ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ І ЗВ'ЯЗКУ

Звіт

з дисципліни Проектний Практикум

Лабораторна робота №3

на тему: «Проектування і реалізація програми з масивами об'єктів, вказівниками та посиланням на них»»

Виконав: студент групи ІПЗ-3.04

Бухта М.М

_____

Перевірив: Багачук Д.Г.

_____

Одеса 2023

ВСТУП

Мета роботи – засвоєння поняття масиву об’єктів, вказівників та посилань на об’єкти класів; набуття практичних навичок ‘х оголошення та використання.

ЗАВДАННЯ

Розрбити і реалізувати програмний додаток нарахування заробітної плати співробітникам фірми. Для збереження даних про співробітників використати масиви об’єктів, а для доступу до функцій – вказівники. Для цього:

1. Розробити і реалізувати програмно клас «Особа»
2. Розробити і реалізувати програмно функції, які враховують таки вимоги:
   2.1. Кожному співробітнику заробітна плата нараховується, виходячи з тарифної сітки, у якій закладено таки позиції: «Спеціаліст», «Спеціаліст 1 категорії», «Спеціаліст 2 категорії», «Спеціаліст вищої категорії», «Начальник відділу», «Директор».
   2.2. Кожному співробітнику призначається додатково до ставки премія за принципом: якщо заробітна плата менша від середньої заробітної плати усіх співробітників, то розмір премії становить 50% від заробітної плати, в інших випадках – 30% від заробітної плати.
   2.3. Реалізувати програмно функцію, яка визначає суму, що видається суму, що видається на руки (заробітна плата плюс премія).
3. Продемонструвати роботи створеного класу і його функцій в основній частині програми

Порядок виконання роботи

1. Для реалізації програмного коду оголосити та визначити клас особа. У цього класі слід оголосити закриті змінні, що зберігають такі дані:

прізвище та ініціали співробітника, заробітну плату, премію, ставку, позицію з тарифної сітки.

2. Додати у клас відкриті конструктори. Один з конструкторів оголосити без параметрів, інший з параметрами для ініціалізації об'єктів класу. Конструктор не повинен ініціалізації значення ставки значення ставки, натомість ван має викликати функцію, яка її визначає.
3. Додати у клас деструктор.
4. Додати у клас функцію визначення ставки відповідно до позиції у тарифній сітці.
5. Додати у клас функцію призначення премій. Врахувати, що премія призначається за принципом: якщо заробітна плата менша, ніж середня зарплата усіх співробітників, то розмір премії становить 50% від заробітної плати, в інших випадках – 30% від заробітної плати.
6. Додати функцію визначення заробітної плати.
7. Додати функції виведення даних про співробітника.
8. В основній частині програми створити масив об'єктів. Доступ до елементів масиву показати через вказівник.
9. Програма повинна містить меню, яке дозволить перевірити основні функції, створені у програмі.

КОД ПРОГРАМИ

main.cpp

```
/***********************************
 * Laboratory work #3;             *
 * Student Bukhta Mykyta;          *
 * Grade: 3;                       *
 * Group Software Engineering 3.04; *
 ***********************************
 */

#include "MainMenu.hpp"

#include <iostream>

using namespace lab_3;

int main(int argc, char **argv) {
    MainMenu main_menu;
```

```
    main_menu.draw();

    return 0;
}
```

IMenu.hpp

```
/************************************
 * Laboratory work #3;              *
 * Student Bukhta Mykyta;           *
 * Grade: 3;                        *
 * Group Software Engineering 3.04; *
 ************************************
 */

#ifndef BUKHTAMYKYTA_LAB3_IMENU_HPP
#define BUKHTAMYKYTA_LAB3_IMENU_HPP

#include <inttypes.h>

namespace lab_3 {

class IMenu {
public:
    virtual void draw(void) = 0;
};

}   // !lab_3;

#endif  // !BUKHTAMYKYTA_LAB3_IMENU_HPP;
```

MainMenu.hpp

```
/************************************
 * Laboratory work #3;              *
 * Student Bukhta Mykyta;           *
 * Grade: 3;                        *
 * Group Software Engineering 3.04; *
 ************************************
 */

#ifndef BUKHTAMYKYTA_LAB3_MAINMENU_HPP
#define BUKHTAMYKYTA_LAB3_MAINMENU_HPP

#include "IMenu.hpp"
#include "Employee.hpp"

#include <deque>
#include <memory>
```

```
namespace lab_3 {

class MainMenu : public IMenu {
private:
    enum class TAB_ENUM {
        ADD_WORKER = 1,
        OUTPUT_WORKER_INFO,
        OUTPUT_WORKER_BONUS_INFO
    };

public:
    MainMenu(void);
    virtual ~MainMenu(void) = default;

    void draw(void) override;

private:
    void init_employees(void);
    void handle_tab(int16_t tab);
    void open_add_worker_submenu(void);
    void output_workers_info(void) const;
    void output_workers_bonus_info(void) const;

private:
    std::deque<std::shared_ptr<Employee>> m_employees;
};

}   // !lab_3;

#endif  // !BUKHTAMYKYTA_LAB3_MAINMENU_HPP;
```

MainMenu.cpp

```
#include "MainMenu.hpp"
#include "SalaryCalculator.hpp"

#include <iostream>
#include <limits>

namespace lab_3 {

MainMenu::MainMenu(void) {
    init_employees();
}

void MainMenu::draw(void) {
    static int16_t tab{0};
    std::cout << "1) Add worker;" << std::endl;
```

```
    std::cout << "2) Output worker info;"  << std::endl;
    std::cout << "3) Output worker info with bonuses;" << std::endl;
    std::cout << "Input: "; std::cin >> tab;
    handle_tab(tab);
}

void MainMenu::init_employees(void) {
    m_employees.resize(10);

    m_employees[0] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#1"}),
        common_types::POSITION::COMMON_SPECIALIST);
    m_employees[1] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#2"}),
        common_types::POSITION::COMMON_SPECIALIST);
    m_employees[2] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#3"}),
        common_types::POSITION::COMMON_SPECIALIST);
    m_employees[3] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#4"}),
        common_types::POSITION::FIRST_CATEGORY_SPECIALST);
    m_employees[4] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#5"}),
        common_types::POSITION::FIRST_CATEGORY_SPECIALST);
    m_employees[5] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#6"}),
        common_types::POSITION::SECOND_CATEGORY_SPECIALST);
    m_employees[6] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#7"}),
        common_types::POSITION::SECOND_CATEGORY_SPECIALST);
    m_employees[7] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#8"}),
        common_types::POSITION::HIGH_CATEGORY_SPECIALIST);
    m_employees[8] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#9"}),
        common_types::POSITION::BRANCH_MANAGER);
    m_employees[9] =
std::make_shared<Employee>(std::move(common_types::FullName{"Worker", "#10"}),
        common_types::POSITION::DIRECTOR);
}

void MainMenu::handle_tab(int16_t tab) {
    switch(static_cast<TAB_ENUM>(tab)) {
    case TAB_ENUM::ADD_WORKER:
        open_add_worker_submenu();
        break;
    case TAB_ENUM::OUTPUT_WORKER_INFO:
        output_workers_info();
```

```
        break;
    case TAB_ENUM::OUTPUT_WORKER_BONUS_INFO:
        output_workers_bonus_info();
        break;
    default:
        std::wcout << "Unknown value!" << std::endl;
    }

    this->draw();
}

void MainMenu::open_add_worker_submenu(void) {
    common_types::FullName full_name;
    uint16_t position;

    std::cout << "Input First Name:\n";
    std::getline(std::cin,
full_name.first_name).ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
    std::cout << "Input Second Name:\n";
    std::getline(std::cin, full_name.second_name);
    std::cout << "Input Middle Name:\n";
    std::getline(std::cin, full_name.middle_name);
    do {    // Input position number;
        std::cout << "Input position list:" << std::endl
        << "0 - COMMON_SPECIALIST" << std::endl
        << "1 - FIRST_CATEGORY_SPECIALST" << std::endl
        << "2 - SECOND_CATEGORY_SPECIALST" << std::endl
        << "3 - HIGH_CATEGORY_SPECIALIST" << std::endl
        << "4 - BRANCH_MANAGER" << std::endl
        << "5 - DIRECTOR" << std::endl
        << "Input: ";
    std::cin >> position;
    }   while (position >=
static_cast<uint16_t>(common_types::POSITION::SIZE));

    m_employees.push_back(std::make_shared<Employee>(full_name,
static_cast<common_types::POSITION>(position)));
}

void MainMenu::output_workers_info(void) const{
    for (const auto &item : m_employees) {
        std::cout << item->full_name() << '\t' <<
common_types::to_string(item->position())
            << "\tsalary: " << item->salary() << std::endl;
    }
}
```

```
void MainMenu::output_workers_bonus_info(void) const{
    auto salary_calculator = SalaryCalculator::instance();

    for (const auto &item : m_employees) {
        std::cout << item->full_name() << '\t' <<
common_types::to_string(item->position())
            << "\tsalary with bonus: " << salary_calculator-
>calculate_with_bonus(*item, m_employees) << std::endl;
    }
}

}   // !lab_3;
```

Employee.hpp

```
/***********************************
 * Laboratory work #3;             *
 * Student Bukhta Mykyta;          *
 * Grade: 3;                       *
 * Group Software Engineering 3.04; *
 ***********************************
 */

#ifndef BUKHTAMYKYTA_LAB3_EMPLOYEE_HPP
#define BUKHTAMYKYTA_LAB3_EMPLOYEE_HPP

#include "CommonTypes.hpp"

#include <inttypes.h>

namespace lab_3 {

class Employee {
public:
    Employee(common_types::POSITION position =
common_types::POSITION::COMMON_SPECIALIST);
    Employee(const common_types::FullName &full_name, common_types::POSITION
position = common_types::POSITION::COMMON_SPECIALIST);
    virtual ~Employee(void) = default;

    // Getters starts;

    virtual uint32_t salary(void) const noexcept;
    virtual common_types::POSITION position(void) const noexcept;
    virtual common_types::FullName full_name(void) const;

    // Getters ends;

    // Setters starts;
```

```
    virtual void set_salary(uint32_t val) noexcept;
    virtual void set_position(common_types::POSITION val) noexcept;
    virtual void set_full_name(const common_types::FullName &val);
    virtual void set_full_name(common_types::FullName &&val);

    // Setters ends;

private:
    uint32_t m_salary;
    common_types::POSITION m_position;
    common_types::FullName m_full_name;
};

}   // !lab_3;

#endif  // !BUKHTAMYKYTA_LAB3_EMPLOYEE_HPP;
```

Employee.cpp

```
#include "Employee.hpp"
#include "SalaryCalculator.hpp"

namespace lab_3 {

Employee::Employee(common_types::POSITION position)
    : Employee({}, position)
{

}

Employee::Employee(const common_types::FullName &full_name,
common_types::POSITION position)
    : m_full_name(full_name), m_position(position)
{
    auto salary_calculator = SalaryCalculator::instance();
    this->set_salary(salary_calculator->calculate(*this));
}

// Getters starts;

uint32_t Employee::salary(void) const noexcept {
    return m_salary;
}

common_types::POSITION Employee::position(void) const noexcept {
    return m_position;
}
```

```cpp
common_types::FullName Employee::full_name(void) const {
    return m_full_name;
}

// Getters ends;

// Setters starts;

void Employee::set_salary(uint32_t val) noexcept {
    m_salary = val;
}

void Employee::set_position(common_types::POSITION val) noexcept {
    m_position = val;
}

void Employee::set_full_name(const common_types::FullName &val) {
    m_full_name = val;
}

void Employee::set_full_name(common_types::FullName &&val) {
    m_full_name = std::move(val);
}

// Setters ends;

}   // !lab_3;
```

SalaryCalculator.hpp

```cpp
/***********************************
 * Laboratory work #3;             *
 * Student Bukhta Mykyta;          *
 * Grade: 3;                       *
 * Group Software Engineering 3.04; *
 ***********************************
 */

#ifndef BUKHTAMYKYTA_LAB3_SALARYCALCULATOR_HPP
#define BUKHTAMYKYTA_LAB3_SALARYCALCULATOR_HPP

#include "Employee.hpp"

#include <unordered_map>
#include <deque>
#include <memory>
#include <inttypes.h>

namespace lab_3 {
```

```
class SalaryCalculator {
public:
    static SalaryCalculator* instance(void);
    uint32_t calculate(const Employee &employee) const;
    uint32_t calculate_with_bonus(const Employee &employee, const
std::deque<std::shared_ptr<Employee>> employees) const;

private:
    SalaryCalculator(void);

private:
    std::unordered_map<common_types::POSITION, int32_t> m_salaries;
};

}   // !lab_3;

#endif  // !BUKHTAMYKYTA_LAB3_SALARYCALCULATOR_HPP;
```

SalaryCalculator.cpp

```
#include "SalaryCalculator.hpp"

namespace lab_3 {

SalaryCalculator* SalaryCalculator::instance(void) {
    static SalaryCalculator instance;
    return &instance;
}

uint32_t SalaryCalculator::calculate(const Employee &employee) const {
    auto salary_rate = m_salaries.find(employee.position());

    if (salary_rate == m_salaries.end()) {
        return 0;
    }

    return salary_rate->second;
}

uint32_t SalaryCalculator::calculate_with_bonus(const Employee &employee,
const std::deque<std::shared_ptr<Employee>> employees) const {
    uint64_t monthly_salary{0};
    uint32_t avarage_salary{0};
    uint32_t bonus{0};
    uint32_t employee_salary{employee.salary()};

    for (const auto &empl : employees) {
        monthly_salary += empl->salary();
```

```
    }
    avarage_salary = monthly_salary / employees.size();

    employee_salary < avarage_salary
        ? bonus = employee_salary / 2
        : bonus = employee_salary / 100 * 30;

    return employee_salary + bonus;
}

SalaryCalculator::SalaryCalculator(void) {
    m_salaries[common_types::POSITION::COMMON_SPECIALIST] = 10000;
    m_salaries[common_types::POSITION::FIRST_CATEGORY_SPECIALST] = 12500;
    m_salaries[common_types::POSITION::SECOND_CATEGORY_SPECIALST] = 15000;
    m_salaries[common_types::POSITION::HIGH_CATEGORY_SPECIALIST] = 17500;
    m_salaries[common_types::POSITION::BRANCH_MANAGER] = 20000;
    m_salaries[common_types::POSITION::DIRECTOR] = 22500;
}

}   // !lab_3;
```

CommonTypes.hpp

```
/***********************************
 * Laboratory work #3;             *
 * Student Bukhta Mykyta;          *
 * Grade: 3;                       *
 * Group Software Engineering 3.04; *
 ***********************************
 */

#ifndef BUKHTAMYKYTA_LAB3_COMMONTYPES_HPP
#define BUKHTAMYKYTA_LAB3_COMMONTYPES_HPP

#include <string>
#include <ostream>

namespace lab_3 {
namespace common_types {

struct FullName {
    std::string first_name;
    std::string second_name;
    std::string middle_name;

    friend std::ostream& operator<<(std::ostream &out, const FullName &value);
};

enum class POSITION {
```

```
    COMMON_SPECIALIST,
    FIRST_CATEGORY_SPECIALST,
    SECOND_CATEGORY_SPECIALST,
    HIGH_CATEGORY_SPECIALIST,
    BRANCH_MANAGER,
    DIRECTOR,
    SIZE
};

std::string to_string(POSITION val);

}   // !common_types;
}   // !lab_3;

#endif  // !BUKHTAMYKYTA_LAB3_COMMONTYPES_HPP;
```

CommonTypes.cpp

```
#include "CommonTypes.hpp"

namespace lab_3 {
namespace common_types {

std::ostream& operator<<(std::ostream &out, const FullName &value) {
    out << value.second_name << ' ' << value.first_name;
    if (value.middle_name.size()) {
        out << ' ' << value.middle_name;
    }

    return out;
}

std::string to_string(POSITION val) {
    switch(val) {
    case POSITION::COMMON_SPECIALIST:
        return "COMMON_SPECIALIST";
    case POSITION::FIRST_CATEGORY_SPECIALST:
        return "FIRST_CATEGORY_SPECIALST";
    case POSITION::SECOND_CATEGORY_SPECIALST:
        return "SECOND_CATEGORY_SPECIALST";
    case POSITION::HIGH_CATEGORY_SPECIALIST:
        return "HIGH_CATEGORY_SPECIALIST";
    case POSITION::BRANCH_MANAGER:
        return "BRANCH_MANAGER";
    case POSITION::DIRECTOR:
        return "DIRECTOR";
    }

    return {};
```

```
}

}   // !common_types;
}   // !lab_3;
```

РЕЗУЛЬТАТ ВИКОНАННЯ

```
1) Add worker;
2) Output worker info;
3) Output worker info with bonuses;
Input: 1
Input First Name:
TestName
Input Second Name:
TestSecondName
Input Middle Name:
SomeMiddleNmae
Input position list:
0 - COMMON_SPECIALIST
1 - FIRST_CATEGORY_SPECIALST
2 - SECOND_CATEGORY_SPECIALST
3 - HIGH_CATEGORY_SPECIALIST
4 - BRANCH_MANAGER
5 - DIRECTOR
Input: 4
1) Add worker;
2) Output worker info;
3) Output worker info with bonuses;
Input: 2
#1 Worker       COMMON_SPECIALIST       salary: 10000
#2 Worker       COMMON_SPECIALIST       salary: 10000
#3 Worker       COMMON_SPECIALIST       salary: 10000
#4 Worker       FIRST_CATEGORY_SPECIALST        salary: 12500
#5 Worker       FIRST_CATEGORY_SPECIALST        salary: 12500
#6 Worker       SECOND_CATEGORY_SPECIALST       salary: 15000
#7 Worker       SECOND_CATEGORY_SPECIALST       salary: 15000
#8 Worker       HIGH_CATEGORY_SPECIALIST        salary: 17500
#9 Worker       BRANCH_MANAGER  salary: 20000
#10 Worker      DIRECTOR        salary: 22500
TestSecondName  SomeMiddleNmae  BRANCH_MANAGER  salary: 20000
1) Add worker;
2) Output worker info;
3) Output worker info with bonuses;
Input: 3
#1 Worker       COMMON_SPECIALIST       salary with bonus: 15000
#2 Worker       COMMON_SPECIALIST       salary with bonus: 15000
#3 Worker       COMMON_SPECIALIST       salary with bonus: 15000
#4 Worker       FIRST_CATEGORY_SPECIALST        salary with bonus: 18750
#5 Worker       FIRST_CATEGORY_SPECIALST        salary with bonus: 18750
#6 Worker       SECOND_CATEGORY_SPECIALST       salary with bonus: 19500
#7 Worker       SECOND_CATEGORY_SPECIALST       salary with bonus: 19500
#8 Worker       HIGH_CATEGORY_SPECIALIST        salary with bonus: 22750
#9 Worker       BRANCH_MANAGER  salary with bonus: 26000
#10 Worker      DIRECTOR        salary with bonus: 29250
TestSecondName  SomeMiddleNmae  BRANCH_MANAGER  salary with bonus: 26000
```

КОНТРОЛЬНІ ЗАВДАННЯ ТА ЗАПИТАННЯ

1. Як оголосити вказівник на об’єкт?
   Type *obj_ptr = &obj; // Декларація та ініціалізація вказівника
2. Що таке вказівник this?
   Це вказівник на осередок пам’яті, де ініціалізований об’єкт типу класа
3. Як звернутися у програмі до елементів масиву об’єктів.
   Щоб звернутися до елементу масиву любого об’єкта, треба узяти вказівник на перший елемент масиву та додати до нього індекс елемента:
   Type type_array[10];
   (type_array + 3)->do_some_method(); // звернення до четвертого елемента масиву
   *(type_array + 3).do_some_method(); // або
   type_array[3].so_some_method // або
4. Як отримати доступ елементу масива об’єктів до відкритих членів класу.

Щоб отримати доступ до відкрити членів класу масива, треба звернутися до потрібного елементу масиву, як показано у питанні 3 та через крапку або стрілку (відповідно до типу об’єкту: вказівник чи об’єкт) обратитися до відкритого члену класу. Приклад у питання 3.

5. Якими способами можна виконати ініціалізацію масиву об’єктів?
   a. Під час декларації масиву:
      int32_t array[5] = {1, 2, 3, 4, 5};
   b. Через цикл
      int32_t array[5];
      for (int32_t i{0}; i < 5; ++i) { array[i] = i; }
   c. memset();
      int32_t array[5];
      memset(array, 0, 5 * sizeof(*array));
6. Як оголосити двомірний масив об’єктів?
   Потрібно два рази указати розмір масиву. Перший – кількість рядків, другий – кількість стовпців
   int32_t array[5][2]; // масив 5x2

7. Дано прототип конструктора myclass(int, char, char*). Оголосить масив чотирьох об’єктів
   myclass myclass_arr[4];