**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Специальность

1-40 04 01 «Информатика и технологии программирования»

Кафедра «Информатика»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

По дисциплине «Конструирование программ»

На тему: «**УЧЕТ ЗАКАЗОВ СЕРВИСНОГО ЦЕНТРА»**

Исполнитель: студент гр. ИП-21

В. О. Фещенко

Руководитель: преподаватель

Н. В. Самовендюк

Дата проверки:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите:­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии  
по защите курсового проекта:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2023

**CОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc154406081)

[ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РЕШЕНИЯ 3](#_Toc154406082)

[РАЗРАБОТКА ИНФОЛОГИЧЕСКОЙ МОДЕЛИ 3](#_Toc154406083)

[ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 4](#_Toc154406084)

[ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ 4](#_Toc154406085)

[ЗАКЛЮЧЕНИЕ 5](#_Toc154406086)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 5](#_Toc154406087)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА 6](#_Toc154406088)

**ВВЕДЕНИЕ**

Цель работы: Разработка программы "Учет заказов сервисного центра" с функционалом администратора и пользователя для управления информацией о заказах и обеспечения легкого доступа к необходимым данным.

Данное курсовое задание предполагает разработку программного решения для сервисного центра, которое позволит учет заказов и обеспечит удобный доступ к информации о них. Система должна иметь функционал как для администратора, так и для обычных пользователей.

# Функционал администратора включает в себя:

# Управление пользователями, включая возможность добавления и удаления пользователей.

# Редактирование данных, включая добавление новых записей, удаление заданной записи и редактирование заданной записи.

# Сохранение изменений в файл и возможность отмены изменений.

# Функционал пользователя включает:

# Вывод общего списка заказов, отсортированного по дате приемки с помощью быстрой сортировки.

# Вывод списка заказов, находящихся в процессе ремонта.

# Поиск информации о заказах по заданному наименованию изделия с возможностью поиска по вхождению строки.

# Результатом работы должно быть готовое программное решение, которое позволяет авторизованным пользователям выполнять необходимые операции с данными заказов, администрировать систему и обеспечивать удобный доступ к информации о заказах.

# Для достижения этой цели потребуется изучение основ программирования, разработка архитектуры программы, реализация функционала с учетом требований задания, а также тестирование и отладка программы. В конечном итоге, выполнение данной курсовой работы позволит разработать полезное программное решение, которое может быть использовано в реальной среде сервисных центров для учета заказов и облегчения рабочего процесса.

# **1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РЕШЕНИЯ**

**1.1 Связный список**

Связный список представляет собой более гибкую систему для хранения данных, в которой вовсе не используются массивы. Вместо них мы получаем для каждого элемента память, используя операцию new, а затем соединяем (или связываем) элементы данных между собой, используя указатели. Элементы списка не обязаны храниться в смежных участках памяти, как это происходит с массивами; они могут быть разбросаны по всему пространству памяти.

Возьмем для примера класс *linlklist*. Элементы списка представлены структурой *link*. Каждый элемент содержит целое число, представляющее собой данные, и указатель на следующий элемент списка. Указатель на сам список хранится в начале списка. Структура списка показана на рисунке 1

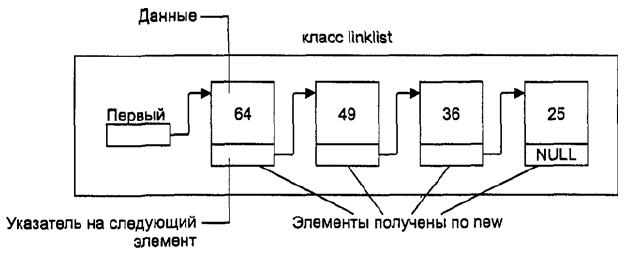


Рисунок 1 — Связный список

Структура link имеет следующее строение:

struct link // один элемент списка

{

int data; // некоторые данные

link\* next; // указатель на следующую структуру

};

Строение класса *linklist* представлено на рисунке 2

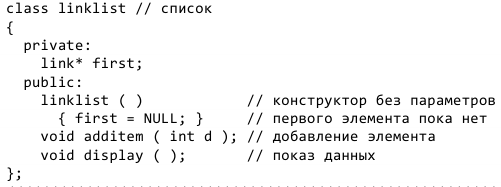


Рисунок 2 — Класс *linklist*

Реализация добавления нового элемента в связный список основывается на замене адреса, содержащегося в указателе *first*, на адрес нового элемента, при этом старый адрес указателя *first* превратится в адрес второго элемента списка. Этот процесс иллюстрирует рисунок 3

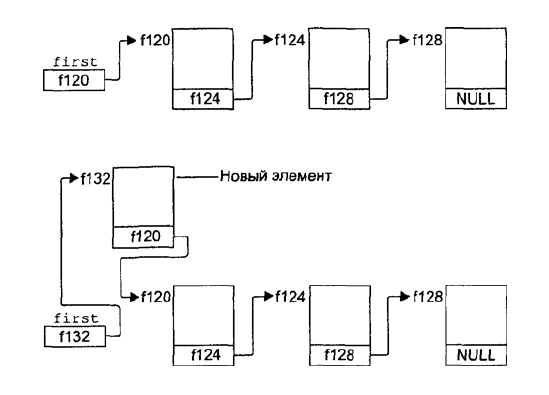


Рисунок 3 — Вставка нового элемента в связный список

Когда список уже создан, легко получить хранящиеся в нем значения (или выполнить операции с ними). Все, что нам нужно, это следовать от одного указателя *next* к другому до тех пор, пока его значение не станет равным NULL, означающим конец списка. В строке функции *display()*

cout << endl << current=data;

выводится значение, содержащееся в переменной *data*, и

current = current->next;

перемещает нас к следующему элементу списка до тех пор, пока

current != NULL;

Связные списки — это наиболее часто встречающаяся после массивов структура для хранения данных. В них, в отличие от массивов,

устранена возможная потеря памяти. Недостаток списков в том, что для поиска

определенного элемента списка необходимо пройти сквозь весь список от его

начала до нужного элемента. И это может занять много времени. А к элементам массива мы можем получить мгновенный доступ, используя индекс элемента, известный заранее.

**1.2 Бинарное дерево**

Бинарное дерево является структурой данных, состоящей из узлов, в которых каждый узел может иметь не более двух дочерних узлов: левый и правый. Каждый узел содержит значение (ключ) и ссылки на его дочерние узлы. Бинарное дерево обладает следующими основными свойствами:

* Свойство упорядоченности: Значение в левом поддереве каждого узла меньше значения самого узла, а значение в правом поддереве больше. Это свойство позволяет эффективно выполнять операции поиска, вставки и удаления элементов.
* Свойство рекурсивной структуры: Каждое поддерево является бинарным деревом, то есть оно также состоит из узлов, каждый из которых может иметь левое и правое поддерево.

Бинарные деревья широко используются для решения различных задач, включая поиск, сортировку, построение алгоритмов компрессии данных и многое другое.

Операции над бинарным деревом:

* Поиск: Начиная с корневого узла, сравниваем искомое значение с текущим узлом. Если значение совпадает, поиск успешен. Если значение меньше текущего узла, переходим к левому поддереву; если значение больше, переходим к правому поддереву. Процесс повторяется до нахождения искомого значения или пустого поддерева.
* Вставка: Новый узел вставляется в бинарное дерево в соответствии с его значением. Сравнивая значение нового узла с текущим узлом, определяем, в какое поддерево нужно перейти. Если поддерево пустое, новый узел становится его корневым узлом. Если поддерево не пустое, повторяем процесс вставки для этого поддерева.
* Удаление: При удалении узла нужно учитывать несколько случаев:

1. Если узел является листом (не имеет дочерних узлов), он просто удаляется.
2. Если у узла есть только один дочерний узел, то данный узел заменяется этим дочерним узлом.
3. Если у узла есть два дочерних узла, то можно заменить его на наибольший элемент из левого поддерева или наименьший элемент из правого поддерева.

В нашем случае ключом дерева является дата приемки заказа, что позволяет при построении дерева одновременно сортировать данные и выводить их на консоль в нужном нам порядке.

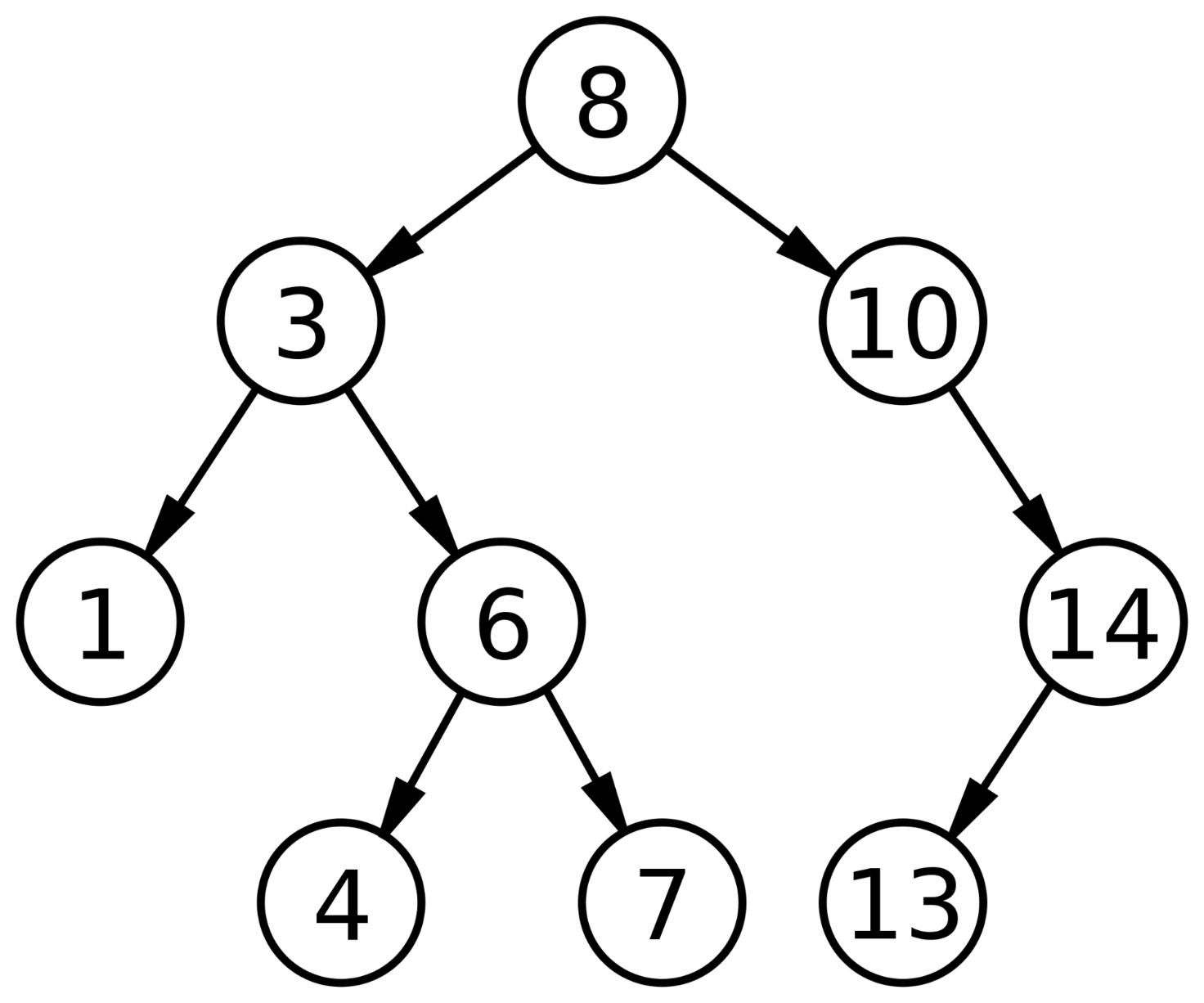


Рисунок 3 — модель бинарного дерева

**1.3 Быстрая сортировка**

"Быстрая сортировка", хоть и была разработана более 40 лет назад, является наиболее широко применяемым и одним их самых эффективных алгоритмов.

Метод основан на подходе "разделяй-и-властвуй". Общая схема такова:

1. из массива выбирается некоторый опорный элемент a[i],
2. запускается процедура разделения массива, которая перемещает все ключи, меньшие, либо равные a[i], влево от него, а все ключи, большие, либо равные a[i] - вправо,
3. теперь массив состоит из двух подмножеств, причем левое меньше, либо равно правого,
4. для обоих подмассивов: если в подмассиве более двух элементов, рекурсивно запускаем для него ту же процедуру.

В конце получится полностью отсортированная последовательность.

Иллюстрация данного алгоритма представлена на рисунках 4 и 5

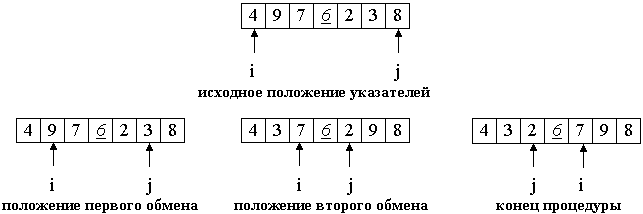


Рисунок 4 — быстрая сортировка

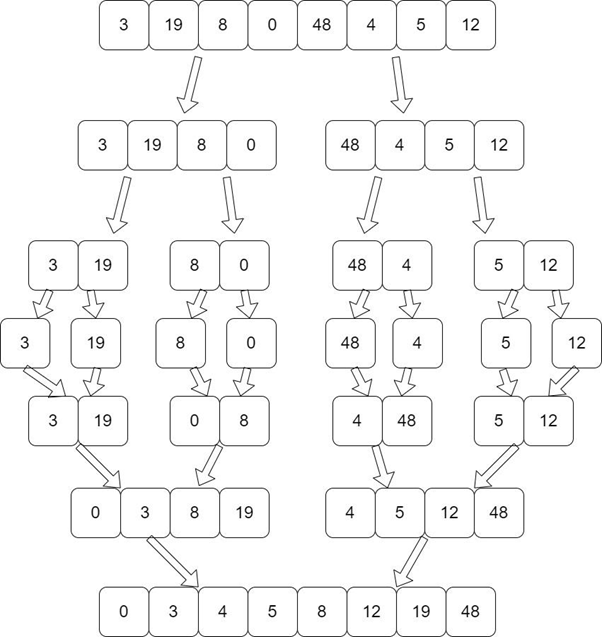


Рисунок 5 — быстрая сортировка

**1.4 Работа с файлами С++**

Большинству программ требуется сохранять данные на диске и считывать их. Работа с дисковыми файлами подразумевает наличие еще одного набора специальных классов: *ifstream* для ввода и *ofstream* для вывода. Класс *fstream* осуществляет и ввод, и вывод. Объекты этих классов могут быть ассоциированы с дисковыми файлами, а их методы — использоваться для обмена данными с ними.

Следующий пример на рисунке 6 демонстрирует запись символа, целого числа, числа типа

double и двух объектов типа string в дисковый файл. Вывод на экран не производится.

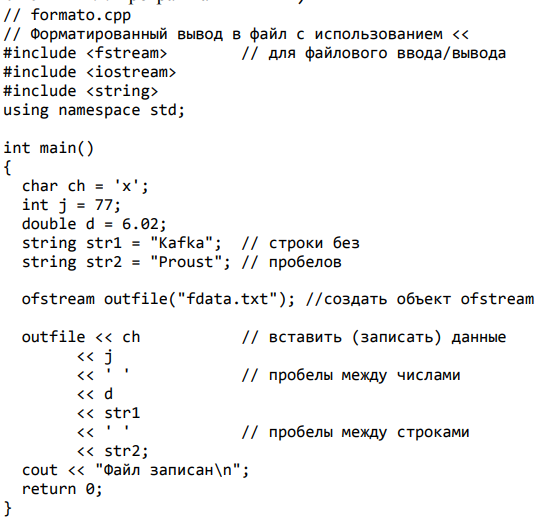


Рисунок 6 — пример записи данных в файл

Здесь мы определили объект *outfile* в качестве компонента класса *ofstream*.

В то же время мы инициализировали его файлом FDATA.TXT. Инициализация

резервирует для дискового файла с данным именем различные ресурсы и получает доступ (или открывает файл) к нему. Если файл не существует, он создается. Если файл уже существует, он переписывается — новые данные в нем

заменяют старые. Объект *outfile* ведет себя подобно *cout* из предыдущих программ, поэтому можно использовать операцию вставки (<<) для вывода переменных любого из стандартных типов в файл. Это так замечательно работает

потому, что оператор вставки перегружен в классе *ostream*, который является

родительским для *ofstream*.

Прочитать файл, можно с использованием объекта типа *ifstream*, инициализированного именем файла. Файл автоматически открывается при создании объекта. Затем данные из него можно считать с помощью

оператора извлечения (>>). Пример показан на рисунке 7

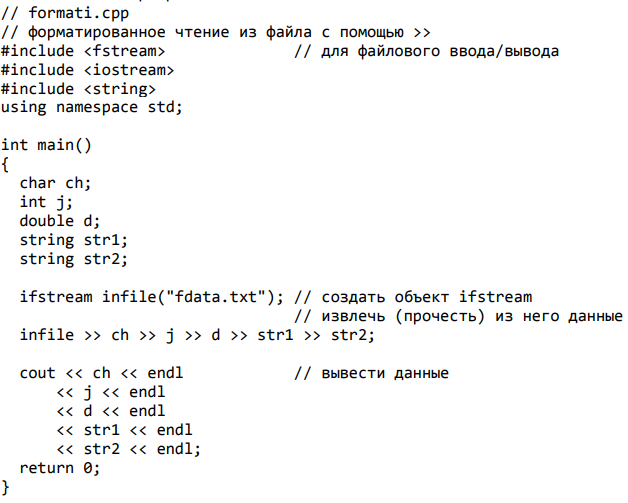


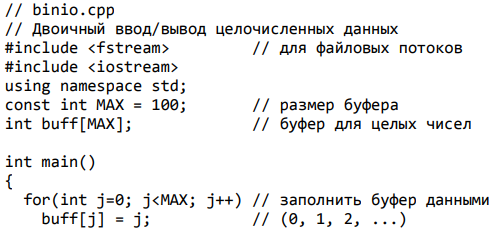
Рисунок 7 — пример чтения из файла

Объект типа *ifstream*, который мы назвали *infile*, действует примерно так же, как *cin* в предыдущих программах. Если мы корректно вставляли данные в файл, то

извлечь их не составит никаких проблем. Мы просто сохраняем их в соответствующих переменных и выводим на экран.

Форматированный файловый ввод/вывод чисел целесообразно использовать только при их небольшой величине и малом количестве. В противном случае, конечно, гораздо эффективнее использовать двоичный ввод/вывод, при котором числа хранятся таким же образом, как в ОП компьютера, а не в виде символьных строк. Целочисленные значения занимают 4 байта, тогда как текстовая версия числа, например «12345», занимает 5 байтов. Значения типа *float* также всегда занимают 4 байта. А форматированная версия «6,02314е13» занимает 10 байтов.

В следующем примере показано, как в бинарном виде массив целых чисел записывается в файл и читается из него. При этом используются две функции — *write()* (метод класса *ofstream*), а также *read()* (метод *ifstream*). Эти функции думают о данных в терминах байтов (тип *char*). Им все равно, как организованы данные, что онисобой представляют, — они просто переносят байты из буфера в файл и обратно.Параметрами этих функций являются адрес буфера и его длина. Адрес долженбыть вычислен с использованием *reinterpret\_cast* относительно типа *char\**. Вторымпараметром является длина в байтах (а не число элементов данных в буфере).



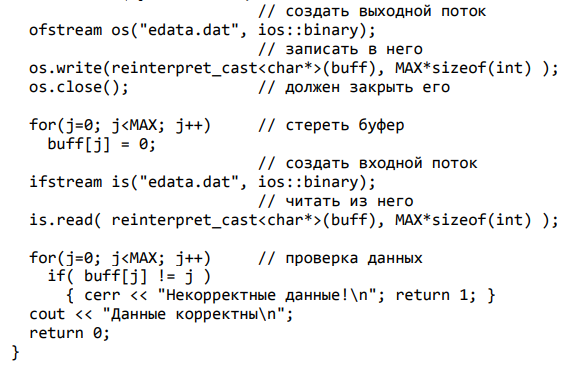


Рисунок 8 — пример двоичного ввода

При работе с бинарными данными в качестве второго параметра *write()* и *read()* следует использовать *ios::binary*. Это необходимо по той причине, что текстовый режим, используемый по умолчанию, допускает несколько вольное обращение с данными. Например, специальный символ *'\n'* занимает два байта (на самом деле это и есть два действия — перевод каретки и перевод строки). Это делает текст более удобным для чтения в *DOS* утилитами типа *TYPE*, но для бинарных данных такой подход не годится вовсе, так как любой байт, которому не повезло иметь *ASCII-код 10*, переводится двумя байтами. Аргумент *ios::binary* — типичный пример бита состояния. Мы еще будем говорить об этом при обсуждении функции *ореn()* немного позднее в этой главе.

Оператор *reinterpret\_cast* используется для того, чтобы буфер данных типа *int* выглядел для функций *read()* и *write()* как буфер типа *char*. *reinterpret\_cast* KaK бы говорит компилятору: «Я знаю, что тебе это не понравится,

и все-таки я это сделаю». Он изменяет тип данных в определенной области памяти, совершенно не задумываясь о том, имеет это смысл или нет. Поэтому вопрос целесообразности использования этого оператора остается целиком на совести программиста. Можно использовать *reinterpret\_cast* для превращения указателей в данные типа *int* и обратно. Это небезопасное занятие, но порой оно необходимо.

**1.5 Шифрование**

Шифрование — это процесс преобразования данных в такой формат, который не может быть прочитан или понятен без знания специального ключа или алгоритма расшифровки. Шифрование может использоваться для защиты конфиденциальности данных при их хранении или передаче. Существует множество алгоритмов шифрования, включая с открытым ключом и закрытым, а также симметричные алгоритмы (используют один и тот же ключ для шифрования и расшифровки) и асимметричные алгоритмы (используют разные ключи для шифрования и расшифровки):

* RSA (Riverst-Shamir-Adleman)
* AES (Advanced Encryption Standard)
* DES (Data Encryption Standard), а также Triple DES (3DE)
* ECC (Elliptic Curve Cryptography)
* DSA (Digital Signature Algorithm)
* Blowfish
* Twofish

Рассмотрим шифрование на примере алгоритма SHA-256. Попробуем зашифровать сообщение “hello world”.

**Шаг 1. Предварительная обработка**

1. Преобразуем «hello world» в двоичный вид:

01101000 01100101 01101100 01101100 01101111 00100000 01110111 01101111

01110010 01101100 01100100

1. Добавим одну единицу:

01101000 01100101 01101100 01101100 01101111 00100000 01110111 01101111

01110010 01101100 01100100 1

1. Заполняем нулями до тех пор, пока данные не станут кратны 512 без последних 64 бит (в нашем случае 448 бит):

01101000 01100101 01101100 01101100 01101111 00100000 01110111 01101111

01110010 01101100 01100100 10000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

1. Добавим 64 бита в конец, где 64 бита — целое число с порядком байтов big-endian, обозначающее длину входных данных в двоичном виде. В нашем случае 88, в двоичном виде — «1011000».

01101000 01100101 01101100 01101100 01101111 00100000 01110111 01101111

01110010 01101100 01100100 10000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 01011000

Теперь у нас есть ввод, который всегда будет без остатка делиться на 512.

**Шаг 2. Инициализация значений хеша (h)**

Создадим 8 значений хеша. Это константы, представляющие первые 32 бита дробных частей квадратных корней первых 8 простых чисел: 2, 3, 5, 7, 11, 13, 17, 19.

h0 := 0x6a09e667

h1 := 0xbb67ae85

h2 := 0x3c6ef372

h3 := 0xa54ff53a

h4 := 0x510e527f

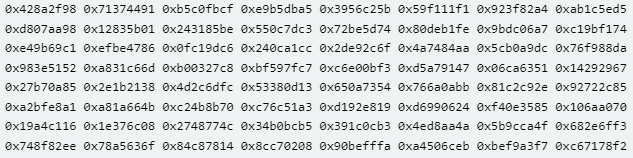
h5 := 0x9b05688c

h6 := 0x1f83d9ab

h7 := 0x5be0cd19

**Шаг 3. Инициализация округлённых констант (k)**

Создадим ещё немного констант, на этот раз их 64. Каждое значение — это первые 32 бита дробных частей кубических корней первых 64 простых чисел (2–311).

Рисунок 9 — инициализация округленных констант

**Шаг 4. Основной цикл**

Следующие шаги будут выполняться для каждого 512-битного «куска» входных данных. Наша тестовая фраза «hello world» довольно короткая, поэтому «кусок» всего один. На каждой итерации цикла мы будем изменять значения хеш-функций h0–h7, чтобы получить окончательный результат.

**Шаг 5. Создаём очередь сообщений (w)**

1. Копируем входные данные из шага 1 в новый массив, где каждая запись является 32-битным словом:

01101000011001010110110001101100 01101111001000000111011101101111

01110010011011000110010010000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000001011000

1. Добавляем ещё 48 слов, инициализированных нулями, чтобы получить массив w[0…63]:

01101000011001010110110001101100 01101111001000000111011101101111

01110010011011000110010010000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000001011000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

...

...

00000000000000000000000000000000 00000000000000000000000000000000

3. Изменяем нулевые индексы в конце массива, используя следующий алгоритм:

For i from w[16…63]:s0 = (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15] righthift 3)s1 = (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2] righthift 10)w [i] = w[i-16] + s0 + w[i-7] + s1

Давайте посмотрим, как это работает для w[16]:

w[1] rightrotate 7:

01101111001000000111011101101111 -> 11011110110111100100000011101110

w[1] rightrotate 18:

01101111001000000111011101101111 -> 00011101110110111101101111001000

w[1] rightshift 3:

01101111001000000111011101101111 -> 00001101111001000000111011101101

s0 = 11011110110111100100000011101110 XOR 00011101110110111101101111001000 XOR 00001101111001000000111011101101

s0 = 11001110111000011001010111001011

w[14] rightrotate 17:

00000000000000000000000000000000 -> 00000000000000000000000000000000

w[14] rightrotate19:

00000000000000000000000000000000 -> 00000000000000000000000000000000

w[14] rightshift 10:

00000000000000000000000000000000 -> 00000000000000000000000000000000

s1 = 00000000000000000000000000000000 XOR 00000000000000000000000000000000 XOR 00000000000000000000000000000000

s1 = 00000000000000000000000000000000

w[16] = w[0] + s0 + w[9] + s1

w[16] = 01101000011001010110110001101100 + 11001110111000011001010111001011 + 00000000000000000000000000000000 + 00000000000000000000000000000000

// сложение рассчитывается по модулю 2^32

w[16] = 00110111010001110000001000110111

Это оставляет нам 64 слова в нашей очереди сообщений (w):

01101000011001010110110001101100 01101111001000000111011101101111

01110010011011000110010010000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000000000000

00000000000000000000000000000000 00000000000000000000000001011000

00110111010001110000001000110111 10000110110100001100000000110001

11010011101111010001000100001011 01111000001111110100011110000010

00101010100100000111110011101101 01001011001011110111110011001001

00110001111000011001010001011101 10001001001101100100100101100100

01111111011110100000011011011010 11000001011110011010100100111010

10111011111010001111011001010101 00001100000110101110001111100110

10110000111111100000110101111101 01011111011011100101010110010011

00000000100010011001101101010010 00000111111100011100101010010100

00111011010111111110010111010110 01101000011001010110001011100110

11001000010011100000101010011110 00000110101011111001101100100101

10010010111011110110010011010111 01100011111110010101111001011010

11100011000101100110011111010111 10000100001110111101111000010110

11101110111011001010100001011011 10100000010011111111001000100001

11111001000110001010110110111000 00010100101010001001001000011001

00010000100001000101001100011101 01100000100100111110000011001101

10000011000000110101111111101001 11010101101011100111100100111000

00111001001111110000010110101101 11111011010010110001101111101111

11101011011101011111111100101001 01101010001101101001010100110100

00100010111111001001110011011000 10101001011101000000110100101011

01100000110011110011100010000101 11000100101011001001100000111010

00010001010000101111110110101101 10110000101100000001110111011001

10011000111100001100001101101111 01110010000101111011100000011110

10100010110101000110011110011010 00000001000011111001100101111011

11111100000101110100111100001010 11000010110000101110101100010110

**Шаг 6. Цикл сжатия**

Инициализируем переменные a, b, c, d, e, f, g, h и установим их равными текущим значениям хеша соответственно. h0, h1, h2, h3, h4, h5, h6, h7.

Запустим цикл сжатия, который будет изменять значения a…h . Цикл выглядит следующим образом:

for i from 0 to 63S1 = (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)ch = (e and f) xor ((not e) and g)temp1 = h + S1 + ch + k[i] + w[i]S0 = (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)maj = (a and b) xor (a and c) xor (b and c)temp2 := S0 + majh = gg = ff = ee = d + temp1d = cc = bb = aa = temp1 + temp2

Давайте пройдём первую итерацию. Сложение рассчитывается по модулю 2^32:

a = 0x6a09e667 = 01101010000010011110011001100111

b = 0xbb67ae85 = 10111011011001111010111010000101

c = 0x3c6ef372 = 00111100011011101111001101110010

d = 0xa54ff53a = 10100101010011111111010100111010

e = 0x510e527f = 01010001000011100101001001111111

f = 0x9b05688c = 10011011000001010110100010001100

g = 0x1f83d9ab = 00011111100000111101100110101011

h = 0x5be0cd19 = 01011011111000001100110100011001

e rightrotate 6:

01010001000011100101001001111111 -> 11111101010001000011100101001001

e rightrotate 11:

01010001000011100101001001111111 -> 01001111111010100010000111001010

e rightrotate 25:

01010001000011100101001001111111 -> 10000111001010010011111110101000

S1 = 11111101010001000011100101001001 XOR 01001111111010100010000111001010 XOR 10000111001010010011111110101000

S1 = 00110101100001110010011100101011

e and f:

01010001000011100101001001111111

& 10011011000001010110100010001100 =

00010001000001000100000000001100

not e:

01010001000011100101001001111111 -> 10101110111100011010110110000000

(not e) and g:

10101110111100011010110110000000

& 00011111100000111101100110101011 =

00001110100000011000100110000000

ch = (e and f) xor ((not e) and g)

= 00010001000001000100000000001100 xor 00001110100000011000100110000000

= 00011111100001011100100110001100

// k[i] is the round constant

// w[i] is the batch

temp1 = h + S1 + ch + k[i] + w[i]

temp1 = 01011011111000001100110100011001 + 00110101100001110010011100101011 + 00011111100001011100100110001100 + 1000010100010100010111110011000 + 01101000011001010110110001101100

temp1 = 01011011110111010101100111010100

a rightrotate 2:

01101010000010011110011001100111 -> 11011010100000100111100110011001

a rightrotate 13:

01101010000010011110011001100111 -> 00110011001110110101000001001111

a rightrotate 22:

01101010000010011110011001100111 -> 00100111100110011001110110101000

S0 = 11011010100000100111100110011001 XOR 00110011001110110101000001001111 XOR 00100111100110011001110110101000

S0 = 11001110001000001011010001111110

a and b:

01101010000010011110011001100111

& 10111011011001111010111010000101 =

00101010000000011010011000000101

a and c:

01101010000010011110011001100111

& 00111100011011101111001101110010 =

00101000000010001110001001100010

b and c:

10111011011001111010111010000101

& 00111100011011101111001101110010 =

00111000011001101010001000000000

maj = (a and b) xor (a and c) xor (b and c)

= 00101010000000011010011000000101 xor 00101000000010001110001001100010 xor 00111000011001101010001000000000

= 00111010011011111110011001100111

temp2 = S0 + maj

= 11001110001000001011010001111110 + 00111010011011111110011001100111

= 00001000100100001001101011100101

h = 00011111100000111101100110101011

g = 10011011000001010110100010001100

f = 01010001000011100101001001111111

e = 10100101010011111111010100111010 + 01011011110111010101100111010100

= 00000001001011010100111100001110

d = 00111100011011101111001101110010

c = 10111011011001111010111010000101

b = 01101010000010011110011001100111

a = 01011011110111010101100111010100 + 00001000100100001001101011100101

= 01100100011011011111010010111001

Все расчёты выполняются ещё 63 раза, изменяя переменные а…h. В итоге мы должны получить следующее:

h0 = 6A09E667 = 01101010000010011110011001100111

h1 = BB67AE85 = 10111011011001111010111010000101

h2 = 3C6EF372 = 00111100011011101111001101110010

h3 = A54FF53A = 10100101010011111111010100111010

h4 = 510E527F = 01010001000011100101001001111111

h5 = 9B05688C = 10011011000001010110100010001100

h6 = 1F83D9AB = 00011111100000111101100110101011

h7 = 5BE0CD19 = 01011011111000001100110100011001

a = 4F434152 = 001001111010000110100000101010010

b = D7E58F83 = 011010111111001011000111110000011

c = 68BF5F65 = 001101000101111110101111101100101

d = 352DB6C0 = 000110101001011011011011011000000

e = 73769D64 = 001110011011101101001110101100100

f = DF4E1862 = 011011111010011100001100001100010

g = 71051E01 = 001110001000001010001111000000001

h = 870F00D0 = 010000111000011110000000011010000

**Шаг 7. Изменяем окончательные значения**

После цикла сжатия, но ещё внутри основного цикла, мы модифицируем значения хеша, добавляя к ним соответствующие переменные a…h. Как обычно, всё сложение происходит по модулю 2^32.

h0 = h0 + a = 10111001010011010010011110111001

h1 = h1 + b = 10010011010011010011111000001000

h2 = h2 + c = 10100101001011100101001011010111

h3 = h3 + d = 11011010011111011010101111111010

h4 = h4 + e = 11000100100001001110111111100011

h5 = h5 + f = 01111010010100111000000011101110

h6 = h6 + g = 10010000100010001111011110101100

h7 = h7 + h = 11100010111011111100110111101001

**Шаг 8. Получаем финальный хеш**

И последний важный шаг — собираем всё вместе.

digest = h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7

= B94D27B9934D3E08A52E52D7DA7DABFAC484EFE37A5380EE9088F7ACE2EFCDE9

# **2 РАЗРАБОТКА ИНФОЛОГИЧЕСКОЙ МОДЕЛИ**

**2.1 Постановка задачи**

Задача курсовой работы заключается в том что нужно разработать информационную модель по заданной предметной области. В самом начале нужно выбрать на основе какой структуры будет информационная модель: Контейнер STL, списки(однонаправленные или двунаправленные), стек, очередь, бинарное дерево и т.д.

Каждому варианту даётся своя предметная область и в зависимости от неё описываются поля и ключ для поиска в нашей системе. Вид сортировки у каждого свой, в зависимости от варианта.

Информационная модель основана на бинарном дереве, которое составляется с соответствующим полем класса, по которому должна происходить сортировка, вследствие этого отпадает необходимость реализации быстрой сортировкой.

В информационной модели должна авторизация пользователя и соответствующая проверка на соответствие *username* и *password*, ролевая система вида: пользователь, администратор. Функционал администратора предусматривает возможность работы с пользователями, а также возможность редактирования данных (вставка новой записи, удаление записи, редактирование заданной записи, сохранение изменений в файл, отмена изменений). Функционал пользователя включает индивидуальное задание:

* Вывести общий список заказов, отсортированный по дате приемки
* Вывести список заказов, находящихся в ремонте.
* Предусмотреть поиск информации для заданного наименования изделия (вхождение строки).

**2.2 Описание предметной области**

Для курсовой работы дана такая предметная область как сервисный центр. Информационная модель содержит информацию о заказах. Данные о заказах содержат наименование ремонтируемого изделия (телевизор и т.д.), марку изделия, Ф.И.О. владельца, телефон владельца, стоимость ремонта, дату приемки, дату выдачи, статус (выполнен или нет).

Всё это необходимо указать для того, чтобы человек, ответственный за ремонт изделий клиентов имел представление о количестве заказов, их клиентах, и срок, который ему дан на ремонт. А также, в случае чего имеется возможность позвонить клиенту и обговорить его заказ. Данная программа облегчит учет заказов в сервисном центре. Администратор центра, в свою очередь, имеет возможность добавить новые заказы, либо изменить старые, если в них была ошибка, а также возможность удалить старые записи.

Именно по причинам изложенным выше без медицинской карты врачам в больницах и поликлинниках никак не обойтись.

* 1. **Схема возможностей пользователей:**

В данном подразделе представлены возможности наших пользователей (админов). Возможности пользователя показаны на рисунке 10

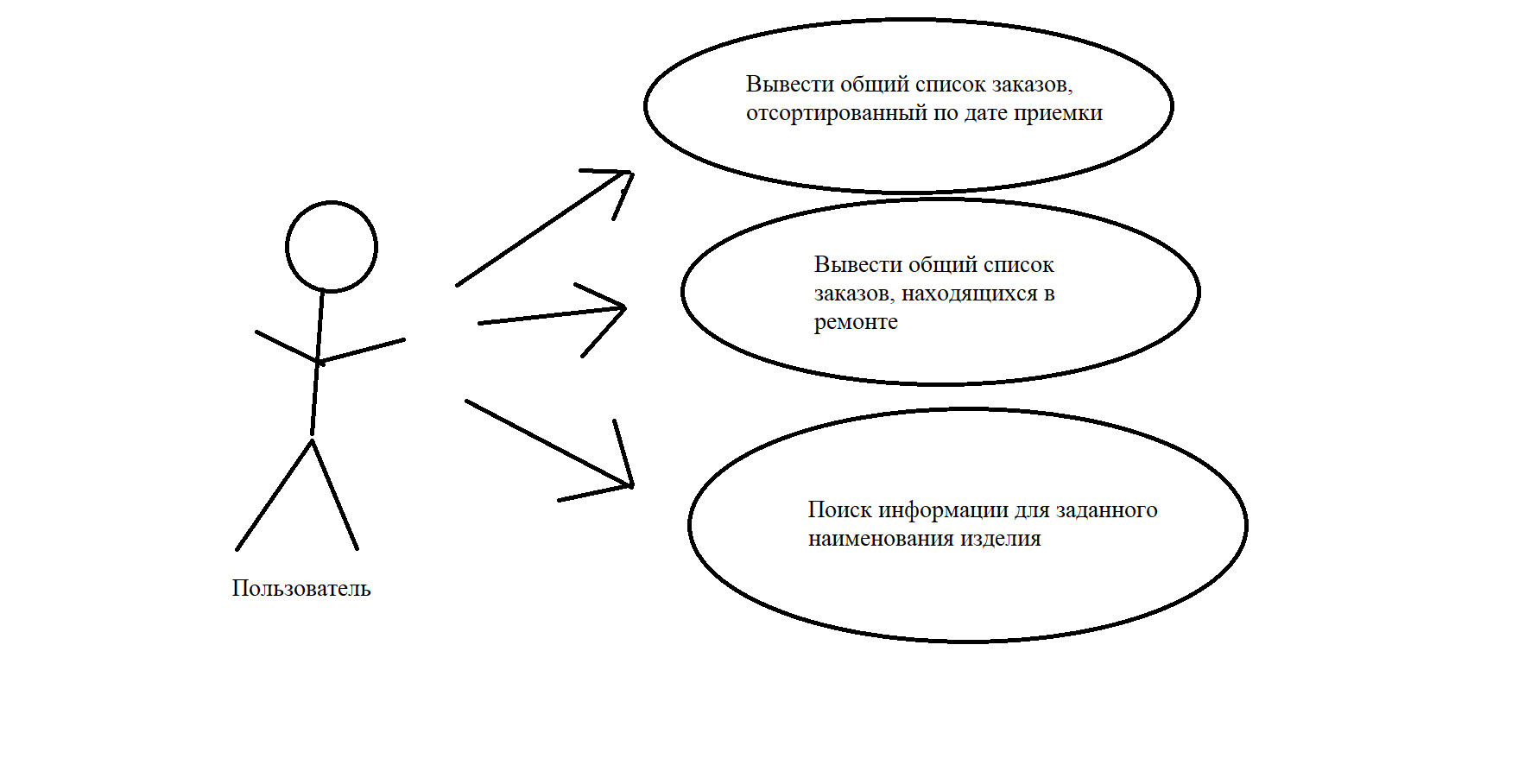


Рисунок 10 — возможности обычного пользователя

Возможности администратора представлены на рисунке 11

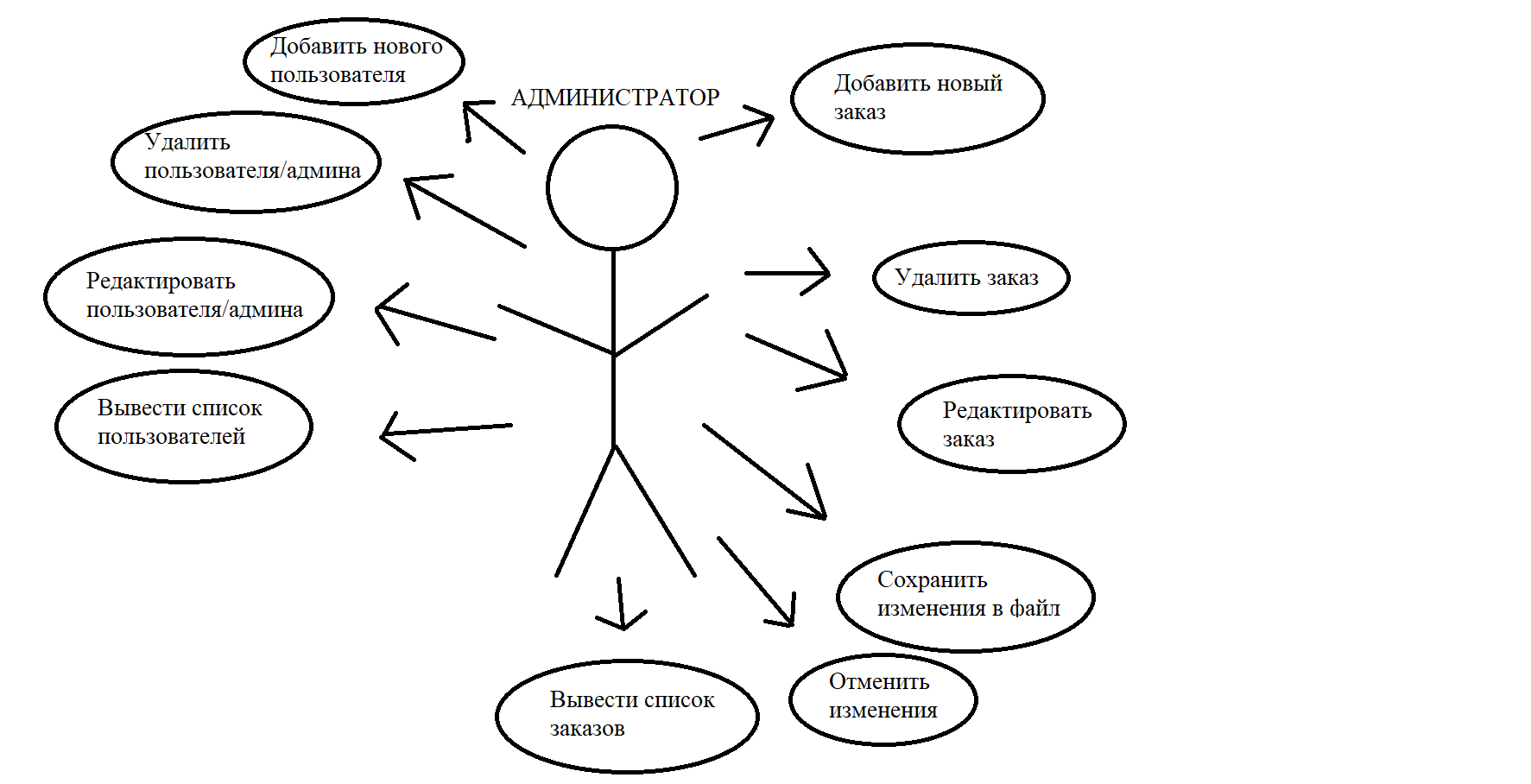


Рисунок 11 — возможности администратора сервисного центра

# **3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

* 1. **Основные файлы**

Программа состоит из 16 файлов, 8 из которых *cpp*, 6 заголовочных файла типа *h,* а также 2 бинарных файла типа *bin*.

Заголовочные файлы типа h:

− *user.h*, создание класса *User*, геттеры и сеттеры для полей класса, создание дочернего класса *Admin*;

− *TreeNode.h*, в нём создается класс *TreeNodeUser* и *TreeNodeOrder* для соответствующих бинарных деревьев;

− *order.h*, в котором создается заказ, прототипы геттеров и сеттеров, а также прототипы перегрузки операторов ввода и вывода;

− *binary\_tree.h*, создается бинарное дерево заказов и пользователей, а также соответствующие этим классам прототипы методов;

− *SHA256.h*, создание класса *SHA256,* который будет использоваться для шифрования паролей пользователей;

− *LoginScreen.h*, создание класса *LoginScreen*, а также прототипы методов;

Основные файлы типа *cpp*:

− *user.cpp*, в нём представлена реализация методов админа для работы с пользователями;

− *TreeNode.cpp*, в нём находятся реализации конструкторов для *TreeNodeUser* и *TreeNodeOrder*;

− *order.cpp*, в нём представлены реализации геттеров, сеттеров, а также перегрузки операторов ввода и вывода;

− *binary tree user.cpp*, в нем реализованы все методы для работы с бинарным деревом пользователей;

− *binary tree order.cpp*, в нем реализованы все методы для работы с бинарным деревом ордеров;

− *SHA256.cpp*, в нем реализован алгоритм шифрования SHA256;

− *LoginScreen.cpp*, в нем реализованы методы для авторизации и вывода меню пользователя и админа;

− *main.cpp*, в нём представлено создание объекта класса *LoginScreen* и вызов метода *Authorization*;

Бинарные файлы типа *bin*:

− *orders.bin*, сюда записывается вся информация о заказах;

− *users.*bin, сюда записывается вся информация о пользователях;

**3.2 Основные функции**

В таблице 1 представлены функции файла *binary tree user.cpp*

Таблица 1 − включающая в себя функции из файла *binary tree user.cpp*

|  |  |
| --- | --- |
| Название функции | Описание |
| 1 | 2 |
| *void BinaryTreeUser::PrintUsers()* | Вывод всех юзеров на консоль |
| *void BinaryTreeUser::Insert(User\* user) { root = InsertUser(root, user); }* | Метод для добавления нового пользователя в дерево |
| *void BinaryTreeUser::DeleteUserTree()* | Удаление дерева пользователей |
| *void BinaryTreeUser::DeleteUserNode(int\* count\_to\_find)* | Удаление пользователя из списка по заданному номеру в списке |
| *void EditUserNode(int\* count);* | Редактирование пользователя из списка по заданному номеру из списка |
| *int GetNameNumber(std::string s);* | Получает номер в списке пользователей текущего авторизованного пользователя в системе |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| *int SearchUserName(const std::string& name);* | Поиск имени на совпадение с уже существующим именем пользователя |
| *int SearchUsers(const std::string& name, const std::string& password);* | Поиск на существование такого *name* и *password* в *users.bin* |
| *void SaveUsersToFile(const std::string& filename);* | Сохранение пользователей в файл |
| *void LoadUsersFromFile(const std::string& filename);* | Загрузка пользователей из файла |

В таблице 2 представлены функции файла *binary tree order.cpp*

Таблица 2 – включающая в себя функции из файла *binary tree order.cpp*

|  |  |
| --- | --- |
| Название функции | Описание |
| 1 | 2 |
| *void Insert(Order\* order);* | Функция добавления нового заказа в бинарное дерево |
| *void PrintOrders();* | Функция для вывода бинарного дерева заказов. |
| *void SaveOrdersToFile(const std::string& filename);* | Функция сохранения заказов в бинарный файл |
| *void LoadOrdersFromFile(const std::string& filename);* | Функция загрузки заказов из бинарного файла |
| *void PrintRepairingOrders();* | Функция для вывода заказов, находящихся в ремонте |
| *Void SearchOrdersByProductName(const std::string& \_product\_name);* | Функция для поиска наименования изделия |
| *void DeleteOrderTree();* | Функция удаления бинарного дерева заказов |
| *void DeleteOrderNode(int\* count);* | Функция удаления заказа по номеру из общего списка заказов |
| *void EditOrderNode(int\* count);* | Функция для редактирования заказа по номеру из общего списка заказов |

В таблице 3 представлены функции файла *LoginScreen.cpp*

Таблица 3 – включающая в себя функции из файла *LoginScreen.cpp*

|  |  |
| --- | --- |
| Название функции | Описание |
| 1 | 2 |
| *void LoginScreen::Authorization()* | Функция авторизации, в которой реализован ввод имени пользователя и пароля |
| *void LoginScreen::AdminMenu()* | Функция, содержащая в себе весь пользовательский интерфейс для администратора |
| *void LoginScreen::UserMenu()* | Функция, содержащая в себе весь пользовательский интерфейс для пользователя |

В таблице 4 представлены функции файла *user.cpp*

Таблица 4 – включающая в себя функции из файла *user.cpp*

|  |  |
| --- | --- |
| Название функции | Описание |
| 1 | 2 |
| *void Admin::AddNewOrder(BinaryTreeOrder\* tree)* | Функция, которую вызывает администратор для добавления нового заказа |
| *void Admin::AddNewUser(BinaryTreeUser\* tree)* | Функция, которую вызывает администратор для добавления нового пользователя |

* 1. **Алгоритм программы**

На рисунке 12 представлена общая блок схема на которой показана работа пользователей

Администратор

Пользователь

Выбор администратора

Выбор пользователя

|  |
| --- |
| 1.Добавление нового заказа |
| 2.Удаление заказа |
| 3.Редактирование заказа |
| 4.Сохранение изменений в файл |
| 5.Отмена изменений  6. Вывод информации о заказах |
| 7.Добавление нового пользователя/админа |
| 8.Удаление пользователя/админа |
| 9.Редактирование пользователя/админа |
| 10.Сохранение изменений в файл |
| 11.Отмена изменений |
| 12.Вывести список всех пользователей |
| 12.Выход |

|  |
| --- |
| 1.Вывод общего списка заказов |
| 2.Вывод списка заказов находящихся в ремонте |
| 3.Поиск информации о заказах для заданного наименования изделия |
| 4.Выход |

При выборе Выход

Рисунок 12—схема работы с пользователями

# **4 ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ**

Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text

# **5 ЗАКЛЮЧЕНИЕ**

Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text Sample text

# **6 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Р. Лафоре “Объектно-ориентированное программирование в С++”

Учебник – 4-ое издание, 2004 г., с. 447 − 450

1. Веб приложение «Пошагово объясняем, как работает алгоритм хеширования SHA-2 (SHA-256)» Электрон. Данные.: Режим доступа: https://tproger.ru/translations/sha-2-step-by-step Дата доступа: 28.11.2023

# ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА

**Файл order.h**

#pragma once

#include <iostream>

#include <iomanip>

#include <string>

class Order

{

private:

std::string \_product\_name;

std::string \_product\_brand;

std::string \_owner\_full\_name;

std::string \_owner\_phone\_number;

double \_repair\_cost;

std::string \_acceptance\_date;

std::string \_issue\_date;

bool \_status;

public:

Order();

Order(const std::string& \_product\_name, const std::string& \_product\_brand, const std::string& \_owner\_full\_name, const std::string& \_owner\_phone\_number, double \_repair\_cost, const std::string& \_acceptance\_date, const std::string& \_issue\_date, bool \_status);

friend std::ostream& operator << (std::ostream& os, const Order& order);

friend std::istream& operator >> (std::istream& in, Order& order);

std::string GetProductName();

std::string GetProductBrand();

std::string GetOwnerFullName();

std::string GetOwnerPhoneNumber();

double GetRepairCost();

std::string GetAcceptanceDate();

std::string GetIssueDate();

bool GetStatus();

void SetProductName(std::string name);

void SetProductBrand(std::string name);

void SetOwnerFullName(std::string name);

void SetOwnerPhoneNumber(std::string phone\_number);

void SetRepairCost(double cost);

void SetAcceptanceDate(std::string date);

void SetIssueDate(std::string date);

void SetStatus(bool status);

};

**Файл order.cpp**

#include "order.h"

Order::Order()

{

\_product\_name = "Undefined";

\_product\_brand = "Undefined";

\_owner\_full\_name = "Undefined";

\_owner\_phone\_number = "Undefined";

\_repair\_cost = 0;

\_acceptance\_date = "Undefined";

\_issue\_date = "Undefined";

\_status = false;

}

Order::Order(const std::string& \_product\_name, const std::string& \_product\_brand, const std::string& \_owner\_full\_name, const std::string& \_owner\_phone\_number, double \_repair\_cost, const std::string& \_acceptance\_date, const std::string& \_issue\_date, bool \_status)

: \_product\_name(\_product\_name), \_product\_brand(\_product\_brand), \_owner\_full\_name(\_owner\_full\_name), \_owner\_phone\_number(\_owner\_phone\_number), \_repair\_cost(\_repair\_cost), \_acceptance\_date(\_acceptance\_date), \_issue\_date(\_issue\_date), \_status(\_status)

{

}

std::string Order::GetProductName() { return this->\_product\_name; }

std::string Order::GetProductBrand() { return this->\_product\_brand; }

std::string Order::GetOwnerFullName() { return this->\_owner\_full\_name; }

std::string Order::GetOwnerPhoneNumber() { return this->\_owner\_phone\_number; }

double Order::GetRepairCost() { return this->\_repair\_cost; }

std::string Order::GetAcceptanceDate() { return this->\_acceptance\_date; }

std::string Order::GetIssueDate() { return this->\_issue\_date; }

bool Order::GetStatus() { return this->\_status; }

void Order::SetProductName(std::string name) { this->\_product\_name = name; }

void Order::SetProductBrand(std::string name) { this->\_product\_brand = name; }

void Order::SetOwnerFullName(std::string name) { this->\_owner\_full\_name = name; }

void Order::SetOwnerPhoneNumber(std::string phone\_number) { \_owner\_phone\_number = phone\_number; }

void Order::SetRepairCost(double cost) { this->\_repair\_cost = cost; }

void Order::SetAcceptanceDate(std::string date) { this->\_acceptance\_date = date; }

void Order::SetIssueDate(std::string date) { this->\_issue\_date = date; }

void Order::SetStatus(bool status) { this->\_status = status; }

std::ostream& operator << (std::ostream& os, const Order& order)

{

os << std::left << std::setw(30) << order.\_product\_name

<< std::setw(30) << order.\_product\_brand

<< std::setw(30) << order.\_owner\_full\_name

<< std::setw(20) << order.\_owner\_phone\_number

<< std::setw(15) << order.\_repair\_cost

<< std::setw(17) << order.\_acceptance\_date

<< std::setw(17) << order.\_issue\_date

<< std::setw(10) << (order.\_status ? "Completed" : "Not completed") << std::endl;

return os;

}

std::istream& operator >> (std::istream& in, Order& order)

{

try

{

in.ignore(in.rdbuf()->in\_avail());

in.clear();

std::cout << "Product Name: ";

std::getline(in, order.\_product\_name);

std::cout << "Product Brand: ";

std::getline(in, order.\_product\_brand);

std::cout << "Owner Name: ";

std::getline(in, order.\_owner\_full\_name);

std::cout << "Owner phone number: ";

std::getline(in, order.\_owner\_phone\_number);

std::cout << "Repair cost: ";

in >> order.\_repair\_cost;

while (in.fail())

{

in.clear();

while (in.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

std::cout << "Repair cost: ";

in >> order.\_repair\_cost;

}

in.ignore(in.rdbuf()->in\_avail());

std::cout << "Acceptance date: ";

std::getline(in, order.\_acceptance\_date);

std::cout << "Issue date: ";

std::getline(in, order.\_issue\_date);

std::cout << "Status: ";

in >> order.\_status;

in.clear();

return in;

}

catch (int code)

{

std::cout << "Error code: " << code << std::endl;

}

}

**Файл user.h**

#pragma once

#ifndef USER\_H

#define USER\_H

#include <iostream>

#include <conio.h>

#include "SHA256.h"

class BinaryTreeOrder;

class BinaryTreeUser;

using namespace std;

class User

{

string name;

string password;

bool isAdmin;

public:

User();

User(const string& name, const string& password, bool isAdmin);

void SetLogin(string name) { this->name = name; }

void SetPassword(string password) { this->password = password; }

void SetIsAdmin(bool isAdmin) { this->isAdmin = isAdmin; }

string GetLogin() { return name; }

string GetPassword() { return password; }

bool GetIsAdmin() { return isAdmin; }

};

class Admin : User

{

public:

void AddNewOrder(BinaryTreeOrder\* tree);

void AddNewUser(BinaryTreeUser\* tree);

Admin() = default;

};

#endif // !USER\_H

**Файл user.cpp**

#include "binary\_tree.h"

User::User(const string& name, const string& password, bool isAdmin)

: name(name), password(password), isAdmin(isAdmin)

{

};

User::User() = default;

void Admin::AddNewOrder(BinaryTreeOrder\* tree)

{

Order\* order = new Order();

std::cout << "Enter data: " << std::endl;

cin >> \*order;

tree->Insert(order);

system("cls");

std::cout << "The data has been added succesfully " << std::endl;

}

void Admin::AddNewUser(BinaryTreeUser\* tree)

{

User\* user = new User();

std::cout << "Enter data: " << std::endl;

cout << "Enter username: ";

string s;

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

getline(cin, s);

user->SetLogin(s);

cout << "Enter user's password: ";

string password;

char ch;

while ((ch = \_getch()) != '\r')

{

if (ch == '\b')

{

if (!password.empty())

{

password.pop\_back();

std::cout << "\b \b";

}

}

else

{

password.push\_back(ch);

std::cout << "\*";

}

}

std::cout << std::endl;

SHA256 sha;

sha.update(password);

password.clear();

std::array<uint8\_t, 32> digest = sha.digest();

user->SetPassword(SHA256::toString(digest));

cout << "Set a role (1 - admin, 0 - not admin): ";

bool b;

cin >> b;

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

user->SetIsAdmin(b);

tree->Insert(user);

tree->PrintUsers();

}

**Файл TreeNode.h**

#pragma once

#ifndef TREENODE\_H

#define TREENODE\_H

#include "user.h"

#include "order.h"

class TreeNodeUser {

public:

User\* data;

TreeNodeUser\* left;

TreeNodeUser\* right;

TreeNodeUser(User\* data);

};

class TreeNodeOrder {

public:

Order\* data;

TreeNodeOrder\* left;

TreeNodeOrder\* right;

TreeNodeOrder(Order\* data);

};

#endif // !TREENODE\_H

**Файл TreeNode.cpp**

#include "TreeNode.h"

TreeNodeUser::TreeNodeUser(User\* data)

{

this->data = data;

left = nullptr;

right = nullptr;

}

TreeNodeOrder::TreeNodeOrder(Order\* data)

{

this->data = data;

left = nullptr;

right = nullptr;

}

**Файл SHA256.h**

#pragma once

#ifndef SHA256\_H

#define SHA256\_H

#include <string>

#include <array>

class SHA256 {

public:

SHA256();

void update(const uint8\_t \* data, size\_t length);

void update(const std::string &data);

std::array<uint8\_t, 32> digest();

static std::string toString(const std::array<uint8\_t, 32> & digest);

private:

uint8\_t m\_data[64];

uint32\_t m\_blocklen;

uint64\_t m\_bitlen;

uint32\_t m\_state[8]; //A, B, C, D, E, F, G, H

static constexpr std::array<uint32\_t, 64> K = {

0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,

0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5,

0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,

0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174,

0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,

0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da,

0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,

0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967,

0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,

0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85,

0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,

0xd192e819,0xd6990624,0xf40e3585,0x106aa070,

0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,

0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,

0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,

0x90befffa,0xa4506ceb,0xbef9a3f7,0xc67178f2

};

static uint32\_t rotr(uint32\_t x, uint32\_t n);

static uint32\_t choose(uint32\_t e, uint32\_t f, uint32\_t g);

static uint32\_t majority(uint32\_t a, uint32\_t b, uint32\_t c);

static uint32\_t sig0(uint32\_t x);

static uint32\_t sig1(uint32\_t x);

void transform();

void pad();

void revert(std::array<uint8\_t, 32> & hash);

};

#endif

**Файл SHA256.cpp**

#include "SHA256.h"

#include <cstring>

#include <sstream>

#include <iomanip>

SHA256::SHA256(): m\_blocklen(0), m\_bitlen(0) {

m\_state[0] = 0x6a09e667;

m\_state[1] = 0xbb67ae85;

m\_state[2] = 0x3c6ef372;

m\_state[3] = 0xa54ff53a;

m\_state[4] = 0x510e527f;

m\_state[5] = 0x9b05688c;

m\_state[6] = 0x1f83d9ab;

m\_state[7] = 0x5be0cd19;

}

void SHA256::update(const uint8\_t \* data, size\_t length) {

for (size\_t i = 0 ; i < length ; i++) {

m\_data[m\_blocklen++] = data[i];

if (m\_blocklen == 64) {

transform();

// End of the block

m\_bitlen += 512;

m\_blocklen = 0;

}

}

}

void SHA256::update(const std::string &data) {

update(reinterpret\_cast<const uint8\_t\*> (data.c\_str()), data.size());

}

std::array<uint8\_t,32> SHA256::digest() {

std::array<uint8\_t,32> hash;

pad();

revert(hash);

return hash;

}

uint32\_t SHA256::rotr(uint32\_t x, uint32\_t n) {

return (x >> n) | (x << (32 - n));

}

uint32\_t SHA256::choose(uint32\_t e, uint32\_t f, uint32\_t g) {

return (e & f) ^ (~e & g);

}

uint32\_t SHA256::majority(uint32\_t a, uint32\_t b, uint32\_t c) {

return (a & (b | c)) | (b & c);

}

uint32\_t SHA256::sig0(uint32\_t x) {

return SHA256::rotr(x, 7) ^ SHA256::rotr(x, 18) ^ (x >> 3);

}

uint32\_t SHA256::sig1(uint32\_t x) {

return SHA256::rotr(x, 17) ^ SHA256::rotr(x, 19) ^ (x >> 10);

}

void SHA256::transform() {

uint32\_t maj, xorA, ch, xorE, sum, newA, newE, m[64];

uint32\_t state[8];

for (uint8\_t i = 0, j = 0; i < 16; i++, j += 4) { // Split data in 32 bit blocks for the 16 first words

m[i] = (m\_data[j] << 24) | (m\_data[j + 1] << 16) | (m\_data[j + 2] << 8) | (m\_data[j + 3]);

}

for (uint8\_t k = 16 ; k < 64; k++) { // Remaining 48 blocks

m[k] = SHA256::sig1(m[k - 2]) + m[k - 7] + SHA256::sig0(m[k - 15]) + m[k - 16];

}

for(uint8\_t i = 0 ; i < 8 ; i++) {

state[i] = m\_state[i];

}

for (uint8\_t i = 0; i < 64; i++) {

maj = SHA256::majority(state[0], state[1], state[2]);

xorA = SHA256::rotr(state[0], 2) ^ SHA256::rotr(state[0], 13) ^ SHA256::rotr(state[0], 22);

ch = choose(state[4], state[5], state[6]);

xorE = SHA256::rotr(state[4], 6) ^ SHA256::rotr(state[4], 11) ^ SHA256::rotr(state[4], 25);

sum = m[i] + K[i] + state[7] + ch + xorE;

newA = xorA + maj + sum;

newE = state[3] + sum;

state[7] = state[6];

state[6] = state[5];

state[5] = state[4];

state[4] = newE;

state[3] = state[2];

state[2] = state[1];

state[1] = state[0];

state[0] = newA;

}

for(uint8\_t i = 0 ; i < 8 ; i++) {

m\_state[i] += state[i];

}

}

void SHA256::pad() {

uint64\_t i = m\_blocklen;

uint8\_t end = m\_blocklen < 56 ? 56 : 64;

m\_data[i++] = 0x80; // Append a bit 1

while (i < end) {

m\_data[i++] = 0x00; // Pad with zeros

}

if(m\_blocklen >= 56) {

transform();

memset(m\_data, 0, 56);

}

// Append to the padding the total message's length in bits and transform.

m\_bitlen += m\_blocklen \* 8;

m\_data[63] = m\_bitlen;

m\_data[62] = m\_bitlen >> 8;

m\_data[61] = m\_bitlen >> 16;

m\_data[60] = m\_bitlen >> 24;

m\_data[59] = m\_bitlen >> 32;

m\_data[58] = m\_bitlen >> 40;

m\_data[57] = m\_bitlen >> 48;

m\_data[56] = m\_bitlen >> 56;

transform();

}

void SHA256::revert(std::array<uint8\_t, 32> & hash) {

// SHA uses big endian byte ordering

// Revert all bytes

for (uint8\_t i = 0 ; i < 4 ; i++) {

for(uint8\_t j = 0 ; j < 8 ; j++) {

hash[i + (j \* 4)] = (m\_state[j] >> (24 - i \* 8)) & 0x000000ff;

}

}

}

std::string SHA256::toString(const std::array<uint8\_t, 32> & digest) {

std::stringstream s;

s << std::setfill('0') << std::hex;

for(uint8\_t i = 0 ; i < 32 ; i++) {

s << std::setw(2) << (unsigned int) digest[i];

}

return s.str();

}

**Файл LoginScreen.h**

#pragma once

class LoginScreen

{

public:

LoginScreen();

~LoginScreen();

void Authorization();

void AdminMenu();

void UserMenu();

private:

};

**Файл LoginScreen.cpp**

#include "LoginScreen.h"

#include "SHA256.h"

#include "binary\_tree.h"

#include <iostream>

#include <conio.h>

LoginScreen::LoginScreen()

{

}

LoginScreen::~LoginScreen()

{

}

void LoginScreen::Authorization()

{

int flag = 1;

while (flag)

{

std::string username;

std::cout << "Enter username: ";

std::cin >> username;

std::string password;

std::cout << "Enter your password: ";

char ch;

while ((ch = \_getch()) != '\r')

{

if (ch == '\b')

{

if (!password.empty())

{

password.pop\_back();

std::cout << "\b \b";

}

}

else

{

password.push\_back(ch);

std::cout << "\*";

}

}

std::cout << std::endl;

SHA256 sha;

sha.update(password);

password.clear();

std::array<uint8\_t, 32> digest = sha.digest();

// проверка на верность логина и пароля в файле

BinaryTreeUser\* tree\_user = new BinaryTreeUser();

tree\_user->LoadUsersFromFile("../data/users.bin");

if (tree\_user->SearchUsers(username, SHA256::toString(digest)) == 1) // 1 - admin, 2 - user, 0 - no such user

{

flag = 0;

system("cls");

std::cout << "Welcome, " << username << std::endl;

this->AdminMenu();

}

else

{

if (tree\_user->SearchUsers(username, SHA256::toString(digest)) == 2)

{

flag = 0;

system("cls");

std::cout << "Welcome, " << username << std::endl;

this->UserMenu();

}

else

{

std::cout << "Error: wrong password or username" << std::endl;

}

}

}

}

void LoginScreen::AdminMenu()

{

BinaryTreeUser\* tree\_user = new BinaryTreeUser();

tree\_user->LoadUsersFromFile("../data/users.bin");

Admin\* admin = new Admin();

BinaryTreeOrder\* tree\_order = new BinaryTreeOrder();

tree\_order->LoadOrdersFromFile("../data/orders.bin");

int choice;

do

{

std::cout << "Menu:" << std::endl;

std::cout << "---------------------------|" << std::endl;

std::cout << "1. Add new Order |" << std::endl;

std::cout << "2. Delete Order |" << std::endl;

std::cout << "3. Edit Order |" << std::endl;

std::cout << "4. Save Changes (orders) |" << std::endl;

std::cout << "5. Cancel Changes (orders) |" << std::endl;

std::cout << "6. Show orders |" << std::endl;

std::cout << "---------------------------|" << std::endl;

std::cout << "7. Add new user/admin |" << std::endl;

std::cout << "8. Delete user/admin |" << std::endl;

std::cout << "9. Edit user/admin |" << std::endl;

std::cout << "10. Save Changes (users) |"<< std::endl;

std::cout << "11. Cancel Changes (users) |" << std::endl;

std::cout << "12. Show users |" << std::endl;

std::cout << "---------------------------|" << std::endl;

std::cout << "0. Exit |" << std::endl;

std::cout << "---------------------------|" << std::endl;

std::cout << "Enter your choice: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> choice;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice;

}

switch (choice) {

case 1:

system("cls");

admin->AddNewOrder(tree\_order);

break;

case 2:

system("cls");

tree\_order->PrintOrders();

std::cout << "Enter a number of order to delete: ";

int number;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> number;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> number;

}

tree\_order->DeleteOrderNode(&number);

break;

case 3:

system("cls");

tree\_order->PrintOrders();

std::cout << "Enter a number of order to edit: ";

int number2;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> number2;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> number2;

}

tree\_order->EditOrderNode(&number2);

break;

case 4:

system("cls");

tree\_order->SaveOrdersToFile("../data/orders.bin");

break;

case 5:

system("cls");

std::cout << "Are you sure ?" << std::endl;

std::cout << "1. Yes" << std::endl;

std::cout << "2. No" << std::endl;

int choice2;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

do

{

std::cin >> choice2;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice2;

}

switch (choice2)

{

case 1:

system("cls");

std::cout << "Success" << std::endl;

tree\_order->DeleteOrderTree();

tree\_order->LoadOrdersFromFile("../data/orders.bin");

choice2 = 0;

break;

case 2:

system("cls");

std::cout << "Canceled..." << std::endl;

choice2 = 0;

break;

default:

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice2 != 0);

break;

case 6:

system("cls");

tree\_order->PrintOrders();

std::cout << "Press any key to continue" << std::endl;

\_getch();

system("cls");

break;

case 7:

system("cls");

admin->AddNewUser(tree\_user);

break;

case 8:

system("cls");

tree\_user->PrintUsers();

std::cout << "Enter a number of user to delete: ";

int number3;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> number3;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> number3;

}

tree\_user->DeleteUserNode(&number3);

break;

case 9:

system("cls");

tree\_user->PrintUsers();

std::cout << "Enter a number of user to edit: ";

int number4;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> number4;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> number4;

}

tree\_user->EditUserNode(&number4);

break;

case 10:

system("cls");

tree\_user->SaveUsersToFile("../data/users.bin");

break;

case 11:

system("cls");

std::cout << "Are you sure ?" << std::endl;

std::cout << "1. Yes" << std::endl;

std::cout << "2. No" << std::endl;

int choice4;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

do

{

std::cin >> choice4;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice4;

}

switch (choice4)

{

case 1:

system("cls");

std::cout << "Success" << std::endl;

tree\_user->DeleteUserTree();

tree\_user->LoadUsersFromFile("../data/users.bin");

choice4 = 0;

break;

case 2:

system("cls");

std::cout << "Canceled..." << std::endl;

choice4 = 0;

break;

default:

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice4 != 0);

break;

case 12:

system("cls");

tree\_user->PrintUsers();

std::cout << "Press any key to continue" << std::endl;

\_getch();

system("cls");

break;

case 0:

system("cls");

std::cout << "Are you sure ?" << std::endl;

std::cout << "1. Yes" << std::endl;

std::cout << "2. No" << std::endl;

int choice3;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

do

{

std::cin >> choice3;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice3;

}

switch (choice3)

{

case 1:

choice3 = 0;

break;

std::cout << "Exiting..." << std::endl;

case 2:

choice = 1;

choice3 = 0;

break;

default:

choice3 = 1;

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice3 != 0);

break;

default:

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice != 0);

}

void LoginScreen::UserMenu()

{

std::string name;

BinaryTreeOrder\* tree\_order = new BinaryTreeOrder();

tree\_order->LoadOrdersFromFile("../data/orders.bin");

int choice;

do {

std::cout << "Menu:" << std::endl;

std::cout << "------------------------------------------------------|" << std::endl;

std::cout << "1. Display a general list of orders |" << std::endl;

std::cout << "2. Display a list of orders under repair |" << std::endl;

std::cout << "3. Searching for information for a given product name |" << std::endl;

std::cout << "------------------------------------------------------|" << std::endl;

std::cout << "0. Exit |" << std::endl;

std::cout << "------------------------------------------------------|" << std::endl;

std::cout << "Enter your choice: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> choice;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice;

}

switch (choice) {

case 1:

system("cls");

tree\_order->PrintOrders();

std::cout << "Press any key to continue" << std::endl;

\_getch();

system("cls");

break;

case 2:

system("cls");

tree\_order->PrintRepairingOrders();

std::cout << "Press any key to continue" << std::endl;

\_getch();

system("cls");

break;

case 3:

system("cls");

std::cout << "Enter product name to search: ";

std::cin >> name;

tree\_order->SearchOrdersByProductName(name);

break;

case 0:

system("cls");

std::cout << "Are you sure ?" << std::endl;

std::cout << "1. Yes" << std::endl;

std::cout << "2. No" << std::endl;

int choice2;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

do

{

std::cin >> choice2;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice2;

}

switch (choice2)

{

case 1:

choice2 = 0;

break;

std::cout << "Exiting..." << std::endl;

case 2:

system("cls");

std::cout << "Canceled..." << std::endl;

choice = 1;

choice2 = 0;

break;

default:

choice2 = 1;

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice2 != 0);

break;

default:

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice != 0);

}

**Файл binary\_tree.h**

#pragma once

#ifndef BINARY\_TREE\_H

#define BINARY\_TREE\_H

#include "TreeNode.h"

#include <fstream>

#include <string>

#include <iomanip>

class TreeNodeOrder;

class TreeNodeUser;

//extern std::string OrdersFileName = "../data/orders.bin";

//extern std::string UsersFileName = "../data/users.bin";

class BinaryTreeUser

{

private:

TreeNodeUser\* root;

TreeNodeUser\* InsertUser(TreeNodeUser\* node, User\* user);

void PrintUsersPrivate(TreeNodeUser\* node, int\* count);

int SearchUsersPrivate(TreeNodeUser\* node, const std::string& name, const std::string& password);

void SaveUsersToFilePrivate(TreeNodeUser\* node, std::ofstream\* outFile);

void DeleteUserTreePrivate(TreeNodeUser\* node);

TreeNodeUser\* DeleteUserNodePrivate(TreeNodeUser\* node, int\* count\_to\_find, int\* count);

TreeNodeUser\* FindMinNode(TreeNodeUser\* node);

void EditUserNodePrivate(TreeNodeUser\* node, int\* count\_to\_find, int\* count);

public:

BinaryTreeUser();

void LoadUsersFromFile(const std::string& filename);

void SaveUsersToFile(const std::string& filename);

int SearchUsers(const std::string& name, const std::string& password);

void Insert(User\* user);

void PrintUsers();

void DeleteUserTree();

void DeleteUserNode(int\* count);

void EditUserNode(int\* count);

};

class BinaryTreeOrder {

private:

TreeNodeOrder\* root;

TreeNodeOrder\* InsertOrder(TreeNodeOrder\* node, Order\* order);

void PrintOrdersPrivate(TreeNodeOrder\* node, int\* count);

void PrintRepairingOrdersPrivate(TreeNodeOrder\* node, int\* count);

void SearchOrdersByProductNamePrivate(TreeNodeOrder\* node, const std::string& \_product\_name, int\* count);

void SaveOrdersToFilePrivate(TreeNodeOrder\* node, std::ofstream\* outFile);

void DeleteOrderTreePrivate(TreeNodeOrder\* node);

TreeNodeOrder\* DeleteOrderNodePrivate(TreeNodeOrder\* node, int\* count\_to\_find, int\* count);

TreeNodeOrder\* FindMinNode(TreeNodeOrder\* node);

void EditOrderNodePrivate(TreeNodeOrder\* node, int\* count\_to\_find, int\* count);

public:

BinaryTreeOrder();

void Insert(Order\* order);

void PrintOrders();

void SaveOrdersToFile(const std::string& filename);

void LoadOrdersFromFile(const std::string& filename);

void PrintRepairingOrders();

void SearchOrdersByProductName(const std::string& \_product\_name);

void DeleteOrderTree();

void DeleteOrderNode(int\* count);

void EditOrderNode(int\* count);

};

#endif // !BINARY\_TREE\_H

**Файл binary tree user.cpp**

#include "binary\_tree.h"

BinaryTreeUser::BinaryTreeUser() { root = nullptr; }

void BinaryTreeUser::Insert(User\* user) { root = InsertUser(root, user); }

void BinaryTreeUser::PrintUsers()

{

int count = 1;

int\* countptr = &count;

std::cout << std::left << std::setw(3) << "#" << std::setw(20) << std::left << "Login" << std::setw(64) << std::left << "Password" << std::setw(1) << std::left << " " << "Role" << std::endl;

std::cout << std::setfill('-') << std::setw(88 + 10) << "-" << std::endl;

std::cout << std::setfill(' ');

PrintUsersPrivate(root, countptr);

}

void BinaryTreeUser::DeleteUserTree()

{

DeleteUserTreePrivate(root);

root = nullptr;

}

void BinaryTreeUser::DeleteUserNode(int\* count\_to\_find)

{

int count = 1;

int\* countptr = &count;

root = DeleteUserNodePrivate(root, count\_to\_find, countptr);

}

void BinaryTreeUser::EditUserNode(int\* count\_to\_find)

{

int count = 1;

int\* countptr = &count;

EditUserNodePrivate(root, count\_to\_find, countptr);

}

void BinaryTreeUser::SaveUsersToFile(const std::string& filename)

{

std::ofstream outFile(filename, std::ios::binary | std::ios::out);

if (outFile.is\_open())

{

SaveUsersToFilePrivate(root, &outFile);

outFile.close();

std::cout << "Data has been written to the file." << std::endl;

}

else {

std::cout << "Failed to open the file for writing." << std::endl;

}

}

TreeNodeUser\* BinaryTreeUser::InsertUser(TreeNodeUser\* node, User\* user)

{

if (node == nullptr) {

return new TreeNodeUser(user);

}

else

{

if (user->GetLogin() < node->data->GetLogin())

{

node->left = InsertUser(node->left, user);

}

else

{

node->right = InsertUser(node->right, user);

}

}

return node;

}

void BinaryTreeUser::PrintUsersPrivate(TreeNodeUser\* node, int\* count)

{

if (node != nullptr)

{

PrintUsersPrivate(node->left, count);

std::cout << std::left << std::setw(3) << (\*count)++ << std::setw(20) << std::left << node->data->GetLogin() << std::setw(64) << std::left << node->data->GetPassword() << std::setw(1) << std::left << " " << (node->data->GetIsAdmin() ? "admin" : "user") << std::endl;

PrintUsersPrivate(node->right, count);

}

}

void BinaryTreeUser::LoadUsersFromFile(const std::string& filename)

{

std::ifstream inFile(filename, std::ios::binary | std::ios::in);

if (inFile.is\_open())

{

while (inFile.good())

{

User\* user = new User();

// Чтение login

int buf\_size = 0;

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

std::string buf;

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

user->SetLogin(buf);

if (buf == "")

{

inFile.close();

break;

}

// Read password

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

user->SetPassword(buf);

// Read IsAdmin

bool temp\_bool;

inFile.read(reinterpret\_cast<char\*>(&temp\_bool), sizeof(bool));

user->SetIsAdmin(temp\_bool);

this->Insert(user);

}

inFile.close();

}

else {

std::cout << "Failed to open the file for reading." << std::endl;

}

}

void BinaryTreeUser::SaveUsersToFilePrivate(TreeNodeUser\* node, std::ofstream\* outFile)

{

if (node != nullptr)

{

// Запись \_owner\_phone\_number

int name\_size = node->data->GetLogin().size();

outFile->write(reinterpret\_cast<char\*>(&name\_size), sizeof(int));

outFile->write(node->data->GetLogin().c\_str(), name\_size);

// Запись \_issue\_date

int password\_size = node->data->GetPassword().size();

outFile->write(reinterpret\_cast<char\*>(&password\_size), sizeof(int));

outFile->write(node->data->GetPassword().c\_str(), password\_size);

// Запись \_status

bool temp2 = node->data->GetIsAdmin();

outFile->write(reinterpret\_cast<char\*>(&temp2), sizeof(bool));

SaveUsersToFilePrivate(node->left, outFile);

SaveUsersToFilePrivate(node->right, outFile);

}

}

void BinaryTreeUser::DeleteUserTreePrivate(TreeNodeUser\* node)

{

if (node == nullptr)

{

return;

}

DeleteUserTreePrivate(node->left);

DeleteUserTreePrivate(node->right);

delete node;

}

TreeNodeUser\* BinaryTreeUser::DeleteUserNodePrivate(TreeNodeUser\* node, int\* count\_to\_find, int\* count)

{

if (node != nullptr)

{

node->left = DeleteUserNodePrivate(node->left, count\_to\_find, count);

if (\*count\_to\_find == \*count)

{

if (node->left == nullptr && node->right == nullptr) {

(\*count)++;

delete node;

return nullptr;

}

else if (node->left == nullptr) {

TreeNodeUser\* temp = node->right;

(\*count)++;

delete node;

return temp;

}

else if (node->right == nullptr) {

TreeNodeUser\* temp = node->left;

(\*count)++;

delete node;

return temp;

}

else {

TreeNodeUser\* minRightNode = FindMinNode(node->right);

\*(node->data) = \*(minRightNode->data);

node->right = DeleteUserNodePrivate(node->right, count\_to\_find, count);

(\*count)++;

}

}

else

{

(\*count)++;

node->right = DeleteUserNodePrivate(node->right, count\_to\_find, count);

}

}

else

{

return nullptr;

}

return node;

}

TreeNodeUser\* BinaryTreeUser::FindMinNode(TreeNodeUser\* node)

{

if (node == nullptr) {

return nullptr;

}

while (node->left != nullptr) {

node = node->left;

}

return node;

}

void BinaryTreeUser::EditUserNodePrivate(TreeNodeUser\* node, int\* count\_to\_find, int\* count)

{

if (node != nullptr)

{

EditUserNodePrivate(node->left, count\_to\_find, count);

if (\*count\_to\_find == \*count)

{

int choice;

std::string s;

double d;

bool b;

do

{

std::cout << "What do you want to change ?" << std::endl;

std::cout << "1. Username" << std::endl;

std::cout << "2. Password" << std::endl;

std::cout << "3. Role" << std::endl;

std::cout << "0. Exit" << std::endl;

std::cout << "Enter your choice: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> choice;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

}

SHA256 sha;

switch (choice) {

case 1:

system("cls");

std::cout << "Enter a new username: ";

std::getline(std::cin, s);

node->data->SetLogin(s);

break;

case 2:

system("cls");

std::cout << "Enter a new password: ";

std::getline(std::cin, s);

sha.update(s);

s.clear();

std::array<uint8\_t, 32> digest = sha.digest();

node->data->SetPassword(SHA256::toString(digest));

break;

case 3:

system("cls");

std::cout << "Enter a new role (1 - admin, 0 - not admin): ";

std::cin >> b;

node->data->SetIsAdmin(b);

std::cin.clear();

system("cls");

break;

case 0:

system("cls");

std::cout << "Are you sure ?" << std::endl;

std::cout << "1. Yes" << std::endl;

std::cout << "2. No" << std::endl;

int choice3;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

do

{

std::cin >> choice3;

while (cin.fail())

{

//in.ignore(in.rdbuf()->in\_avail());

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice3;

}

switch (choice3)

{

case 1:

choice3 = 0;

std::cout << "Exiting..." << std::endl;

break;

case 2:

choice = 1;

choice3 = 0;

break;

default:

choice3 = 1;

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice3 != 0);

break;

default:

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice != 0);

}

else

{

(\*count)++;

EditUserNodePrivate(node->right, count\_to\_find, count);

}

}

else

{

return;

}

return;

}

int BinaryTreeUser::SearchUsersPrivate(TreeNodeUser\* node, const std::string& name, const std::string& password)

{

if (node != nullptr)

{

if (SearchUsersPrivate(node->left, name, password) == 1) return 1;

else if (SearchUsersPrivate(node->left, name, password) == 2) return 2;

int found\_position\_name = node->data->GetLogin().find(name);

int found\_position\_password = node->data->GetPassword().find(password);

if ((found\_position\_name != string::npos && found\_position\_name + name.length() == node->data->GetLogin().length())

&& (found\_position\_password != string::npos && found\_position\_password + password.length() == node->data->GetPassword().length()))

{

if (node->data->GetIsAdmin())

{

return 1;

}

else return 2;

std::cout << "Login: " << node->data->GetLogin() << "Password: " << node->data->GetPassword() << "Is Admin: " << node->data->GetIsAdmin() << std::endl;

}

if (SearchUsersPrivate(node->right, name, password) == 1) return 1;

else if (SearchUsersPrivate(node->left, name, password) == 2) return 2;

}

return 0;

}

int BinaryTreeUser::SearchUsers(const std::string& name, const std::string& password) { return SearchUsersPrivate(root, name, password); }

**Файл binary tree order.cpp**

#include "binary\_tree.h"

BinaryTreeOrder::BinaryTreeOrder() { root = nullptr; }

void BinaryTreeOrder::Insert(Order\* order) { root = InsertOrder(root, order); }

void BinaryTreeOrder::PrintOrders()

{

int count = 1;

int\* countptr = &count;

std::cout << std::left << std::setw(3) << "#" << std::setw(30) << "Product Name"

<< std::setw(30) << "Product Brand"

<< std::setw(30) << "Owner Full Name"

<< std::setw(20) << "Phone Number"

<< std::setw(15) << "Repair Cost"

<< std::setw(17) << "Acceptance Date"

<< std::setw(17) << "Issue Date"

<< std::setw(10) << "Status" << std::endl;

// Вывод разделительной строки

std::cout << std::setfill('-') << std::setw(150) << "-" << std::endl;

std::cout << std::setfill(' ');

PrintOrdersPrivate(root, countptr);

}

void BinaryTreeOrder::PrintRepairingOrders()

{

int count = 1;

int\* countptr = &count;

std::cout << std::left << std::setw(3) << "#" << std::setw(30) << "Product Name"

<< std::setw(30) << "Product Brand"

<< std::setw(30) << "Owner Full Name"

<< std::setw(20) << "Phone Number"

<< std::setw(15) << "Repair Cost"

<< std::setw(17) << "Acceptance Date"

<< std::setw(17) << "Issue Date"

<< std::setw(10) << "Status" << std::endl;

// Вывод разделительной строки

std::cout << std::setfill('-') << std::setw(150) << "-" << std::endl;

std::cout << std::setfill(' ');

PrintRepairingOrdersPrivate(root, countptr);

}

void BinaryTreeOrder::SearchOrdersByProductName(const std::string& \_product\_name)

{

int count = 1;

int\* countptr = &count;

std::cout << std::left << std::setw(3) << "#" << std::setw(30) << "Product Name"

<< std::setw(30) << "Product Brand"

<< std::setw(30) << "Owner Full Name"

<< std::setw(20) << "Phone Number"

<< std::setw(15) << "Repair Cost"

<< std::setw(17) << "Acceptance Date"

<< std::setw(17) << "Issue Date"

<< std::setw(10) << "Status" << std::endl;

// Вывод разделительной строки

std::cout << std::setfill('-') << std::setw(150) << "-" << std::endl;

std::cout << std::setfill(' ');

SearchOrdersByProductNamePrivate(root, \_product\_name, countptr);

if (count == 1)

{

system("cls");

std::cout << "No such orders found" << std::endl;

}

}

void BinaryTreeOrder::SaveOrdersToFile(const std::string& filename)

{

std::ofstream outFile(filename, std::ios::binary | std::ios::out);

if (outFile.is\_open())

{

SaveOrdersToFilePrivate(root, &outFile);

outFile.close();

std::cout << "Data has been written to the file." << std::endl;

}

else {

std::cout << "Failed to open the file for writing." << std::endl;

}

}

void BinaryTreeOrder::DeleteOrderTree()

{

DeleteOrderTreePrivate(root);

root = nullptr;

}

void BinaryTreeOrder::DeleteOrderNode(int\* count\_to\_find)

{

int count = 1;

int\* countptr = &count;

root = DeleteOrderNodePrivate(root, count\_to\_find, countptr);

}

void BinaryTreeOrder::EditOrderNode(int\* count\_to\_find)

{

int count = 1;

int\* countptr = &count;

EditOrderNodePrivate(root, count\_to\_find, countptr);

if (\*count\_to\_find == \*countptr)

{

std::cout << "No such object" << std::endl;

}

}

bool compareDates(const std::string& date1, const std::string& date2);

TreeNodeOrder\* BinaryTreeOrder::InsertOrder(TreeNodeOrder\* node, Order\* order)

{

if (node == nullptr) {

return new TreeNodeOrder(order);

}

else

{

if (compareDates(order->GetAcceptanceDate(), node->data->GetAcceptanceDate()))

{

node->left = InsertOrder(node->left, order);

}

else

{

node->right = InsertOrder(node->right, order);

}

}

return node;

}

bool compareDates(const std::string& date1, const std::string& date2)

{

try

{

int day1 = std::stoi(date1.substr(0, 2));

int month1 = std::stoi(date1.substr(3, 2));

int year1 = std::stoi(date1.substr(6, 4));

int day2 = std::stoi(date2.substr(0, 2));

int month2 = std::stoi(date2.substr(3, 2));

int year2 = std::stoi(date2.substr(6, 4));

if (year1 != year2) {

return year1 < year2;

}

else if (month1 != month2) {

return month1 < month2;

}

else {

return day1 < day2;

}

}

catch (const std::invalid\_argument& e)

{

std::cerr << "Error: Invalid date format." << std::endl;

}

}

void BinaryTreeOrder::PrintOrdersPrivate(TreeNodeOrder\* node, int\* count)

{

if (node != nullptr)

{

PrintOrdersPrivate(node->right, count);

std::cout << std::left << std::setw(3) << (\*count)++ << \*(node->data) << std::endl;

PrintOrdersPrivate(node->left, count);

}

}

void BinaryTreeOrder::PrintRepairingOrdersPrivate(TreeNodeOrder\* node, int\* count)

{

if (node != nullptr)

{

PrintRepairingOrdersPrivate(node->right, count);

if (!node->data->GetStatus())

{

std::cout << std::left << std::setw(3) << (\*count)++ << \*(node->data) << std::endl;

}

PrintRepairingOrdersPrivate(node->left, count);

}

}

void BinaryTreeOrder::SearchOrdersByProductNamePrivate(TreeNodeOrder\* node, const std::string& \_product\_name, int\* count)

{

if (node != nullptr)

{

SearchOrdersByProductNamePrivate(node->right, \_product\_name, count);

if (node->data->GetProductName().find(\_product\_name) != std::string::npos)

{

std::cout << std::left << std::setw(3) << (\*count)++ << \*(node->data) << std::endl;

}

SearchOrdersByProductNamePrivate(node->left, \_product\_name, count);

}

}

void BinaryTreeOrder::DeleteOrderTreePrivate(TreeNodeOrder\* node)

{

if (node == nullptr)

{

return;

}

DeleteOrderTreePrivate(node->left);

DeleteOrderTreePrivate(node->right);

delete node;

}

TreeNodeOrder\* BinaryTreeOrder::DeleteOrderNodePrivate(TreeNodeOrder\* node, int\* count\_to\_find, int\* count)

{

if (node != nullptr)

{

node->right = DeleteOrderNodePrivate(node->right, count\_to\_find, count);

if (\*count\_to\_find == \*count)

{

if (node->left == nullptr && node->right == nullptr) {

(\*count)++;

delete node;

return nullptr;

}

else if (node->right == nullptr) {

TreeNodeOrder\* temp = node->left;

(\*count)++;

delete node;

return temp;

}

else if (node->left == nullptr) {

TreeNodeOrder\* temp = node->right;

(\*count)++;

delete node;

return temp;

}

else {

TreeNodeOrder\* minRightNode = FindMinNode(node->right);

\*(node->data) = \*(minRightNode->data);

node->right = DeleteOrderNodePrivate(node->right, count\_to\_find, count);

(\*count)++;

}

}

else

{

(\*count)++;

node->left = DeleteOrderNodePrivate(node->left, count\_to\_find, count);

}

}

else

{

return nullptr;

}

return node;

}

TreeNodeOrder\* BinaryTreeOrder::FindMinNode(TreeNodeOrder\* node)

{

if (node == nullptr) {

return nullptr;

}

while (node->left != nullptr) {

node = node->left;

}

return node;

}

void BinaryTreeOrder::EditOrderNodePrivate(TreeNodeOrder\* node, int\* count\_to\_find, int\* count)

{

if (node != nullptr)

{

EditOrderNodePrivate(node->right, count\_to\_find, count);

if (\*count\_to\_find == \*count)

{

(\*count)++; // во избежании повторного выполнения условия

int choice;

std::string s;

double d;

bool b;

do

{

system("cls");

std::cout << \*(node)->data << std::endl;

std::cout << "What do you want to change ?" << std::endl;

std::cout << "1. Product Name" << std::endl;

std::cout << "2. Product Brand" << std::endl;

std::cout << "3. Owner Full Name" << std::endl;

std::cout << "4. Owner Phone Number" << std::endl;

std::cout << "5. Repair Cost" << std::endl;

std::cout << "6. Acceptance Date" << std::endl;

std::cout << "7. Issue Date" << std::endl;

std::cout << "8. Status" << std::endl;

std::cout << "0. Exit" << std::endl;

std::cout << "Enter your choice: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> choice;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice;

}

switch (choice) {

case 1:

system("cls");

std::cout << "Enter a new product name: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::getline(std::cin, s);

node->data->SetProductName(s);

break;

case 2:

system("cls");

std::cout << "Enter a new product brand: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::getline(std::cin, s);

node->data->SetProductBrand(s);

break;

case 3:

system("cls");

std::cout << "Enter a new owner full name: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::getline(std::cin, s);

node->data->SetOwnerFullName(s);

break;

case 4:

system("cls");

std::cout << "Enter a new owner phone number: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::getline(std::cin, s);

node->data->SetOwnerPhoneNumber(s);

break;

case 5:

system("cls");

std::cout << "Enter a new repair cost: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> d;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> d;

}

node->data->SetRepairCost(d);

break;

case 6:

system("cls");

std::cout << "Enter a new acceptance date (please, use the dd.mm.yyyy format): ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::getline(std::cin, s);

node->data->SetAcceptanceDate(s);

system("cls");

break;

case 7:

system("cls");

std::cout << "Enter a new issue date (please, use the dd.mm.yyyy format): ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::getline(std::cin, s);

node->data->SetIssueDate(s);

system("cls");

break;

case 8:

system("cls");

std::cout << "Enter a new status: ";

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

std::cin >> b;

node->data->SetStatus(b);

std::cin.clear();

system("cls");

break;

case 0:

system("cls");

std::cout << "Are you sure ?" << std::endl;

std::cout << "1. Yes" << std::endl;

std::cout << "2. No" << std::endl;

int choice3;

std::cin.ignore(std::cin.rdbuf()->in\_avail());

std::cin.clear();

do

{

std::cin >> choice3;

while (cin.fail())

{

cin.clear();

while (cin.get() != '\n');

std::cout << "Invalid value. Try again" << std::endl;

cin >> choice3;

}

switch (choice3)

{

case 1:

choice3 = 0;

system("cls");

std::cout << "Exiting..." << std::endl;

break;

case 2:

choice = 1;

choice3 = 0;

break;

default:

choice3 = 1;

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice3 != 0);

break;

default:

std::cout << "Invalid choice. Please try again." << std::endl;

}

} while (choice != 0);

}

else

{

(\*count)++;

EditOrderNodePrivate(node->left, count\_to\_find, count);

}

}

else

{

return;

}

return;

}

void BinaryTreeOrder::SaveOrdersToFilePrivate(TreeNodeOrder\* node, std::ofstream\* outFile)

{

if (node != nullptr)

{

// Запись \_product\_name

int product\_name\_size = (node->data)->GetProductName().size();

outFile->write(reinterpret\_cast<char\*>(&product\_name\_size), sizeof(int));

outFile->write(node->data->GetProductName().c\_str(), product\_name\_size);

// Запись \_product\_brand

int product\_brand\_size = (node->data)->GetProductBrand().size();

outFile->write(reinterpret\_cast<char\*>(&product\_brand\_size), sizeof(int));

outFile->write(node->data->GetProductBrand().c\_str(), product\_brand\_size);

// Запись \_owner\_full\_name

int owner\_full\_name\_size = node->data->GetOwnerFullName().size();

outFile->write(reinterpret\_cast<char\*>(&owner\_full\_name\_size), sizeof(int));

outFile->write(node->data->GetOwnerFullName().c\_str(), owner\_full\_name\_size);

// Запись \_owner\_phone\_number

int owner\_phone\_number\_size = node->data->GetOwnerPhoneNumber().size();

outFile->write(reinterpret\_cast<char\*>(&owner\_phone\_number\_size), sizeof(int));

outFile->write(node->data->GetOwnerPhoneNumber().c\_str(), owner\_phone\_number\_size);

// Запись \_repair\_cost

double temp = node->data->GetRepairCost();

outFile->write(reinterpret\_cast<char\*>(&temp), sizeof(double));

// Запись \_acceptance\_date

int acceptance\_date\_size = node->data->GetAcceptanceDate().size();

outFile->write(reinterpret\_cast<char\*>(&acceptance\_date\_size), sizeof(int));

outFile->write(node->data->GetAcceptanceDate().c\_str(), acceptance\_date\_size);

// Запись \_issue\_date

int issue\_date\_size = node->data->GetIssueDate().size();

outFile->write(reinterpret\_cast<char\*>(&issue\_date\_size), sizeof(int));

outFile->write(node->data->GetIssueDate().c\_str(), issue\_date\_size);

// Запись \_status

bool temp2 = node->data->GetStatus();

outFile->write(reinterpret\_cast<char\*>(&temp2), sizeof(bool));

SaveOrdersToFilePrivate(node->left, outFile);

SaveOrdersToFilePrivate(node->right, outFile);

}

}

void BinaryTreeOrder::LoadOrdersFromFile(const std::string& filename)

{

std::ifstream inFile(filename, std::ios::binary | std::ios::in);

if (inFile.is\_open())

{

while (inFile.good())

{

Order\* order = new Order();

// Чтение \_product\_name

int buf\_size = 0;

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

std::string buf;

buf.clear();

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

order->SetProductName(buf);

if (buf.empty() || buf\_size == 1)

{

inFile.close();

break;

}

// Read \_product\_brand

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

order->SetProductBrand(buf);

// Read \_owner\_full\_name

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

order->SetOwnerFullName(buf);

// Read \_owner\_phone\_number

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

order->SetOwnerPhoneNumber(buf);

// Read \_repair\_cost

double temp\_double;

inFile.read(reinterpret\_cast<char\*>(&temp\_double), sizeof(double));

order->SetRepairCost(temp\_double);

// Read \_acceptance\_date

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

order->SetAcceptanceDate(buf);

// Read \_issue\_date

inFile.read(reinterpret\_cast<char\*>(&buf\_size), sizeof(int));

buf.resize(buf\_size);

inFile.read(&buf[0], buf\_size);

order->SetIssueDate(buf);

// Read \_issue\_date

bool temp\_bool;

inFile.read(reinterpret\_cast<char\*>(&temp\_bool), sizeof(bool));

order->SetStatus(temp\_bool);

this->Insert(order);

}

inFile.close();

}

else {

std::cout << "Failed to open the file for reading." << std::endl;

}

}

**Файл main.cpp**

#include "binary\_tree.h"

#include <conio.h>

#include <Windows.h>

#include "SHA256.h"

#include "LoginScreen.h"

int main()

{

LoginScreen ls;

ls.Authorization();

return 0;

}