**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»

по теме: «Разработка прикладных программ на языке С++»

Выполнил:

студент 2 курса, гр. МПМ-122

Якунин К.А.

Руководитель:

Романенков А.М.

Москва, 2024 г.

Содержание

[Введение 3](#_Toc169123937)

[Основная часть 4](#_Toc169123938)

[Задание 3 4](#_Toc169123939)

[Задание 4 9](#_Toc169123940)

[Задание 5 13](#_Toc169123941)

[Задание 11 17](#_Toc169123942)

[Задание 12 21](#_Toc169123943)

[Заключение 26](#_Toc169123944)

[Список использованных источников 27](#_Toc169123945)

[Приложение 1 28](#_Toc169123946)

# 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 для некоторой MMORPG системы усиления игровых персонажей с помощью случайных элементов экипировки. 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 1.

*Таблица 1. Описание использованных средств языка.*

|  |  |  |
| --- | --- | --- |
| № | Средство | Назначение |
| 1. | iostream | Управление операциями ввода и вывода |
| 2. | vector | Вводит класс динамического массива std::vector для хранения и управления динамическими массивами |
| 3. | string | Обработка и манипуляции со строками |
| 4. | random | Предоставляет различные генераторы случайных чисел, распределения и другие утилиты для работы со случайными числами |
| 5. | unordered\_map | Предоставляет std::unordered\_map, неупорядоченную ассоциативную коллекцию пар ключ-значение. Основан на хеш-таблицах |

Для начала были определены перечисления: enum CharacterClass, которое определяет возможные классы персонажей, enum SlotType, которое определяет различные типы слотов для экипировки, а также структура Stats, которая определяет структуру для хранения характеристик персонажа и экипировки.

Далее был реализован Класс Equipment, в котором создаются объекты, представляющие различные предметы экипировки для персонажей игры. Каждый объект Equipment хранит информацию о своем типе слота (например, шлем, оружие и т. д.), названии, а также статистике, которую он предоставляет персонажу (здоровье, броня, сила и т. д.).

После этого был описан класс Character, который управляет характеристиками персонажа, обновляет их при экипировке снаряжения, рассчитывает боевые показатели, такие как урон, и применяет способности.

*Листнг 1. Описание класса персонажей.*

class Character {

public:

std::string nickname;

CharacterClass charClass;

Stats baseStats;

Stats totalStats;

int abilityDamage;

std::unordered\_map<SlotType, Equipment> equipmentSlots;

Character(const std::string& name, CharacterClass cls) : nickname(name), charClass(cls), abilityDamage(0) {

baseStats = getDefaultStats(cls);

totalStats = baseStats;

}

void equip(const Equipment& equipment) {

equipmentSlots[equipment.slotType] = equipment;

updateTotalStats();

}

void applyRandomAbility(std::vector<int>& abilityDamages) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(0, abilityDamages.size() - 1);

int index = dis(gen);

abilityDamage = abilityDamages[index];

}

void updateTotalStats() {

totalStats = baseStats;

for (const auto& eq : equipmentSlots) {

totalStats.health += eq.second.stats.health;

totalStats.armor += eq.second.stats.armor;

totalStats.strength += eq.second.stats.strength;

*Продолжение листинга 1. Описание класса персонажей.*

totalStats.intellect += eq.second.stats.intellect;

totalStats.agility += eq.second.stats.agility;

totalStats.accuracy += eq.second.stats.accuracy;

totalStats.luck += eq.second.stats.luck;

totalStats.mastery += eq.second.stats.mastery;

}

}

int calculateDamage() const {

int damage = 0;

switch (charClass) {

case DEFENDER:

damage = totalStats.strength \* 2;

break;

case HEALER:

damage = totalStats.intellect \* 2;

break;

case MELEE\_FIGHTER:

damage = totalStats.agility \* 2 + totalStats.strength;

break;

case RANGED\_FIGHTER:

damage = totalStats.intellect \* 2 + totalStats.agility;

break;

}

damage += totalStats.mastery;

damage += abilityDamage; // Применяем урон от способности

return damage;

}

private:

Stats getDefaultStats(CharacterClass cls) {

switch (cls) {

case DEFENDER:

return { 100, 50, 10, 5, 5, 5, 5, 5 };

case HEALER:

return { 80, 30, 5, 10, 5, 5, 5, 5 };

case MELEE\_FIGHTER:

return { 90, 40, 10, 5, 10, 5, 5, 5 };

case RANGED\_FIGHTER:

return { 70, 20, 5, 10, 10, 5, 5, 5 };

default:

return { 0, 0, 0, 0, 0, 0, 0, 0 };

}

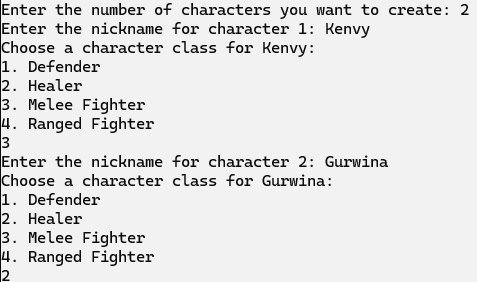
}

};

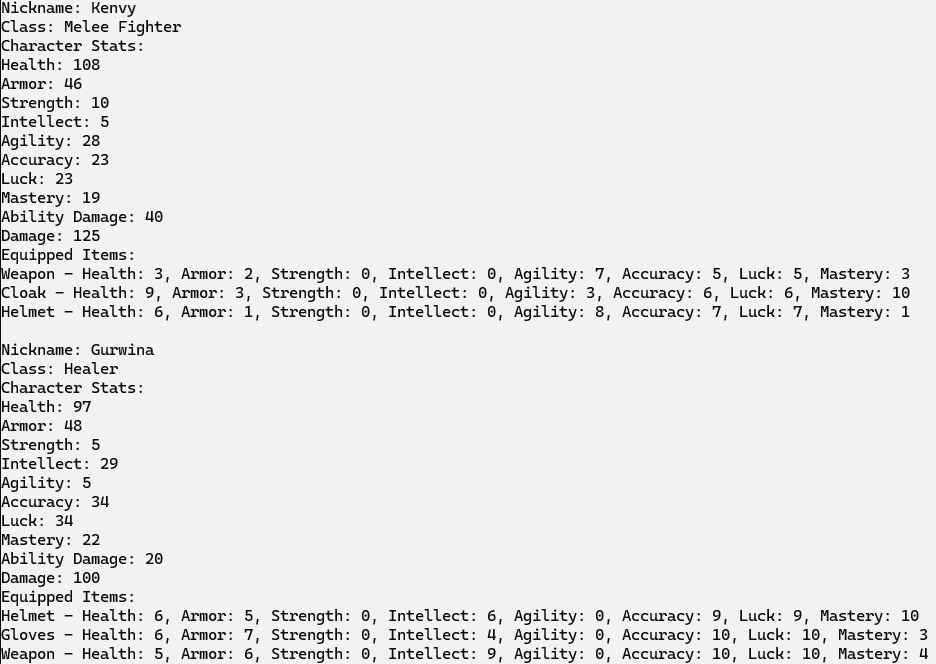
В классе Character реализовано создание персонажа с заданным никнеймом и классом. Класс управляет характеристиками персонажа, включая базовые и полные характеристики, учитывая снаряжение. При добавлении снаряжения обновляются полные характеристики персонажа. Класс также включает методы для расчета урона на основе характеристик и класса персонажа. Кроме того, персонаж может получить случайную способность, урон которой рассчитывается на основе общего урона персонажа и множителя способности.

Дальше для демонстрации и создания персонажей был создан класс Game, в котором реализовано управление созданием и развитием персонажей. Он включает в себя методы для создания персонажа с заданным никнеймом и классом, генерации случайного снаряжения для персонажей, добавления созданных персонажей в список игры, а также добавления и назначения случайных боевых способностей. Основной метод startGame позволяет пользователю создать несколько персонажей, выбрать для них классы и никнеймы, сгенерировать для них снаряжение и способности, а затем вывести их полные характеристики и рассчитанный урон, включая урон от способности.

Результат выполнения программы представлен на рисунках 1 и 2.



*Рисунок 1. Создание нескольких персонажей разных классов.*

**

*Рисунок 2.Созданные персонажи с их характеристиками и снаряжением.*

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" |
| 10. | functional | Позволяет использовать классы и функции для работы с функциональными объектами |
| 11. | limits | Предоставляет типовые ограничения для различных типов данных |

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 (см. листинг 2), 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 подменяет синонимы на канонические слова, если таковые имеются в словаре.

*Листнг 2. Класс TextProcessor.*

class TextProcessor {

private:

SynonymDictionary& dictionary;

public:

TextProcessor(SynonymDictionary& dict) : dictionary(dict) {}

void processFile(const std::string& input\_filename, const std::string& output\_filename, bool automatic\_mode) {

std::ifstream input\_file(input\_filename);

if (!input\_file.is\_open()) {

throw std::runtime\_error("Could not open input file: " + input\_filename);

}

std::ofstream output\_file(output\_filename);

if (!output\_file.is\_open()) {

input\_file.close();

throw std::runtime\_error("Could not open output file: " + output\_filename);

}

std::string line;

while (std::getline(input\_file, line)) {

std::istringstream line\_stream(line);

std::string word;

while (line\_stream >> word) {

std::string canonical\_word = dictionary.getCanonicalWord(word);

if (canonical\_word == word && dictionary.getCanonicalWord(word) == word) {

if (!automatic\_mode) {

std::cout << "Word not in dictionary: " << word << std::endl;

char option;

std::cout << "Add to dictionary? (y/n): ";

std::cin >> option;

if (option == 'y' || option == 'Y') {

std::cout << "Enter canonical word for " << word << ": ";

std::string canon;

std::cin >> canon;

dictionary.addSynonym(canon, word);

*Продолжение листинга 2. Класс TextProcessor.*

}

}

}

output\_file << dictionary.getCanonicalWord(word) << " ";

}

output\_file << "\n";

}

input\_file.close();

output\_file.close();

}

};

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.

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++ |
|  | chrono | Инструменты для работы с временем и длительностями |
|  | ctime | Функции и типы данных для работы со временем и датами в языке C++ |
|  | iomanip | Манипулирование выводом и вводом данных |

Для учёта деятельности сотрудников мы используем словарь employees с ключом "ФИО". Каждый сотрудник имеет список контрактов, содержащих следующие данные:

- Номер контракта;

- Дата начала;

- Дата окончания;

- Количество отработанных часов;

- Стоимость оплаты.

При обработке файла мы считываем его построчно. Когда встречаем левую фигурную скобку, сохраняем предыдущие данные как имя пользователя. Затем анализируем строки, содержащие информацию о договорах.

Функция parseAndStoreData разбирает и структурирует информацию о сотрудниках и их контрактах из текстового файла. Мы разделяем текст по пробельным разделителям и извлекаем следующие значения:

- Номер контракта;

- Время начала исполнения;

- Время завершения исполнения;

- Количество отработанных часов;

- Размер оплаты.

*Листинг 3. Функция parseAndStoreData.*

void parseAndStoreData(const string& filename)

{

ifstream file(filename);

if (!file.is\_open())

{

cout << "Failed to open the file: " << filename << endl;

return;

}

string line;

Employee currentEmployee;

while (getline(file, line))

{

if (line.empty()) continue;

// Считываем ФИО сотрудника

stringstream ss(line);

ss >> currentEmployee.lastName >> currentEmployee.firstName >> currentEmployee.middleName;

// Считываем блок с информацией о договорах сотрудника

getline(file, line); // Пропускаем открывающую фигурную скобку

while (getline(file, line))

{

if (line == "}") break; // Конец блока

// Считываем информацию о договоре

Contract contract;

stringstream ss\_contract(line);

string temp;

ss\_contract >> temp; // Пропускаем "Договор"

ss\_contract >> temp; // Считываем номер договора

*Продолжение листинга 3. Функция parseAndStoreData.*

contract.number = temp;

ss\_contract >> temp; // Пропускаем "нач."

ss\_contract >> contract.startDate;

ss\_contract >> temp; // Пропускаем "кон."

ss\_contract >> contract.endDate;

ss\_contract >> temp; // Пропускаем "Работа"

ss\_contract >> contract.work;

ss\_contract >> temp; // Пропускаем "Стоимость"

ss\_contract >> contract.cost;

currentEmployee.contracts.push\_back(contract);

}

// Сохраняем данные о сотруднике

string key = currentEmployee.lastName + " " + currentEmployee.firstName + " " + currentEmployee.middleName;

employees[key] = currentEmployee;

// Очищаем данные для следующего сотрудника

currentEmployee = Employee();

}

file.close();

}

Эта функция позволяет загрузить данные из файла, создавая объекты сотрудников и добавляя информацию о соответствующих контрактах. Пустые строки пропускаются. При встрече строки с символом "}", текущий сотрудник завершается, а объект сохраняется в словаре employees.

При запуске программы она предлагает пользователю выбрать одно из действий:

- Отобразить общую стоимость всех контрактов для сотрудника.

- Отобразить все контракты для сотрудника.

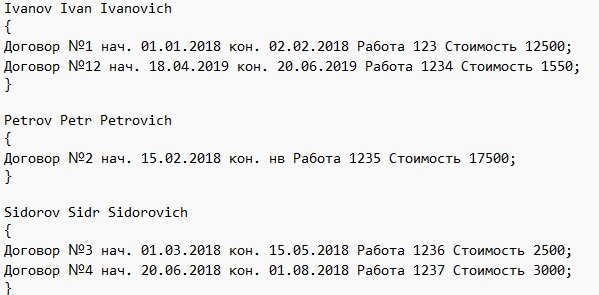
- Найти самый длительный контракт для сотрудника.

- Найти самый дорогой контракт для сотрудника.

- Удалить запись о сотруднике.

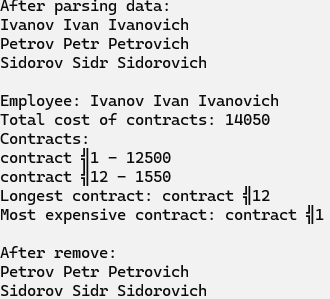
- Завершить выполнение.

Данные входного файла представлены на рисунке 6.

**

*Рисунок 6. Входные данные.*

Демонстрация результатов выполнения различных действий алгоритма:



*Рисунок 7. Результат выполнения.*

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 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 |
|  | algorithm | Алгоритмы для работы с контейнерами в C++ |
|  | random | Предоставляет различные генераторы случайных чисел, распределения и другие утилиты для работы со случайными числами |
|  | forward\_list | Односвязный список |
|  | ctime | Функции и типы данных для работы со временем и датами в языке C++ |
| 7. | unordered\_map | Предоставляет std::unordered\_map, неупорядоченную ассоциативную коллекцию пар ключ-значение. Основан на хеш-таблицах |

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).

Генерация билетов: При выборе типа лотереи пользователем и вводе количества билетов вызывается соответствующий метод для генерации билетов (generateTickets). Этот метод создает билеты с номерами от 1 до заданного количества и добавляет их в соответствующий контейнер.

Перемешивание билетов: После генерации билетов они перемешиваются с помощью метода shuffleTickets. Это делается для обеспечения случайного выбора билетов в розыгрыше.

Проведение розыгрыша: После перемешивания билетов вызывается метод для проведения розыгрыша (conductDraw). В этом методе случайным образом выбирается определенное количество билетов (количество победителей), и каждому из них назначается случайная сумма выигрыша.

Вывод результатов розыгрыша: После проведения розыгрыша вызывается метод для вывода результатов (printWinningTickets). В этом методе отсортированный список выигрышных билетов выводится на экран, указывая номер каждого билета и сумму его выигрыша.

Поиск билета и проверка выигрыша: После вывода результатов пользователь может ввести номер билета для поиска и проверки выигрыша. Вызывается соответствующий метод (checkAndGetValue), который проверяет, есть ли билет с таким номером в списке выигрышных, и если да, то возвращает сумму его выигрыша.

*Листинг 4. Класс LotteryForwardList.*

class LotteryForwardList

{

private:

std::forward\_list<Ticket> tickets;

std::unordered\_map<int, double> prizes;

public:

// Генерация билетов для лотереи

void generateTickets(int numTickets)

{

tickets.clear();

for (int i = numTickets; i >= 1; --i)

{

tickets.emplace\_front(i); // Нумерация билетов с 1

}

}

// Перемешивание билетов

void shuffleTickets()

{

std::vector<Ticket> tempTickets{ std::begin(tickets), std::end(tickets) };

static std::mt19937 mt(std::time(nullptr)); // Генератор случайных чисел

std::shuffle(tempTickets.begin(), tempTickets.end(), mt);

tickets.assign(tempTickets.begin(), tempTickets.end());

}

// Получение случайного билета

Ticket getRandomTicket()

{

Ticket ticket = tickets.front();

tickets.pop\_front();

return ticket;

}

// Поиск билета по номеру

Ticket\* findTicket(int number)

{

auto it = std::find\_if(tickets.begin(), tickets.end(), [number](const Ticket& ticket) {

return ticket.getNumber() == number;

});

if (it != tickets.end())

return &(\*it); // Возвращаем указатель на найденный билет

else

return nullptr; // Билет не найден

}

// Проверка участия билета в розыгрыше и получение суммы выигрыша

*Продолжение листинга 4. Класс LotteryForwardList.*

double checkAndGetValue(const Ticket& ticket)

{

auto it = prizes.find(ticket.getNumber());

if (it != prizes.end())

{

return it->second; // Возвращаем выигрыш билета

}

else

{

return 0; // Билет не участвовал в розыгрыше

}

}

void setPrize(const Ticket& ticket, double prize)

{

prizes[ticket.getNumber()] = prize;

}

bool hasTickets() const

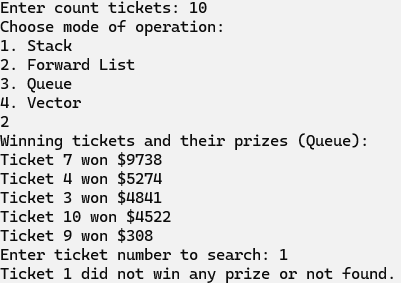
{

return !tickets.empty();

}

};

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 8.



*Рисунок 8. 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 5.

*Таблица 5. Описание использованных средств языка*

|  |  |  |
| --- | --- | --- |
| № | Средство | Назначение |
| 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. | iomanip | Используется для форматирования вывода сообщений в консоль |
| 7. | regex | Используется для проверки формата |

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

- Имя пользователя

- Дата и время отправки

- Содержание сообщения

В начале программы определены несколько структур и классов:

Структура Message представляет отдельное сообщение и содержит информацию о пользователе, времени отправки и тексте сообщения. Она также имеет оператор сравнения <, который используется для сравнения сообщений на основе времени и имени пользователя.

Класс MessageStore представляет собой контейнер для хранения сообщений. Он содержит методы для добавления, удаления и вывода сообщений.

*Листинг 5. Класс MessageStore.*

class MessageStore {

public:

void addMessage(const std::string& user, const std::string& time, const std::string& text) {

messages.emplace(user, time, text);

}

void deleteMessage(const std::string& user, const std::string& time) {

auto it = std::find\_if(messages.begin(), messages.end(), [&](const Message& msg) {

std::string cleanMsgTime = msg.time;

cleanMsgTime.erase(std::remove\_if(cleanMsgTime.begin(), cleanMsgTime.end(), ::isspace), cleanMsgTime.end());

std::string cleanDelTime = time;

cleanDelTime.erase(std::remove\_if(cleanDelTime.begin(), cleanDelTime.end(), ::isspace), cleanDelTime.end());

return msg.user == user && cleanMsgTime == cleanDelTime;

});

if (it != messages.end()) {

messages.erase(it);

std::cout << "Message deleted successfully." << std::endl;

}

else {

std::cout << "Message not found for user: " << user << " at time: " << time << std::endl;

}

}

void deleteMessagesByUser(const std::string& user) {

auto it = messages.begin();

while (it != messages.end()) {

if (it->user == user) {

it = messages.erase(it);

}

else {

++it;

}

}

*Продолжение листинга 5. Класс MessageStore.*

std::cout << "All messages from user: " << user << " have been deleted." << std::endl;

}

void printMessagesByUser(const std::string& user) const {

for (const auto& msg : messages) {

if (msg.user == user) {

std::cout << msg.user << " " << msg.time << ": " << msg.text << std::endl;

}

}

}

void printMessagesByUserInRange(const std::string& user, const std::string& startTime, const std::string& endTime) const {

for (const auto& msg : messages) {

if (msg.user == user && msg.time >= startTime && msg.time <= endTime) {

std::cout << msg.user << " " << msg.time << ": " << msg.text << std::endl;

}

}

}

void printMessagesInRange(const std::string& startTime, const std::string& endTime) const {

for (const auto& msg : messages) {

if (msg.time >= startTime && msg.time <= endTime) {

std::cout << msg.user << " " << msg.time << ": " << msg.text << std::endl;

}

}

}

void printMessages() const {

for (const auto& msg : messages) {

std::cout << msg.user << " " << msg.time << ": " << msg.text << std::endl;

}

}

private:

std::set<Message> messages;

};

Затем определены функции для загрузки сообщений из файлов и главная функция main. В функции main сначала проверяется, были ли переданы в аргументах командной строки имена файлов. Если нет, выводится сообщение о правильном использовании программы.

Затем создается объект MessageStore, и для каждого файла, переданного в аргументах, вызывается функция loadMessagesFromFile, которая загружает сообщения из файла в хранилище.

После загрузки сообщений выводится список загруженных сообщений.

Затем определяются переменные для пользователя, времени и временного диапазона, для которых будет выполнен вывод сообщений.

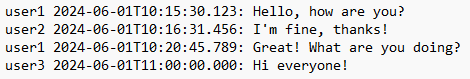
Программа выводит сообщения пользователя, указанного в переменной user, затем выводит сообщения этого пользователя в определенном временном диапазоне, а также выводит все сообщения в этом временном диапазоне.

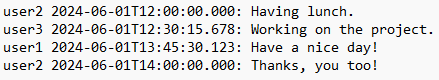
Затем программа пытается удалить сообщение пользователя в указанное время и выводит обновленный список сообщений после попытки удаления.

Затем удаляются все сообщения пользователя и выводится окончательный список сообщений.

Все операции по удалению сообщений реализованы таким образом, что сообщения сравниваются по пользователю и времени, чтобы найти соответствующее сообщение для удаления.

Пример входного файла и результат работы программы представлен на рисунках 9, 10.





*Рисунок 9. Входные файлы.*

**

*Рисунок 10. 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/Kenvymrr/CursovayaC--