**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. КОСЫГИНА**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ЦИФРОВОЙ ТРАНСФОРМАЦИИ**

**Кафедра Искусственного интеллекта, прикладной математики и программирования**

КУРСОВАЯ РАБОТА

по дисциплине: «Языки и методы программирования»

Выполнил:

студент 2 курса, гр. МПМ-122

Поломин С.В.

Руководитель:

Романенков А.М.

Москва, 2024 г.

Содержание

[Введение 3](#_Toc168801788)

[Основная часть 4](#_Toc168801789)

[Задание 3 4](#_Toc168801790)

[Задание 4 7](#_Toc168801791)

[Задание 5 10](#_Toc168801792)

[Задание 11 15](#_Toc168801793)

[Задание 12 19](#_Toc168801794)

[Заключение 24](#_Toc168801795)

[Список использованных источников 25](#_Toc168801796)

[Приложение 1 26](#_Toc168801797)

# Введение

Курс «Языки и методы программирования» ориентирован на улучшение умений в области кодирования. В его контексте была поставлена задача создания серии программ согласно предоставленным спецификациям на языке программирования C++. Также были определены специфические требования к выполнению этих заданий.

Необходимо эффективно управлять и обрабатывать все возможные исключения. Использование функций для принудительного завершения программы на любом этапе их выполнения запрещено. Важно обеспечить разделение логики обработки и хранения данных от их вывода. Данные, вводимые через консоль, должны проходить проверку соответствия ожидаемому типу. Также критически важно следить за возможными проблемами с выделением памяти и гарантировать правильное освобождение всей задействованной динамической памяти. Все компоненты программы должны зависеть от абстракции, а не от конкретной реализации.

Основная цель проекта - не просто проверка уровня теоретического понимания, а также стимуляция углубления практических умений в кодировании, развитие критического мышления и способности к решению комплексных проблем. Достижение целей проекта способствует повышению знаний в сфере C++ и эффективно готовит обучающихся к предстоящим профессиональным задачам в области программной инженерии и разработке ПО.

# Основная часть

# Задание 3

Задача номер 3 заключается в разработке системы усиления игровых персонажей для некоторой MMORPG. В результате выполнения задания были реализованы системы нанесения урона, расчёта характеристик персонажа, экипировки и ее случайной генерации.

Для выполнения задания были использованы средства языка программирования С++ и хранящиеся в них структуры данных, представленные в таблице 4.

*Таблица 4. Описание использованных средств языка.*

|  |  |  |
| --- | --- | --- |
| № | Средство | Назначение |
|  | iostream | Управление операциями ввода и вывода |
|  | vector | Вводит класс динамического массива std::vector для хранения и управления динамическими массивами |
|  | string | Обработка и манипуляции со строками |
|  | random | Предоставляет различные генераторы случайных чисел, распределения и другие утилиты для работы со случайными числами |

Для начала была определена структура Attributes, которая содержит поля для хранения различных характеристик, таких как здоровье, броня, интеллект, сила, ловкость, меткость, удача и навык. Она также имеет конструктор по умолчанию для инициализации всех атрибутов и метод printInfo(), который выводит информацию об атрибутах в стандартный вывод.

Далее был реализован класс Equipment, который содержит имя и атрибуты для представления игровой экипировки. У него есть конструктор по умолчанию и конструктор с параметрами для инициализации имени и атрибутов. Класс также предоставляет геттеры и сеттеры для доступа к имени и атрибутам, а также метод printInfo(), который выводит информацию об экипировке в стандартный вывод.

После чего были описаны базовый класс персонажа и классы определённых персонажей, таких как Защитник, Боец ближнего боя и боец дальнего боя. Листинг описания базового класса и класса бойца ближнего боя представлен на следующей странице.

*Листнг 5. Описание классов персонажей.*

class Character {

public:

std::string name;

std::string characterClass;

Attributes baseAttributes;

std::vector<std::shared\_ptr<Equipment>> equipment;

// Чисто виртуальная функция для расчета урона

virtual void calculateDamage() = 0;

void addEquipment(std::shared\_ptr<Equipment> eq) {

equipment.push\_back(eq);

}

// Подсчет общих характеристик с учетом экипировки

Attributes getTotalAttributes() {

Attributes total = baseAttributes;

for (auto& eq : equipment) {

total.health += eq->attributes.health;

total.armor += eq->attributes.armor;

total.intelligence += eq->attributes.intelligence;

total.strength += eq->attributes.strength;

total.agility += eq->attributes.agility;

total.accuracy += eq->attributes.accuracy;

total.luck += eq->attributes.luck;

total.skill += eq->attributes.skill;

}

return total;

};

};

class MeleeFighter : public Character {

public:

MeleeFighter(const std::string& name) {

this->name = name;

this->characterClass = "Melee Fighter";

// Устанавливаем базовые атрибуты для бойца ближнего боя

this->baseAttributes = Attributes(80, 30, 10, 25, 20, 10, 5, 10);

}

void calculateDamage() override {

// Получаем общие атрибуты с учетом экипировки

Attributes totalAttributes = getTotalAttributes();

// Урон -- комбинация силы и ловкости

int damage = totalAttributes.strength\*2 + totalAttributes.agility;

damage += damage \* totalAttributes.accuracy / 100;

damage += damage \* totalAttributes.skill / 100;

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(1, 100);

if (dis(gen) <= totalAttributes.luck) {

damage \*= 2; // Критический удар удваивает урон

}

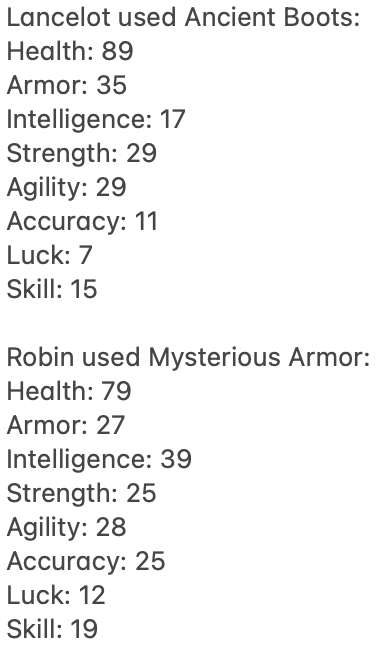
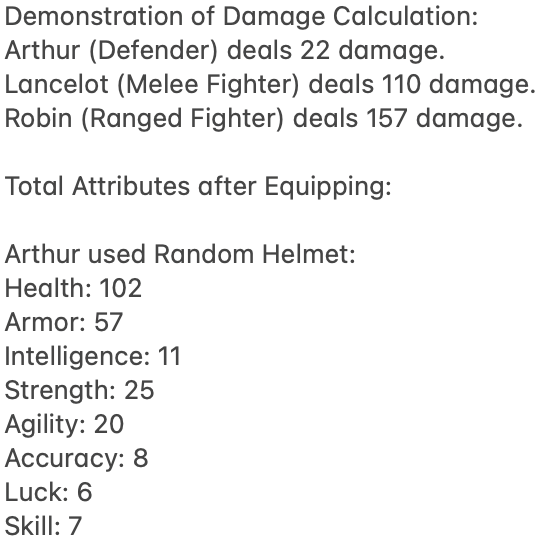
std::cout << this->name << " (" << this->characterClass << ") deals " << damage << " damage." << std::endl;

}

};

Для каждого класса персонажей реализована функция подсчёта урона calculateDamage, которая считает урон наносимый персонажем в зависимости от его начальных характеристик и дополнительных атрибутов и навыков, усиливающих каждый из классов. Также у бойцов ближнего и дальнего боя реализована механика нанесения критического урона относительно их атрибута luck.

Результат выполнения программы представлен на рисунке 10.



*Рисунок 10. Результат подсчета харрактеристик персонажей.*

Требуемый результат задачи достигнут, был разработан и реализован алгоритм подсчета усиления игровых персонажей для MMORPG. Детали реализации находятся в приложении 1.

# Задание 4

Задача номер 4 заключается в разработке программного обеспечения которое осуществляет подмену слов в тексте, используя для этого словарь синонимов. Для выполнения задания были использованы средства языка программирования С++ и хранящиеся в них структуры данных, представленные в таблице 2.

Таблица 2. Описание использованных средств языка

|  |  |  |
| --- | --- | --- |
| № | Средство | Назначение |
|  | iostream | Управление операциями ввода и вывода |
|  | stdexcept | Содержит стандартные классы исключений, которые можно использовать для обработки ошибок. |
|  | string | Обработка и манипуляции со строками |
|  | unordered\_set | Хранит уникальные элементы в неупорядоченном виде |
|  | sstream | Предоставляет классы для работы со строковыми потоками |
|  | fstream | Чтение и запись данных в файлы |
|  | vector | Вводит класс динамического массива std::vector для хранения и управления динамическими массивами |
|  | unordered\_map | Предоставляет std::unordered\_map, неупорядоченную ассоциативную коллекцию пар ключ-значение. Основан на хеш-таблицах |
|  | stack | Структура данных "стек" |

Программа предназначена для работы с текстовыми файлами и обладает следующим функционалом:

Замена слов на стандартные термины: В каждом текстовом документе найденные слова из словаря синонимов подменяются на соответствующие стандартные термины.

Режим автоматической обработки: слова, отсутствующие в словаре синонимов, остаются без изменений. Все оригинальные слова сохраняются как есть.

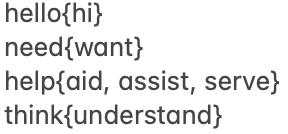
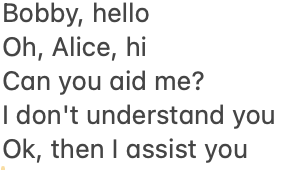
Режим обучения: для слов, не найденных в словаре синонимов, пользователю предлагается несколько действий: добавить как синоним к существующему стандартному термину, создать новый стандартный термин или произвести замену на уже существующий.

Откат последовательности операций: функция позволяет отменить последние N изменений, где N — это число, указывающее на количество последних операций, подлежащих отмене.

Стабильность работы: программа должна функционировать без сбоев в любых условиях.

Сохранение словаря: словарь синонимов должен сохраняться после завершения работы и восстанавливаться при следующем запуске программы.

При старте программа предлагает пользователю выбрать одно из действий: активировать режим автоматической обработки, начать ручной режим, дополнить словарь новым синонимом, внести каноническое слово с синонимами, отменить недавние изменения, сохранить результаты и завершить работу. На входе у программы два файла: "input.txt", который содержит текст (см. рисунок 3), и "synonyms.txt", в котором перечислены синонимы (см. рисунок 4).



*Рис. 3. Входные данные. Рис 4. Словарь синонимов*

Программа инициирует обработку текста в соответствии с выбором пользователя. Для этого создается объект класса TextProcessor, который занимается анализом текста, а также используется набор для хранения слов, отсутствующих в словаре синонимов. В случае если режим автоматической обработки выключен и обнаружены неизвестные слова, программа уведомляет пользователя о них. Обработка файла осуществляется построчно: программа считывает слова, ищет для каждого из них соответствующее каноническое слово в словаре, и в случае, если слово отсутствует в словаре и включен ручной режим, предлагает пользователю добавить его. В автоматическом режиме программа автоматически подменяет синонимы на канонические слова, если таковые имеются в словаре. В программе также присутствует класс UndoManager, который отвечает за функцию отмены операций. Он хранит в стеке функции, которые необходимы для реализации отмены операций. Когда возникает необходимость в отмене одной или нескольких операций, соответствующие функции извлекаются из стека и выполняются (см. листинг 3).

*Листнг 3. Класс UndoManager*

class UndoManager {

private:

// Стек для хранения функций, которые будут вызываться для отмены действий

std::stack<std::function<void()>> undo\_stack;

public:

// Метод для добавления действия отмены в стек

void addUndoAction(const std::function<void()>& action) {

undo\_stack.push(action);

}

// Метод для отмены последних N действий

void undoLastActions(int N) {

// Циклично выполняем и удаляем с верхушки стека, пока не достигнем N или стек не станет пустым

for (int i = 0; i < N && !undo\_stack.empty(); ++i) {

// Выполнение фунции отмены

undo\_stack.top()();

// Удаление выполненной функции отмены из стека

undo\_stack.pop();

}

}

// Метод для проверки, пуст ли стек

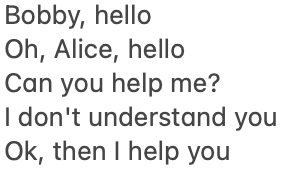
bool isEmpty() const {

return undo\_stack.empty();

}

};

Результат работы программы представлен на рисунке 5.

**

*Рис. 5. Результат выполнения.*

Требуемый результат задачи достигнут, был разработан метод, осуществляющий замену слов в тексте на синонимы. Детали реализации находятся в приложении 1.

# Задание 5

Задача номер 5 заключается в разработке программы для анализа текстового файла, в котором фиксируются данные о работе персонала определённой компании. Приложение должно читать данные из текстового файла, формировать ассоциативный контейнер с записями, где ключевым набором полей является ФИО сотрудников. Для этого был создан комплекс функций, покрывающий все необходимые аспекты функциональности. Для выполнения задания были использованы средства языка программирования С++ и хранящиеся в них структуры данных, представленные в таблице 3.

*Таблица 3. Описание использованных средств языка.*

|  |  |  |
| --- | --- | --- |
| № | Средство | Назначение |
|  | iostream | Управление операциями ввода и вывода |
|  | fstream | Чтение и запись данных в файлы |
|  | string | Обработка и манипуляции со строками |
|  | map | Для хранения списка сотрудников с доступом к их информации по ключам |
|  | sstream | Предоставляет классы для работы со строковыми потоками |
|  | vector | Вводит класс динамического массива std::vector для хранения и управления динамическими массивами |
|  | algorithm | Алгоритмы для работы с контейнерами в C++ |
|  | limits | Информация о лимитах различных типов данных в языке C++ |
|  | ctime | Функции и типы данных для работы со временем и датами в языке C++ |

Для учёта деятельности сотрудников используется словарь с ключом "ФИО". Для каждого сотрудника сохраняется список контрактов, каждый из которых содержит следующие поля:

- Номер контракта;

- Дата начала;

- Дата окончания;

- Количество отработанных часов;

- Стоимость оплаты.

Файл обрабатывается построчно. При обнаружении левой фигурной скобки программа сохраняет предыдущие данные в качестве имени пользователя и затем анализирует строки, содержащие информацию о договорах.

Для извлечения и структурирования информации о сотрудниках и их контрактах из текстового файла, записанного в определенном формате, используется функция loadDeals. В ней выполняется разбиение текста по следующим пробельным разделителям:

* значение, стоящее после слова «договор», сохраняется как номер;
* значение, стоящее после слова «нач.», сохраняется как время начала исполнения;
* значение, стоящее после слова «кон.», сохраняется как время завершения исполнения;
* значение, стоящее после слова «работа», сохраняется как количество рабочих часов;

- значение, стоящее после слова «стоимость», сохраняется как размер оплаты.

Эта функция позволяет загрузить данные из файла, создавая объекты сотрудников и добавляя информацию о соответствующих контрактах. Пустые строки пропускаются. Если строка заканчивается на «}», создается новый объект Employee (листинг 4), его имя инициализируется, и затем объект сохраняется в map employees. Если строка содержит «}», текущий сотрудник завершается. В противном случае строка считается описанием контракта для текущего сотрудника; она парсится с использованием istringstream для проверки корректности формата строки контракта.

*Листинг 4. Employee.*

struct Employee {

std::string fullname;

std::vector<Contract> contracts;

Employee(std::string fn) : fullname(fn) {}

double total\_cost() {

double sum = 0;

for (const auto& contract : contracts) {

sum += contract.cost;

}

return sum;

}

Contract\* most\_expensive\_contract() {

if (contracts.empty()) return nullptr;

return &\*std::max\_element(contracts.begin(), contracts.end(), [&](const Contract& a, const Contract& b) {

return a.cost < b.cost;

});

}

Contract\* longest\_contract() {

if (contracts.empty()) return nullptr; // Если нет контрактов, возвращаем nullptr

// Лямбда-функция для парсинга даты из строки

auto parse\_date = [](const std::string& date) -> std::tm {

std::tm tm = {}; // Инициализируем структуру std::tm нулями

std::istringstream ss(date);

if (date != "нв") ss >> std::get\_time(&tm, "%d.%m.%Y"); // Парсим дату, если она не равна "нв" (настоящее время)

return tm; // Возвращаем структуру std::tm

};

return &\*std::max\_element(contracts.begin(), contracts.end(), [&](const Contract& a, const Contract& b) {

std::tm start\_a = parse\_date(a.start\_date);

std::tm end\_a = a.end\_date == "нв" ? std::tm() : parse\_date(a.end\_date);

std::tm start\_b = parse\_date(b.start\_date);

std::tm end\_b = b.end\_date == "нв" ? std::tm() : parse\_date(b.end\_date);

auto duration\_a = mktime(&end\_a) - mktime(&start\_a);

auto duration\_b = mktime(&end\_b) - mktime(&start\_b);

return duration\_a < duration\_b;

});

}

};

При запуске программы она предлагает пользователю выбрать одно из действий:

- Отобразить общую стоимость всех контрактов для сотрудника.

- Отобразить все контракты для сотрудника.

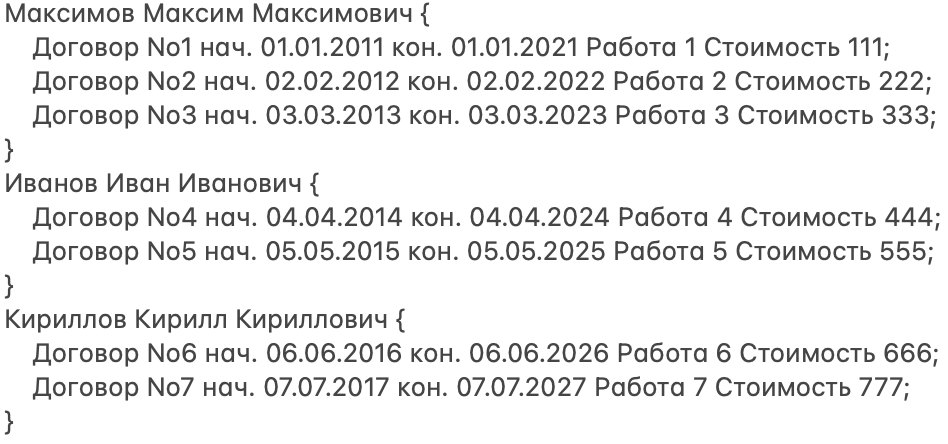
- Найти самый длительный контракт для сотрудника.

- Найти самый дорогой контракт для сотрудника.

- Удалить запись о сотруднике.

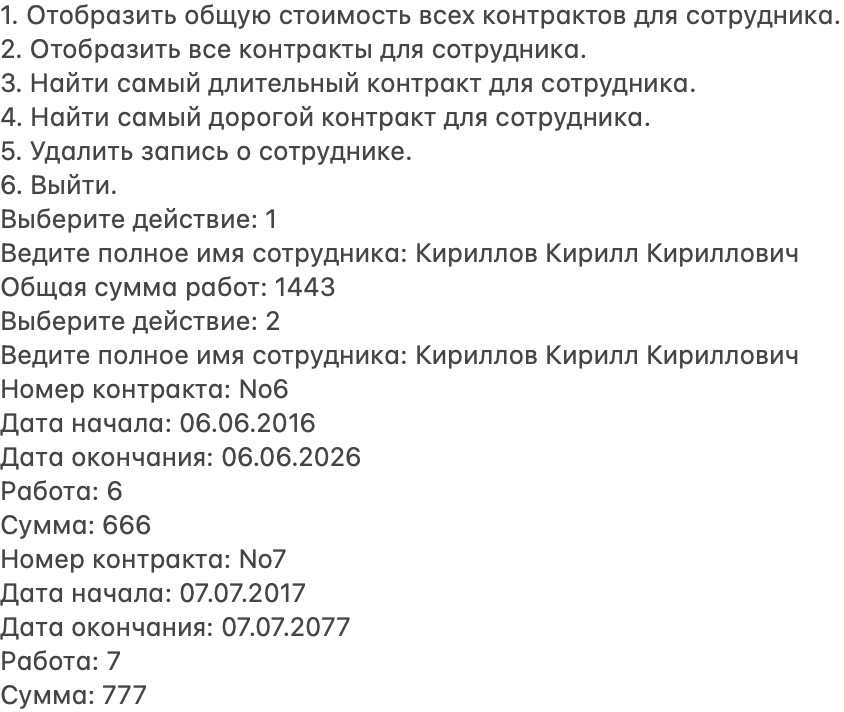
- Завершить выполнение.

Данные входного файла представлены на рисунке 6.

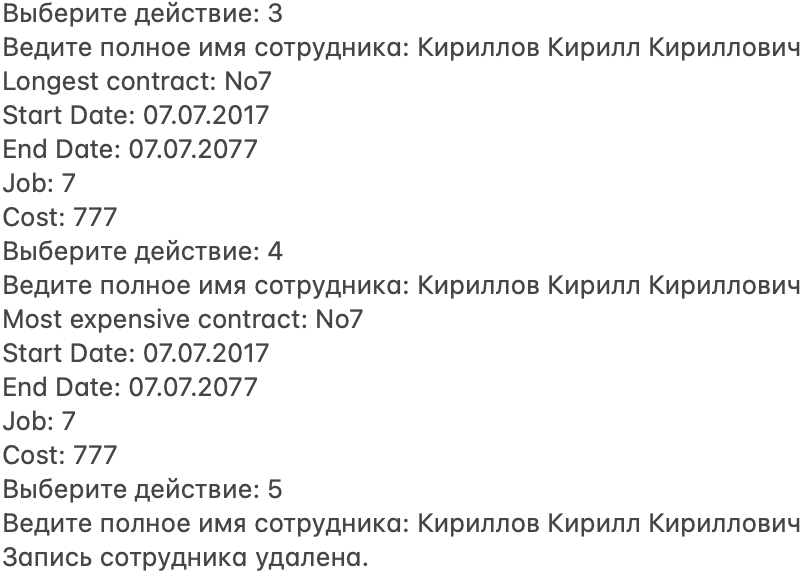
**

*Рисунок 6. Входные данные.*

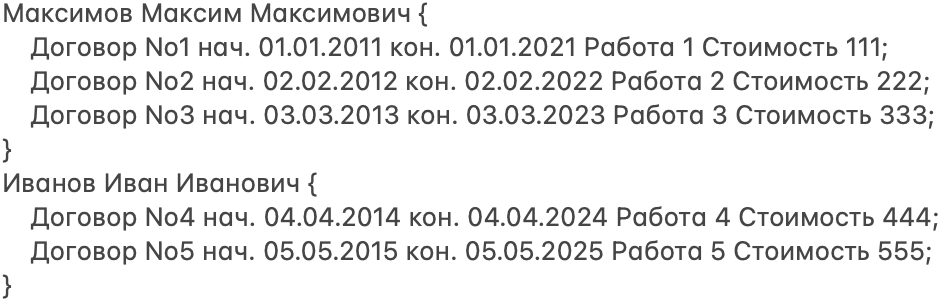
Демонстрация результатов выполнения различных действий алгоритма представлена ниже:



*Рисунок 7. Результат выполнения (1,2)*



*Рисунок 8. Результат выполнения (3, 4, 5)*

**

*Рисунок 9. Входной файл после удаления.*

Требуемый результат задачи достигнут, был разработан и реализован алгоритм работы с сотрудниками и их контрактами. Детали реализации находятся в приложении 1.

# Задание 11

Задача номер 11 заключается в разработке программы создание приложения, которое позволит проводить лотерею. В приложении должен быть реализован функционал генерации билетов, моделирования розыгрыша, проверки победителей, системы выигрышей, поиска билетов по различным критериям и сохранения информации о проведенных тиражах.

Для выполнения задания были использованы средства языка программирования С++ и хранящиеся в них структуры данных, представленные в таблице 5.

*Таблица 5. Описание использованных средств языка.*

|  |  |  |
| --- | --- | --- |
| № | Средство | Назначение |
|  | iostream | Управление операциями ввода и вывода |
|  | vector | Вводит класс динамического массива std::vector для хранения и управления динамическими массивами |
|  | set | Предоставляет класс, который реализует структуру данных "множество". Множество хранит уникальные элементы в отсортированном порядке |
|  | random | Предоставляет различные генераторы случайных чисел, распределения и другие утилиты для работы со случайными числами |
|  | forward\_list | Односвязный список |
|  | cstdarg | Макросы и функции для работы с переменным числом аргументов функций |

В первую очередь задаются основные параметры: общий объем и число реализованных билетов. Далее определяется тип контейнера для размещения билетов и фиксации результатов. При инициализации, программа предлагает выбрать режим работы из следующих вариантов: с использованием стека, с применением std::forward\_list, с организацией очереди на базе односвязного списка или через std::vector.

std::vector является динамическим массивом, обеспечивающим быстрый доступ к элементам по индексу и автоматическое управление памятью. Преимущества включают индексный доступ, возможность динамического расширения, совместимость со стандартными алгоритмами STL и удобство взаимодействия с алгоритмами STL.

std::forward\_list представляет собой односвязный список с прямым доступом, выделяющийся более высокой эффективностью операций вставки и удаления по сравнению с массивами, хотя и с определенными оговорками. К его преимуществам относятся быстрая вставка и удаление (время выполнения O(1) при наличии итератора на позицию), а также меньшие затраты памяти по сравнению с двусвязными списками.

Стек представляет структуру данных типа LIFO (Last In, First Out), где новейший элемент удаляется первым. Преимуществами являются легкость в реализации (может основываться на односвязном списке или массиве) и операции высокой эффективности — вставка и удаление элементов производятся за время O(1).

Очередь — это структура данных с принципом FIFO (First In, First Out), где элемент, добавленный первым, будет убран первым. Преимущества включают простоту реализации (может базироваться на двусвязном списке или кольцевом буфере) и эффективные операции — вставка и удаление элементов занимают время O(1).

После реализации вышеприведенных структур данных был описан класс ticket для хранения и использования информации о билете и класс results для хранения информации о лотерее и ее результатах.

Далее следует реализация функции ticket\_generation для генерации билетов.

Параметры функции:

g: Генератор случайных чисел std::mt19937, используемый для обеспечения случайности при выборе номеров билетов.

tickets\_amount: Общее количество билетов, которое должно быть сгенерировано.

tickets: Контейнер (например, очередь), в который будут помещаться сгенерированные билеты.

prizes\_types: Количество различных типов призов.

... : Переменное количество аргументов, где каждая пара аргументов представляет собой количество билетов данного типа приза и соответствующую призовую сумму.

Векторы prizes\_value и amount\_prize\_type используются для записи размеров призов и количества билетов по каждому типу приза соответственно. Информация о числе билетов для различных видов призов и соответствующих денежных наградах извлекается из списка передаваемых аргументов функции.

Далее подсчитывается общее количество выигрывающих билетов, и сохраняется в переменную amount\_of\_prize\_tickets.

Генерируется вектор sh\_numbers, содержащий последовательные номера билетов от 1 до tikets\_amount. Рассчитывается общее количество билетов с призами.

Создается множество numbersContain для хранения уникальных номеров билетов с призами. С помощью генератора случайных чисел и распределения выбираются случайные номера билетов до тех пор, пока множество не заполнится уникальными номерами билетов с призами (см. листинг 6).

Перебор всех билетов: для каждого номера билета проверяется, содержится ли он в множестве номеров с призами. Если да, то создается билет с призовой суммой из prizes\_value и добавляется в контейнер tickets. В противном случае создается билет без приза (с призовой суммой 0) и также добавляется в контейнер. Счетчик counter используется для отслеживания индекса в векторе prizes\_value.

*Листинг 6. Генерация и сохранение выигрывающих билетов.*

std::set<int> numbersContain;

while (numbersContain.size() < amount\_of\_prize\_tickets)

{

std::uniform\_int\_distribution<int> distribution(1, tikets\_amount);

int random = distribution(g);

numbersContain.insert(random);

}

int counter = 0;

for (auto i = 1; i <= tikets\_amount; ++i)

{

auto it = numbersContain.find(sh\_numbers[i-1]);

if (it != numbersContain.end())

{

ticket ticket(sh\_numbers[i-1], prizes\_value[counter]);

tickets.push(ticket);

counter++;

}

else

{

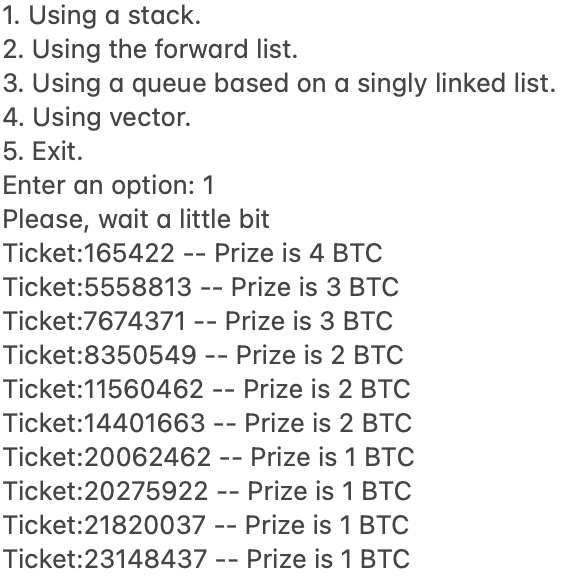
ticket ticket(sh\_numbers[i-1], 0);

tickets.push(ticket);

}

}

Для демонстрации и удобства работы программы был описан визуальный пользовательский интерфейс. Каждую структуру данных можно вызывать из командной строки. Результат работы программы представлен на рисунке 11.



*Рисунок 11. Результат выполнения программы.*

Требуемый результат задачи достигнут, был разработан и реализован алгоритм проведения розыгрыша билетов. Детали реализации представлены в приложении 1.

# Задание 12

Задача номер 12 заключается в разработке программного обеспечения для работы с сообщениями пользователей из соцсети. В ходе выполнения были успешно реализованы все функции интерпретатора, указанные в условиях задания.

Для выполнения задания были использованы средства языка программирования С++ и хранящиеся в них структуры данных, представленные в таблице 1.

*Таблица 1. Описание использованных средств языка*

|  |  |  |
| --- | --- | --- |
| № | Название инструментария | Описание |
| 1 | iostream | Для ввода и вывода данных |
| 2 | fstream | Для работы с файлами |
| 3 | sstream | Предоставляет классы для работы со строковыми потоками |
| 4 | string | Предоставляет класс строк. Включает в себя множество функций для удобной работы со строковыми данными |
| 5 | set | Предоставляет класс, который реализует структуру данных "множество". Множество хранит уникальные элементы в отсортированном порядке |
| 6 | optional | Представляет собой контейнер для значений, которые могут либо присутствовать, либо отсутствовать. Это полезно для функций, которые могут не возвращать значение при определённых условиях |
| 7 | exception | Предоставляет базовый класс исключений, от которого могут быть унаследованы пользовательские исключения. Содержит в себе стандартные классы исключений, которые можно использовать для сигнализирования об ошибках во время выполнения программы |

Сохранение сообщений осуществляется в списке messages, а каждое отдельное сообщение хранится в структуре данных Message, описанной в листинге 1 и содержащей различные поля:

1. Имя пользователя
2. Дата и время отправки
3. Содержание сообщения

Также для структуры перегружены конструктор, оператор присваивания, оператор «<», оператор «==» и реализована функция вывода данных.

*Листинг 1. Структура Message.*

struct Message {  
 std::string username;  
 std::string time;  
 std::string content;  
  
 Message(const std::string& uname, const std::string& tm, const std::string& msg)  
 : username(uname), time(tm), content(msg) {}  
  
 Message& operator=(const Message& other) {  
 if (this != &other) { // Защита от самоприсваивания  
 username = other.username;  
 time = other.time;  
 content = other.content;  
 }  
 return \*this;  
 }  
  
 bool operator<(const Message& other) const {  
 return std::tie(time, username, content) < std::tie(other.time, other.username, other.content);  
 }  
  
 bool operator==(const Message& other) const {  
 return username == other.username && time == other.time && content == other.content;  
 }  
  
 void print() const {  
 std::cout << username << " " << time << ": " << content << std::endl;  
 }  
};

В классе MessageDatabase используется std::set, где ключевыми элементами являются умные указатели std::shared\_ptr<Message> на объекты структуры Message. Также в классе реализованы следующие функции:

addMessage: Принимает умный указатель на сообщение и добавляет его в набор. Это позволяет хранить сообщения в упорядоченном виде.

removeMessage: Удаляет сообщение, соответствующее переданному умному указателю, из набора. Если сообщение не найдено, выводится сообщение об ошибке.

removeAllMessagesFromUser: Удаляет все сообщения от указанного пользователя. Этот метод перебирает все сообщения и удаляет те, которые принадлежат указанному пользователю.

findMessage: Пытается найти сообщение по имени пользователя, временной метке и содержимому. Если сообщение найдено, возвращается умный указатель на него. Если сообщение не найдено, возвращается std::nullopt.

printAllMessagesFromUser: Выводит все сообщения от указанного пользователя. Этот метод перебирает все сообщения и выводит те, которые принадлежат указанному пользователю.

printMessagesFromUserInTimeRange: Выводит сообщения от указанного пользователя, которые попадают в заданный временной диапазон.

printMessagesInTimeRange: Выводит все сообщения, которые попадают в заданный временной диапазон, независимо от пользователя.

Далее представлен фрагмент листинга функции main, демонстрирующий работу программы:

*Листинг 2. Демонстрация работы*

int main() {

try {

MessageDatabase db;

loadMessagesFromFile("dialogs.txt", db);

std::cout << "All messages from user Alice:\n";

db.printAllMessagesFromUser("Alice");

std::cout << "\nMessages from user Bob between 2023-10-07 10:00:00.000 and 2023-10-07 12:00:00.000:\n";

db.printMessagesFromUserInTimeRange("Bob", "2023-10-07 10:00:00.000", "2023-10-07 12:00:00.000");

std::cout << "\nAll messages between 2023-10-07 10:00:00.000 and 2023-10-07 12:00:00.000:\n";

db.printMessagesInTimeRange("2023-10-07 10:00:00.000", "2023-10-07 12:00:00.000");

std::cout << "\nRemoving a specific message from Alice:\n";

auto msgOpt = db.findMessage("Alice", "2023-10-07 11:00:00.000:", "Hello Bob!");

if (msgOpt) {

db.removeMessage(\*msgOpt);

} else {

std::cerr << "Message not found.\n";

}

std::cout << "\nAll messages after removing a specific message:\n";

db.printMessagesInTimeRange("2023-10-07 10:00:00.000", "2023-10-07 12:00:00.000");

std::cout << "\nRemoving all messages from Alice:\n";

db.removeAllMessagesFromUser("Alice");

std::cout << "\nAll messages after removing all messages from Alice:\n";

db.printMessagesInTimeRange("2023-10-07 10:00:00.000", "2023-10-07 12:00:00.000");

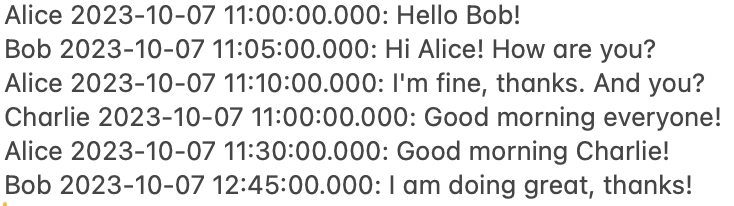
}

...

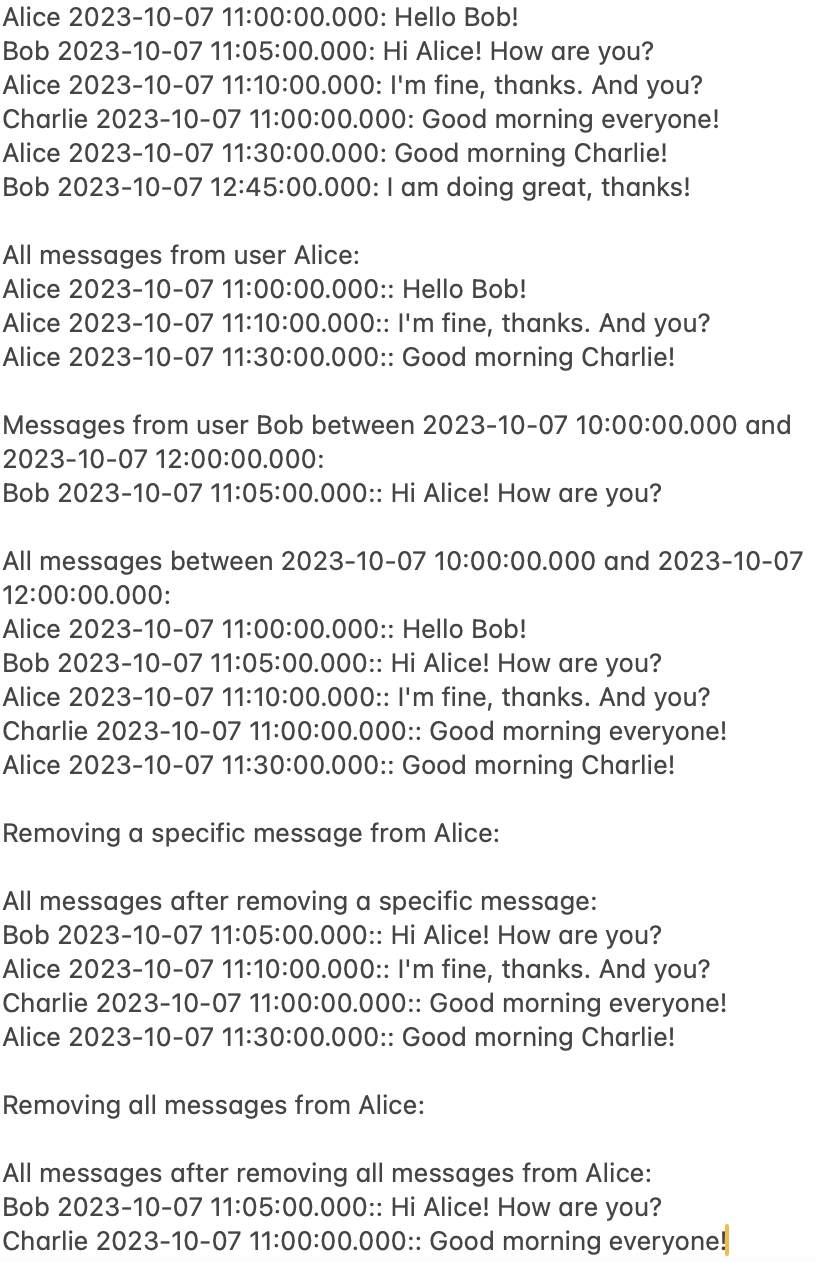
return 0;

}

Пример входного файла и результат работы программы представлен на рисунках ниже:



*Рисунок 1. Входной файл.*

**

*Рисунок 2. Результат выполнения.*

Требуемый результат задачи достигнут, был разработан метод сохранения диалогов пользователей определённой социальной сети. Детали реализации находятся в приложении 1.

# Заключение

В рамках дисциплины "Языки и методы программирования" была успешно решена задача по разработке приложений на C++, удовлетворяя всем предъявленным техническим условиям. В ходе работы над проектами были тщательно изучены и применены ключевые аспекты C++, включая объектно-ориентированное проектирование, управление памятью, файловые операции и использование стандартной библиотеки. Обработка исключений была выполнена с особым вниманием к деталям для обеспечения стабильности кода. Для улучшения архитектуры каждого приложения было принято решение разделить обработку данных и их отображение, что способствовало повышению читаемости кода. Все данные, вводимые пользователем через консоль и параметры, передаваемые в функции, были тщательно проверены на соответствие ожидаемым типам.

Реализация этих проектов не только позволила применить теоретические знания на практике, но и способствовала развитию важных профессиональных умений, таких как написание эффективного и надежного кода, отладка приложений и решение комплексных проблем. Проект подчеркнул значимость и популярность C++ в сфере программирования, подтвердив его выбор как инструмент для создания качественного программного обеспечения.

Полученный опыт и знания заложили крепкий фундамент для будущего профессионального роста и станут ключом к успешному решению профессиональных задач в области IT и программирования.

# Список использованных источников

1. Теренс Чан "Программирование на С++ под Unix." 1999.

2. Кнут, Дональд Э. "Искусство программирования. Том 1." 1968.

3. Кнут, Дональд Э. "Искусство программирования. Том 3." 1973.

4. Джосаттис, Нико. "Стандартная библиотека C++. Второе издание." 2012.

5. Мейерс, Скотт. "Эффективный и современный C++." 2016.

# Приложение 1

Листинг программного кода располагается по ссылке:

https://github.com/sergpolom1n/Kursach.git