

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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  
Кешабян Р. Р.  
Руководитель:  
Романенков А.М.

Москва, 2024 г

# СОДЕРЖАНИЕ

Введение.....	3
1 Реализованные программы.....	4
1.1 Реализация для MMORPG системы усиления персонажа.....	4
1.2 Обработка диалогов в социальной сети.....	7
1.3 Интерпретатор операций над целочисленными массивами.....	9
1.4 Обработка служебных записей сотрудников.....	11
1.5 Приложение для проведения лотереи.....	14
Вывод.....	17
Список использованных источников.....	18
Приложение.....	19

# ВВЕДЕНИЕ

**Актуальность:** В современном мире программирование играет огромную роль во всех сферах деятельности, поэтому данная работа, направленная на изучение ассоциативных контейнеров и работу с ними, является актуальной. Код данной работы полностью реализован на языке программирования C++(стандарт C++14 и выше).

**Цель работы:** Разработка эффективного программного обеспечения для обработки поступающих файлов ассоциативного контейнера.

**Объектом исследования** данной работы является процесс разработки эффективного программного обеспечения.

**Предметом исследования** данной работы является применение объектно-ориентированной парадигмы (сокр. ООП) и библиотеки стандартных шаблонов (сокр. англ. STL – Standard Template Library) языка программирования C++ для разработки программного обеспечения.

## **Задачи:**

- 1) Изучить STL.
- 2) Реализовать код.
- 3) Протестировать программу.
- 4) Исправить ошибки.
- 5) Оптимизировать работу программы.

# 1 Реализованные программы

## 1.1 РЕАЛИЗАЦИЯ ДЛЯ MMORPG СИСТЕМЫ УСИЛЕНИЙ ПЕРСОНАЖА

В данном задании необходимо разработать для некоторой MMORPG систему усиления игровых персонажей с помощью случайных элементов экипировки. Для разработки системы было реализовано следующее:

### 1) Перечисление классов персонажей

*Листинг 1. Перечисление классов персонажей*

```
enum class CharacterClass
{
    Defender,
    Healer,
    MeleeFighter,
    RangedFighter
};
```

### 2) Структура для характеристик персонажа

*Листинг 2. Структура для характеристик персонажа*

```
struct CharacterStats
{
    int health;
    int armor;
    int strength;
    int intelligence;
    int agility;
    int accuracy;
    int luck;
    int mastery;
};
```

### 3) Базовый класс элементов экипировки

*Листинг 3. Базовый класс элементов экипировки*

```
class Equipment
{
public:
    virtual ~Equipment() {}
    virtual void applyStats(CharacterStats& stats) const = 0;
};
```

### 4) Класс для генерации случайной экипировки

*Листинг 4. Класс для генерации случайной экипировки*

```
class EquipmentGenerator
{
public:
    virtual ~EquipmentGenerator() {}
    virtual Equipment* generateEquipment() = 0;
};
```

Далее были созданы классы Weapon и Armor, наследуемые от класса Equipment, в который происходит создание оружия и брони соответственно.

*Листинг 5. Класс оружия*

```
class Weapon : public Equipment
{
    ...
};
```

```
private:
    int damage;
public:
    Weapon(int damage) : damage(damage) {}

    void applyStats(CharacterStats& stats) const override
    {
        stats.strength += damage;
    }
};
```

*Листинг 6. Класс брони*

```
class Armor : public Equipment
{
private:
    int armorValue;
public:
    Armor(int armorValue) : armorValue(armorValue) {}

    void applyStats(CharacterStats& stats) const override
    {
        stats.armor += armorValue;
    }
};
```

Также были реализованы классы, которые генерируют случайное оружие и броню и задают значения их характеристикам.

*Листинг 7. Класс, генерирующий случайное оружие*

```
class WeaponGenerator : public EquipmentGenerator
{
public:
    Equipment* generateEquipment() override
    {
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<> dis(1, 10);
        int damage = dis(gen);
        return new Weapon(damage);
    }
};
```

*Листинг 8. Класс, генерирующий случайную броню*

```
class ArmorGenerator : public EquipmentGenerator
{
public:
    Equipment* generateEquipment() override
    {
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<> dis(1, 20);
        int armorValue = dis(gen);
        return new Armor(armorValue);
    }
};
```

Крайней реализацией стал класс игрового персонажа, в котором происходит

инициализация характеристик персонажа в зависимости от его класса. В данном классе также реализованы функция добавления элемента экипировки, функция подсчета наносимого урона и остальные необходимые функции.

*Листинг 9. Класс игрового персонажа*

```
class Character
{
    ...
    // Инициализация характеристик в зависимости от класса персонажа
    switch (characterClass)
    {
        case CharacterClass::Defender:
            stats.health = 100;
            stats.armor = 50;
            stats.strength = 10;
            stats.intelligence = 0;
            stats.agility = 5;
            break;
        ...
    }
    ...
    int calculateDamage() const
    {
        return stats.strength + stats.mastery;
    }
    int calculateProtection() const
    {
        return stats.health + stats.luck;
    }
    ...
};
```

В функции `main()` создаётся объект класса `Character`, которого снабжают экипировкой и для которого производится подсчет наносимого урона. В конце программы происходит освобождение ранее выделенной динамической памяти.

*Листинг 10. Функция `main()`*

```
int main()
{
    // Создание персонажа
    CharacterClass charclass = CharacterClass(character);
    ...
    switch (character)
    {
    case 1:
        std::cout << "You have chosen the Defender class!\n";
        std::cout << "Damage: " << damage << std::endl;
        std::cout << "Protection: " << protection << std::endl;
        break;
    case 2:
        std::cout << "You have chosen the Healer class!\n";
        std::cout << "Damage: " << damage << std::endl;
        std::cout << "Protection: " << protection << std::endl;
```

```

        std::cout << "Therapy: " << therapy << std::endl;
        break;
    case 3:
        std::cout << "You have chosen the MeleeFighter class!\n";
        std::cout << "Damage: " << damage << std::endl;
        std::cout << "Protection: " << protection << std::endl;
        std::cout << "Endurance: " << endurance << std::endl;
        break;
    case 4:
        std::cout << "You have chosen the RangedFighter class!\n";
        std::cout << "Damage: " << damage << std::endl;
        std::cout << "Protection: " << protection << std::endl;
        std::cout << "Range: " << range << std::endl;
        break;
    default:
        break;
}
...
return 0;
}

```

## 1.2 ОБРАБОТКА ДИАЛОГОВ В СОЦИАЛЬНОЙ СЕТИ

На вход программе подаются файлы с данными диалогов пользователей некой социальной сети. Необходимо реализовать приложение, которое будет выполнять следующие действия:

- 1) Вывода всех сообщений заданного пользователя
- 2) Вывода сообщений заданного пользователя для заданного временного интервала
- 3) Вывод всех сообщений из заданного временного интервала

Для хранения данных о сообщениях каждого пользователя была создана структура Message.

*Листинг 11. Структура Message*

```

struct Message
{
    std::string userName;
    std::string time;
    std::string text;

    Message(const std::string& user, const std::string& t, const
std::string& msg)
        : userName(user), time(t), text(msg) {}

    friend std::ostream& operator<<(std::ostream& os, const Message& msg)
    {
        os << msg.userName << " " << msg.time << msg.text;
        return os;
    }
};

```

Далее создана структура для сравнения сообщений пользователей, в

которой перегружен оператор(), для класса Message.

*Листинг 12. Функтор MessageComparator*

```
struct MessageComparator
{
    bool operator()(const Message* lhs, const Message* rhs) const
    {
        if (lhs->userName != rhs->userName)
        {
            return lhs->userName < rhs->userName;
        }
        return lhs->time < rhs->time;
    }
};
```

В классе MessageProcessor реализованы функции для вывода всех сообщений пользователя, вывода сообщений заданного пользователя для заданного временного интервала, вывод всех сообщений из заданного временного интервала.

*Листинг 12. Класс MessageProcessor*

```
class MessageProcessor final
{
    ...
    void printUserMessages(const std::string& user) const
    {
        ...
    }
    void printUserMessagesInTimeInterval(const std::string& user, const
std::string& startTime, const std::string& endTime) const
    {
        ...
    }
    void printMessagesInTimeInterval(const std::string& startTime, const
std::string& endTime) const
    {
        ...
    }
    void deleteMessage(const std::string& user, const std::string& time)
    {
        ...
    }
    void deleteUserMessages(const std::string& user)
    {
        ...
    }
};
```

В функции main() производится вызов реализованных функций с параметрами, которые задаёт пользователь(имя пользователя и время сообщения). Результат выводится в консоль.



```

Enter the name of the user whose messages you want to see:
Ilya
All messages of user 'Ilya':
Ilya 15:00:00: Where are the allocators?
Ilya 15:30:00: Everything I say will be on the exam.

Enter the name of the user whose messages you want to see in this time interval:
Aleksandr
Enter the start time of the message, accurate to milliseconds: 10:00:00
Enter the end time of the message, accurate to milliseconds: 20:00:00
Messages of user 'Aleksandr' between 10:00:00 and 20:00:00
Aleksandr 16:00:00: Attention - trick.
Aleksandr 16:30:00: I hope this is clear to everyone.

```

Рисунок 1. Результат программы

### 1.3 ИНТЕРПРЕТАТОР ОПЕРАЦИЙ НАД ЦЕЛОЧИСЛЕННЫМИ МАССИВАМИ

Необходимо разработать программу–интерпретатор операций над целочисленными массивами. Программа оперирует целочисленными массивами произвольной длины с именами A, B, ..., Z. Система команд данного интерпретатора (прописные и строчные буквы не различаются) подаётся через файл «instruction.txt».

Чтобы создать интерпретатор понадобилось реализовать функцию, которая по индексу возвращает из строки лексему, это необходимо для того, чтобы программа понимала, какую операцию ей выполнять и под каким именем сохранять массив.

Далее для каждой инструкции было реализована отдельная функция:

Таблица 1. Функции для работы с массивами

Функции	Логика
Load()	Создаёт массив и заполняет его значениями из файла
Save()	Выгружает значения из массива в файл
Rand()	Заполняет массив случайными числами
Concat()	Объединяет два массива
Free()	Очищает массив
Remove()	Удаляет элементы массива начиная с индекса n
Copy()	Копирует значения одного массива в другой
Sort()	Сортирует массив по убыванию(возрастанию)
Permute()	Переставляет элементы массива в случайном порядке
Intersect()	Находит пересечение двух массивов
Xor()	Находит симметрическую разницу
Stat()	Выводит статическую информацию о массиве
Print_from_and_to()	Выводит элементы массива в указанном диапазоне
Print_all()	Выводит все элементы массива

Каждая из данных функция получает на вход в качестве параметров строки, которые являются названиями массивов, строку, которая является строчкой из

файла «instruction.txt», и map, в который сохраняются массивы, возвращают значение типа void. Вот пример реализации одной из функций:

*Листинг 13. Функция сортировки массива*

```
void Sort(std::string& name, std::string& line, std::map<std::string,
std::vector<int>>& arrays)
{
    name = getWord(line, 1);
    name.pop_back();
    char order = name.back();
    name.pop_back();
    if (arrays[name].empty() != true)
    {
        if (order == '+')
        {
            std::sort(arrays[name].begin(), arrays[name].end());
        }
        else if (order == '-')
        {
            std::sort(arrays[name].begin(), arrays[name].end(),
std::greater<int>());
        }
    }
    else
    {
        throw std::logic_error("Array is empty");
    }
    for (auto n : arrays[name])
    {
        std::cout << n << " ";
    }
}
```

Для удобства работы с программой объявление и реализация всех функций была перенесена в отдельный заголовочный файл «Functions.h». В функции main() производится только вызов нужной функции.

*Листинг 14. Функция main()*

```
int main()
{
    ...
    try
    {
        if (instr.is_open())
        {
            while (std::getline(instr, line))
            {
                if (getWord(line, 0) == "LOAD")
                {
                    Load(name, line, arrays);
                }
            }
        }
    }
}
```

```

        if (getWord(line, 0) == "SAVE")
        {
            Save(name, line, arrays);
        }

        if (getWord(line, 0) == "RAND")
        {
            Rand(name, line, arrays);
        }

        if (getWord(line, 0) == "CONCAT")
        {
            Concat(name, name2, line, arrays);
        }

        if (getWord(line, 0)[0] == 'F')
        {
            Freee(name, line, arrays);
        }

        if (getWord(line, 0) == "REMOVE")
        {
            Remove(name, line, arrays);
        }

        ...
        return 0;
    }

```

```

SIZE: 16
Min Value: 1 and its index: 0
Max Value: 300 and its index: 12
Most Frequent: 1
Average: 43
Max Deviation: 257

```

*Рисунок 2. Результат программы*

## 1.4 ОБРАБОТКА СЛУЖЕБНЫХ ЗАПИСЕЙ СОТРУДНИКОВ

На вход программе подаётся файл служебных записей о деятельности сотрудников некоторой фирмы. Необходимо реализовать приложение, которое обрабатывает этот текстовый файл, формируя ассоциативный контейнер, который содержит указатели на записи, считанные из файла. Ключевым набором полей каждой записи является ФИО. Ваша программа должна предоставить возможность для заданного сотрудника, определяемого ключевым набором полей:

- подсчёта стоимости всех договоров этого сотрудника;
- выдачи списка договоров, с которыми работал заданный сотрудник;
- найти самый продолжительный договор данного сотрудника;
- найти самый дорогой договор данного сотрудника.

Необходимо также реализовать удаление информации о сотруднике.

Для хранения информации о сотрудниках и их договоров были реализованы

две структуры с необходимыми полями:

*Листинг 15. Структура для хранения информации о договоре*

```
struct Contract
{
    string number;
    string startDate;
    string endDate;
    string work;
    int cost;
};
```

*Листинг 16. Структура для хранения информации о сотруднике*

```
struct Employee
{
    string lastName;
    string firstName;
    string middleName;
    vector<Contract> contracts;
};
```

Для обработки поступающего файла реализована функция `parseAndStoreData()`, которая читает файл и сохраняет необходимую информацию в выше указанные структуры.

*Листинг 17. Функция обработки файла*

```
void parseAndStoreData(const string& filename)
{
    ...
    while (getline(file, line))
    {
        if (line.empty()) continue;
        if (line.back() == '}')
        {
            string key = currentEmployee.lastName + " " +
currentEmployee.firstName + " " + currentEmployee.middleName;
            employees[key] = currentEmployee;
            ...
        }
        else if (line.back() == ' ')
        {
            stringstream ss(line);
            ss >> currentEmployee.lastName >> currentEmployee.firstName >>
currentEmployee.middleName;
            getline(file, line);
        }
        else
        {
            Contract contract;
            stringstream ss(line);
            string temp;
            ss >> temp;
```

```

        ss >> contract.number;
        ss >> temp;
        ss >> contract.startDate;
        ss >> temp;
        ss >> contract.endDate;
        ss >> temp;
        ss >> contract.work;
        ss >> temp; //
        ss >> contract.cost;
        currentEmployee.contracts.push_back(contract);
    }
}
...
}

```

Также реализованы функции, данные по заданию:

Таблица 2. Функции для работы с поступающей информацией

Функции	Логика
int totalCostOfContracts()	Функция для подсчета стоимости всех договоров сотрудника
vector<Contract> contractsOfWork()	Функция для получения списка договоров сотрудника
Contract longestContract()	Функция для нахождения самого продолжительного договора сотрудника
Contract mostExpensiveContract()	Функция для нахождения самого дорогого договора сотрудника
void removeEmployee()	Функция для удаления информации о сотруднике

В функции main() производится вывод информации в консоль:

Листинг 18. Функция main()

```

int main()
{
    parseAndStoreData("data.txt");
    for (const auto& pair : employees)
    {
        cout << "Employee: " << pair.first << endl;
        cout << "Total cost of contracts: " <<
totalCostOfContracts(pair.second) << endl;

        cout << "Contracts:" << endl;
        for (const auto& contract : contractsOfWork(pair.second))
        {
            cout << contract.number << " - " << contract.cost << endl;
        }

        cout << "Longest contract: " << longestContract(pair.second).number
<< endl;
    }
}

```

```

        cout << "Most expensive contract: " <<
mostExpensiveContract(pair.second).number << endl;

        cout << endl;
    }
    return 0;
}

```

```

Employee: Keshabyan Ruslan Ruslanovich
Total cost of contracts: 13000
Contracts:
13 - 13000
Longest contract: 13
Most expensive contract: 13

Employee: Teterin Geo Tagirov
Total cost of contracts: 15000
Contracts:
4 - 4000
5 - 5000
6 - 6000
Longest contract: 5
Most expensive contract: 6

```

Рисунок 3. Результат работы программы

## 1.5 ПРИЛОЖЕНИЕ ДЛЯ ПРОВЕДЕНИЯ ЛОТЕРЕИ

Необходимо разработать приложение для проведения лотереи (например, аналога Спортлото 6 из 49 или 5 из 36). Приложение должно обеспечивать генерацию билетов для очередного тиража лотереи. Количество генерируемых билетов произвольно и может быть велико (> 20'000'000 шт.). Нужно смоделировать проведение розыгрыша: на каждом ходе проверять, появился ли победитель; предусмотреть систему выигрышей; предоставить возможность поиска билетов по заданным критериям: номеру билета, величине выигрыша.

Реализован класс Ticket, который предоставляет публичные методы для получения номера и выигрыша билета, а также возможность задания выигрыша билета.

Листинг 19. Реализация класса билета

```

class Ticket
{
    ...
public:
    Ticket(int num) : number(num), prize(0) {}

    int getNumber() const { return number; }
    double getPrize() const { return prize; }

    void setPrize(double amount) { prize = amount; }
};

```

Также представлен класс Lottery, в котором реализованы методы генерации

и получения случайного билета:

*Листинг 20. Реализация класса лотереи*

```
class Lottery
{
private:
    std::vector<Ticket> tickets;
public:
    void generateTickets(int numTickets)
    {
        tickets.clear();
        for (int i = 1; i < numTickets; ++i)
        {
            tickets.emplace_back(i);
        }
    }
    Ticket& getRandomTicket()
    {
        static std::mt19937 mt(std::time(nullptr));
        std::uniform_int_distribution<int> dist(0, tickets.size()-1);
        return tickets[dist(mt)];
    }
};
```

Далее разработан класс LotteryDraw, в котором реализованы методы поиска билета по номеру и величине выигрыша.

*Листинг 21. Класс представления тиража*

```
class LotteryDraw
{
    ...
    Ticket* findTicketByNumber(int number)
    {
        auto it = std::find_if(winningTickets.begin(),
winningTickets.end(),
        [number](Ticket* t) { return t->getNumber() == number; });
        return (it != winningTickets.end()) ? *it : nullptr;
    }

    std::vector<Ticket*> findTicketsByPrize(double prize)
    {
        std::vector<Ticket*> result;
        for (auto ticket : winningTickets)
        {
            if (ticket->getPrize() == prize)
            {
                result.push_back(ticket);
            }
        }
        return result;
    }
};
```

В функции main() пользователю предлагается ввести количество билетов в

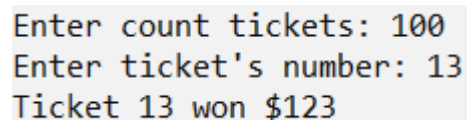
тираже, а также номер своего билета. Далее производится поиск билета, и в консоль выводится результат.

Листинг 21. Функция *main()*

```
int main()
{
    for (const auto& pair : employees)
    {
        cout << "Employee: " << pair.first << endl;
        cout << "Total cost of contracts: " <<
totalCostOfContracts(pair.second) << endl;

        cout << "Contracts:" << endl;
        for (const auto& contract : contractsOfWork(pair.second))
        {
            cout << contract.number << " - " << contract.cost << endl;
        }
        cout << "Longest contract: " << longestContract(pair.second).number
<< endl;
        cout << "Most expensive contract: " <<
mostExpensiveContract(pair.second).number << endl;

        cout << endl;
    }
    return 0;
}
```



```
Enter count tickets: 100
Enter ticket's number: 13
Ticket 13 won $123
```

Рисунок 4. Результат работы программы



## ВЫВОД

В результате данной получены полностью рабочие программы, которые успешно исполняют задания. Также были решены многие проблемы, связанные с выделением и распределением памяти, оптимизации программ и их бесперебойной работы. Все программы успешно прошли тесты. Поставленная цель была достигнута, все задачи были решены в полном объёме. Знания и умения, полученные во время реализации цели данной курсовой работы – это хороший опыт, который будет полезен при дальнейшем освоении профессии.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Эффективное программирование на C++. Практическое программирование на примерах / Му Барбара Э., Кениг Эндрю: Диалектика, 2019. 368 с
- 2) Язык программирования C++. Базовый уровень. Пятое издание / Стенли Б. Липпман, Жози Лажойе, Барабара Э. Му: 2017. 1390 с
- 3) Самоучитель C++. Третье издание / Герберт Шилдт: СПб.: БХВ-Петербург, 2003. — 688 с

## ПРИЛОЖЕНИЕ



QR-Code GitHub

<https://github.com/1Ruslan3/Kurs.git>