для записи по ключу;
- удаление существующей записи по ключу.

Во время работы приложения возможно выполнение также следующих операций:

- добавление/удаление пулов данных;
- добавление/удаление схем данных для заданного пула данных;
- добавление/удаление коллекций данных для заданной схемы данных заданного пула данных.

Поток команд, выполняемых в рамках работы приложения, поступает из файла, путь к которому подаётся в качестве аргумента командной строки. Формат команд в файле определите самостоятельно.

Дополнительные задания, реализованные в курсовой работе:

- Реализуйте интерактивный диалог с пользователем. Пользователь при этом может вводить конкретные команды (формат ввода определите самостоятельно) и подавать на вход файлы с потоком команд;

- Реализуйте механизм, позволяющий выполнять эффективный поиск по различным отношениям порядка на пространстве данных (дублирование объектов данных при этом запрещается). Обеспечьте поиск при помощи указания ключа отношения порядка (в виде строки, подаваемой как параметр поиска);

- Обеспечьте хранение объектов строк, размещённых в объектах данных, на основе структурного паттерна проектирования “Приспособленец”. Дублирования объектов строк для разных объектов (независимо от контекста хранения) при этом запрещены. Доступ к строковому пулу обеспечьте на основе порождающего паттерна проектирования “Одиночка”;

- Реализуйте механизмы сохранения состояния системы хранения данных в файловую систему и восстановления состояния системы хранения данных из файловой системы;

- Реализуйте функционал приложения в виде сервера, запросы на который поступают из клиентских приложений. При этом взаимодействие клиентских приложений с серверным должно быть реализовано посредством средств межпроцессного взаимодействия (IPC). Используемые средства IPC и операционная система определяются вариантом;

- Реализуйте механизмы регистрации авторизации пользователя в системе (на клиентской стороне) и открытия пользовательской сессии (на серверной стороне) через пару значений <логин, пароль>. Пароль при этом должен храниться и передаваться в виде хеша (используйте хеш-функцию SHA-4). Логины пользователей уникальны в рамках системы, могут содержать только символы букв и цифр в количестве [5..15] (обеспечьте валидацию на стороне сервера). Пароль должен содержать не менее 8 символов (обеспечьте валидацию на стороне клиента). Формат хранения и передачи данных для авторизации пользователей определите самостоятельно;

- На основе передаваемого в клиентские запросы токена аутентификации реализуйте различные роли, разграничивающие доступ к выполнению операций в рамках системы:

- администратор - имеет возможности создания новых пользователей; управления (выдача, блокировка) ролями других пользователей (кроме администраторов); управления доступом к схемам данных и доступом к операциям на уровне схем данных для заданных коллекций данных (режимы “только для чтения” и “чтение и модификация”);

- редактор - имеет возможности управления пулами (добавление/удаление), схемами (добавление/удаление) данных, коллекциями данных (добавление/удаление) в соответствии с предоставленными правами доступа;

- пользователь - имеет возможности взаимодействия с коллекциями данных в соответствии с предоставленными правами доступа.

Вариант курсовой работы 29:

Тип данных:

Данные о прохождении пайплайна (id сборки, версия сборки, информация о коммите с которого выполняется сборка (хеш коммита, логин разработчика,

электронная почта разработчика), ссылка на сценарий сборки (путь к файлу), название сборки, информация об ошибках этапа build, информация об ошибках этапа статического анализа кода, информация об ошибках этапа прогона автотестов, ссылка на артефакты сборки (путь к директории)).

Контейнер: красно-чёрное дерево

IPC: Unix message queues

Описание реализованного приложения

В ходе курсовой работы было реализовано приложение, позволяющее создавать, добавлять, хранить, искать и удалять записи данных о прохождении пайплайна разработчиком. Приложение работает согласно архитектуре «клиент-сервер». Со стороны клиента предусмотрена возможность регистрации новых пользователей в системе с разными правами доступа. По умолчанию на сервере существует пользователь *admin* со своим паролем. После запуска приложения он должен авторизоваться в системе. Если этого не сделать, будет выведено сообщение об ошибке. Потом можно будет вводить команды, используемые в интерактивном диалоге. Хранение записей о сборках происходит в структуре данных на основе красно-чёрного дерева на стороне сервера.

Красно-чёрным называется бинарное поисковое дерево, у которого каждому узлу сопоставлен дополнительный атрибут — цвет и для которого выполняются следующие свойства:

- Каждый узел промаркирован красным или чёрным цветом;
- Корень и конечные узлы(листья) дерева — чёрные;
- У красного узла родительский узел — чёрный;
- Все простые пути из любого узла x до листьев содержат одинаковое количество чёрных узлов;
- Чёрный узел может иметь чёрного родителя.

Данное дерево относится к сбалансированным деревьям поиска и очень эффективно в использовании. Операции вставки, удаления и поиска работают в среднем за $O(\log N)$.

Спецификация программы

Ниже представлены исходные файлы приложения и их краткое описание

Таблица 1. Исходные файлы приложения

rbt.h	Описывает реализацию красно-чёрного дерева для хранения пар ключ-значение
stringpool.h	Содержит класс <i>StringPool</i> , который кеширует текстовые строки через строковый пул
pipeline.h	Содержит структуру типа данных согласно варианту
mq.h	Содержит класс <i>MQ</i> , который реализует взаимодействие клиента с сервером через очереди сообщений
sha.hpp	Содержит класс <i>SHA3</i> , который реализует алгоритм хеширования для хранения паролей пользователей
server.cpp	Приложение сервера
client.cpp	Приложение клиента

Описание классов и их методов

Приложение включает в себя несколько классов, которые, взаимодействуя между собой, обеспечивают полноценную работу программы, хранение данных и обработку запросов. Разберём их более подробно.

Класс *StringPool* представляет собой пул строк. Он содержит два приватных поля: `_set`, который является набором строк без повторов и `_len`, который является хеш-картой, отображающей указатель на строку на её длину.

В классе определены приватный конструктор и деструктор, что означает, что объекты этого класса могут быть созданы и уничтожены только внутри самого класса.

Класс также имеет публичный статический метод `instance`, который возвращает ссылку на единственный экземпляр *StringPool*.

Метод `find` принимает строку в качестве аргумента и возвращает указатель на строку в пуле. Если строка уже существует в пуле, метод возвращает указатель на неё. Если строка отсутствует в пуле, она добавляется в `_set`, и метод возвращает указатель на неё.

Метод `length` принимает указатель на строку в качестве аргумента и возвращает её длину. Он ищет указатель в хеш-карте `_len` и, если находит, возвращает соответствующую длину. Если указатель не найден, метод возвращает 0.

Класс *AbstractTree* является абстрактным классом, предназначенным для представления структур данных дерева. Он определяет интерфейс для реализации красно-чёрного дерева.

Класс содержит несколько чисто виртуальных методов:

- `Size`: возвращает размер дерева, то есть количество элементов в нём;
- `Insert`: вставляет элемент с ключом `k` и значением `v` в дерево;
- `Update`: обновляет значение элемента с ключом `k` на значение `v`;
- `Get`: возвращает константную ссылку на значение элемента с ключом `k`;
- `Get`: возвращает ссылку на значение элемента с ключом `k`;
- `ValuesFromRange`: возвращает вектор значений элементов, ключи которых находятся в заданном диапазоне от `minbound` до `maxbound`;
- `Remove`: удаляет элемент с ключом `k` из дерева;
- `Clear`: удаляет все элементы из дерева;
- `Contains`: проверяет, содержит ли дерево элемент с ключом `k`;
- `IsEmpty`: проверяет, пустое ли дерево.

Класс *RBTREE* является наследником класса *AbstractTree*. Он реализует методы, описанные в предыдущем классе. Также в нём реализованы вспомогательные методы для работы с красно-чёрным деревом.

Класс *MQ* представляет собой реализацию очереди сообщений (*Message Queue*). Он содержит три приватных поля: `m_isOK`, которое указывает, успешно ли

была создана очередь сообщений, `key`, которое хранит ключ, используемый для создания очереди сообщений и `msgid`, которое хранит идентификатор очереди сообщений.

Метод `isOk` возвращает значение типа `bool`, указывающее, была ли успешно создана очередь сообщений. Если очередь сообщений была успешно создана, метод возвращает `true`, в противном случае - `false`.

Метод `Create` создаёт очередь сообщений с заданным именем. Он использует функцию `flock` для генерации ключа на основе имени, а затем вызывает функцию `msgget` для создания очереди сообщений с полученным ключом. Если создание очереди прошло успешно, метод возвращает `true`, в противном случае - `false`.

Метод `Unlink` удаляет очередь сообщений, если она была создана. Он использует функцию `msgctl` с флагом `IPC_RMID` для удаления очереди.

Метод `Read` считывает сообщение из очереди сообщений и сохраняет его в буфер. Он использует функцию `msgrcv` для чтения сообщения. Параметр `count` указывает максимальный размер буфера, и после выполнения метода содержит фактическое количество байт, прочитанных из очереди.

Метод `Write` записывает сообщение из буфера в очередь сообщений. Он использует функцию `msgsnd` для отправки сообщения. Параметр `count` указывает размер сообщения.

Статический метод `Sleep` выполняет задержку в выполнении программы. Он использует функцию `nanosleep` для приостановки выполнения на заданное количество времени (в данном случае, задержка определена константой `QUEUE_POLL_CONSUMER`).

Класс `SHA3` предоставляет интерфейс для вычисления хеш-значения с использованием алгоритма SHA-3. Разберём его публичные поля.

Перечисление `Bits` определяет различные варианты битовых длин (224 бита, 256 битов, 384 бита и 512 битов) для вычисления хеш-значения.

Конструктор создает объект класса `SHA3` с указанным количеством битов по умолчанию равным 256.

Операторы `operator()` и метод `add`: эти функции предназначены для вычисления хеш-значения на основе переданных данных.

Метод `getHash` возвращает текущее хеш-значение в виде строки.

Метод `reset` сбрасывает состояние объекта `SHA3`, позволяя повторно использовать его для вычисления хеш-значений.

Также класс имеет несколько приватных полей.

`m_hash` - массив беззнаковых 64-битных целых чисел, используемых для хранения текущего состояния хеш-значения.

`m_numBytes` - используется для отслеживания общего количества обработанных байтов данных. В процессе вычисления хеш-значения алгоритм

обрабатывает данные блоками, и `m_numBytes` помогает отслеживать количество обработанных данных.

`m_blockSize` и `m_bufferSize` - используются для отслеживания размера текущего блока данных и размера буфера, в котором временно хранятся данные перед их обработкой. `m_blockSize` и `m_bufferSize` используются для управления процессом обработки данных и обновления состояния алгоритма.

`m_buffer` - массив байтов, который служит буфером для временного хранения данных, пока не наберётся достаточное количество данных для обработки.

`m_bits` - переменная перечисления `Bits`, которая хранит текущее количество битов, используемых для вычисления хеш-значения. Она определена внутри класса и используется внутри методов класса для настройки алгоритма SHA-3 на вычисление хеша с заданным количеством битов.

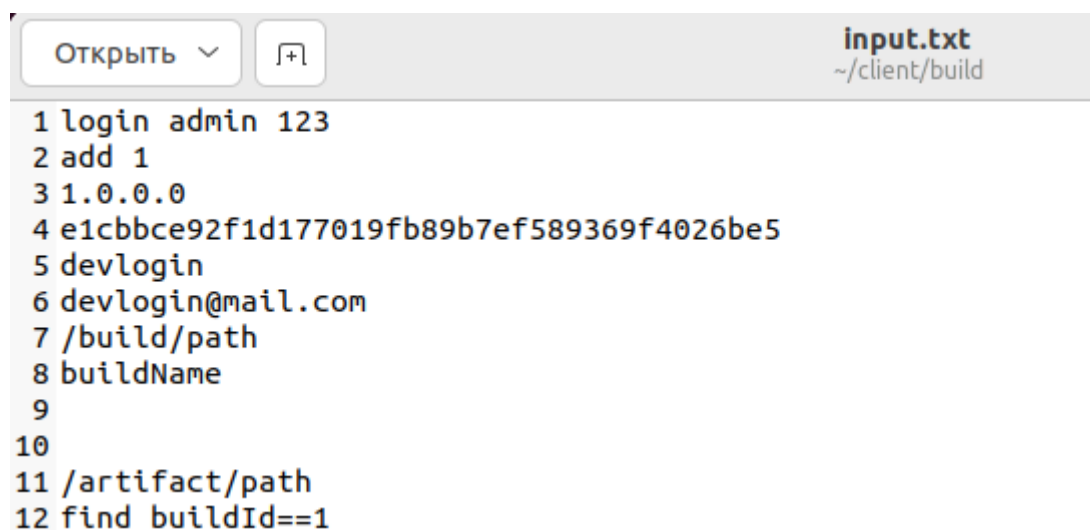
Класс *SHA3* реализует алгоритм SHA-3, который является одним из стандартных алгоритмов хеширования, используемых для создания контрольных сумм данных. Все методы и члены класса предназначены для обеспечения возможности вычисления хеш-значения для данных, переданных в виде байтов или строк.

Руководство пользователя

Перед началом работы приложения необходимо через аргументы командной строки подать на вход файл, в котором будет находиться поток команд, выполняемых в рамках работы программы. После выполнения команд из файла пользователю будет доступен интерактивный режим, в котором он может сам выбирать команды для управления коллекцией данных. Были реализованы следующие команды:

- **add** – добавление записей о прохождении пайплайна;
- **find** – поиск по параметру. В случае успешного нахождения на экран выводятся данные о записи. В противном случае выведется сообщение о том, что запись не найдена;
- **del** – удаление одной или нескольких записей;
- **login** – авторизация на сервере по переданному логину и паролю. Если в логине будут символы, отличные от цифр и букв, а пароль будет меньше 8 символов, то будет выведено сообщение об ошибке;
- **register** – создание пользователя с разными правами доступа. Например, чтобы создать пользователя с правами администратора нужно написать так: **register USERNAME password admin**. Если мы хотим, чтобы пользователь не имел права регистрировать новых пользователей, то вместо последнего параметра указываем **editor** или **user**. Модифицирующие запросы, такие как **add** и **del**, могут выполнять пользователи групп **admin** и **editor**.
- **quit** – закрыть клиент.

Рассмотрим примеры входных файлов с потоком команд и результатом их обработки.

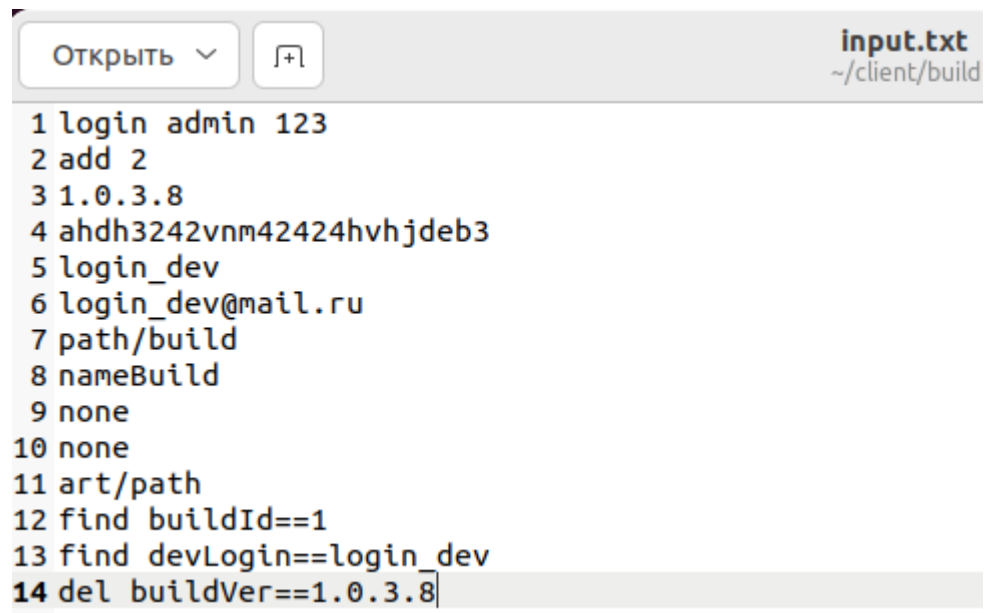


```
1 login admin 123
2 add 1
3 1.0.0.0
4 e1cbbce92f1d177019fb89b7ef589369f4026be5
5 devlogin
6 devlogin@mail.com
7 /build/path
8 buildName
9
10
11 /artifact/path
12 find buildId==1
```

Рисунок1. Входной файл с потоком команд

```
Hello! Please login and then select one of the data management commands:
1.add
2.find
3.del
4.register
OK: login successful.
buildId: "1"
buildVer: "1.0.0.0"
commitHash: "e1cbbce92f1d177019fb89b7ef589369f4026be5"
devLogin: "devlogin"
devEmail: "devlogin@mail.com"
buildPath: "/build/path"
buildName: "buildName"
errorsStaticAnalyze: ""
errorsAutoTests: ""
pathArtifact: "/artifact/path"
```

Рисунок 2. Полученный вывод



```
1 login admin 123
2 add 2
3 1.0.3.8
4 ahdh3242vnm42424hvhjdeb3
5 login_dev
6 login_dev@mail.ru
7 path/build
8 nameBuild
9 none
10 none
11 art/path
12 find buildId==1
13 find devLogin==login_dev
14 del buildVer==1.0.3.8
```

Рисунок 3. Входной файл с потоком команд

```
Hello! Please login and then select one of the data management commands:
1.add
2.find
3.del
4.register
OK: login successful.
ERR: not found.
buildId: "2"
buildVer: "1.0.3.8"
commitHash: "ahdh3242vnm42424hvhjdeb3"
devLogin: "login_dev"
devEmail: "login_dev@mail.ru"
buildPath: "path/build"
buildName: "nameBuild"
errorsStaticAnalyze: "none"
errorsAutoTests: "none"
pathArtifact: "art/path"

OK: 1 rows affected.
```

Рисунок 4. Полученный вывод

Рассмотрим работу интерактивного диалога, в котором пользователь сам отправляет запросы к данным.

Листинг 1. Работа интерактивного режима

```
Hello! Please login and then select one of the data management commands:
1.add
2.find
3.del
4.register
> login admin 123
OK: login successful.
> add 3
2.0.4
e13hvavfja324gfgfhj3g2cg232e
alex
alex@gmail.com
path/build
tasks
not
not
artifact/path
> find buildId==3
buildId: "3"
buildVer: "2.0.4"
commitHash: "e13hvavfja324gfgfhj3g2cg232e"
devLogin: "alex"
devEmail: "alex@gmail.com"
buildPath: "path/build"
buildName: "tasks"
errorsStaticAnalyze: "not"
errorsAutoTests: "not"
pathArtifact: "artifact/path"
> register user2 qwerty-0 user
OK: register successful.
> login user2 qwerty-0
```

```

OK: login successful.
> del buildId==3
ERR: access denied.
> find buildId==2
ERR: not found.
> login admin 123
OK: login successful.
> add 5
1.6
ehgh2ghgfhg3324bcgh2v4
ivan
ivan@mail.ru
build/path
fundament
none
none
path/art
> find buildId!=2
buildId: "3"
buildVer: "2.0.4"
commitHash: "e13hvavfja324gfgfhj3g2cg232e"
devLogin: "alex"
devEmail: "alex@gmail.com"
buildPath: "path/build"
buildName: "tasks"
errorsStaticAnalyze: "not"
errorsAutoTests: "not"
pathArtifact: "artifact/path"

buildId: "5"
buildVer: "1.6"
commitHash: "ehgh2ghgfhg3324bcgh2v4"
devLogin: "ivan"
devEmail: "ivan@mail.ru"
buildPath: "build/path"
buildName: "fundament"
errorsStaticAnalyze: "none"
errorsAutoTests: "none"
pathArtifact: "path/art"
> del buildVer==2.0.4
OK: 1 rows affected.
> find buildId==3
ERR: not found.
> register user2 qwerty-0 user
ERR: login is taken.
> register user3 12345678 editor
OK: register successful.
> login user3 12345678
OK: login successful.
> del buildId==5
OK: 1 rows affected.
> find buildId==5
ERR: not found.
> quit

```

Здесь мы видим использование всех основных команд для взаимодействия с коллекцией данных.

Разберём связь между клиентом и сервером. В начале своей работы сервер создаёт очередь сообщений. Это выполняется с использованием объекта MQ. Эта очередь служит механизмом связи между клиентами и сервером. Каждый клиент будет писать запросы в эту очередь, и сервер будет читать их оттуда.

Клиенты создают сообщения типа PLR_MESSAGE и отправляют их в очередь сервера. Каждое сообщение PLR_MESSAGE содержит информацию о типе запроса, параметрах запроса и, возможно, токене аутентификации пользователя (если требуется).

Сервер ожидает запросы, читая их из очереди сообщений. После получения запроса сервер вызывает соответствующую функцию для обработки запроса (processMessage), и в этой функции выполняются соответствующие операции с базой данных или другие действия.

После обработки запроса сервер создаёт одно или несколько сообщений PLR_MESSAGE, которые содержат ответы на запросы или сообщения об ошибках. Эти сообщения отправляются обратно клиентам через очередь сообщений.

При закрытии сервера те записи, что были добавлены, сохраняются в файлы base.dat на стороне сервера.

Таким образом, клиент и сервер взаимодействуют обменом сообщениями через очередь сообщений. Клиенты отправляют запросы на сервер, сервер обрабатывает их и отправляет ответы обратно клиентам. Этот механизм обеспечивает асинхронное и безопасное взаимодействие между клиентами и сервером, а также позволяет серверу обрабатывать несколько запросов параллельно.

Вывод

В ходе выполнения курсовой работы было реализовано приложение, позволяющее выполнять операции над коллекциями данных заданных типов и контекстами их хранения.

Программа обеспечивает взаимодействие клиента с сервером через очереди сообщений. Со стороны клиента предусмотрена регистрация новых пользователей с разными правами доступа, а со стороны сервера – открытие пользовательской сессии и сохранение записей с данными в серверные файлы.

Программа использует эффективные механизмы работы с памятью, включая пул строк для оптимизации использования ресурсов.

Эта программа является основным результатом курсового проектирования.

Приложение

<https://github.com/Kladmen34/FundAlg/tree/main/CourseWork>