# МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Кемеровский государственный университет»

институт цифры кафедра цифровых технологий

# ОТЧЕТ

о выполнении

Лабораторной работы №6 **«ООП на С++: исключения»**

по курсу «Языки программирования»

студента 1 курса

Коник Ильи Николаевича

(ФИО полностью)

направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

направленность (профиль) подготовки «Интеллектуальные информационные системы и анализ данных»

направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии

направленность (профиль) подготовки «Информатика и компьютерные науки»

Преподаватель:

Дуванов И. О.

Кемерово 2025

**Тема: «ООП на С++: исключения»** *(6 баллов)*

## Задание

1. В соответствии с вариантами задания и приведённой ниже спецификацией реализуйте программу на языке C++.
2. Разработайте тестовые задания, и протестируйте программу одним из методов тестирования.

## Требования к оформлению программ:

* 1. **Содержание**. Программа должна делать только то, что предусмотрено заданием. Не нужно выполнять лишнюю работу.
  2. **Спецификация.** В преамбуле программы в комментариях указывать сведения:
     + Кто выполнил.
     + Что делает программа (кратко).
     + Что на входе, имена входных файлов указываются.
     + Что на выходе (что является результатов работы программы).

# Ввод и вывод

* + - Приглашение пользователю. (Например, сколько чисел, какого типа и через какой разделитель нужно вводить).
    - Контрольный вывод (все введенные данные выводить на экран, и только после этого выполнять необходимые вычисления.)
    - «Защита от дурака». Проверять вводимые данные на корректность. Например, если необходимо считать количество чего – то, то эта величина не может быть отрицательной и т.д.
  1. **Структура кода.** Набираемый код должен быть хорошо структурированным. Использовать:
     + Отступы.
     + Комментарии – поясняют решение программы.
     + Осмысленные названия переменных.
     + Пояснения о назначении переменных в комментариях (кроме счетчиков).

# Декомпозиция кода

* + - Функциональная
    - Объектно-ориентированная

# Многофайловые проекты

* + - Классы определять в отдельном h-файле, а все методы классов – в соответствующем cpp - файле. Созданный заголовочный файл подключать к проекту.
    - Методы класса реализовывать не в определении класса.
    - cpp-файл и соответствующий ему h-файл называть одинаково. В качестве

названия выбирать имя того класса, который определен в соответствующем модуле.

## Задания для самостоятельного выполнения:

***Задание 1.***

Напишите функцию ***string get\_password()***, которая спрашивает у пользователя пароль и возвращает его в качестве результата. Функция должна выбрасывать исключения следующих типов:

* LengthError — если длина пароля меньше 9 символов.
* RegisterError — если в пароле все символы одного регистра.
* DigitError — если в пароле нет ни одной цифры.
* ForbiddenLetterError — если в пароле есть символы, которые легко перепутать между собой: «l» (L маленькое), «I» (i большое), «1» (цифра), «o» и «O» (большая и маленькая буквы) и «0» (цифра).

Все исключения должны быть унаследованы от базового — ***PasswordError***.

В функции main добавьте обработку всех исключений. Протестируйте и продемонстрируйте работу программы.

Passworderror.cpp

#include "PasswordError.h"

#include <string>

#include <cctype> // https://learn.microsoft.com/ru-ru/cpp/standard-library/cctype?view=msvc-170

#include <algorithm>

#include <iostream>

const char\* LengthError::what() const noexcept {

    return "Password is too short (less than 9 characters)";

}

const char\* RegisterError::what() const noexcept {

    return "Password contains only one case letters";

}

const char\* DigitError::what() const noexcept {

    return "Password doesn't contain any digits";

}

const char\* ForbiddenLetterError::what() const noexcept {

    return "Password contains forbidden characters (l, I, 1, o, O, 0)";

}

std::string get\_password() {

    std::string password;

    std::cout << "Enter password: ";

    std::cin >> password;

    // Check length

    if (password.length() < 9) {

        throw LengthError();

    }

    // Check register

    bool has\_upper = false, has\_lower = false;

    for (char c : password) {

        if (isupper(c)) has\_upper = true;

        if (islower(c)) has\_lower = true;

    }

    if (!(has\_upper && has\_lower)) {

        throw RegisterError();

    }

    // Check digits

    bool has\_digit = false;

    for (char c : password) {

        if (isdigit(c)) {

            has\_digit = true;

            break;

        }

    }

    if (!has\_digit) {

        throw DigitError();

    }

    // Check forbidden characters

    const std::string forbidden = "lI1oO0";

    for (char c : password) {

        if (forbidden.find(c) != std::string::npos) {

            throw ForbiddenLetterError();

        }

    }

    return password;

}

Passworderror.h

#ifndef PASSWORDERROR\_H

#define PASSWORDERROR\_H

#include <stdexcept>

#include <string>

class PasswordError : public std::exception {

public:

    virtual const char\* what() const noexcept override = 0;

};

class LengthError : public PasswordError {

public:

    const char\* what() const noexcept override;

};

class RegisterError : public PasswordError {

public:

    const char\* what() const noexcept override;

};

class DigitError : public PasswordError {

public:

    const char\* what() const noexcept override;

};

class ForbiddenLetterError : public PasswordError {

public:

    const char\* what() const noexcept override;

};

std::string get\_password();

#endif // PASSWORDERROR\_H

Main.cpp

#include <iostream>

#include "PasswordError.h"

int main() {

    try {

        std::string password = get\_password();

        std::cout << "Password accepted: " << password << std::endl;

    }

    catch (const PasswordError& e) {

        std::cerr << "Error: " << e.what() << std::endl;

    }

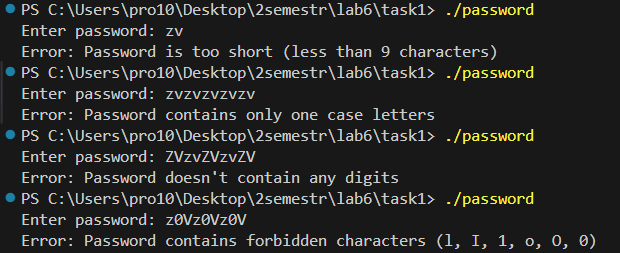
    catch (...) {

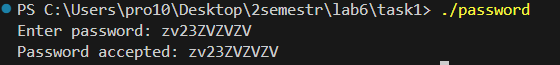
        std::cerr << "Unknown error occurred" << std::endl;

    }

    return 0;

}





## Задание 2.

При разработке компьютерных игр зачастую создаётся очень сложная система классов. Реализуйте следующие классы:

1. Класс Weapon для оружия, который при инициализации должен принимать имя оружия name, наносимый урон damage, а также радиус, на котором оружие может достать до врага – range.

В классе Weapon должны быть реализованы следующие методы:

* + hit(actor, target) – удар персонажем actor персонажа target. Метод должен проверять, жив ли персонаж target, если нет, то выводить сообщение «Враг уже повержен», иначе должна проводится проверка расстояния от actor до target. Если расстояние больше range оружия, тогда надо выводить сообщение «Враг слишком далеко для оружия

<name>». В случае, если target жив и оружие имеет достаточный range, вывести сообщение «Врагу нанесен урон оружием <name> в размере <damage>» и нанести урон target.

* + оператор << – печатает имя оружия.
  + оператор >> – вводит новое оружие.

1. Класс BaseCharacter для создания базовых персонажей, который при инициализации должен принимать pos\_x и pos\_y – позицию персонажа в мире, hp – количество жизней персонажа.

Класс BaseCharacter должен реализовывать методы:

* + move(delta\_x, delta\_y) – для перемещения персонажа на delta\_x и delta\_y в игровом мире.
  + is\_alive() – проверка, жив ли персонаж. Возвращает True, если жив, иначе – False.
  + get\_damage(amount) – убавляет количество жизней персонажа на amount. При достижении 0 или отрицательного значения персонаж считается мёртвым.
  + get\_x(), get\_y() – возвращает текущие координаты персонажа.

1. Класс BaseEnemy для создания противников, который расширяет класс BaseCharacter. При инициализации он должен принимать pos\_x и pos\_y – позицию персонажа в мире, weapon – оружие, hp – количество жизней персонажа. Класс должен использовать конструктор BaseCharacter.

Класс BaseEnemy должен реализовывать следующие методы:

* + hit(target) – для удара персонажа target.
  + оператор << – печатает «Враг на позиции (<pos\_x>, <pos\_y>) с оружием <weapon>».

1. Класс MainHero для создания главного героя, который расширяет класс BaseCharacter. При инициализации он должен принимать pos\_x и pos\_y – позицию персонажа в мире, name – имя персонажа, hp – количество жизней персонажа. Класс должен использовать конструктор BaseCharacter.

Класс MainHero должен реализовывать следующие методы:

* + hit(target) – для удара персонажа target (экземпляр класса BaseEnemy). Метод должен проверять, есть ли у персонажа оружие, если его нет выводить - «Я безоружен», иначе – наносить удар target.
  + add\_weapon(weapon) – добавляет оружие в инвентарь персонажа, если это оружие единственное, то оно должно экипироваться сразу же. Метод должен печатать:

«Подобрал <weapon>».

* + next\_weapon() – для смены оружия. Если оружия нет, то должно выводиться «Я безоружен», если есть только одно оружие – «У меня только одно оружие», иначе должна производиться смена оружия на следующее (в порядке подбора, по кругу) и выводиться сообщение «Сменил оружие на <weapon>».
  + heal(amount) – для повышения количества жизней персонажа на amount. При этом максимальный уровень здоровья главного героя – 200. Также должно выводиться сообщение «Полечился, теперь здоровья <hp>».

Дополнительные требования к программе:

## Реализуйте не менее 2 собственных классов исключений и используйте их в коде.

* Протестируйте работу классов и методов.
* Проведите битву главного героя с несколькими врагами. При желании можно добавить меню взаимодействия с пользователем.

Base\_character.cpp

#include "base\_character.h"

BaseCharacter::BaseCharacter(int pos\_x, int pos\_y, int hp)

    : pos\_x(pos\_x), pos\_y(pos\_y), hp(hp) {}

void BaseCharacter::move(int delta\_x, int delta\_y) {

    pos\_x += delta\_x;

    pos\_y += delta\_y;

}

bool BaseCharacter::is\_alive() const {

    return hp > 0;

}

void BaseCharacter::get\_damage(int amount) {

    hp -= amount;

    if (hp < 0) hp = 0;

}

int BaseCharacter::get\_x() const { return pos\_x; }

int BaseCharacter::get\_y() const { return pos\_y; }

int BaseCharacter::get\_hp() const { return hp; }

base\_character.h

#ifndef BASE\_CHARACTER\_H

#define BASE\_CHARACTER\_H

class Weapon;

class BaseCharacter {

protected:

    int pos\_x;

    int pos\_y;

    int hp;

public:

    BaseCharacter(int pos\_x, int pos\_y, int hp);

    virtual void move(int delta\_x, int delta\_y);

    bool is\_alive() const;

    void get\_damage(int amount);

    int get\_x() const;

    int get\_y() const;

    int get\_hp() const;

};

#endif // BASE\_CHARACTER\_H

Base\_enemy.cpp

#include "base\_enemy.h"

BaseEnemy::BaseEnemy(int pos\_x, int pos\_y, const Weapon& weapon, int hp)

    : BaseCharacter(pos\_x, pos\_y, hp), weapon(weapon) {}

void BaseEnemy::hit(BaseCharacter& target) {

    weapon.hit(\*this, target);

}

std::ostream& operator<<(std::ostream& os, const BaseEnemy& enemy) {

    os << "The enemy on position (" << enemy.pos\_x << ", " << enemy.pos\_y

       << ") with weapon " << enemy.weapon;

    return os;

}

Base\_enemy.h

#ifndef BASE\_ENEMY\_H

#define BASE\_ENEMY\_H

#include "base\_character.h"

#include "weapon.h"

#include <iostream>

class BaseEnemy : public BaseCharacter {

private:

    Weapon weapon;

public:

    BaseEnemy(int pos\_x, int pos\_y, const Weapon& weapon, int hp);

    void hit(BaseCharacter& target);

    friend std::ostream& operator<<(std::ostream& os, const BaseEnemy& enemy);

};

#endif // BASE\_ENEMY\_H

Exceptions.h

#ifndef EXCEPTIONS\_H

#define EXCEPTIONS\_H

#include <stdexcept>

#include <string>

class InvalidWeaponException : public std::runtime\_error {

public:

    InvalidWeaponException(const std::string& msg) : std::runtime\_error(msg) {}

};

class CharacterDeadException : public std::runtime\_error {

public:

    CharacterDeadException(const std::string& msg) : std::runtime\_error(msg) {}

};

#endif // EXCEPTIONS\_H

Main\_hero.cpp

#include "main\_hero.h"

#include "base\_enemy.h"

#include <iostream>

MainHero::MainHero(int pos\_x, int pos\_y, const std::string& name, int hp)

    : BaseCharacter(pos\_x, pos\_y, hp), name(name), current\_weapon\_index(-1) {}

void MainHero::hit(BaseEnemy& target) {

    if (weapons.empty()) {

        std::cout << "I have no weapons to fight with" << std::endl;

        return;

    }

    weapons[current\_weapon\_index].hit(\*this, target);

}

void MainHero::add\_weapon(const Weapon& weapon) {

    weapons.push\_back(weapon);

    if (weapons.size() == 1) {

        current\_weapon\_index = 0;

    }

    std::cout << "Took " << weapon << std::endl;

}

void MainHero::next\_weapon() {

    if (weapons.empty()) {

        std::cout << "I have no weapons to fight with" << std::endl;

        return;

    }

    if (weapons.size() == 1) {

        std::cout << "I have only one weapon" << std::endl;

        return;

    }

    current\_weapon\_index = (current\_weapon\_index + 1) % weapons.size();

    std::cout << "Changed weapon to " << weapons[current\_weapon\_index] << std::endl;

}

void MainHero::heal(int amount) {

    hp += amount;

    if (hp > 200) hp = 200;

    std::cout << "Healed " << amount << " HP. Current HP: " << hp << std::endl;

}

const std::string& MainHero::getName() const { return name; }

main\_hero.h

#ifndef MAIN\_HERO\_H

#define MAIN\_HERO\_H

#include "base\_character.h"

#include "weapon.h"

#include <vector>

#include <string>

class BaseEnemy;

class MainHero : public BaseCharacter {

private:

    std::string name;

    std::vector<Weapon> weapons;

    int current\_weapon\_index;

public:

    MainHero(int pos\_x, int pos\_y, const std::string& name, int hp);

    void hit(BaseEnemy& target);

    void add\_weapon(const Weapon& weapon);

    void next\_weapon();

    void heal(int amount);

    const std::string& getName() const;

};

#endif // MAIN\_HERO\_H

Main.cpp

#include <iostream>

#include "main\_hero.h"

#include "base\_enemy.h"

#include "weapon.h"

#include "exceptions.h"

int main() {

    try {

        // Создаем оружие

        Weapon sword("Sword", 30, 5);

        Weapon bow("Bow", 20, 15);

        Weapon axe("Axe", 40, 3);

        // Создаем героя

        MainHero hero(0, 0, "Edik", 100);

        // Создаем врагов

        BaseEnemy enemy1(3, 4, sword, 50);

        BaseEnemy enemy2(10, 10, bow, 60);

        BaseEnemy enemy3(2, 1, axe, 70);

        // Битва

        std::cout << "The battle starts!" << std::endl;

        // Герой без оружия

        hero.hit(enemy1);

        // Подбираем оружие

        hero.add\_weapon(sword);

        hero.add\_weapon(bow);

        // Атакуем врагов

        hero.hit(enemy1);

        hero.hit(enemy2); // Должен быть слишком далеко для меча

        // Меняем оружие

        hero.next\_weapon();

        hero.hit(enemy2); // Теперь можно достать луком

        // Лечение

        hero.heal(30);

        // Еще оружие

        hero.add\_weapon(axe);

        hero.next\_weapon();

        hero.hit(enemy3);

        // Проверяем состояние врагов

        std::cout << enemy1 << std::endl;

        std::cout << enemy2 << std::endl;

        std::cout << enemy3 << std::endl;

    } catch (const std::exception& e) {

        std::cerr << "Error: " << e.what() << std::endl;

    }

    return 0;

}

Weapon.cpp

#include "weapon.h"

#include "base\_character.h"

#include <cmath>

Weapon::Weapon(const std::string& name, int damage, int range)

    : name(name), damage(damage), range(range) {}

void Weapon::hit(BaseCharacter& actor, BaseCharacter& target) {

    if (!target.is\_alive()) {

        std::cout << "The enemy is already defeated" << std::endl;

        return;

    }

    double distance = sqrt(pow(actor.get\_x() - target.get\_x(), 2) +

                          pow(actor.get\_y() - target.get\_y(), 2));

    if (distance > range) {

        std::cout << "The enemy is too far for weapon " << name << std::endl;

        return;

    }

    std::cout << "The enemy got hit by weapon " << name

              << "and took " << damage << "damage" << std::endl;

    target.get\_damage(damage);

}

std::ostream& operator<<(std::ostream& os, const Weapon& weapon) {

    os << weapon.name;

    return os;

}

std::istream& operator>>(std::istream& is, Weapon& weapon) {

    is >> weapon.name >> weapon.damage >> weapon.range;

    return is;

}

const std::string& Weapon::getName() const { return name; }

int Weapon::getDamage() const { return damage; }

int Weapon::getRange() const { return range; }

weapon.h

#ifndef WEAPON\_H

#define WEAPON\_H

#include <string>

#include <iostream>

#include "base\_character.h"

class BaseCharacter; // Forward declaration

class Weapon {

private:

    std::string name;

    int damage;

    int range;

public:

    Weapon(const std::string& name, int damage, int range);

    void hit(BaseCharacter& actor, BaseCharacter& target);

    friend std::ostream& operator<<(std::ostream& os, const Weapon& weapon);

    friend std::istream& operator>>(std::istream& is, Weapon& weapon);

    const std::string& getName() const;

    int getDamage() const;

    int getRange() const;

};

#endif // WEAPON\_H

# Критерии оценки:

Если программа компилируется и оформлена по требованиям, то за лабораторную работу начисляются баллы.

Максимально можно получить 6 баллов:

* Задание 1 – 2 балла
* Задание 2 – 4 балла