МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №7

по дисциплине «Объектно-ориентированное программирование»

Тема: Написание исключений

Студент гр. 8303 _____ Парфентьев Л.М.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы

Научиться использовать исключения в C++.

Задание

Разработать и реализовать набор исключений. Основные требования к исключениям:

- Исключения покрывают как минимум все тривиальные случаи возникновения ошибки
- Все реализованные исключения обрабатываются в программе
- Исключения должны хранить подробную информацию об ошибке, а не только строку с сообщением об ошибке

Ход выполнения работы

- Было создано 2 класса исключений, возникающих при загрузке файла:
  - `NoSavegameFile`: указанный файл отсутствует;
  - `InvalidSaveFileContents`: не получилось загрузить сохранённую игру.
- Оба класса наследуются от `std::exception` из стандартной библиотеки C++.
- Оба класса содержат имя файла. `InvalidSaveFileContents` также содержит позицию в файле, на которой завершилось считывание содержимого файла.
- Оба класса содержат поле `_msg`, в которое записывается строка с сообщением при первом обращении к методу `what()`. Таким образом, указатель на строку, полученную при помощи `std::string::c_str`, остаётся валидным. Чтобы можно было писать в `_msg` из `const` метода `what()`, это поле объявлено как `mutable`.

Выводы

В ходе выполнения лабораторной работы были изучены исключения в языке C++.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: exceptions.hpp

```
#ifndef _H_EXCEPTIONS_HPP
#define _H_EXCEPTIONS_HPP

#include <string>
#include <sstream>
#include <stdexcept>


class NoSavegameFile: public std::exception {
    std::string _fn;

    mutable std::string _msg {""};

public:
    NoSavegameFile(const std::string &fn)
        :_fn{fn} {}

    virtual const char *
    what() const noexcept override
    {
        if (!_msg.length()) {
            std::ostringstream oss {};
            oss << "Can't open savegame file: " << _fn;
            _msg = oss.str();
        }
        return _msg.c_str();
    }
};

class InvalidSaveFileContents: public std::exception {
    using pos_t = std::istream::pos_type;

    std::string _fn;
    pos_t _pos;

    mutable std::string _msg {""};

public:
    InvalidSaveFileContents(const std::string &fn, pos_t pos)
        :_fn{fn}, _pos{pos} {}

    virtual const char *
    what() const noexcept override
    {
        if (!_msg.length()) {
```

```
            std::ostringstream oss {};
            oss << "Error while reading savegame file " << _fn;
            if (_pos >= 0) {
                oss << " around position " << _pos;
            }
            _msg = oss.str();
        }
        return _msg.c_str();
    }
};

#endif
```

Название файла: main.cpp

```
#include <string.h>

#include <stdexcept>
#include <iostream>
#include <fstream>

#include "demo.hpp"

#include "event_printer.hpp"
#include "game_driver.hpp"
#include "game_rules.hpp"
#include "exceptions.hpp"


void
run_demos(void)
{
    std::cout << "Demo 1\n";
    demo1();

    std::cout << "\nDemo 2\n";
    demo2();

    std::cout << "\nDemo 3\n";
    demo3();

    std::cout << "\nDemo 4\n";
    demo4();

    std::cout << "\nDemo 5\n";
    demo5();

    std::cout << "\nDemo 6\n";
    demo6();

    std::cout << "\nDemo 7\n";
    demo7();
```

```
    std::cout << "\nDemo 8\n";
    demo8();

    std::cout << "\nDemo 9\n";
    demo9();
}

Game *
loadGame(const std::string &load_fn)
{
    std::ifstream f {load_fn};
    if (!f) {
        throw NoSavegameFile {load_fn};
    }
    auto *tab = RestorerTable::defaultTable();
    Storable *s = tab->restore(f);
    delete tab;

    if (auto *lg = dynamic_cast<Game *>(s)) {
        return lg;
    } else {
        delete s;
        throw InvalidSaveFileContents {load_fn, f.tellg()};
    }
}

int
run_game(int argc, char **argv)
{
    std::vector<EventPrinter *> loggers {};
    bool have_stdout = false;

    const char *load_fn = nullptr;

    for (int i = 1; i < argc; ++i) {
        if (!strcmp(argv[i], "-log")) {
            char *fn = argv[++i];
            if (!strcmp(fn, "-")) {
                loggers.push_back(new LoggingEventPrinter {std::cout});
                have_stdout = true;
            } else {
                auto *of = new std::ofstream {fn};
                if (!*of) {
                    std::cerr << "Failed to open file: " << fn << "\n";
                    return 1;
                }
                loggers.push_back(new LoggingEventPrinter {of});
            }
        } else if (!strcmp(argv[i], "-load")) {
            load_fn = argv[++i];
        } else {
```

```
            std::cerr << "Unknown option: " << argv[i] << "\n";
            return 1;
        }
    }

    GameDriver<DefaultRules> drv {};

    for (auto *logger: loggers) {
        drv.addLogger(logger);
    }

    if (!have_stdout) {
        drv.setPrinter(new EventPrinter {std::cout});
    }

    if (load_fn) {
        try {
            drv.resetFrom(loadGame(load_fn));
        } catch (std::exception &e) {
            std::cerr << "An error occurred while loading savegame:\n"
                      << e.what() << "\n";
            return 1;
        }
    } else {
        drv.reset();
    }

    drv.run();

    return 0;
}

int
main(int argc, char **argv)
{
    if (argc == 2
        && !strcmp(argv[1], "-demo")) {
        run_demos();
        return 0;
    }

    return run_game(argc, argv);
}
```