**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра інтелектуальних технологій**

**ЛАБОРАТОРНА РОБОТА №5**

з дисципліни «Об'єктно-орієнтоване програмування»

Тема роботи: «Бібліотека шаблонів»

Варіант №20

Виконала студентка

групи АнД-21

Радоманова С.П.

Перевірила:

Москаленко Н.В.

**Київ-2024**

Мета роботи є закріплення знань та набуття практичних навичок:

− створення класів, екземпляри яких будуть зберігатися в контейнерах

бібліотеки STL мови С++;

− створення функторів та лямбда-функцій;

− обробки виключних ситуацій;

− використання колекцій java;

− з технології серіалізації об’єктів java.

Перша частина звіту (програма С++).

Варіант 5. Worker: ПІБ, Посада (для переліку посад використати enum), Рік вступу на роботу, зарплата.

Опис структури класу:

**Клас:** Worker  
**Призначення:** Описує основні характеристики працівника, включаючи ім'я, прізвище, по батькові, посаду, рік прийому на роботу та зарплату. Реалізує функціонал для збереження, обробки та відображення інформації про працівника.

### **Атрибути:**

* char\* Name — ім'я працівника.
* char\* Surname — прізвище працівника.
* char\* Patronymic — по батькові працівника.
* position Position — посада працівника (Junior, Middle, Senior, Lead).
* int year — рік прийому на роботу.
* int salary — зарплата працівника.

### **Методи:**

#### **Геттери:**

* const char\* getName() — повертає ім'я працівника.
* const char\* getSurname() — повертає прізвище працівника.
* const char\* getPatronymic() — повертає по батькові працівника.
* position getPosition() — повертає посаду працівника.
* int getYear() — повертає рік прийому на роботу.
* int getSalary() — повертає зарплату працівника.

#### **Сеттери:**

* void setName(const char\* name) — встановлює ім'я працівника.
* void setSurname(const char\* surname) — встановлює прізвище працівника.
* void setPatronymic(const char\* patronymic) — встановлює по батькові працівника.
* void setPosition(position pos) — встановлює посаду працівника.
* void setYear(int y) — встановлює рік прийому на роботу.
* void setSalary(int sal) — встановлює зарплату працівника.

#### **Інші методи:**

* void printDetails() — виводить у консоль детальну інформацію про працівника.
* void writeToFile(const string& filename) — записує інформацію про працівника у файл.
* void readFromFile(const string& filename) — зчитує інформацію про працівника з файлу.

#### **Перевантаження операторів:**

* friend istream& operator>>(istream& is, Worker& p) — перевантаження оператора >> для введення інформації про працівника.
* friend ostream& operator<<(ostream& os, const Worker& w) — перевантаження оператора << для виведення інформації про працівника.

**Клас:** MyException  
**Призначення:** Реалізує власний клас виключень із можливістю збереження повідомлення про помилку.

### **Атрибути:**

* string message — повідомлення про помилку.

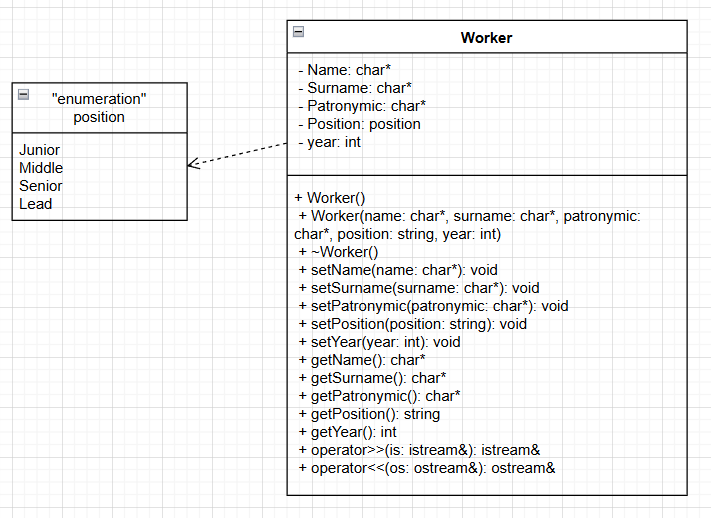
### **Методи:**

* MyException(const string& msg) — конструктор, що ініціалізує повідомлення.
* const char\* what() const override — повертає повідомлення про помилку.

**Додаткове:**

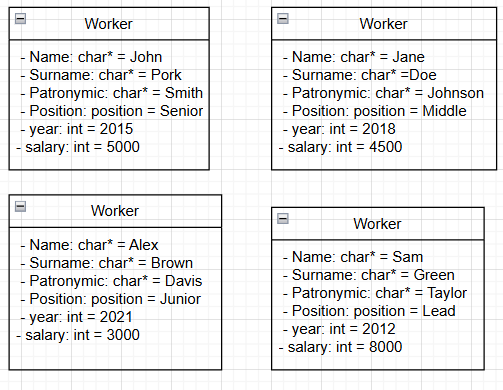
* enum position { Junior, Middle, Senior, Lead } — перелік для визначення посади працівника.
* Перевантажено оператор >> для enum position.

UML Діаграма класів:

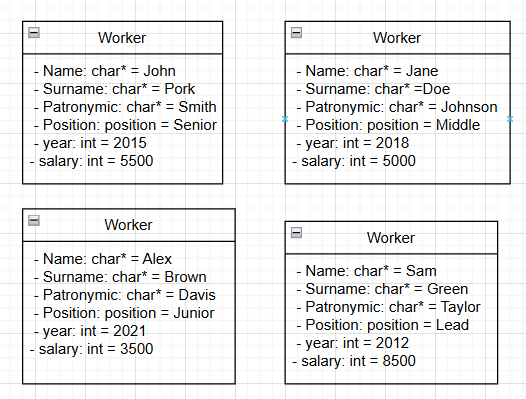


### 

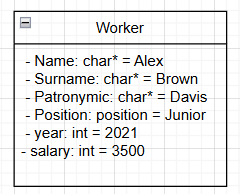
Спочатку створюємо 4 екземпляри класу:



Далі, за допомогою алгоритму, що модифікує for\_each з лямбда-функцією збільшуємо зарплати всіх працівників на 500.



Тепер використовуємо алгоритм видалення: Застосовуємо remove\_if з предикатом для видалення працівників із зарплатою більше 4000.



Надалі також використовуємо такі алгоритми:

Алгоритм перестановки:

* + За допомогою shuffle перемішуємо працівників випадковим чином.

Алгоритм сортування:

* + Сортуємо працівників за зарплатою з використанням функтора SalaryComparator.

Алгоритм для впорядкованих діапазонів:

* + Перевіряємо, чи є список працівників відсортованим за допомогою is\_sorted.

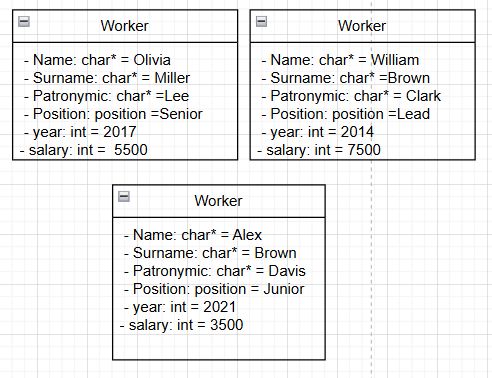
Чисельний алгоритм:

* + Використовуємо accumulate для підсумовування зарплат усіх працівників.

### Також в програмі використовуємо:

Вставний ітератор:

* + Для демонстрації вставки нових працівників у новий список використано вставний ітератор (insert\_iterator).



### Адаптер зворотного ітератора:

* + Виводимо працівників у зворотному порядку за допомогою ітераторів rbegin() та rend().

Потоковий ітератор:

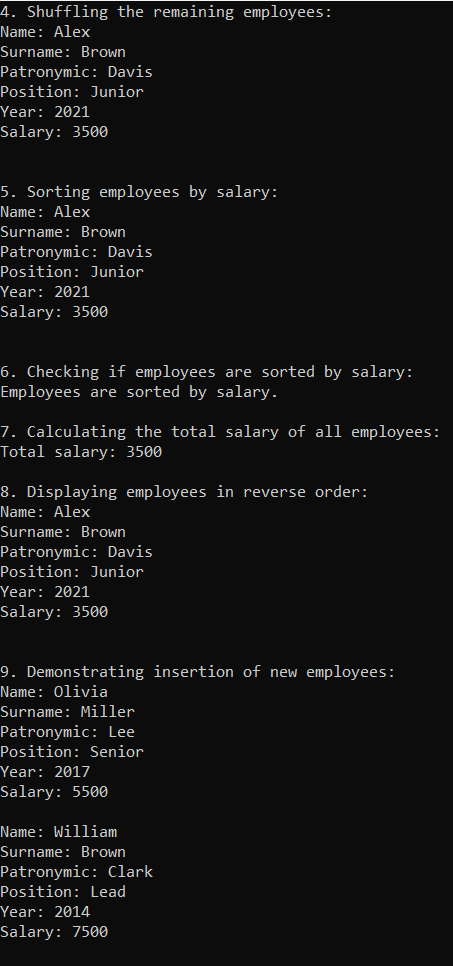
* + Для виведення колекції працівників у потік використовуємо ostream\_iterator.

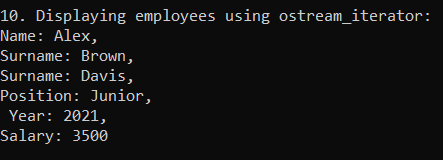
Алгоритм, що не модифікує:

* + За допомогою max\_element знаходимо працівника з максимальною зарплатою.

### 

Тестові приклади:



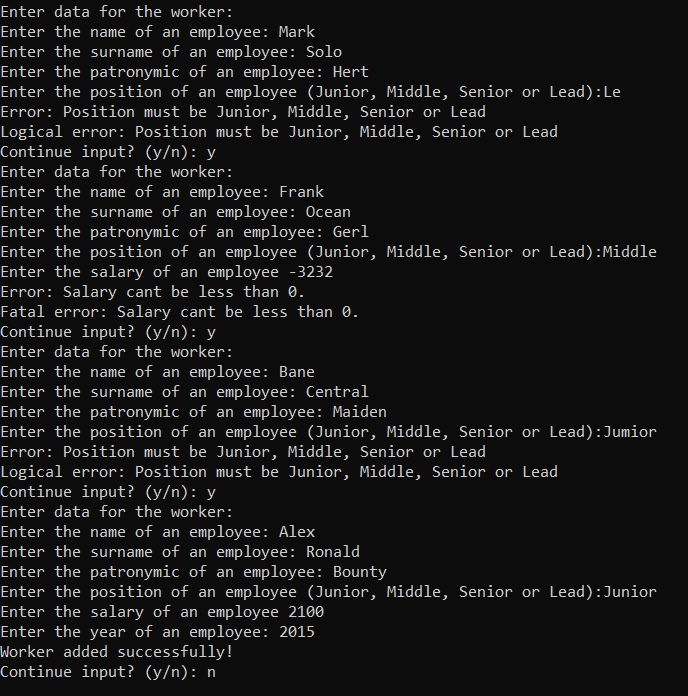


### 

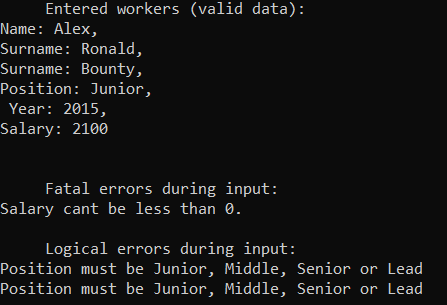
### 

### Тепер проведемо аналіз виключних ситуацій, які виникли під час роботи програми.

На вхід отримуємо такі дані:



Отримуємо результат:



В цьому завданні також використовуємо власний маніпулятор (мій додає табуляцію)

Нам успішно вивелась кількість базових помилок та помилок, спровокованих власним виключенням.

### Текст програми мовою С++:

**Worker.h**

#include <iostream>

#include <random>

#include <fstream>

#include <stdexcept> // Для invalid\_argument

using namespace std;

#pragma once

enum position { Junior, Middle, Senior, Lead };

//Власний клас виключень

class MyException : public exception {

private:

string message; // Член класу для зберігання повідомлення про помилку

public:

// Конструктор класу, який ініціалізує повідомлення

MyException(const string& msg) : message(msg) {}

// Переозначення методу what() для повернення повідомлення

const char\* what() const override {

return message.c\_str(); // Повертає C-стрічку з повідомленням

}

};

// Перевантаження оператор вводу для enum

istream& operator>>(istream& is, position& pos)

{

string input;

is >> input;

//Обробка власного класу виключень

try {

if (input != "Junior" && input != "Middle" && input != "Senior" && input != "Lead")

{

// Генерація виключення

throw MyException("Position must be Junior, Middle, Senior or Lead");

}

}

catch (const MyException& e) {

// Обробка виключення

cerr << "Error: " << e.what() << '\n';

throw;

}

// Порівнюємо введене значення та присвоюємо відповідний елемент enum

if (input == "Junior") {

pos = position::Junior;

}

else if (input == "Middle") {

pos = position::Middle;

}

else if (input == "Senior") {

pos = position::Senior;

}

else if (input == "Lead") {

pos = position::Lead;

}

return is;

}

class Worker {

private:

char\* Name;

char\* Surname;

char\* Patronymic;

position Position;

int year;

int salary;

public:

// Конструктор

Worker(const char\* name, const char\* surname, const char\* patronymic, position pos, int y, int sal) {

Name = new char[strlen(name) + 1];

strcpy\_s(Name, strlen(name) + 1, name);

Surname = new char[strlen(surname) + 1];

strcpy\_s(Surname, strlen(surname) + 1, surname);

Patronymic = new char[strlen(patronymic) + 1];

strcpy\_s(Patronymic, strlen(patronymic) + 1, patronymic);

Position = pos;

year = y;

salary = sal;

}

// Конструктор копіювання

Worker(const Worker& other) {

Name = new char[strlen(other.Name) + 1];

strcpy\_s(Name, strlen(other.Name) + 1, other.Name);

Surname = new char[strlen(other.Surname) + 1];

strcpy\_s(Surname, strlen(other.Surname) + 1, other.Surname);

Patronymic = new char[strlen(other.Patronymic) + 1];

strcpy\_s(Patronymic, strlen(other.Patronymic) + 1, other.Patronymic);

Position = other.Position;

year = other.year;

salary = other.salary;

}

// Оператор присвоєння

Worker& operator=(const Worker& other) {

if (this != &other) {

delete[] Name;

delete[] Surname;

delete[] Patronymic;

Name = new char[strlen(other.Name) + 1];

strcpy\_s(Name, strlen(other.Name) + 1, other.Name);

Surname = new char[strlen(other.Surname) + 1];

strcpy\_s(Surname, strlen(other.Surname) + 1, other.Surname);

Patronymic = new char[strlen(other.Patronymic) + 1];

strcpy\_s(Patronymic, strlen(other.Patronymic) + 1, other.Patronymic);

Position = other.Position;

year = other.year;

salary = other.salary;

}

return \*this;

}

// Сеттер для Name

void setName(const char\* name) {

delete[] Name;

Name = new char[strlen(name) + 1];

strcpy\_s(Name, strlen(name) + 1, name);

}

// Сеттер для Surname

void setSurname(const char\* surname) {

delete[] Surname;

Surname = new char[strlen(surname) + 1];

strcpy\_s(Surname, strlen(surname) + 1, surname);

}

// Сеттер для Patronymic

void setPatronymic(const char\* patronymic) {

delete[] Patronymic;

Patronymic = new char[strlen(patronymic) + 1];

strcpy\_s(Patronymic, strlen(patronymic) + 1, patronymic);

}

// Сеттер для Position

void setPosition(position pos) {

Position = pos;

}

// Сеттер для year

void setYear(int y) {

year = y;

}

// Сеттер для salary

void setSalary(int sal) {

salary = sal;

}

// Геттери

const char\* getName() const { return Name; }

const char\* getSurname() const { return Surname; }

const char\* getPatronymic() const { return Patronymic; }

int getSalary() const { return salary; }

int getYear() const { return year; }

const position getPosition() const { return Position; }

// Метод для виведення даних

void printDetails() const {

cout << "Name: " << Name << "\n";

cout << "Surname: " << Surname << "\n";

cout << "Patronymic: " << Patronymic << "\n";

cout << "Position: ";

switch (Position) {

case 0:

cout << "Junior\n";

break;

case 1:

cout << "Middle\n";

break;

case 2:

cout << "Senior\n";

break;

case 3:

cout << "Lead\n";

break;

}

cout << "Year: " << year << '\n';

cout << "Salary: " << salary << "\n" << "\n";

}

// Деструктор

~Worker() {

delete[] Name;

delete[] Surname;

delete[] Patronymic;

}

//перевантаження оператору вводу >>

friend istream& operator>>(istream& is, Worker& p) {

char buffer[256]; // Тимчасовий буфер для введення рядків

cout << "Enter the name of an employee: ";

is >> buffer; // Читаємо у тимчасовий буфер

delete[] p.Name; // Видаляємо стару пам’ять

p.Name = new char[strlen(buffer) + 1]; // Виділяємо нову

strcpy\_s(p.Name, strlen(buffer) + 1, buffer); // Копіюємо дані

cout << "Enter the surname of an employee: ";

is >> buffer;

delete[] p.Surname;

p.Surname = new char[strlen(buffer) + 1];

strcpy\_s(p.Surname, strlen(buffer) + 1, buffer);

cout << "Enter the patronymic of an employee: ";

is >> buffer;

delete[] p.Patronymic;

p.Patronymic = new char[strlen(buffer) + 1];

strcpy\_s(p.Patronymic, strlen(buffer) + 1, buffer);

cout << "Enter the position of an employee (Junior, Middle, Senior or Lead):";

is >> p.Position;

cout << "Enter the salary of an employee ";

is >> p.salary;

try {

if (p.salary < 0)

{

throw invalid\_argument("Salary cant be less than 0.");

}

}

catch (const invalid\_argument& e) {

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

// Скидаємо стан потоку та очищаємо буфер

is.clear(); // Скидає флаг помилки вводу

is.ignore(numeric\_limits<streamsize>::max(), '\n'); // Очищує залишок буфера

throw;

}

cout << "Enter the year of an employee: ";

is >> p.year;

try { // Перевірка на помилку вводу

if (cin.fail()) {

throw invalid\_argument("Year must be a valid number.");

}

// Перевірка на правильність діапазону року

if (p.year < 1900 || p.year > 2024) {

throw out\_of\_range("A year must be in range [1900; 2024].");

}

}

catch (const invalid\_argument& e) {

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

// Скидаємо стан потоку та очищаємо буфер

is.clear(); // Скидає флаг помилки вводу

is.ignore(numeric\_limits<streamsize>::max(), '\n'); // Очищує залишок буфера

throw;

}

catch (const out\_of\_range& e) {

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

// Скидаємо стан потоку та очищаємо буфер

is.clear(); // Скидає флаг помилки вводу

is.ignore(numeric\_limits<streamsize>::max(), '\n'); // Очищує залишок буфера

throw;

}

}

//перевантаження оператору виводу <<

friend ostream& operator<<(ostream& os, const Worker& w) {

os << "Name: " << w.Name << ", \n"

<< "Surname: " << w.Surname << ", \n"

<< "Surname: " << w.Patronymic << ", \n"

<< "Position: ";

if (w.Position == 0)

{

os << "Junior,\n ";

}

else if (w.Position == 1)

{

os << "Middle,\n";

}

else if (w.Position == 2)

{

os << "Senior,\n";

}

else if (w.Position == 3)

{

os << "Lead,\n";

}

os << "Year: " << w.year << ", \n";

os << "Salary: " << w.salary << "\n";

return os;

}

void writeToFile(const string& filename) {

ofstream ofs(filename); // Створюємо об'єкт вихідного потоку для запису у файл

if (ofs) { // Перевіряємо, чи вдалось відкрити файл

ofs << \*this; // Використовуємо перевантажений оператор << для запису об'єкта в файл

ofs.close(); // Закриваємо файл після запису

}

else {

cerr << "Error opening file for writing.\n"; // Якщо файл не вдалося відкрити, виводимо помилку

}

}

void readFromFile(const string& filename) {

ifstream ifs(filename); // Створюємо об'єкт вхідного потоку для читання з файлу

if (ifs) { // Перевіряємо, чи вдалось відкрити файл

ifs >> \*this; // Використовуємо перевантажений оператор >> для читання об'єкта з файлу

ifs.close(); // Закриваємо файл після читання

}

else {

cerr << "Error opening file for reading.\n"; // Якщо файл не вдалося відкрити, виводимо помилку

}

}

};

**main:**

#include <iostream>

#include <cstring>

#include <list>

#include <algorithm>

#include <numeric>

#include <functional>

#include <random>

#include <iterator>

#include <fstream>

#include <stdexcept> // Для invalid\_argument

#include "Worker.h"

using namespace std;

// Маніпулятор для додавання табуляції

ostream& add\_tab(ostream& os)

{

os << " ";

return os;

}

// Функтор для порівняння зарплати

struct SalaryComparator {

// Оператор виклику (перевантаження operator())

// Використовується для порівняння двох працівників по їх зарплаті

bool operator()(const Worker& w1, const Worker& w2) const {

return w1.getSalary() < w2.getSalary(); // Порівнюємо зарплати

}

};

// Лямбда-функція для визначення працівників з зарплатою більше 4000

auto salaryGreaterThan4000 = [](const Worker& w) {

return w.getSalary() > 4000; // Повертає true, якщо зарплата працівника більше 4000

};

int main() {

// Створення списку працівників

list<Worker> employees = {

Worker("John", "Pork", "Smith", Senior, 2015, 5000),

Worker("Jane", "Doe", "Johnson", Middle, 2018, 4500),

Worker("Alex", "Brown", "Davis", Junior, 2021, 3000),

Worker("Sam", "Green", "Taylor", Lead, 2012, 8000)

};

// 1. Алгоритм, що не модифікує: Пошук максимального елемента за допомогою лямбда-функції

cout << "1. Finding the employee with the highest salary:\n";

auto max\_elem = max\_element(employees.begin(), employees.end(), [](const Worker& a, const Worker& b) {

return a.getSalary() < b.getSalary(); // Порівнюємо зарплати

});

if (max\_elem != employees.end()) {

max\_elem->printDetails();

}

// 2. Алгоритм, що модифікує: Збільшення зарплати кожного працівника на 500

cout << "\n2. Increasing all employee salaries by 500:\n";

for\_each(employees.begin(), employees.end(), [](Worker& w) {

w.setSalary(w.getSalary() + 500);

});

for (const auto& worker : employees) {

worker.printDetails();

}

// 3. Алгоритм видалення: Видалення працівників із зарплатою більше 4000

cout << "\n3. Removing employees with salary > 4000:\n";

employees.remove\_if(salaryGreaterThan4000);

for (const auto& worker : employees) {

worker.printDetails();

}

// 4. Алгоритм перестановки: Перемішування працівників

cout << "\n4. Shuffling the remaining employees:\n";

vector<Worker> temp(employees.begin(), employees.end());

random\_device rd;

mt19937 g(rd());

shuffle(temp.begin(), temp.end(), g);

employees.assign(temp.begin(), temp.end());

for (const auto& worker : employees) {

worker.printDetails();

}

// 5. Алгоритм сортування: Сортування працівників за зарплатою

cout << "\n5. Sorting employees by salary:\n";

employees.sort(SalaryComparator());

for (const auto& worker : employees) {

worker.printDetails();

}

// 6. Алгоритм для впорядкованих діапазонів: Перевірка на відсортованість

cout << "\n6. Checking if employees are sorted by salary:\n";

if (is\_sorted(employees.begin(), employees.end(), SalaryComparator())) {

cout << "Employees are sorted by salary.\n";

}

else {

cout << "Employees are not sorted by salary.\n";

}

// 7. Чисельний алгоритм: Підсумовування зарплат

cout << "\n7. Calculating the total salary of all employees:\n";

int total\_salary = accumulate(employees.begin(), employees.end(), 0, [](int sum, const Worker& w) {

return sum + w.getSalary();

});

cout << "Total salary: " << total\_salary << "\n";

// Демонстрація зворотного ітератора

cout << "\n8. Displaying employees in reverse order:\n";

for (auto it = employees.rbegin(); it != employees.rend(); ++it) {

it->printDetails();

}

// Демонстрація вставки елементів за допомогою вставного ітератора

cout << "\n9. Demonstrating insertion of new employees:\n";

list<Worker> newEmployees;

insert\_iterator<list<Worker>> inserter(newEmployees, newEmployees.begin());

\*inserter = Worker("Olivia", "Miller", "Lee", Senior, 2017, 5500);

\*inserter = Worker("William", "Brown", "Clark", Lead, 2014, 7500);

for (const auto& worker : newEmployees) {

worker.printDetails();

}

// Демонстрація потокового ітератора

cout << "\n10. Displaying employees using ostream\_iterator:\n";

copy(employees.begin(), employees.end(), ostream\_iterator<Worker>(cout, "\n"));

list<Worker> validWorkers; // Об'єкти, введені коректно

list<string> fatalErrors; // Введення з фатальними помилками

list<string> logicalErrors; // Введення з логічними помилками

char continueInput = 'y';

while (continueInput == 'y')

{

try {

Worker w("", "", "", Junior, 0, 0); // Початковий об'єкт

cout << "Enter data for the worker:" << endl;

cin >> w; // Спроба введення працівника

// Якщо введення успішне, додаємо до списку коректних

validWorkers.push\_back(w);

cout << "Worker added successfully!" << endl;

}

catch (const MyException& e) {

// Логічна помилка, додаємо опис до logicalErrors

logicalErrors.push\_back(e.what());

cerr << "Logical error: " << e.what() << endl;

}

catch (const exception& e) {

// Фатальна помилка, додаємо опис до fatalErrors

fatalErrors.push\_back(e.what());

cerr << "Fatal error: " << e.what() << endl;

}

cout << "Continue input? (y/n): ";

cin >> continueInput;

}

// Виведення результатів

cout << '\n' << add\_tab << "Entered workers (valid data):" << endl;

for (const auto& worker : validWorkers) {

cout << worker << endl;

}

cout << '\n' << add\_tab << "Fatal errors during input:" << endl;

for (const auto& error : fatalErrors) {

cout << error << endl;

}

cout << '\n' << add\_tab << "Logical errors during input:" << endl;

for (const auto& error : logicalErrors) {

cout << error << endl;

}

return 0;

}

### 

### 