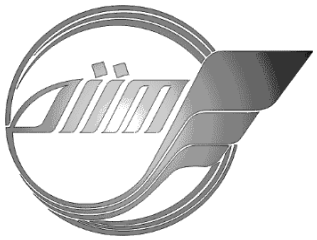
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



Дніпровський національний університет
залізничного транспорту імені академіка В. Лазаряна

Кафедра «Комп'ютерні інформаційні технології»

Лабораторна робота №10

з дисципліни «Об'єктно-орієнтоване програмування»

на тему: «Виключення»

Виконав:
студент гр.ПЗ1911
Сафонов Д. Є.
Прийняла:
Демидович І.М.

Дніпро, 2021

Тема. Виключення

Завдання. Написати програму на мові С++з декількох функцій, яка містить генерацію та відстеження виняткових ситуацій. Передбачити використання поліморфізму спадкування для класів-виключень.

Індивідуальне завдання. Клас та функцію обрати за власними вподобаннями для довільної предметної області.

Текст програми.

“main.cpp”

```
#include "exceptions.h"
#include "menu_choice.h"
#include "menu_io.h"
#include "read.h"
#include <cstddef>
#include <iostream>
#include <vector>

auto main() -> int {
	std::vector<int> items;

	auto item_idx_getter = [&] () -> std::string {return "size of array is: " + std::to_string(items.size()) + ", choose index: ";};
	std::vector<menu_choice> choices {
		menu_choice(
			"Insert value @index.",
			item_idx_getter,
			[&] (size_t idx) {
				if (idx > items.size()) {
					throw bad_item_idx();
				}
				std::cout << "Input a integer to insert @" << idx + 1 << ":" << std::endl;
				auto i = read<int>();
				items.insert(items.begin() + idx, i);
			}
		),
		menu_choice(
			"Erase value @index.",
			item_idx_getter,
			[&] (size_t idx) {
				if (idx >= items.size()) {
					throw bad_item_idx();
				}
				items.erase(items.begin() + idx);
```

```
                }
            )
        };

        auto is_running = true;
        while (is_running) {
            try {
                std::cout << "array: ";
                for (auto i : items) {
                    std::cout << i << " ";
                }
                std::cout << std::endl;
                auto menu_idx = read_menu_index(choices);
                auto item_idx = read_item_index(choices[menu_idx]);
                choices[menu_idx](item_idx);
            } catch (bad_str& e) {
                std::cout << "Invalid input!" << std::endl;
            } catch (bad_menu_idx& e) {
                std::cout << "Out of range menu choice!" << std::endl;
            } catch (bad_item_idx& e) {
                std::cout << "Out of range item choice!" << std::endl;
            }

            bool got_choice = false;

            while (!got_choice) {
                try {
                    is_running = read_exit_choice();
                    got_choice = true;
                } catch (bad_str& e) {
                    std::cout << "Invalid input, try again." << std::endl;
                }
            }
    }
}
```

“execptions.h”

```
#ifndef EXCEPTIONS_H

#define EXCEPTIONS_H

#include <exception>

class bad_str : std::exception {};
class bad_menu_idx : std::exception {};
class bad_item_idx : std::exception {};

#endif
```

“menu_choice.h”

```
#ifndef MENU_CHOICE_H

#define MENU_CHOICE_H

#include <cstddef>
#include <functional>
#include <string>

class menu_choice {
	std::string str;
	std::function<std::string()> prompt;
	std::function<void(size_t)> callable;
public:
	menu_choice(std::string str, std::function<std::string()> prompt, std::function<void(size_t)>
callable)
		: str(std::move(str)), prompt(std::move(prompt)), callable(std::move(callable)) {}

	void operator() (size_t idx) const {this->callable(idx);}
	[[nodiscard]] auto getStr() const -> std::string {return this->str;}
	[[nodiscard]] auto getPrompt() const -> std::string {return this->prompt();}
};
#endif
```

“menu_io.h”

```
/// @file menu_io.h

#ifndef MENU_IO_H
#define MENU_IO_H

#include "menu_choice.h"
#include <exception>
#include <vector>

/**
 * @brief Gets a >=0 index from std::cin.
 * @param[in] choices Vector of menu choices for user to choose from
 * @return Correct index to acces element from choices
 * @throws bad_menu_idx Thrown if user input is out of available range(size of choices vector)
 */
auto read_menu_index(const std::vector<menu_choice>& choices) -> size_t;


/**
 * @brief Reads a size_t value from std::cin.
 * @param[in] mc An object to getPrompt() from
 * @return size_t from std::cin
 * @throws bad_str Thrown if any errors occured while reading from std::cin
 * @details
 * Prints mc.getPrompt() to std::cout.
 * Reads size_t from std::cin, returns decremented value.
 * In case of any errors throws bad_str exception.
 */
auto read_item_index(const menu_choice& mc) -> size_t;


/**
 * @brief Returns false if program should stop it's process.
 * @return bool interpolated from std::cin
 * @throws bad_str Thrown in case of wrong input from std::cin
 * @details
 * Prompts user from std::cout.
 * Reads value from std::cin.
 * If value is 'Y' or 'y' returns true.
 * If value is 'N' or 'n' returns false.
 * In case of any other inputs throws bad_str exception.
 */
auto read_exit_choice() -> bool;
#endif
```

“read.h”

```
/// @file read.h

#ifndef READ_H
#define READ_H

#include "exceptions.h"
#include <iostream>
#include <menu_io.h>

/**
 * @brief Returns T object read from std::cin.
 * @tparam T Type to be read from std::cin
 * @return T object read from std::cin
 * @throws bad_str Thrown if any errors occured while reading from std::cin
 * @details
 * Tries to read T object from std::cin.
 * In case of any errors throws bad_str exception.
 */
template<class T>
auto read() -> T {
	T res;
	std::cin >> res;
	if (!std::cin || std::cin.get() != '\n') {
		std::cin.clear();
		while (std::cin.get() != '\n') {}
		throw bad_str();
	}
	return res;
}

#endif
```

“menu_io.cpp”

```
#include "exceptions.h"

#include "menu_choice.h"
#include "menu_io.h"
#include "read.h"
#include <iostream>

auto read_menu_index(const std::vector<menu_choice>& choices) -> size_t {
	for (size_t i = 0; i < choices.size(); i++) {
		std::cout << i + 1 << ". " << choices[i].getStr() << std::endl;
	}
	auto r = read<size_t>();
	if (r > choices.size()) {
		throw bad_menu_idx();
	}
	return r - 1;
}


auto read_item_index(const menu_choice& mc) -> size_t {
	std::cout << mc.getPrompt() << std::endl;
	return read<size_t>() - 1;
}


auto read_exit_choice() -> bool {
	std::cout << "Do you want to continue?(y/n)" << std::endl;
	auto res = read<char>();
	if (res == 'y' || res == 'Y') {
		return true;
	}
	if (res == 'n' || res == 'N') {
		return false;
	}
	throw bad_str();
}
```

Приклад роботи програми.

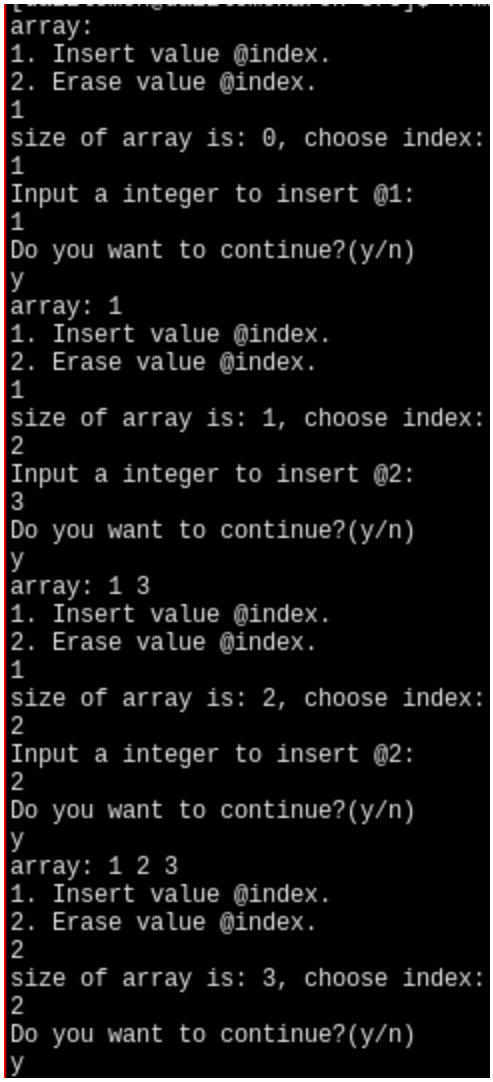


Рисунок 1(Коректні вхідні дані)

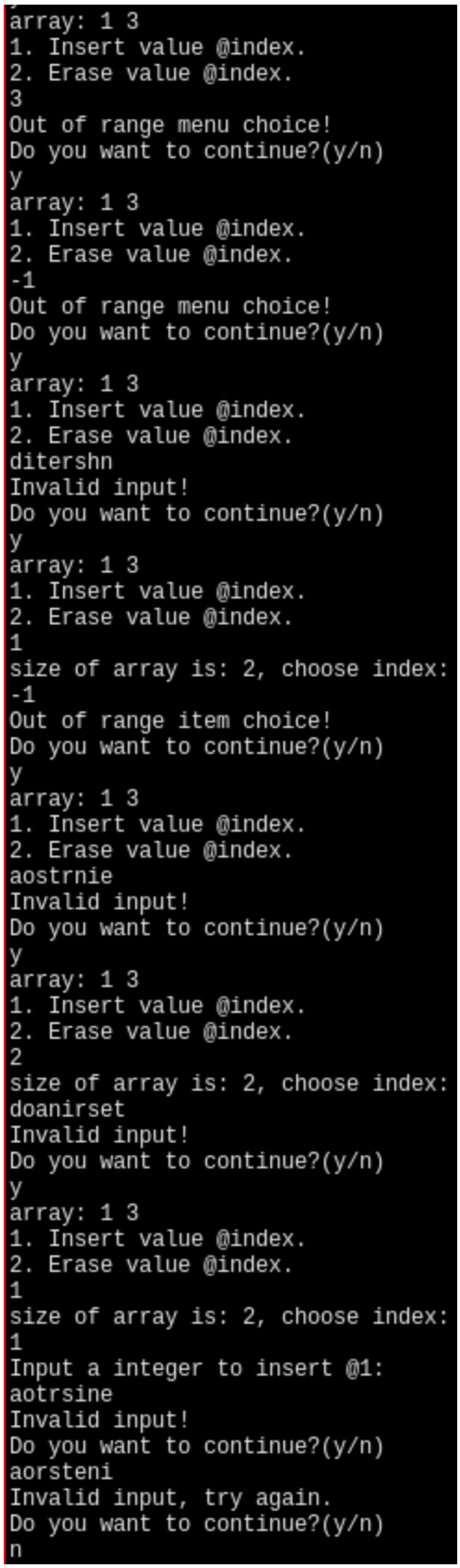


Рисунок 2(Некоректні вхідні дані)

Висновки.

Виключення у C++ та інших мовах програмування потрібні для спрощення обробки виняткових ситуацій та помилок. Але важливо розуміти коли доречно їх використання, не треба використовувати їх для перевірки коректності програміста, та того, як він використовує функцію. Найкраще їх використовувати для перевірки вхідних даних на граничні значень.