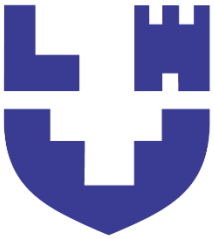
Міністерство освіти і науки України



ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Методичні вказівки до лабораторних занять
для здобувачів першого (бакалаврського) рівня вищої освіти
освітньої програми «Комп’ютерна інженерія»
галузь знані F Інформаційні технології
спеціальності F7 Комп’ютерна інженерія
денної та заочної форм навчання

Луцьк 2026

УДК 004.041(07)
О 13

Рекомендовано до видання вченою радою факультету КІТ ЛНТУ,
протокол № від « » 20 26 року.

Голова вченої ради факультету КІТ Інна КОНДІУС

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ

Директор бібліотеки Наталія ПОЛІЩУК

Розглянуто і схвалено на засіданні кафедри комп'ютерної інженерії та охоронних систем ЛНТУ,
протокол № від « 06 » « 11 » 20 26 року.

Завідувач кафедри комп'ютерної інженерії та охоронних систем ЛНТУ Тарас ТЕРЛЕЦЬКИЙ

Укладач: Петро ПЕХ, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та охоронних систем ЛНТУ

Рецензент: Світлана ЛАВРЕНЧУК, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та охоронних систем ЛНТУ

Відповідальний за випуск: Тарас ТЕРЛЕЦЬКИЙ, кандидат технічних наук, доцент комп'ютерної інженерії та охоронних систем ЛНТУ

О 13 Об'єктно-орієнтоване програмування: методичні вказівки до лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Комп'ютерна інженерія» галузі знань F Інформаційні технології спеціальності F7 Комп'ютерна інженерія денної та заочної форм навчання / уклад. П.А.Пех. Луцьк: ЛНТУ, 2026.186 с.

Методичне видання до лабораторних занять з дисципліни «Об'єктно-орієнтоване програмування» складено відповідно до діючої програми курсу.

Призначене для здобувачів вищої освіти спеціальності F7 Комп'ютерна інженерія освітньої програми «Комп'ютерна інженерія».

ЗМІСТ

ВСТУП *4*

ЛАБОРАТОРНА РОБОТА №1. ТЕМА «ОСОБЛИВОСТІ МОВИ С++ ТА ОСНОВИ ООП» *6*

ЛАБОРАТОРНА РОБОТА №2. ТЕМА «КЛАСИ ТА ОБ'ЄКТИ» *14*

ЛАБОРАТОРНА РОБОТА №3. ТЕМА «ІНКАПСУЛЯЦІЯ ТА МЕТОДИ КЛАСІВ» *24*

ЛАБОРАТОРНА РОБОТА №4. ТЕМА «КОНСТРУКТОРИ, ДЕСТРУКТОРИ ТА ЖИТТЄВИЙ ЦИКЛ ОБ'ЄКТІВ» *33*

ЛАБОРАТОРНА РОБОТА №5. ТЕМА «ДИНАМІЧНА ПАМ'ЯТЬ ТА КЕРУВАННЯ РЕСУРСАМИ» *44*

ЛАБОРАТОРНА РОБОТА №6. ТЕМА «ПЕРЕВАНТАЖЕННЯ ФУНКЦІЙ ТА ОПЕРАЦІЙ» *54*

ЛАБОРАТОРНА РОБОТА №7. ТЕМА «ДРУЖНІ ФУНКЦІЇ ТА ДРУЖНІ КЛАСИ» *61*

ЛАБОРАТОРНА РОБОТА №8. ТЕМА "УСПАДКУВАННЯ КЛАСІВ *69*

ЛАБОРАТОРНА РОБОТА №9. ТЕМА «ПОЛІМОРФІЗМ, ВІРТУАЛЬНІ ФУНКЦІЇ ТА АБСТРАКТНІ КЛАСИ *77*

ЛАБОРАТОРНА РОБОТА №10. ТЕМА «ШАБЛОНИ ТА УЗАГАЛЬНЕНЕ ПРОГРАМУВАННЯ» *91*

ЛАБОРАТОРНА РОБОТА №11. ТЕМА «БІБЛІОТЕКА СТАНДАРТНИХ ШАБЛОНІВ STL: КОНТЕЙНЕРИ ТА ІТЕРАТОРИ» *111*

ЛАБОРАТОРНА РОБОТА №12. ТЕМА «БІБЛІОТЕКА СТАНДАРТНИХ ШАБЛОНІВ STL: АЛГОРИТМИ, ПРЕДИКАТИ ТА ЛЯМБДА_ФУНКЦІЇ» *122*

ЛАБОРАТОРНА РОБОТА №13. ТЕМА "ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ ТА ПОТОКОВЕ ВВЕДЕННЯ-ВИВЕДЕННЯ *138*

ЛАБОРАТОРНА РОБОТА №14. ТЕМА «ФАЙЛИ ТА UML-МОДЕЛЮВАННЯ ООП-СИСТЕМ» *154*

ЛАБОРАТОРНА РОБОТА №15. ТЕМА «БАГАТОФАЙЛОВІ ПРОГРАМИ ТА СТРУКТУРА ООП-ПРОЄКТУ *172*

РЕКОМЕНДОВАНА ЛІТЕРАТУРА *195*

ВСТУП

Метою цих методичних вказівок є забезпечення студентів спеціальності F7 «Комп'ютерна інженерія» методичними матеріалами, необхідними під час виконання лабораторних робіт з дисципліни «Об'єктно-орієнтоване програмування». Організація ефективного виконання лабораторних робіт спрямована на оволодіння студентами необхідними навиками у вирішенні складних завдань за рахунок використання сучасного програмного забезпечення.

Сформульована вище мета досягається за рахунок:

- поглибленого вивчення основ алгоритмізації інформаційних процесів;
- оволодіння сучасними технологіями програмування;
- вивчення принципів функціонування сучасних систем програмування, що дозволяють швидко і на сучасному рівні створювати прикладне програмне забезпечення;
- оволодіння практичними навичками розробки надійних і ефективних програм в Visual Studio та інших сучасних середовищах програмування.

Приступаючи до виконання лабораторної роботи з дисципліни «Об'єктно-орієнтоване програмування», студент повинен:

- опрацювати теоретичний матеріал з даної теми в обсязі, передбаченим навчальним планом для студентів даної спеціальності;
- уміти володіти комп'ютером на достатньому рівні;
- мати навички роботи з операційною системою Windows та офісними додатками, володіти знаннями, отриманими під час вивчення дисциплін «Програмування» иа «Структури даних та алгоритми».

Знання, отримані студентами в рамках виконання лабораторних робіт з дисципліни «Об'єктно-орієнтоване програмування», стануть базою для вивчення інших дисциплін фундаментального та професійного блоку.

Дані методичні вказівки містять 15 лабораторних робіт. З цією метою студент має створити папку з назвою lab_works, у якій у підлеглих папках з назвами lab_work_n (де n – номер лабораторної роботи), мають бути збережені розроблені програмні проекти під назвами lab_work_n_m (де m – номер завдання даної лабораторної роботи).

Окрім завдань лабораторної роботи, виконання яких наводяться у тексті методичних вказівок, студент повинен виконати завдання за індивідуальним варіантом, що задається в кінці кожної теми. З цією метою студент має створити папку з назвою ind_works, у якій у підлеглих папках з назвами ind_work_n (де n – номер індивідуальної роботи), мають бути збережені розроблені програмні проекти під назвами ind_work_n_m (де m – номер завдання даної роботи).

Проекти повинні бути протестовані за різних наборів вхідних даних. Невідлагоджені та непротестовані програмні проекти до захисту не допускаються.

До програм, розроблених у процесі виконання лабораторних робіт з дисципліни «Об'єктно-орієнтоване програмування» висувається ряд вимог, яких студент має дотримуватися. Програмні проекти повинні:

- бути виконані у відповідності до умов завдання;
- бути самостійно розробленими та протестованими;
- ґрунтуватися на результатах самостійної роботи (дослідженнях);
- бути оформленими за стандартами і мати необхідний обсяг;
- бути виконаними і захищеними в зазначені терміни.

Програмні проекти, виконані не за варіантом, до захисту не допускаються.

ЛАБОРАТОРНА РОБОТА №1. ТЕМА «ОСОБЛИВОСТІ МОВИ С++ ТА ОСНОВИ ООП»

МЕТА РОБОТИ

1. Вивчити формати оголошення класів та об’єктів засобами мови С++.
2. З’ясувати суть принципу інкапсуляції методів та даних.
3. З’ясувати призначення конструкторів та дестукторів об’єктів класу.
4. Вивчити способи доступу до об’єктів класу.

ХІД РОБОТИ

Завдання 1.1. Завантажити, вивчити та протестувати програму, яка демонструє принцип інкапсуляції відкритих методів (функцій) та закритих даних в одному об'єкті класу

```
// lab_work_01_01.cpp Deitel_9_1_9_3

// Програма демонструє принцип інкапсуляції
// відкритих методів (функцій) та закритих даних в одному об'єкті класу
// з метою захисту цих закритих даних від несанкцонованого доступу
користувачів.
// З цією метою пропонується клас Time (варіант 1).

// Клас Time дозволяє встановити значення та вивести на друк час
// (у вигляді послідовності чисел, які означають годину, хвилину, секунду)
// у двох форматах - всесвітньому ( 00:00:00) та стандартному ( 00:00:00
AM/PM).
// Клас Time має чотири відкриті елементи-функції (методи):
// Time, setTime, printUniversal, printStandard
// та три закриті елементи-дані: hour, minute, second.

// Конструктор класу Time ініціалізує закриті дані нульовими значеннями.
// Дані класу повинні бути коректними (узгодженими) -
// нульові значення закритих даних класу Time є корректними.
// Закриті дані не слід ініціалізувати безпосередньо в розділі private,
// це слід робити за допомогою функцій класу, зокрема, конструктора.

// Функція setTime(int h, int m, int s) встановлює
// коректні значення закритих даних hour, minute, second,
// слідкуючи за тим, щоб величина hour лежала в межах (0-24),
// величина minute лежала в межах (0-59), величина second лежала в межах
(0-59).
// Під час спроби ввести некоректне значення кожного закритого даного,
// воно автоматично замінюється на нуль.

// Функція printUniversal() виводить на друк час
// у всесвітньому форматі ( 00:00:00).
// Наприклад, 18:23:45 буде надруковано 18:23:45
// 99:99:99 буде надруковано 00:00:00

// Функція printStandard() виводить на друк час
// у стандартному форматі ( 00:00:00 AM/PM).
```

```
// Наприклад, 18:23:45 буде надруковано у вигляді 06:23:45 PM
// 99:99:99 буде надруковано12:00:00 AM

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Time
{
public:
    Time() { // конструктор
        hour = minute = second = 0;
    }; // кінець конструктор
    void setTime(int, int, int); // встановити годину, хвилину, секунду
    void printUniversal(); // надрукувати час у форматі всесвітнього часу
    void printStandard(); // надрукувати час у форматі стандартного часу

private:
    int hour = 0; // 0-23 (24-годинний формат часу)
    int minute; // 0-59
    int second; // 0- 59
};

// втановити значення часу у всесвітньому форматі
// переконатися, що всі дані узгоджені
// недопустимі значення даних обнуляються
void Time::setTime(int h, int m, int s) {
    hour = (h >= 0 && h < 24) ? h : 0; // перевірка годин
    minute = (m >= 0 && m < 60) ? m : 0; // перевірка хвилин
    second = (s >= 0 && s < 60) ? s : 0; // перевірка секунд
} // кінець функції setTime()

// надрукувати значення часу у всесвітньому форматі (HH:MM:SS)
void Time::printUniversal() {
    cout << setfill('0') << setw(2) << hour << ":"
        << setw(2) << minute << ":" << setw(2) << second;
} // кінець функції printUniversal()

// надрукувати значення часу у стандартному форматі (HH:MM:SS AM або PM)
void Time::printStandard() {
    cout << ((hour == 0) || (hour == 12) ? 12 : hour % 12) << ":" <<
        setfill('0') << setw(2) << minute << ":" << setw(2)
        << second << (hour < 12 ? " AM" : " PM");
} // кінець функції printStandard()

int main()
{   // Очищення екрана
    system("cls");
```

```
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Time t; // створити обєкт t класу Time

    // вивести вхідні значення обєкта t
    cout << "The initial Universal time is ";
    t.printUniversal();
    cout << "\nThe initial standard time is ";
    t.printStandard();

    // встановити інший час
    t.setTime(13, 27, 6);

    // вивести нові значення обєкта t
    cout << "\n\nUniversal time after setTime is ";
    t.printUniversal();
    cout << "\nStandard time after setTime is ";
    t.printStandard();

    // Спроба некоректного задання часу
    t.setTime(99, 99, 99);

    // вивести нові значення обєкта t після задання некоретних значень часу
    cout << "\n\nAfter attempting invalid settings:"
        << "\nUniversal time is ";
    t.printUniversal(); // 00:00 : 00
    cout << "\nStandard time is ";
    t.printStandard(); //12:00:00 AM
    cout << endl;

    return 0;

}

The initial Universal time is 00:00:00 The initial standard time is
12:00:00 AM

Universal time after setTime is 13:27:06 Standard time after setTime is
1:27:06 PM

After attempting invalid settings:
Universal time is 00:00:00
Standard time is 12:00:00 AM
```

Завдання 1.2. Завантажити, вивчити та протестувати програму, яка демонструє три форми доступу до елемент-функцій об'єкта: за допомогою імені об'єкта та операції крапка (.); за допомогою посилання на об'єкт та операції крапка (.); за допомогою вказівника на об'єкт та операції стрілка (->)

```
// lab_work_01_02.cpp Deitel_9_4

// Програма демонструє три форми доступу до елемент-функцій об'єкта:
// за допомогою імені об'єкта та операції крапка (.)
// за допомогою посилання на об'єкт та операції крапка (.)
// за допомогою вказівника на об'єкт та операції стрілка (->)

// Клас Count складається з двох відкритих функцій доступу до закритих
даних:
// setX(int value)- встановлює значення x та print() - виводить на друк
значення x
// та власне розділу закритих даних, що містить одне закрите дане x цілого
типу

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

// визначення класу Count
class Count {
public: // відкриті дані небезпечні!
    // функція, яка встановлює значення закритого елемента даних x
    void setX(int value) {
        x = value;
    } // кінець визначення функції setX

    // функція, яка виводить на друк значення закритого елемента даних x
    void print() {
        cout << x << endl;
    } // кінець визначення функції print

private:
    int x; // значення закритого елемента даних
}; // кінець визначення класу Count


int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Count counter; // створення об'єкта counter класу Count
    Count* counterPtr = &counter; // створення вказівника на об'єкт
*counterPtr класу Count
```

```
    Count& counterRef = counter; // створення посилання на об'єкт
&counterRef класу Count

    cout << "Set x to 1 and print using the object's name: ";
    counter.setX(1); // встановити елемент даних x в 1
    counter.print(); // викликати елемент-функцію print

    cout << "Set x to 2 and print using a reference to an object: ";
    counterRef.setX(2); // встановити елемент даних x в 2
    counterRef.print(); // викликати елемент-функцію print

    cout << "Set x to 3 and print using a pointer to an object: ";
    counterPtr->setX(3); // встановити елемент даних x в 3
    counterPtr->print(); // викликати елемент-функцію print

    cout << "\n\nHello World!\n";
    return 0;

}

Set x to 1 and print using the object's name: 1
Set x to 2 and print using a reference to an object: 2
Set x to 3 and print using a pointer to an object: 3
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1.
Створити програму для обчислення площі та периметра прямокутника. Реалізувати окремі функції для обчислення площі та периметра. Параметри функцій передати **за значенням**.

Варіант 2.
Створити програму для обчислення площі та довжини кола. Використати функції з параметрами за **замовчуванням**. Пояснити призначення параметрів за замовчуванням.

Варіант 3.
Створити програму для обміну значеннями двох змінних. Реалізувати функцію `swapValues()`, яка отримує параметри **за посиланням**. Показати результат до та після обміну.

Варіант 4.
Створити програму для обміну значеннями двох змінних. Реалізувати дві версії функції: з передаванням параметрів **за адресою** та **за посиланням**. Порівняти ці способи.

Варіант 5.
Створити програму для знаходження більшого з двох чисел. Реалізувати **перевантажені функції** для типів `int` та `double`.

Варіант 6.
Створити програму для знаходження максимального значення серед двох, трьох та чотирьох чисел. Використати **перевантаження функцій**.

Варіант 7.
Створити програму для обчислення степеня числа. Реалізувати перевантажені функції для цілого та дійсного числа. Передбачити показник степеня **за замовчуванням**, рівний 2.

Варіант 8.
Створити програму для обчислення середнього арифметичного двох, трьох та чотирьох чисел. Використати **перевантаження функцій**.

Варіант 9.
Створити програму для роботи з температурою. Реалізувати функції перетворення температури з градусів Цельсія у Фаренгейти та навпаки. Продемонструвати використання **простору імен**.

Варіант 10.
Створити програму для обчислення площі геометричних фігур: прямокутника, трикутника та кола. Для кожної фігури реалізувати окрему **перевантажену функцію** `area()`.

Варіант 11.
Створити програму для роботи з двома змінними. Реалізувати функції збільшення значення змінної на задану величину та обміну значень. Показати використання **посилань**.

Варіант 12.
Створити програму для знаходження суми елементів масиву. Реалізувати функцію, яка отримує масив та його розмір. Додатково реалізувати функцію для зміни елементів масиву через **посилання/адреси**.

Варіант 13.
Створити програму для знаходження мінімального та максимального елементів масиву. Реалізувати функцію, яка повертає результати через **параметри-посилання**.

Варіант 14.
Створити програму для підрахунку кількості парних та непарних елементів масиву. Для отримання двох результатів використати **параметри за посиланням**.

Варіант 15.
Створити програму для сортування трьох чисел за зростанням. Реалізувати функцію `sortThree()`, яка змінює значення змінних через **посилання**.

Варіант 16.
Створити програму для роботи з інформацією про студента: ім'я, вік, середній бал. Реалізувати окремі функції для введення, виведення та зміни даних. Пояснити, які елементи програми можна було б об'єднати в майбутньому в клас.

Варіант 17.
Створити програму для роботи з автомобілем: марка, рік випуску, об'єм двигуна. Реалізувати функції введення та виведення інформації. Пояснити, як цю програму можна було б реалізувати за допомогою **ООП**.

Варіант 18.
Створити програму для роботи з банківським рахунком: номер, власник, баланс. Реалізувати функції поповнення та зняття коштів із використанням **посилань**. Пояснити необхідність **інкапсуляції** таких даних.

Варіант 19.
Створити програму для обчислення заробітної плати працівника. Використати функцію з параметрами за замовчуванням для встановлення премії. Продемонструвати переваги такого підходу.

Варіант 20.
Створити програму-калькулятор. Реалізувати перевантажені функції для виконання операцій з `int` та `double`. Передбачити додавання, віднімання, множення та ділення.

Варіант 21.
Створити програму для роботи з прямокутником. Реалізувати функції обчислення площі, периметра та діагоналі. Параметри передавати **за значенням та за посиланням** і порівняти ці способи.

Варіант 22.
Створити програму для роботи з дробами. Реалізувати функції додавання та віднімання двох дробів. Використати **посилання** для повернення результату через параметр функції.

Варіант 23.
Створити програму для конвертації одиниць довжини. Реалізувати перевантажені функції для роботи з різними числовими типами. Використати простір імен для групування функцій конвертації.

Варіант 24.
Створити програму для обчислення вартості покупки. Функція повинна отримувати ціну та кількість товару, а знижку — як параметр **за замовчуванням**. Передбачити можливість зміни ціни через посилання.

Варіант 25.
Створити програму для роботи з координатами точки на площині. Реалізувати функції для обчислення відстані від точки до початку координат та відстані між двома точками. Використати **перевантаження функцій**.

Варіант 26.
Створити програму для роботи з часом: години, хвилини, секунди. Реалізувати функцію додавання двох значень часу з передаванням параметрів **за посиланням**. Передбачити нормалізацію секунд і хвилин.

Варіант 27.
Створити програму для роботи з товаром: назва, ціна, кількість. Реалізувати функції обчислення вартості товару та зміни ціни. Пояснити, як принцип **інкапсуляції** дозволив би захистити ціну та кількість від некоректної зміни.

Варіант 28.
Створити програму для обчислення параметрів трикутника. Реалізувати перевантажені функції для знаходження площі трикутника різними способами. Використати **простір імен** для функцій роботи з геометричними фігурами.

Варіант 29.
Створити програму для демонстрації різних способів передавання параметрів. Для однієї задачі реалізувати три функції: передавання **за значенням, за адресою та за посиланням**. Пояснити різницю між отриманими результатами.

Варіант 30.
Створити невелику програму предметної області на вибір: **«Автомобіль», «Студент», «Товар», «Банківський рахунок» або «Працівник»**. Реалізувати її процедурним способом із використанням функцій, посилань, параметрів за замовчуванням та перевантаження функцій. Окремо описати, як цю програму можна було б вдосконалити за допомогою принципів **ООП**.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №2. ТЕМА «КЛАСИ ТА ОБ'ЄКТИ»

МЕТА РОБОТИ

1. Ознайомитися з конструкторами за замовчуванням.
2. Ознайомитися з конструкторами з параметрами.
3. Ознайомитися з конструкторами копіювання.
4. Ознайомитися з деструкторами класу.
5. Вивчити порядок виклику констукторів та деструкторів класу під час роботи програми.

ХІД РОБОТИ

Завдання 2.1. Завантажити, вивчити та протестувати програму, яка демонструє різницю між функціями доступу, якими є функції set, get, print та функціями перевірки певних умов та сервісними (допоміжними) функціями.

```
// lab_work_02_01.cpp Deitel_9_5_9_7

// Програма демонструє різницю між функціями доступу,
// якими є функції set, get, print та функції перевірки певних умов
// та сервісними (допоміжними) функціями.
// Сервісні функції, як правило, не входять до складу відкритих функцій
// Сервісні функції допомагають виконувати операції відкритим функціям
доступу.

// З цією метою пропонується клас SalesPerson, який,
// використовуючи сервісну функцію totalAnnualSales() , знаходить річний
// сумарний обсяг продаж (у разі, якщо місячні обсяги продаж відомі).

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>

using namespace std;

class SalesPerson {  // визначення класу SalesPerson
public:
    SalesPerson();  // конструктор
    void getSilesFromUser(); // ввести з клавіатури суми продаж
    void setSales(int, double); // встановити суму продаж для № місяця
    void printAnnualSales(); // підсумувати і надрукувати суму продаж
private:
    double totalAnnualSales(); // прототип допоміжної функції
    double sales[12]; // 12 сум місячних продаж
}; // кінець визначення класу SalesPerson

// визначення конструктора класу SalesPerson
SalesPerson::SalesPerson() {
    for (int i = 0; i < 12; i++)
```

```
        sales[i] = 0.0;
} // кінець визначення конструктора класу SalesPerson

//
void SalesPerson::getSilesFromUser() {
    double salesFigure;
    for (int i = 1; i <= 12; i++) {
        cout << "Enter sales amount for month" << i << ": ";
        cin >> salesFigure;
        setSales(i, salesFigure);
    } // кінець for
} // кінець функції getSilesFromUser

// встановити одну з 12-ти сум;
// функція віднімає від значення місяця 1,
// щоб отримати правильний індекс масиву
void SalesPerson::setSales(int month, double amount) {
    // перевірити коректність значень місяця і суми продаж
    if (month >= 1 && month <= 12 && amount > 0)
        sales[month - 1] = amount; // adjust for subscripts 0 - 11
    else // некоректне значення місяця або суми
        cout << "Invalid month or sales figure" << endl;
} // кінець функції setSales

// надрукувати річну суму (за допомогою допоміжної суми)
void SalesPerson::printAnnualSales() {
    cout << setprecision(2) << fixed <<
        "\nThe total annual sales are: $"
        << totalAnnualSales() << endl; // виклик допоміжної функції
} // кінець функції printAnnualSales


// закрита допоміжна функція для підсумування продаж за рік
double SalesPerson::totalAnnualSales() {
    double total = 0.0;
    for (int i = 0; i < 12; i++) // підсумувати продажі за рік
        total += sales[i]; // додати продажі i - го місяця до total
    return total;
} // кінець функції totalAnnualSales


int main()
{   // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    SalesPerson s; // створити об'єкт класу SalesPerson
    s.getSilesFromUser(); // зверніть увагу на просту структуру:
```

```
    s.printAnnualSales(); //  у головній функції немає керуючих операторів

    return 0;

}

Enter sales amount for month1: 5314.76
Enter sales amount for month2: 4292.38
Enter sales amount for month3: 4589.83
Enter sales amount for month4: 3534.03
Enter sales amount for month5: 4376.34
Enter sales amount for month6: 5698.45
Enter sales amount for month7: 4439.22
Enter sales amount for month8: 5893.57
Enter sales amount for month9: 4909.67
Enter sales amount for month10: 5123.45
Enter sales amount for month11: 4024.97
Enter sales amount for month12: 5923.92
The total annual sales are: $58120.59
```

Завдання 2.2. Завантажити, вивчити та протестувати програму, яка демонструє роботу конструкторів класу з аргументами за замовчуванням

```
// lab_work_02_02.cpp  Deitel_9_8_9_10

// Програма демонструє роботу конструкторів класу
// з аргументами за замовчуванням.
// З цією метою пропонується удосконалений клас Time (варіант 2).

// Кожен клас може мати лише один конструктор за замовчуванням.
// У нашій програмі стрічка оголошує конструктор Time
// з трьома аргументами цілого типу, встановлюючи їх
// початкові значення у нуль. Це прототип конструктора.
// У визначеннні конструктора (стрічки ) він викликає функцію
// setTime(int h, int m, int s), яка приймаючи значення параметрів
// h, m, s, передає ці значення даним закритого тиипу hour, minute, second,
// викликаючи до роботи три функції setHour, setMinute, setSecond.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Time
{
public:
    Time(int = 0, int = 0, int = 0); // конструктор за замовчуванням

    // set-функції
```

```
    void setTime(int, int, int); // встановити годину, хвилину, секунду
    void setHour(int); // встановити годину (після перевірки)
    void setMinute(int); // встановити годину (після перевірки)
    void setSecond(int); // встановити годину (після перевірки)

    // get-функції
    int getHour(); // отримати годину
    int getMinute(); //отримати годину
    int getSecond(); // отримати годину

    void printUniversal(); // надрукувати час у форматі всесвітнього часу
    void printStandard(); // надрукувати час у форматі стандартного часу

private:
    int hour; // 0-23 (24-годинний формат часу)
    int minute; // 0-59
    int second; // 0- 59
}; // кінець визначення класу Time

// визначення конструктора класу Time
// ініціалізує кожен елемент даних нулем
// гарантує, що обєкти створюються в узгодженому стані
Time::Time(int hr, int min, int sec) {
    setTime(hr, min, sec); // перевірити і встановити час
} // кінець конструктора класу Time

// встановити новий час (у всесвітнньому форматі);
// переконатися, що дані узгоджені -
// некоректні елементи обнуляються
void Time::setTime(int h, int m, int s) {
    setHour(h); // встановити закрите поле годин
    setMinute(m); //встановити закрите поле хвилин
    setSecond(s); // встановити закрите поле секунд
} // кінець функції setTime

// встановити значення годин
void Time::setHour(int h) {
    hour = (h >= 0 && h < 24) ? h : 0; // встановити годину
} // кінець функції setHour

// встановити значення хвилин
void Time::setMinute(int m) {
    minute = (m >= 0 && m < 60) ? m : 0; // встановити хвилину
} // кінець функції setMinute

// встановити значення секунд
void Time::setSecond(int s) {
    second = (s >= 0 && s < 60) ? s : 0; // встановити секунду
} // кінець функції setSecond

// повернути значення годин
```

```
int Time::getHour() {
    return hour;
} // кінець функції getHour

// повернути значення хвилин
int Time::getMinute() {
    return minute;
} // кінець функції getMinute

// повернути значення секунд
int Time::getSecond() {
    return second;
} // кінець функції getSecond

// надрукувати значення часу у всесвітньому форматі (HH:MM:SS)
void Time::printUniversal() {
    cout << setfill('0') << setw(2) << getHour() << ":"
        << setw(2) << getMinute() << ":" << setw(2) << getSecond();
} // кінець функції printUniversal()

// надрукувати значення часу у стандартному форматі (HH:MM:SS AM або PM)
void Time::printStandard() {
    cout << ((hour == 0) || (hour == 12) ? 12 : getHour() % 12) << ":" <<
        setfill('0') << setw(2) << getMinute() << ":" << setw(2)
        << getSecond() << (hour < 12 ? " AM" : " PM");
} // кінець функції printStandard()

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Time t1; // обєкт t1 - всі аргументи встановлюються за замовчуванням
    Time t2(2); // обєкт t2 - година задана, хвилина та секунда за
замовчуванням
    Time t3(21, 34); // обєкт t3 - година та хвилина задані, секунда за
замовчуванням
    Time t4(12, 25, 42); // обєкт t4 - всі аргументи задані
    Time t5(27, 74, 99); // обєкт t5 - всі аргументи задані некоректно

    cout << "Object t1: constructured with all arguments defaulted\n  ";
    t1.printUniversal(); // 00:00:00
    cout << "\n  ";
    t1.printStandard(); //12:00:00 AM
    cout << "\n";

    cout << "\n\nObject t2: hour specified, minute and second defaulted\n
";
    t2.printUniversal(); // 02:00:00
```

```
    cout << "\n  ";
    t2.printStandard(); // 02:00:00 AM
    cout << "\n";

    cout << "\n\nObject t3: hour and minute specified, second defaulted\n
";
    t3.printUniversal(); // 21:34:00
    cout << "\n  ";
    t3.printStandard(); // 9:34:00 PM
    cout << "\n";

    cout << "\n\nObject t4: all arguments specified correctly\n  ";
    t4.printUniversal(); // 12:25:42
    cout << "\n  ";
    t4.printStandard(); // 12:25:42 PM
    cout << "\n";

    cout << "\n\nObject t5: all arguments specified uncorrectly\n  ";
    t5.printUniversal(); // 00:00:00
    cout << "\n  ";
    t5.printStandard(); // 12:00:00 AM
    cout << "\n";

    return 0;
}
```

```
Object t1: constructured with all arguments defaulted
  00:00:00
  12:00:00 AM

Object t2: hour specified, minute and second defaulted
  02:00:00
  2:00:00 AM

Object t3: hour and minute specified, second defaulted
  21:34:00
  9:34:00 PM

Object t4: all arguments specified correctly
  12:25:42
  12:25:42 PM

Object t5: all arguments specified uncorrectly
  00:00:00
  12:00:00 AM
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Студент

Створити клас `Student`, який містить поля: прізвище, ім'я, номер групи, вік, середній бал. Реалізувати методи введення та виведення інформації. Створити об'єкт класу та продемонструвати роботу з його полями і методами.

Варіант 2. Автомобіль
Створити клас `Car`, який містить марку, модель, рік випуску, об'єм двигуна та ціну. Реалізувати методи введення, виведення та зміни ціни. Частину методів визначити всередині класу, а частину — поза класом.

Варіант 3. Книга
Створити клас `Book`, який містить назву книги, автора, рік видання та ціну. Реалізувати методи введення та виведення інформації. Створити кілька об'єктів класу та порівняти їхні ціни.

Варіант 4. Працівник
Створити клас `Employee`, який містить прізвище, посаду, стаж роботи та заробітну плату. Реалізувати метод обчислення заробітної плати з урахуванням премії. Дані класу зробити закритими (`private`).

Варіант 5. Товар
Створити клас `Product`, який містить назву товару, артикул, ціну та кількість. Реалізувати метод обчислення загальної вартості товару. Створити масив об'єктів та знайти найдорожчий товар.

Варіант 6. Прямокутник
Створити клас `Rectangle` з полями довжини та ширини. Реалізувати методи обчислення площі, периметра та діагоналі. Методи визначити **поза межами класу**.

Варіант 7. Коло
Створити клас `Circle` з полем радіуса. Реалізувати методи обчислення площі та довжини кола. Створити масив об'єктів і визначити коло з найбільшою площею.

Варіант 8. Банківський рахунок
Створити клас `BankAccount`, який містить номер рахунку, ім'я власника та баланс. Реалізувати методи поповнення рахунку, зняття коштів та виведення інформації. Поле балансу зробити `private`.

Варіант 9. Точка
Створити клас `Point`, який описує точку на площині координатами `x` та `y`. Реалізувати метод обчислення відстані від точки до початку координат. Створити функцію, яка отримує два об'єкти `Point` та визначає відстань між ними.

Варіант 10. Трикутник
Створити клас `Triangle`, який містить довжини трьох сторін. Реалізувати методи перевірки існування трикутника, обчислення периметра та площі. Передбачити передавання об'єкта класу у функцію.

Варіант 11. Час

Створити клас `Time`, який містить години, хвилини та секунди. Реалізувати методи введення, виведення та переведення часу в секунди. Створити функцію, яка отримує об'єкт `Time` і повертає кількість секунд.

Варіант 12. Дата
Створити клас `Date`, який містить день, місяць та рік. Реалізувати методи введення та виведення дати. Створити функцію, яка отримує два об'єкти `Date` і визначає, яка дата є пізнішою.

Варіант 13. Студентська група
Створити клас `Student` та масив із 5 об'єктів цього класу. Клас повинен містити прізвище, групу та середній бал. Визначити студента з найвищим середнім балом.

Варіант 14. Автопарк
Створити клас `Car` та масив із 5 автомобілів. Для кожного автомобіля зберігати марку, рік випуску, пробіг та ціну. Вивести автомобілі, які мають пробіг менше заданого значення.

Варіант 15. Бібліотека
Створити клас `Book` та масив об'єктів цього класу. Для кожної книги зберігати назву, автора та рік видання. Реалізувати функцію, яка отримує масив об'єктів і повертає книгу з найновішим роком видання.

Варіант 16. Телефон
Створити клас `Phone`, який містить виробника, модель, об'єм пам'яті та ціну. Реалізувати методи введення та виведення. Створити функцію, яка отримує об'єкт `Phone` і змінює його ціну.

Варіант 17. Комп'ютер
Створити клас `Computer`, який містить процесор, об'єм оперативної пам'яті, об'єм накопичувача та ціну. Реалізувати методи для виведення характеристик. Створити масив комп'ютерів і визначити найдорожчий.

Варіант 18. Квартира
Створити клас `Apartment`, який містить адресу, площу, кількість кімнат та вартість. Реалізувати метод обчислення вартості одного квадратного метра. Створити функцію, яка отримує два об'єкти та визначає дорожчу квартиру.

Варіант 19. Товарний склад
Створити клас `Product`, який містить назву, ціну та кількість товару на складі. Реалізувати метод збільшення та зменшення кількості товару. Створити масив об'єктів і визначити товари, кількість яких менша за задану.

Варіант 20. Працівники підприємства
Створити клас `Employee`, який містить прізвище, посаду, зарплату та стаж. Створити масив об'єктів. Реалізувати функцію, яка отримує масив працівників і повертає об'єкт працівника з найбільшим стажем.

Варіант 21. Вказівник на об'єкт

Створити клас `Car` з полями марки, моделі та ціни. Створити об'єкт класу та **вказівник на цей об'єкт**. Здійснити доступ до полів і методів через вказівник за допомогою оператора `->`.

Варіант 22. Посилання на об'єкт
Створити клас `Student` з полями прізвища, групи та середнього бала. Створити функцію, яка отримує об'єкт класу **за посиланням** і змінює його середній бал. Продемонструвати результат до та після виклику функції.

Варіант 23. Функція, що повертає об'єкт
Створити клас `Rectangle`. Реалізувати функцію, яка отримує довжину та ширину і **повертає об'єкт `Rectangle`**. Вивести параметри отриманого об'єкта та його площу.

Варіант 24. Порівняння об'єктів
Створити клас `Book`. Реалізувати функцію, яка отримує два об'єкти `Book` і повертає той об'єкт, ціна якого більша. Продемонструвати повернення об'єкта з функції.

Варіант 25. Копіювання об'єкта
Створити клас `Product` з полями назви, ціни та кількості. Створити один об'єкт, а потім створити другий об'єкт на основі першого. Змінити дані другого об'єкта та продемонструвати, що перший об'єкт не змінився.

Варіант 26. Масив об'єктів і функція
Створити клас `Student` та масив із 10 об'єктів. Реалізувати окрему функцію, яка отримує масив об'єктів і його розмір та виводить студентів, середній бал яких перевищує задане значення.

Варіант 27. Об'єкт як параметр і результат
Створити клас `Point`. Реалізувати функцію, яка отримує **два об'єкти `Point`** та повертає новий об'єкт `Point`, координати якого є середніми арифметичними координат початкових точок.

Варіант 28. `private`, `public`, `protected`
Створити клас `Employee`, у якому продемонструвати використання специфікаторів доступу `private`, `public` та `protected`. Створити методи для роботи з відповідними даними. Пояснити, до яких членів класу можна отримати доступ безпосередньо.
Тут успадкування ще не потрібно реалізовувати — достатньо продемонструвати призначення трьох специфікаторів доступу та пояснити особливості `protected`.

Варіант 29. Методи всередині та поза класом
Створити клас `Circle`. Один метод визначити **безпосередньо всередині класу**, а інші методи визначити **поза класом** за допомогою оператора `::`. Продемонструвати створення об'єкта та виклик усіх методів.

Варіант 30. Комплексне завдання

Створити клас на вибір: `Car`, `Student`, `Book`, `Product`, `Employee` або `BankAccount`.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №3. ТЕМА «ІНКАПСУЛЯЦІЯ ТА МЕТОДИ КЛАСІВ»

МЕТА РОБОТИ

1. Вивчити порядок виклику конструкторів та деструкторів класу.
2. Навчитися копіювати об’єкти засобами мови С/С++.
3. Вивчити способи виклику константних та неконстантних елемент-функцій за допомогою об’єктів класу.

ХІД РОБОТИ

Завдання 3.1. Завантажити, вивчити та протестувати програму, яка демонструє порядок виклику конструкторів та деструкторів класу

```
// lab_work_03_01.cpp Deitel_9_11_9_13

// Програма демонструє порядок виклику конструкторів та
деструкторів.

// З цією метою розроблено клас CreateAndDestroy,
// який забезпечує створення об'єктів (виконує конструктор)
// та їх руйнування після завершення роботи програми (виконує
деструктор).
// Конструктор за замовчуванням має два параметри -
// ID об'єкта та повідомлення, яке характеризує цей об'єкт.
// Об'єкти можуть бути створені в атоматній чи статичній пам'яті.
// Крім того, за місцем їх створення, об'єкти можуть бути
глобальними чи локальними.
// Ще однією особливістю програми є можливість створення об'єктів
// як безпосередньо за допомогою конструктора,
// так і за допомогою допоміжної функції create.


#include <iostream>
#include <iomanip>
#include <conio.h>
#include <string.h>
#include <windows.h>

using namespace std;

class CreateAndDestroy
{
public:
    CreateAndDestroy(int, string); // конструктор за замовчуванням
    ~CreateAndDestroy(); // деструктор

private:
    int objectID; // ID - значення для обєкта класу
    string message; // повідомленя, яке описує обєкт
}; // кінець визначення класу CreateAndDestroy
```

```
// визначення конструктора класу CreateAndDestroy
CreateAndDestroy::CreateAndDestroy(int ID, string messageString) {
    objectID = ID; // встановити ідентифікаційний номер обєкта
    message = messageString; // встановити повідомлення, яке
описує обєкт

    cout << "object " << objectID << " constructor runs " <<
message << endl;
} // кінець визначення конструктора класу CreateAndDestroy

// визначення деструктора класу CreateAndDestroy
CreateAndDestroy::~CreateAndDestroy() {
    // для деяких обєктів виводити \n, що покращує читабельність
    cout << (objectID == 1 || objectID == 6 ? "\n" : "");
    cout << "object " << objectID << " destructor runs " <<
message << endl;
} // кінець визначення деструктора класу CreateAndDestroy

int b, a = 1;
CreateAndDestroy first(1, "(global before main)");


void create(void); // прототип функції

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
    CreateAndDestroy second(2, "(local automatic in main)");
    static CreateAndDestroy third(3, "(local static in main)");

    create();

    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
    CreateAndDestroy fourth(4, "(local automatic in main)");
    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;

    return 0;

}

// функція для створення обєктів
void create(void) {
    cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
    CreateAndDestroy fifth(5, "(local automatic in create)");
    static CreateAndDestroy sixth(6, "(local static in create)");
```

```
    CreateAndDestroy seventh(7, "(local automatic in create)");
    cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
} // кінець визначення функції create

MAIN FUNCTION: EXECUTION BEGINS
object 2 constructor runs (local automatic in main)
object 3 constructor runs (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
object 5 constructor runs (local automatic in create)
object 6 constructor runs (local static in create)
object 7 constructor runs (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
object 7 destructor runs (local automatic in create)
object 5 destructor runs (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
object 4 constructor runs (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
object 4 destructor runs (local automatic in main)
object 2 destructor runs (local automatic in main)
object 6 destructor runs (local static in create)
object 3 destructor runs (local static in main)
object 1 destructor runs (global before main)
```

Завдання 3.2. Завантажити, вивчити та протестувати програму, яка демонструє демонструє виникнення прихованої помилки під час спроби повернути константне посилання на закритий елемент даних класу, що призводить до порушення принципу інкапсуляції.

```
// ind_work_03_02.cpp Deitel_9_14_9_16

// Програма демонструє виникнення прихованої помилки
// під час спроби повернути константне посилання
// на закритий елемент даних класу, що призводить
// до порушення принципу інкапсуляції.
// З цією метою пропонується клас Time (варіант 3).

// Посилання на об'єкт є псевдонімом для імені об'єкта.
// тому воно може з'являтися у лівій частині оператора присвоєння.
// Тому посилання на об'єкт є цілком законною величиною lvalue,
// яка може отримувати значення (її можна змінювати).
// З іншого боку, за визначенням будь-яка елемент-функція класу
// може повертати посилання на об'єкт класу.
// Далі можна змінити його значення на інше (модифікувати),
// що фактично означатиме зміну значення закритого даного,
// тому може виникнути прихована помилка.
//
// Наша програма демонструє повернення елемент-функцією
```

```
// int & Time::badSetHour(int) (яка оголошена в стрічці та
визначена
// у стрічках ) посилання на закритий елемент даних.
// Таке повернення по суті робить виклик елемент-функції
badSetHour
// псевдонімом закритого елемента даних hour.
// А це дозволяє користувачу змінювати яким завгодно чином закриті
дані,
// чого в принципі не повинно бути.
// Така ж проблема виникає при поверненні елеиент-функцією
// вказівника на закритий елемент даних.
// Це є порушенням принципу інкапсуляції.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Time
{
public:
    Time(int = 0, int = 0, int = 0); // конструктор за
замовчуванням

    void setTime(int, int, int); // встановити годину, хвилину,
секунду
    int getHour(); // отримати годину
    int& badSetHour(int); // небезпечне повернення посилання

private:
    int hour; // 0-23 (24-годинний формат часу)
    int minute; // 0-59
    int second; // 0- 59
}; // кінець визначення класу Time

// визначення конструктора класу Time
// за замовчуванням ініціалізує кожен елемент закритих даних нулем
// для встановлення інших значень змінних класу викликає функцію
setTime
Time::Time(int hr, int min, int sec) {
    setTime(hr, min, sec); // перевірити і встановити час
} // кінець конструктора класу Time

// встановити новий час (у всесвітнньому форматі);
// переконатися, що дані узгоджені -
// некоректні елементи обнуляються
void Time::setTime(int h, int m, int s) {
    hour = (h >= 0 && h < 24) ? h : 0; // перевірити і встановити
годину
```

```
    minute = (m >= 0 && m < 60) ? m : 0; // перевірити і
встановити хвилину
    second = (s >= 0 && s < 60) ? s : 0; // перевірити і
встановити секунду
} // кінець функції setTime

// повернути значення годин
int Time::getHour() {
    return hour;
} // кінець функції getHour

// ЦЕ ПОГАНИЙ СТИЛЬ ПРОГРАМУВАННЯ:
// повернути посилання на закритий елемент даних
int& Time::badSetHour(int hh) {
    hour = (hh >= 0 && hh < 24) ? hh : 0; // перевірити і
встановити  годину
    return hour; // небезпечне повернення посилання
} // кінець функції badSetHour

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Time t; // обєкт t - всі аргументи встановлюються за
замовчуванням

    // проініціалізувати hourRef посиланням,
    // яке повертається функцією badSetHour
    int& hourRef = t.badSetHour(20); // 20 - коректний час

    cout << "Valid hour before modification: " << hourRef;
    hourRef = 30; // // встановлення через hourRef поганого
значення в t
    cout << "\nInvalid hour after modification: " << t.getHour();

    // Небезпечно: Виклик, який повертає посилання,
    // і його можна використовувати в якості lvalue!
    t.badSetHour(12) = 74; // це ще одне присвоєння поганого числа

    cout <<
"\n\n**********************************************************\n"
        << "POOR PROGRAMMING PRACTICE!!!!!!!!!!!!!!!\n"
        << "t.badSetHour(12) as an lvalue, invalid hour: "
        << t.getHour()
        <<
"\n**********************************************************\n"
<< endl;
```

```
    return 0;

}

Valid hour before modification: 20
Invalid hour after modification: 30
*******************************************************
POOR PROGRAMMING PRACTICE!!!!!!!!!!!!!!!
t.badSetHour(12) as an lvalue, invalid hour: 74
*******************************************************
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Студент
Створити клас `Student`, який містить закриті поля: прізвище, ім'я, номер групи та середній бал. Реалізувати відкриті методи-аксесори для отримання значень полів та методи-модифікатори для їх зміни. Заборонити встановлення середнього бала, меншого за 0 або більшого за 100.

Варіант 2. Автомобіль
Створити клас `Car` із закритими полями: марка, модель, рік випуску, об'єм двигуна та ціна. Реалізувати `getter` і `setter` для кожного поля. При зміні ціни не дозволяти встановлювати від'ємне значення.

Варіант 3. Банківський рахунок
Створити клас `BankAccount` із закритими полями: номер рахунку, ім'я власника та баланс. Реалізувати методи `deposit()` та `withdraw()`. Заборонити безпосередню зміну балансу. Реалізувати метод `getBalance()` як `const`-метод.

Варіант 4. Товар
Створити клас `Product`, який містить закриті поля: назва, артикул, ціна та кількість. Реалізувати методи доступу до даних. Забезпечити перевірку ціни та кількості при їх зміні. Реалізувати `getTotalCost()` як `const`-метод.

Варіант 5. Працівник
Створити клас `Employee` із закритими полями: прізвище, посада, зарплата та стаж. Реалізувати методи-аксесори та методи-модифікатори. Зарплата не може бути від'ємною. Метод обчислення річного доходу зробити `const`.

Варіант 6. Прямокутник
Створити клас `Rectangle` із закритими полями довжини та ширини. Реалізувати методи `getArea()`, `getPerimeter()` та `getDiagonal()` як `const`-методи. Методи зміни довжини та ширини повинні перевіряти коректність значень.

Варіант 7. Коло
Створити клас `Circle` із закритим полем радіуса. Реалізувати `getter` та `setter` для радіуса. Заборонити встановлення радіуса, меншого або рівного нулю. Методи обчислення площі та довжини кола зробити `const`.

Варіант 8. Книга
Створити клас `Book` із закритими полями: назва, автор, рік видання та ціна. Реалізувати методи доступу до даних. При встановленні року видання перевіряти його допустимість. Створити `const`-метод для виведення інформації про книгу.

Варіант 9. Температура
Створити клас `Temperature`, який зберігає температуру в градусах Цельсія у закритому полі. Реалізувати методи встановлення та отримання температури, а також `const`-методи перетворення в градуси Фаренгейта та Кельвіна. Заборонити значення нижче абсолютного нуля.

Варіант 10. Дата
Створити клас `Date` із закритими полями дня, місяця та року. Реалізувати методи доступу до даних та їх зміни з перевіркою коректності. Реалізувати `const`-метод для виведення дати.

Варіант 11. Лічильник студентів
Створити клас `Student`, який містить інформацію про студента та **статичне поле**, що зберігає кількість створених об'єктів класу. Реалізувати **статичний метод** для отримання цієї кількості.
Продемонструвати зміну значення лічильника при створенні об'єктів.

Варіант 12. Автомобілі
Створити клас `Car` зі статичним полем `count`, яке зберігає кількість створених автомобілів. Реалізувати статичний метод `getCount()`. Продемонструвати роботу лічильника.

Варіант 13. Товари
Створити клас `Product`, який має статичне поле для підрахунку кількості створених товарів. Реалізувати статичний метод для отримання цього значення. Дані конкретного товару повинні залишатися нестатичними.

Варіант 14. Працівники
Створити клас `Employee`, який містить інформацію про працівника та статичне поле для підрахунку кількості працівників. Реалізувати статичний метод `getEmployeeCount()`.

Варіант 15. Банківські рахунки
Створити клас `BankAccount`. Передбачити статичне поле, яке зберігає загальну кількість створених рахунків. Реалізувати статичний метод отримання цього значення.
Додатково реалізувати `const`-метод для отримання поточного балансу.

Варіант 16. Геометричні об'єкти
Створити клас `Rectangle` зі статичним полем, яке підраховує кількість створених прямокутників. Реалізувати статичний метод для отримання кількості об'єктів. Методи обчислення площі та периметра зробити `const`.

Варіант 17. Замовлення
Створити клас `Order`, який містить номер замовлення, назву товару та його вартість. Реалізувати статичний лічильник кількості створених замовлень. Створити статичний метод отримання наступного номера замовлення.

Варіант 18. Книга та бібліотека
Створити клас `Book`, який містить назву, автора та рік видання. Додати статичне поле для підрахунку кількості створених книг. Реалізувати статичний метод отримання кількості книг.

Варіант 19. Комп'ютер
Створити клас `Computer`, який містить процесор, обсяг оперативної пам'яті, обсяг диска та ціну. Реалізувати статичне поле для підрахунку кількості створених комп'ютерів та статичний метод для отримання цього значення. Методи виведення характеристик зробити `const`.

Варіант 20. Телефон
Створити клас `Phone` із закритими полями: виробник, модель, пам'ять та ціна. Додати статичне поле для підрахунку кількості створених телефонів. Реалізувати `getter/setter`, статичний метод `getCount()` та `const`-метод для виведення інформації.

Варіант 21. Курс
Створити клас `Course`, який містить закриті поля: назву курсу, викладача, кількість годин та вартість. Реалізувати методи-аксесори та методи-модифікатори. Не дозволяти встановлювати від'ємну кількість годин або вартість. Реалізувати `const`-метод для виведення інформації про курс.

Варіант 22. Телефонний номер
Створити клас `PhoneNumber`, який містить закриті поля: код країни, код оператора та номер. Реалізувати методи для отримання та зміни окремих частин номера. Створити `const`-метод для виведення повного номера.

Варіант 23. Автомобіль
Створити клас `Car`, який містить закриті поля: марка, модель, рік випуску та пробіг. Реалізувати методи-аксесори та модифікатори. Пробіг не може зменшуватися. Реалізувати `const`-метод для визначення віку автомобіля.

Варіант 24. Книга
Створити клас `Book`, який містить закриті поля: назва, автор, кількість сторінок та ціна. Реалізувати методи для отримання та зміни даних. Кількість сторінок та ціна повинні бути додатними. Реалізувати `const`-метод для обчислення вартості однієї сторінки.

Варіант 25. Працівник
Створити клас `Employee`, який містить закриті поля: прізвище, посада, зарплата та стаж роботи. Реалізувати аксесори та модифікатори. Зарплата та стаж не можуть бути від'ємними. Реалізувати `const`-метод для обчислення заробітної плати за рік.

Варіант 26. Прямокутник
Створити клас `Rectangle`, який містить закриті поля довжини та ширини. Реалізувати методи зміни розмірів із перевіркою їх коректності. Реалізувати `const`-методи для обчислення площі, периметра та діагоналі.

Варіант 27. Студент
Створити клас `Student`, який містить закриті поля: прізвище, група та середній бал. Реалізувати аксесори та модифікатори. Середній бал повинен знаходитися в межах від 0 до 100. Реалізувати `const`-метод для визначення, чи є студент відмінником.

Варіант 28. Товар
Створити клас `Product`, який містить закриті поля: назва, ціна та кількість. Реалізувати методи зміни ціни та кількості з перевіркою даних. Створити `const`-метод для обчислення загальної вартості товару.

Варіант 29. Банківський рахунок
Створити клас `BankAccount`, який містить закриті поля: номер рахунку, власник та баланс. Реалізувати методи поповнення та зняття коштів. Не дозволяти зняття суми, яка перевищує баланс. Метод отримання балансу зробити `const`.

Варіант 30. Геометрична точка
Створити клас `Point`, який містить закриті координати `x` та `y`. Реалізувати аксесори та модифікатори координат. Створити `const`-метод для обчислення відстані від точки до початку координат. Додатково реалізувати **статичне поле**, яке підраховує кількість створених об'єктів класу, та **статичний метод** для отримання цього значення.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №4. ТЕМА «КОНСТРУКТОРИ, ДЕСТРУКТОРИ ТА ЖИТТЄВИЙ ЦИКЛ ОБ'ЄКТІВ»

МЕТА РОБОТИ

1. Вивчити способи виклику константних та неконстантних елемент-функцій за допомогою об'єктів класу.
2. Вивчити взаємодію константних та неконстантних елемент-функцій з константними та неконстантними об'єктами
3. Навчитися ініціалізувати константні дані класу за допомогою ініціалізаторів.

ХІД РОБОТИ

Завдання 4.1. Завантажити, вивчити та протестувати програму, яка демонструє взаємодію константних та неконстантних елемент-функцій з константнми та неконстантними об'єктами

```
// lab_work_04_01.cpp  Deitel_10_1_10_3

// Програма демонструє взаємодію
// константних та неконстантних елемент-функцій
// з константнми та неконстантними об'єктами.
// Помилка трансляції коду виникає, коли:
// константний об'єкт викликає неконстантну елемент-функцію.
// З цією метою пропонується клас Time (варіант 4).

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Time
{
public:
    Time(int = 0, int = 0, int = 0); // конструктор за замовчуванням

    // set-функції
    void setTime(int, int, int); // встановити годину, хвилину, секунду
    void setHour(int); // встановити годину (після перевірки)
    void setMinute(int); // встановити годину (після перевірки)
    void setSecond(int); // встановити годину (після перевірки)

    // get-функції, які, як правило, оголошуються як const
    int getHour() const; // отримати годину
    int getMinute() const; //отримати годину
    int getSecond() const; // отримати годину

    // функції виведення на друк, які, як правило, оголошуються як const
```

```
    void printUniversal() const; // надрукувати час у форматі всесвітнього
часу
    void printStandard(); // надрукувати час у форматі стандартного часу;
мала би бути const

private:
    int hour; // 0-23 (24-годинний формат часу)
    int minute; // 0-59
    int second; // 0- 59
}; // кінець визначення класу Time

// визначення конструктора класу Time
// ініціалізує кожен елемент закритих даних нулем
// для встановлення інших (ненульових) значень викликає функцію setTime
Time::Time(int hour, int minute, int second) {
    setTime(hour, minute, second); // перевірити і встановити час
} // кінець конструктора класу Time

// встановити новий час (у всесвітнньому форматі);
// переконатися, що дані узгоджені -
// некоректні елементи обнуляються
void Time::setTime(int hour, int minute, int second) {
    setHour(hour); // встановити закрите поле годин
    setMinute(minute); //встановити закрите поле хвилин
    setSecond(second); // встановити закрите поле секунд
} // кінець функції setTime

// встановити значення годин
void Time::setHour(int h) {
    hour = (h >= 0 && h < 24) ? h : 0; // встановити годину
} // кінець функції setHour

// встановити значення хвилин
void Time::setMinute(int m) {
    minute = (m >= 0 && m < 60) ? m : 0; // встановити хвилину
} // кінець функції setMinute

// встановити значення секунд
void Time::setSecond(int s) {
    second = (s >= 0 && s < 60) ? s : 0; // встановити секунду
} // кінець функції setSecond


// повернути значення годин
int Time::getHour() const {
    return hour;
} // кінець функції getHour

// повернути значення хвилин
int Time::getMinute() const {
    return minute;
```

```
} // кінець функції getMinute

// повернути значення секунд
int Time::getSecond() const {
    return second;
} // кінець функції getSecond


// надрукувати значення часу у всесвітньому форматі (HH:MM:SS)
void Time::printUniversal() const {
    cout << setfill('0') << setw(2) << getHour() << ":"
        << setw(2) << getMinute() << ":" << setw(2) << getSecond();
} // кінець функції printUniversal()

// надрукувати значення часу у стандартному форматі (HH:MM:SS AM або PM)
void Time::printStandard() {
    cout << ((hour == 0) || (hour == 12) ? 12 : getHour() % 12) << ":" <<
        setfill('0') << setw(2) << getMinute() << ":" << setw(2)
        << getSecond() << (hour < 12 ? " AM" : " PM");
} // кінець функції printStandard()


int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Time wakeup(6, 45, 0); // неконстантний обєкт обєкт wakeup
    const Time noon(12, 0, 0); // константний обєкт обєкт noon

    //                                ОБЄКТ          ЕЛЕМЕНТ-ФУНКЦІЯ
    wakeup.setHour(12);       //  неконстантний      неконстантна
    //   noon.setHour(12);          //   константний       неконстантна

    wakeup.getHour();         //  неконстантний      константна
    noon.getMinute();         //   константний       константна

    noon.printUniversal();    //   константний       константна
    //    noon.printStandard();     //   константний      неконстнтна

    cout << endl;
    // закоментовані випадки noon.setHour(12)
    // та noon.printStandard() компілюються з помилками
    return 0;

}

12:00:00
```

Завдання 4.2. Створити програму, яка демонструє схему ініціалізації константного елемента закритих даних за допомогою ініціалізатора елемента. З цією метою пропонується клас Increment. Клас Increment дозволяє многократно збільшувати деяку початкову величину count на одне і те ж значення increment.

```
// Ind_work_04_02.cpp  Deitel_10_4_10_6

// Програма демонструє схему ініціалізації
// константного елемента закритих даних
// за допомогою ініціалізатора елемента.
// З цією метою пропонується клас Increment (варіант 1).
// Клас Increment дозволяє многократно збільшувати деяку
// початкову величину count на одне і те ж значення increment.

// Визначення конструктора (стрічки  ) містить
// список ініціалізаторів елементів,
// який ініціалізує елементи даних класу  Increment:
// неконстантне ціле count;
// константне ціле increment;
// Після списку формальних параметрів конструктора,
// взятого у круглі дужки,
// ставимо знак двокрапка (:)
// і далі записуємо список ініціалізаторів  даних,
// причому окремі елементи списку розділяються комами.
// кожен ініціалізатор елемента складається з імені елемента даних
// після якого у круглих дужках записується
// початкове значення цього елемента.
// У нашому прикладі це означає, що
// величина count ініціалізується значенням c;
// величина increment ініціалізується i.
// Після останнього елемента списку записується
// тіло конструктора у круглих дужках.

// Зауважимо, що за описаною схемою ініціалізуються:
// константні елементи даних;
// елементи даних, які є посиланнями;
// елементи-об'єкти;
// базові частини похідних класів.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Increment
{
public:
    Increment(int c = 0, int i = 1); // конструктор за замовчуванням
```

```
    // визначення функції addIncrement
    void addIncrement() { //
        count += increment;
    } // кінець функції addIncrement

      // функція виведення на друк
    void print() const; // надрукувати count та increment


private:
    int count; //
    const int increment; // постійне значення приросту (інкременту)
}; // кінець визначення класу Time

// визначення конструктора класу Increment
// ініціалізує кожен елемент закритих даних
// для встановлення інших (ненульових) значень викликає функцію setTime
Increment::Increment(int c, int i)
    : count(c), // функція-ініціалізатор неконстантного значення c
    increment(i) { // функція-ініціалізатор константного значення i

    // порожнє тіло конструктора
} // кінець конструктора класу Increment

// функція виведення на друк
void Increment::print() const { // надрукувати count та increment
    cout << "count = " << count << ";  increment = " << increment << endl;
} // кінець функції виведення на друк print

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Increment value(10, 5);

    cout << "Before incrementing: ";
    value.print();
    cout << endl;

    for (int j = 1; j <= 3; j++) {
        value.addIncrement();
        cout << "After increment " << j << ":";
        value.print();
    }
    cout << endl;

    Increment value1;
```

```
    cout << "Before incrementing: ";
    value1.print();
    cout << endl;

    for (int j = 1; j <= 3; j++) {
        value1.addIncrement();
        cout << "After increment " << j << ":";
        value1.print();
    }
    cout << endl;

    return 0;

}
```

```
Before incrementing: count = 10;  increment = 5

After increment 1:count = 15;  increment = 5
After increment 2:count = 20;  increment = 5
After increment 3:count = 25;  increment = 5

Before incrementing: count = 0;  increment = 1

After increment 1:count = 1;  increment = 1
After increment 2:count = 2;  increment = 1
After increment 3:count = 3;  increment = 1
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Студент
Створити клас `Student`, який містить прізвище, ім'я, групу та середній бал. Реалізувати **конструктор за замовчуванням** та **конструктор із параметрами**. Продемонструвати створення об'єктів обома способами.

Варіант 2. Автомобіль
Створити клас `Car` з полями: марка, модель, рік випуску та ціна. Реалізувати конструктор за замовчуванням і конструктор із параметрами. Поля класу ініціалізувати у **списку ініціалізації конструктора**.

Варіант 3. Книга
Створити клас `Book`, який містить назву, автора, рік видання та ціну. Реалізувати **перевантажені конструктори**:
без параметрів;
з назвою та автором;
з усіма параметрами.
Продемонструвати створення об'єктів кожним конструктором.

Варіант 4. Прямокутник

Створити клас `Rectangle` з полями довжини та ширини. Реалізувати конструктор за замовчуванням та конструктор із параметрами. Створити методи обчислення площі та периметра.

Варіант 5. Коло
Створити клас `Circle` з полем радіуса. Реалізувати два конструктори: без параметрів та з параметром `radius`. Передбачити перевірку коректності радіуса. Реалізувати метод обчислення площі.

Варіант 6. Працівник
Створити клас `Employee` з полями: прізвище, посада, зарплата та стаж. Реалізувати три перевантажені конструктори:
конструктор за замовчуванням;
конструктор із прізвищем та посадою;
конструктор з усіма даними.
Вивести інформацію про створені об'єкти.

Варіант 7. Товар
Створити клас `Product` з полями назви, ціни та кількості. Реалізувати конструктор із параметрами та **конструктор копіювання**. Створити один об'єкт, скопіювати його в інший та продемонструвати результати.

Варіант 8. Банківський рахунок
Створити клас `BankAccount` з номером рахунку, ім'ям власника та балансом. Реалізувати конструктор за замовчуванням та конструктор із параметрами. Реалізувати методи поповнення та зняття коштів.

Варіант 9. Точка
Створити клас `Point`, який містить координати `x` та `y`. Реалізувати:
конструктор за замовчуванням;
конструктор із параметрами;
конструктор копіювання.
Продемонструвати створення точки кожним способом.

Варіант 10. Дата
Створити клас `Date` з полями день, місяць та рік. Реалізувати конструктор за замовчуванням та конструктор із параметрами. У конструкторі перевірити коректність значень дня та місяця.

Варіант 11. Телефон
Створити клас `Phone`, який містить виробника, модель, обсяг пам'яті та ціну. Реалізувати конструктор із параметрами та **конструктор копіювання**. Створити початковий об'єкт, його копію та змінити дані копії.
Пояснити, чи змінився початковий об'єкт.

Варіант 12. Комп'ютер

Створити клас `Computer`, який містить процесор, оперативну пам'ять, обсяг диска та ціну. Реалізувати конструктор із параметрами та конструктор копіювання. Створити кілька об'єктів і продемонструвати копіювання.

Варіант 13. Книга
Створити клас `Book` із полями назви, автора та року видання. Реалізувати конструктор копіювання. Створити об'єкт книги та його копію. Змінити рік видання копії та перевірити, чи змінився оригінал.

Варіант 14. Трикутник
Створити клас `Triangle` з трьома сторонами. Реалізувати конструктор із параметрами та конструктор копіювання. Додати методи обчислення периметра та площі.

Варіант 15. Час
Створити клас `Time`, який містить години, хвилини та секунди. Реалізувати конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Передбачити нормалізацію значень часу.

Варіант 16. Демонстрація життєвого циклу
Створити клас `TestObject`, який містить ідентифікатор об'єкта. У конструкторі виводити повідомлення:
`Об'єкт створено`
а в деструкторі:
`Об'єкт знищено`.
Створити декілька об'єктів і простежити порядок виклику конструкторів та деструкторів.

Варіант 17. Об'єкти в різних областях видимості
Створити клас `Object`, конструктор і деструктор якого виводять повідомлення на екран.
У функції `main()` створити:
один глобальний об'єкт;
один локальний об'єкт;
об'єкт у додатковому блоці { }.
За повідомленнями конструктора та деструктора дослідити **порядок створення та знищення об'єктів**.

Варіант 18. Об'єкти у функції
Створити клас `Demo`, який має конструктор та деструктор із повідомленнями. Реалізувати функцію, яка створює локальний об'єкт цього класу. Викликати функцію кілька разів і дослідити життєвий цикл створених об'єктів.

Варіант 19. Масив об'єктів
Створити клас `Student`, який має конструктор та деструктор із повідомленнями. Створити масив із 5 об'єктів класу. Дослідити порядок виклику конструкторів та деструкторів для елементів масиву.

Варіант 20. Вкладена область видимості
Створити клас `Car` із конструктором та деструктором, які виводять інформацію про створення та знищення об'єкта.
У функції `main()` створити об'єкти:
до вкладеного блоку;
всередині вкладеного блоку;
після вкладеного блоку.
Визначити порядок виклику конструкторів і деструкторів.

Варіант 21. Студент
Створити клас `Student`, поля: прізвище, група, середній бал.
Реалізувати:
конструктор за замовчуванням;
параметризований конструктор;
копіювальний конструктор;
деструктор;
метод виведення даних про студента.
Продемонструвати створення об'єктів різними конструкторами.

Варіант 22. Автомобіль
Створити клас `Car`, поля: марка, модель, рік випуску.
Реалізувати:
конструктор за замовчуванням;
параметризований конструктор;
копіювальний конструктор;
деструктор;
метод виведення інформації.
Показати порядок виклику конструкторів і деструктора.

Варіант 23. Книга
Створити клас `Book`, поля: назва, автор, рік видання.
Реалізувати конструктори за замовчуванням, параметризований та копіювальний конструктор. Передбачити деструктор і метод виведення інформації.
Поля класу ініціалізувати за допомогою **списку ініціалізації конструктора**.

Варіант 24. Прямокутник
Створити клас `Rectangle`, поля: довжина і ширина.
Реалізувати:
конструктор за замовчуванням;
параметризований конструктор;
копіювальний конструктор;
деструктор;
метод обчислення площі;
метод виведення розмірів прямокутника.
Продемонструвати роботу всіх конструкторів.

Варіант 25. Працівник

Створити клас `Employee`, поля: ім'я, посада, заробітна плата.
Реалізувати конструктор за замовчуванням, параметризований та копіювальний конструктор, а також деструктор.
Створити кілька об'єктів і вивести інформацію про них.

Варіант 26. Коло
Створити клас `Circle`, поле: радіус.
Реалізувати:
конструктор за замовчуванням;
параметризований конструктор;
копіювальний конструктор;
деструктор;
метод обчислення площі;
метод обчислення довжини кола.
Показати створення копії об'єкта за допомогою копіювального конструктора.

Варіант 27. Банківський рахунок
Створити клас `BankAccount`, поля: номер рахунку, власник, баланс.
Реалізувати три конструктори:
за замовчуванням;
параметризований;
копіювальний.
Додати деструктор та метод виведення інформації про рахунок.
Продемонструвати створення об'єктів у різних конструкціях програми.

Варіант 28. Точка
Створити клас `Point`, поля `x` та `y`.
Реалізувати:
конструктор за замовчуванням;
параметризований конструктор;
копіювальний конструктор;
деструктор;
метод виведення координат;
метод обчислення відстані від початку координат.
Продемонструвати копіювання об'єкта.

Варіант 29. Товар
Створити клас `Product`, поля: назва, ціна, кількість.
Реалізувати конструктор за замовчуванням, параметризований та копіювальний конструктор, а також деструктор.
Додати метод обчислення загальної вартості товару та метод виведення інформації.
Для ініціалізації полів використати **список ініціалізації**.

Варіант 30. Час
Створити клас `Time`, поля: години, хвилини, секунди.
Реалізувати:
конструктор за замовчуванням;
параметризований конструктор;

копіювальний конструктор;
деструктор;
метод виведення часу;
метод переведення часу в загальну кількість секунд.
Продемонструвати створення об'єктів за допомогою різних конструкторів.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №5. ТЕМА «ДИНАМІЧНА ПАМ'ЯТЬ ТА КЕРУВАННЯ РЕСУРСАМИ»

МЕТА РОБОТИ

1. Проаналізувати виникнення помилки під час спроби ініціалізувати константний елемент даних шляхом звичайного присвоєння.
2. Ознайомитися з використанням об'єктів одного класу у якості елементів даних іншого класу

ХІД РОБОТИ

Завдання 5.1. Завантажити, вивчити та протестувати програму, яка демонструє виникнення помилки під час спроби ініціалізувати константний елемент даних шляхом звичайного присвоєння

```
// lab_work_05_01.cpp Deitel_10_7_10_9

// Програма демонструє виникнення помилки, яка виникає під час
спроби ініціалізувати
// константний елемент даних шляхом звичайного присвоєння.
// З цією метою пропонується клас Increment (варіант 2).

// Константні елементи класу (об'єкти та змінні)
// а також ті, що оголошені як посилання,
// необхідно ініціалізувати , використовуючи синтаксис
// ініціалізатора елемента; присвоєння у тілі
// конструктора для таких даних не допускаються.


#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Increment
{
public:
    Increment(int c = 0, int i = 1); // конструктор за
замовчуванням

    // визначення функції addIncrement
    void addIncrement() { //
        count += increment;
    } // кінець функції addIncrement

      // функція виведення на друк
    void print() const; // надрукувати count та increment
```

```
private:
    int count; //
    const int increment; // постійне значення приросту
(інкременту)
}; // кінець визначення класу Time

// визначення конструктора класу Increment
// правильно ініціалізує елемент закритих даних count
// не правильно ініціалізує елемент закритих даних count
// обовязково у конструкторі має бути ініціалізатор для закритих
константних даних
Increment::Increment(int c, int i) {
    count = c; // присвоєння допускається, оскільки c не є
константою
    increment = 1;  // ПОМИЛКА! Присвоєння не допускається,
оскільки i є константою
} // кінець конструктора класу Increment

// функція виведення на друк
void Increment::print() const { // надрукувати count та increment
    cout << "count = " << count << ";  increment = " << increment
<< endl;
} // кінець функції виведення на друк print

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Increment value(10, 5);

    cout << "Before incrementing: ";
    value.print();
    cout << endl;

    for (int j = 1; j <= 3; j++) {
        value.addIncrement();
        cout << "After increment " << j << ":";
        value.print();
    }
    cout << endl;

    Increment value1;

    cout << "Before incrementing: ";
    value1.print();
    cout << endl;
```

```
    for (int j = 1; j <= 3; j++) {
        value1.addIncrement();
        cout << "After increment " << j << ":";
        value1.print();
    }
    cout << endl;

    return 0;

}
```

При завантаженні програми виникає помилка. Усунути її!

Завдання 5.2. Завантажити, вивчити та протестувати програму, яка демонструє використання об'єктів одного класу у якості елементів даних іншого класу

```
// lab_work_05_02.cpp Deitel_10_10_10_14

// Програма демонструє використання об'єктів одного класу
// у якості елементів даних іншого класу.

// З цією метою пропонуються два класи:
// клас Date (варіант 2) та клас Employee.
// Клас Date дозволяє вивести дату у форматі mm/dd/yyyy.
// Клас Employee виводить інформацію про працівника:
// прізвище (lastName), ім'я (firstName),
// дату народження (birthDate), дату прийняття на
роботу(hireDate).


// Два константні елементи класу Employee
// є одночасно об'єктами класу Date.
// Саме тому тут використовуються ініціалізатори даних.


#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <cstring>

#include <windows.h>
using namespace std;

class Date
{
public:
    Date(int = 1, int = 1, int = 1900); // конструктор за
замовчуванням
    void print() const; // надрукувати дату у форматі
місяць/день/рік
```

```
    ~Date(); // передбачено для пыдтвердження порядку
деструкцыъ
private:
    int month; // 1 - 12 (січень - грудень)
    int day; // 0-31 в залежності від місяця
    int year; // будь-який рік

    // допоміжна функція для перевірки коректності дня
місяця
    int checkDay(int) const;
}; // кінець визначення класу Date

// визначення конструктора класу Date
// підтверджує корректність місяця
// викликає допоміжну функцію checkDay для перевірки дня
місяця
Date::Date(int mn, int dy, int yr) {
    if (mn > 0 && mn <= 12)
        month = mn; // ініціалізує значення місяця
    else {
        month = 1; //некоректне значення місяця замінюється
на 1
        cout << "Invalid month (" << mn << ") set to 1. \n";
    } // кінець else

    year = yr; // рік варто було б перевірити також
    day = checkDay(dy); //  перевірити день

    // вивести обєкт класу Date, щоб позначити виклик
конструктора
    cout << "Date object constructor for date ";
    print();
    cout << endl;
} // кінець визначення конструктора класу Date


// надрукувати дату у форматі mm/dd/yyyy
void Date::print() const {
    cout << month << '/' << day << '/' << year;
} // кінець визначення функції print


// вивести повідомлення, яке б свідчило про виклик
деструктора
Date::~Date() {
    cout << "Date object destructor for date ";
    print();
    cout << endl;
} // кінець визначення деструктора ~Date

// допоміжна функція для підтвердження коректності дня
місяця і року
// (виконує обробку високосних років)
```

```
int Date::checkDay(int testDay) const {
    static   const int daysPerMonth[13] =
    { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
    // перевірити коректність testDay для вказаного місяця
    if (testDay > 0 && testDay <= daysPerMonth[month])
        return testDay;
    // перевірити 29 лютого для високосного року
    if (month == 2 && testDay == 29 && (year % 400 == 0 ||
(year % 4 == 0 && year % 100 != 0)))
        return testDay;
    cout << "Invalid day (" << testDay << ") set to 1.\n";
    return 1; // залишити обєкт у коректному стані
} // кінець визначення функції checkDay


class Employee {
public:
    Employee(const char* const, const char* const,
        const Date&, const Date&);

    void print() const; // надрукувати обєкт Employe
    ~Employee(); //деструктор - передбачено для
підтвердження порядку деструкції

private:
    char firstName[25];
    char lastName[25];
    const Date birthDate; // композиція: елемент - обєкт
    const Date hireDate; // композиція: елемент - обєкт
}; // кінець класу Employee

// конструктор
Employee::Employee(const char* const first, const char*
const last,
    const Date& dateOfBirth, const Date& dateOfHire)
    : birthDate(dateOfBirth), // ініціалізувати константну
величину birthDate
    hireDate(dateOfHire) {  // ініціалізувати константну
величину hireDate

    // копіювати first в firstName: переконатися що стрічка
поміститься
    int length = strlen(first);
    length = (length < 25 ? length : 24);
    strncpy_s(firstName, first, length);
    firstName[length] = '\0';

    // копіювати last в lastName: переконатися що стрічка
поміститься
    length = strlen(last);
    length = (length < 25 ? length : 24);
    strncpy_s(lastName, last, length);
    lastName[length] = '\0';
```

```
    // вивести обєкт , щоб переконатися про виклик
конструктора
    cout << "Employee object constructor: "
        << firstName << ' ' << lastName << endl;
} //  кінець конструктора класу Employee

// надрукувати обєкт Employee
void Employee::print() const {
    cout << lastName << ", " << firstName << " Hired: ";
    hireDate.print();
    cout << " Birthday: ";
    birthDate.print();
    cout << endl;
} // кінець функції print

// надрукувати обєкт Employee, щоб підтвердити виклик
деструктора
Employee::~Employee() {
    cout << "Employee object destructor: "
        << lastName << ", " << firstName << endl;
} // кінець деструктора

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Date birth(7, 24, 1949);
    Date hire(3, 12, 1988);
    Employee manager("Bob", "Blue", birth, hire);
    cout << endl;

    manager.print();

    cout << "\nTest Date constructor with invalid
values:\n";
    Date lastDayOff(14, 35, 1994); // некоректний день та
місяць
    cout << endl;

    return 0;
}

Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Blue

Blue, Bob Hired: 3/12/1988 Birthday: 7/24/1949

Test Date constructor with invalid values:
```

```
Invalid month (14) set to 1.
Invalid day (35) set to 1.
Date object constructor for date 1/1/1994

Date object destructor for date 1/1/1994
Employee object destructor: Blue, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Створити клас `Student`, який містить ім’я студента та динамічний масив оцінок. Реалізувати конструктор, деструктор і конструктор копіювання. Забезпечити глибоке копіювання масиву. Вивести дані двох об’єктів після копіювання.

Варіант 2.
Створити клас `Book`, який містить назву книги та динамічний масив цін на примірники. Реалізувати виділення пам’яті в конструкторі та її звільнення в деструкторі. Реалізувати глибоке копіювання об’єктів.

Варіант 3.
Створити клас `Product`, який містить назву товару та динамічний масив його цін за останні місяці. Реалізувати конструктор, деструктор і конструктор копіювання. Перевірити, що зміна масиву одного об’єкта не впливає на інший.

Варіант 4.
Створити клас `Car`, який містить марку автомобіля та динамічний масив витрат пального за декілька поїздок. Реалізувати правильне виділення та звільнення пам’яті. Забезпечити глибоке копіювання.

Варіант 5.
Створити клас `Group`, який містить назву групи та динамічний масив балів студентів. Реалізувати конструктор, деструктор і конструктор копіювання. Визначити середній бал групи.

Варіант 6.
Створити клас `Temperature`, який містить назву міста та динамічний масив температур за тиждень. Реалізувати динамічне виділення пам’яті та її звільнення. Реалізувати глибоке копіювання об’єкта.

Варіант 7.
Створити клас `Warehouse`, який містить назву складу та динамічний масив кількості товарів. Реалізувати конструктор і деструктор. Створити копію об’єкта та перевірити незалежність двох масивів.

Варіант 8.
Створити клас `Employee`, який містить ім’я працівника та динамічний масив його заробітної плати за кілька місяців. Реалізувати правило трьох: деструктор, конструктор копіювання та оператор присвоєння.

Варіант 9.
Створити клас `Exam`, який містить назву дисципліни та динамічний масив оцінок студентів. Реалізувати конструктор, деструктор і конструктор копіювання. Визначити максимальну оцінку.

Варіант 10.
Створити клас `Team`, який містить назву команди та динамічний масив кількості набраних очок у матчах. Реалізувати правильне керування пам’яттю та глибоке копіювання.

Варіант 11.
Створити клас `Cinema`, який містить назву фільму та динамічний масив кількості проданих квитків за кілька сеансів. Реалізувати конструктор, деструктор і конструктор копіювання.

Варіант 12.
Створити клас `Athlete`, який містить ім’я спортсмена та динамічний масив його результатів у змаганнях. Реалізувати правило трьох. Визначити найкращий результат.

Варіант 13.
Створити клас `Course`, який містить назву курсу та динамічний масив оцінок слухачів. Реалізувати динамічну пам’ять, деструктор і глибоке копіювання. Визначити середню оцінку.

Варіант 14.
Створити клас `Apartment`, який містить номер квартири та динамічний масив показників споживання електроенергії за місяці. Реалізувати конструктор, деструктор і конструктор копіювання.

Варіант 15.
Створити клас `Game`, який містить назву гри та динамічний масив результатів гравця за кілька рівнів. Реалізувати глибоке копіювання та перевірити незалежність копій.

Варіант 16.
Створити клас `Patient`, який містить ім’я та динамічний масив результатів вимірювань певного показника. Реалізувати конструктор, деструктор і оператор присвоєння. Забезпечити правильне звільнення пам’яті.

Варіант 17.
Створити клас `Flight`, який містить номер рейсу та динамічний масив кількості пасажирів за декілька рейсів. Реалізувати правило трьох. Визначити максимальну кількість пасажирів.

Варіант 18.
Створити клас `Shop`, який містить назву магазину та динамічний масив щоденної виручки. Реалізувати динамічний масив, конструктор копіювання та деструктор. Визначити загальну виручку.

Варіант 19.
Створити клас `Weather`, який містить назву міста та динамічний масив температур за декілька днів. Реалізувати правило трьох і визначити найвищу температуру.

Варіант 20.
Створити клас `CarService`, який містить назву автомобіля та динамічний масив вартості ремонтів. Реалізувати конструктор, деструктор і конструктор копіювання. Визначити загальну вартість ремонтів.

Варіант 21.
Створити клас `Library`, який містить назву бібліотеки та динамічний масив кількості виданих книг за місяці. Реалізувати глибоке копіювання та правильне звільнення пам’яті.

Варіант 22.
Створити клас `Worker`, який містить ім’я працівника та динамічний масив кількості відпрацьованих годин за тиждень. Реалізувати правило трьох. Визначити загальну кількість годин.

Варіант 23.
Створити клас `Hotel`, який містить назву готелю та динамічний масив кількості зайнятих номерів за декілька днів. Реалізувати конструктор, деструктор і конструктор копіювання.

Варіант 24.
Створити клас `Bus`, який містить номер маршруту та динамічний масив кількості пасажирів за кілька рейсів. Реалізувати глибоке копіювання. Визначити середню кількість пасажирів.

Варіант 25.
Створити клас `Farm`, який містить назву господарства та динамічний масив урожайності за кілька років. Реалізувати конструктор, деструктор і оператор присвоєння. Забезпечити коректне керування пам’яттю.

Варіант 26.
Створити клас `Restaurant`, який містить назву ресторану та динамічний масив щоденної кількості відвідувачів. Реалізувати правило трьох. Визначити день із найбільшою кількістю відвідувачів.

Варіант 27.
Створити клас `School`, який містить назву школи та динамічний масив кількості учнів у класах. Реалізувати динамічне виділення пам’яті, деструктор і глибоке копіювання.

Варіант 28.
Створити клас `University`, який містить назву університету та динамічний масив кількості студентів за факультетами. Реалізувати конструктор копіювання, оператор присвоєння та деструктор.

Варіант 29.
Створити клас `BusStation`, який містить назву станції та динамічний масив кількості відправлених автобусів за декілька днів. Реалізувати правило трьох і визначити загальну кількість автобусів.

Варіант 30.
Створити клас `Competition`, який містить назву змагання та динамічний масив

результатів учасників. Реалізувати конструктор, деструктор, конструктор копіювання та оператор присвоєння. Перевірити правильність глибокого копіювання.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №6. ТЕМА «ПЕРЕВАНТАЖЕННЯ ФУНКЦІЙ ТА ОПЕРАЦІЙ»

МЕТА РОБОТИ

1. Вивчити методику перевантаження функцій та операцій.
2. Вивчити методику використання дружніх функцій.

ХІД РОБОТИ

Завдання 6.1. Завантажити, вивчити та протестувати програму, яка демонструє використання дружньої функції для модифікації закритих даних класу

```
// ind_work_06_01.cpp Deitel_10_15
//
// Програма демонструє використання дружньої функції
// для модифікації закритих даних класу.
// З цією метою пропонується клас Count (варіант 2).

// Дружні функції класу визначаються поза областю дії цього класу.
// У визначенні клау наводитьсялише прототип дружньої функції.
// Дружня функція має право доступу
// до відкритих та закритих елементів класу.

// У нашій програмі створена дружня функція setX,
// прототип якої наведено у стрічці , а визначення дано у стрічках .
// Ця функція модифікує значення закритого елемента даних x класу Count

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

// визначення класу Count
class Count {
    friend void setX(Count&, int); // оголошення дружньої функції
public: // відкриті дані
    // конструктор
    Count()
        : x(0) { // виклик функції-ініціалізатора x(0)
        // порожнє тіло конструктора
    } // кінець оголошення конструктора

    // функція, яка виводить на друк значення закритого елемента даних x
    void print() const {
        cout << x << endl;
    } // кінець визначення функції print

private:
    int x; // значення закритого елемента даних
```

```
}; // кінець визначення класу Count

// функція setX, яка модифікує значення закритого елемента даних Count
// оскільки функція setX оголошена дружньою до класу Count
void setX(Count& c, int val) {
    c.x = val; // це дозволяється, оскільки setX - дружня функція
} // кінець функції setX

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Count counter; // створення об'єкта counter класу Count

    cout << "counter.x after instantiation: ";
    counter.print(); // викликати елемент-функцію print

    setX(counter, 8); // встановити значення x за допомогою функції-друга
    cout << "countr.x after call to setX friend function: ";
    counter.print();

    return 0;

}

counter.x after instantiation: 0
countr.x after call to setX friend function: 8
```

Завдання 6.2. Завантажити, вивчити та протестувати програму, яка демонструє спробу використання недружньої функції для модифікації закритих даних класу

```
// lab_work_05_01.cpp  Deitel_10_16

// Програма демонструє спробу використання недружньої функції
// для модифікації закритих даних класу.
// З цією метою пропонується клас Count (варіант 3).

// Недружні функції класу визначаються поза областю дії цього класу.
// У визначенні класу не наводиться прототип недружньої функції.
// Недружня функція не має право доступу
// до відкритих та закритих елементів класу.

// У нашій програмі створена недружня функція cannotsetX,
// визначення якої дано у стрічках .
// Ця функція не може модифікувати значення
// закритого елемента даних x класу Count. Виникає помилка.
```

```
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

// визначення класу Count
class Count {

public: // відкриті дані
    // конструктор
    Count()
        : x(0) { // виклик функції-ініціалізатора x(0)
        // порожнє тіло конструктора
    } // кінець оголошення конструктора

    // функція, яка виводить на друк значення закритого елемента даних x
    void print() const {
        cout << x << endl;
    } // кінець визначення функції print

private:
    int x; // значення закритого елемента даних
}; // кінець визначення класу Count

// функція cannotsetX, яка намагається модифікувати значення закритого
елемента даних Count
// оскільки функція setX оголошена дружньою до класу Count
void cansetX(Count& c, int val) {
    c.x = val; // ПОМИЛКА! Не дозволяється звертатися до закритого елемента
даних Count
} // кінець функції cannotsetX

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Count counter; // створення об'єкта counter класу Count

    cout << "counter.x after instantiation: ";
    counter.print(); // викликати елемент-функцію print

    cansetX(counter, 3); // cannotsetX не є функцією-другом
    cout << "countr.x after call to setX friend function: ";
    counter.print();

    return 0;
```

```
}
```

Помилка виникає під час спроби використання недружньої функції для модифікації закритих даних класу. Як її усунути?

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Загальні вимоги до всіх варіантів
Створити відповідний клас із **закритими полями**.
Реалізувати конструктор та необхідні методи доступу до даних.
Продемонструвати **перевантаження функцій/методів** — не менше двох функцій з однаковою назвою, але різними параметрами.
Перевантажити **не менше двох операторів** відповідно до свого варіанта.
Для операторів +, >, == продемонструвати їх роботу для **двох об'єктів або об'єкта та значення**.
У main() створити кілька об'єктів класу та продемонструвати роботу всіх перевантажених функцій і операторів.
Заборонено використовувати готові бібліотеки чи конструкції, які безпосередньо виконують поставлені операції замість власної реалізації.

Варіант 1. Книга
Створити клас Book, що містить назву, автора та кількість сторінок. Реалізувати перевантажені методи для виведення інформації про книгу без параметрів та з параметром — кількістю примірників. Перевантажити оператори + для збільшення кількості сторінок та == для порівняння книг за кількістю сторінок.
Варіант 2. Працівник
Створити клас Worker, що містить ім'я, посаду та заробітну плату. Реалізувати перевантажені методи для зміни заробітної плати: на фіксовану суму та на відсоток. Перевантажити оператори > для порівняння зарплат та + для збільшення зарплати на задану суму.
Варіант 3. Автомобіль
Створити клас Car, що містить марку, модель та пробіг. Реалізувати перевантажені методи для збільшення пробігу на задану кількість кілометрів та на стандартне значення. Перевантажити оператори > для порівняння пробігу та + для додавання кілометрів до пробігу.
Варіант 4. Товар
Створити клас Product, що містить назву, ціну та кількість. Реалізувати перевантажені методи зміни ціни: на задану суму та на заданий відсоток. Перевантажити оператори == для порівняння товарів за ціною та + для збільшення кількості.
Варіант 5. Студент
Створити клас Student, що містить ім'я, групу та середній бал. Реалізувати перевантажені методи додавання оцінки: однією оцінкою та кількома оцінками. Перевантажити оператори > для порівняння студентів за середнім балом та == для перевірки однаковості середніх балів.
Варіант 6. Банківський рахунок
Створити клас BankAccount, що містить номер рахунку, власника та баланс. Реалізувати перевантажені методи поповнення рахунку: на задану суму та на суму з урахуванням бонусу. Перевантажити оператори + для поповнення балансу та > для порівняння балансів.

Варіант 7. Телефон
Створити клас Phone, що містить модель, обсяг пам’яті та ціну. Реалізувати перевантажені методи зміни ціни на фіксовану суму та на відсоток. Перевантажити оператори > для порівняння цін та == для порівняння обсягу пам’яті.
Варіант 8. Кімната
Створити клас Room, що містить довжину, ширину та висоту. Реалізувати перевантажені методи обчислення площі: без параметрів та для заданої кількості однакових кімнат. Перевантажити оператори > для порівняння площ та + для збільшення довжини кімнати.
Варіант 9. Прямокутник
Створити клас Rectangle, що містить довжину та ширину. Реалізувати перевантажені методи обчислення площі та периметра: без параметрів і з коефіцієнтом масштабування. Перевантажити оператори == для порівняння площ та + для збільшення розмірів прямокутника.
Варіант 10. Коло
Створити клас Circle, що містить радіус. Реалізувати перевантажені методи встановлення радіуса: цілим та дійсним числом. Перевантажити оператори > для порівняння площ кіл та + для збільшення радіуса на задане значення.
Варіант 11. Точка
Створити клас Point, що містить координати x та y. Реалізувати перевантажені методи переміщення точки: на задані x і y та на однакову величину по обох координатах. Перевантажити оператори + для переміщення точки та == для порівняння координат.
Варіант 12. Час
Створити клас Time, що містить години та хвилини. Реалізувати перевантажені методи встановлення часу: за годинами і хвилинами та лише за хвилинами. Перевантажити оператори + для додавання хвилин та > для порівняння двох моментів часу.
Варіант 13. Дата
Створити клас Date, що містить день, місяць і рік. Реалізувати перевантажені методи встановлення дати: трьома параметрами та лише роком. Перевантажити оператори == для порівняння дат та > для визначення пізнішої дати.
Варіант 14. Температура
Створити клас Temperature, що містить температуру в градусах Цельсія. Реалізувати перевантажені методи встановлення температури для int та double. Перевантажити оператори + для зміни температури та > для порівняння температур.
Варіант 15. Відстань
Створити клас Distance, що містить відстань у кілометрах і метрах. Реалізувати перевантажені методи встановлення відстані: у кілометрах та в метрах. Перевантажити оператори + для додавання відстаней та > для їх порівняння.
Варіант 16. Гроші
Створити клас Money, що містить гривні та копійки. Реалізувати перевантажені методи встановлення суми: окремо гривні й копійки та одним дійсним числом. Перевантажити оператори + для додавання сум та > для порівняння.
Варіант 17. Товарний кошик
Створити клас Basket, що містить кількість товарів та загальну вартість. Реалізувати перевантажені методи додавання товару: за кількістю та за кількістю і ціною. Перевантажити оператори + для додавання товарів та > для порівняння вартості кошиків.
Варіант 18. Фільм

Створити клас Film, що містить назву, жанр і тривалість у хвилинах. Реалізувати перевантажені методи зміни тривалості: на задану кількість хвилин та на задану кількість годин. Перевантажити оператори > для порівняння тривалості та == для порівняння назв.

Варіант 19. Пісня

Створити клас Song, що містить назву, виконавця та тривалість. Реалізувати перевантажені методи зміни тривалості: у хвилинах та секундах. Перевантажити оператори > для порівняння тривалості та == для порівняння пісень за назвою.

Варіант 20. Комп'ютер

Створити клас Computer, що містить модель, обсяг оперативної пам'яті та ціну. Реалізувати перевантажені методи збільшення оперативної пам'яті: на задану кількість ГБ та до заданого значення. Перевантажити оператори > для порівняння обсягу пам'яті та + для збільшення пам'яті.

Варіант 21. Ноутбук

Створити клас Laptop, що містить модель, діагональ екрана та ціну. Реалізувати перевантажені методи зміни ціни: на суму та на відсоток. Перевантажити оператори > для порівняння цін та == для порівняння діагоналей.

Варіант 22. Автомобільний двигун

Створити клас Engine, що містить об'єм двигуна, потужність та максимальні оберти. Реалізувати перевантажені методи збільшення потужності: на задане значення та на відсоток. Перевантажити оператори > для порівняння потужності та == для порівняння об'єму.

Варіант 23. Велосипед

Створити клас Bicycle, що містить модель, кількість передач та ціну. Реалізувати перевантажені методи зміни передачі: на одну передачу та на задану кількість. Перевантажити оператори > для порівняння кількості передач та + для збільшення кількості передач.

Варіант 24. Книга в бібліотеці

Створити клас LibraryBook, що містить назву, автора та кількість днів прокату. Реалізувати перевантажені методи збільшення терміну прокату: на один день та на задану кількість днів. Перевантажити оператори > для порівняння термінів прокату та == для порівняння назв.

Варіант 25. Працівник підприємства

Створити клас Employee, що містить ім'я, стаж та зарплату. Реалізувати перевантажені методи збільшення стажу: на один рік та на задану кількість років. Перевантажити оператори > для порівняння стажу та + для збільшення зарплати.

Варіант 26. Квартира

Створити клас Apartment, що містить площу, кількість кімнат та вартість. Реалізувати перевантажені методи зміни вартості: на фіксовану суму та на відсоток. Перевантажити оператори > для порівняння площ та == для порівняння кількості кімнат.

Варіант 27. Посилка

Створити клас Parcel, що містить довжину, ширину, висоту та масу. Реалізувати перевантажені методи обчислення об'єму: без параметрів та для заданого коефіцієнта. Перевантажити оператори > для порівняння маси та + для збільшення маси.

Варіант 28. Валіза

Створити клас Suitcase, що містить об'єм, масу та кількість речей. Реалізувати перевантажені методи додавання речей: однієї речі та заданої кількості.

Перевантажити оператори > для порівняння маси та + для збільшення кількості речей.

Варіант 29. Замовлення

Створити клас Order, що містить номер замовлення, кількість товарів та загальну суму. Реалізувати перевантажені методи додавання товару: із зазначенням лише ціни та із зазначенням ціни й кількості. Перевантажити оператори > для порівняння сум замовлень та + для збільшення суми.

Варіант 30. Автосервіс

Створити клас Service, що містить назву послуги, вартість та тривалість виконання. Реалізувати перевантажені методи зміни вартості: на фіксовану суму та на відсоток. Перевантажити оператори > для порівняння вартості послуг та == для порівняння їх тривалості.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №7. ТЕМА «ДРУЖНІ ФУНКЦІЇ ТА ДРУЖНІ КЛАСИ»

МЕТА РОБОТИ

1. Вивчити використання вказівника this для модифікації закритих даних класу.
2. Вивчити використання вказівника this для реалізації каскадних викликів функцій.
3. Ознайомитися з методикою роботи з динамічною пам’яттю.

ХІД РОБОТИ

Завдання 7.1. Завантажити, вивчити та протестувати програму, яка демонструє явне та неявне використання вказівника this для модифікації закритих даних класу

```
// lab_work_07_01.cpp  Deitel_10_17

// Програма демонструє явне та неявне використання
// вказівника this для модифікації закритих даних класу.
// З цією метою пропонується клас Test (варіант 1).
// Клас Test описує об'єкти, які мають елемент-функцію print()
// та одне закрите дане цілого типу x.
// Значення x формується конструктором об'єкта класу
// і виводится на друк.

// Відомо, що функції об'єкта мають доступ
// до даних цього об'єкта.
// Звідки ж функції знають, даними якого об'єкта
// вони можуть маніпулювати?
// Кожен об'єкт має доступ до своєї початкової адреси
// за допомогою вказівника this.
// (цей вказівник не є складовою об'єкта
// і не впливає на результат операції sizeof об'єкта).
// Точніше, вказівник this передається компілятором в об'єкт
// як неявний аргумент виклику кожної нестатичної його функції.
// Статичним елементам-функціям класу
// не передається неявний вказівник this.

// Об'єкти використовують вказівник this для доступу
// до своїх елементів даних і елементів функцій
// як явно, так і неявно.
// Тип вказівник this залежить від типу об'єкта, а також від того,
// чи оголошена константною елемент-функція,
// в якій використовується вказівник this.

// У нашій програмі елемент-функція print()
// спочатку друкує значення  x (стрічка ),
// використовуючи вказівник this неявним чином.
// Далі функція print()  використовує дві явні форми
// для доступу до x через вказівник this:

#include <iostream>
```

```
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

// визначення класу Test
class Test {
public: // відкриті дані
    Test(int = 0); // конструктор
    void print() const;
private: // закриті дані
    int x; // значення закритого елемента даних
}; // кінець визначення класу Test

// визначення конструктора
Test::Test(int value)
    : x(value) {
    // порожнє тіло конструктора
} // кінець визначення конструктора

// функція, яка виводить на друк значення
//закритого елемента даних x, використовуючи
// явний та неявний вказівник this
void Test::print() const {
    // неявне використання вказівник this
    // для доступу до елемента x
    cout << "  x=" << x << endl;

    // явне використання вказівник this
    // для доступу до елемента x
    cout << "\nthis->x=" << this->x << endl;

    // явне використання вказівник this
    // для доступу до елемента x
    // дужки у конструкції (*this) обов'язкові
    cout << "\n(*this).x=" << (*this).x << endl;
} // кінець визначення функції print

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Test testObject(12); // створення об'єкта testObject класу
Test
    testObject.print();

    return 0;
```

```
}

x=12
this->x=12
(*this).x=12
```

Завдання 7.2. Завантажити, вивчити та протестувати програму, яка демонструє використання вказівника this для реалізації каскадних викликів функцій

```
// ind_work_07_02.cpp : This file contains the 'main' function.
Program execution begins and ends there.

// deitel_10_18_10_20

// Програма демонструє використання вказівника this
// для реалізації каскадних викликів функцій.
// З цією метою пропонується клас Time (варіант 5).

// Каскадний виклик функцій - це коли один оператор
// викликає декілька функцій відразу.
// При цьому кожна з цих функцій оператором return
// має повертати *this з типом Time&.
// У програмі функції setTime, setHour, setMinute, setSecond
// змінені так, що кожна з них тепер
// повертає посилання &Time  на об'єкт класу Time.
// Саме тому це дозволяє організувати каскадний виклик функцій.
//


#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Time
{
public:
    Time(int = 0, int = 0, int = 0); // конструктор за
замовчуванням

    // set-функції, які допускають каскадуваання
    Time& setTime(int, int, int); // встановити годину, хвилину,
секунду
    Time& setHour(int); // встановити годину (після перевірки)
    Time& setMinute(int); // встановити годину (після перевірки)
    Time& setSecond(int); // встановити годину (після перевірки)

    // get-функції, які, як правило, оголошуються як const
```

```
    int getHour() const; // отримати годину
    int getMinute() const; //отримати годину
    int getSecond() const; // отримати годину

    // функції виведення на друк, які, як правило, оголошуються як
const
    void printUniversal() const; // надрукувати час у форматі
всесвітнього часу
    void printStandard() const; // надрукувати час у форматі
стандартного часу; мала би бути const

private:
    int hour; // 0-23 (24-годинний формат часу)
    int minute; // 0-59
    int second; // 0- 59
}; // кінець визначення класу Time з каскадуванням

// визначення конструктора класу Time
// ініціалізує кожен елемент закритих даних нулем
// для встановлення інших (ненульових) значень викликає функцію
setTime
Time::Time(int hr, int min, int sec) {
    setTime(hr, min, sec); // перевірити і встановити час
} // кінець конструктора класу Time

// встановити новий час (у всесвітнньому форматі);
// переконатися, що дані узгоджені -
// некоректні елементи обнуляються
Time& Time::setTime(int h, int m, int s) {  // !!! повертає Time&
    setHour(h); // встановити закрите поле годин
    setMinute(m); //встановити закрите поле хвилин
    setSecond(s); // встановити закрите поле секунд
    return *this; // забезпечує каскадування
} // кінець функції setTime

// встановити значення годин
Time& Time::setHour(int h) {  // !!! повертає Time&
    hour = (h >= 0 && h < 24) ? h : 0; // встановити годину
    return *this; // забезпечує каскадування
} // кінець функції setHour

// встановити значення хвилин
Time& Time::setMinute(int m) { // !!! повертає Time&
    minute = (m >= 0 && m < 60) ? m : 0; // встановити хвилину
    return *this; // забезпечує каскадування
} // кінець функції setMinute

// встановити значення секунд
Time& Time::setSecond(int s) { // !!! повертає Time&
    second = (s >= 0 && s < 60) ? s : 0; // встановити секунду
    return *this; // забезпечує каскадування
```

```
} // кінець функції setSecond

// повернути значення годин
int Time::getHour() const {
    return hour;
} // кінець функції getHour

// повернути значення хвилин
int Time::getMinute() const {
    return minute;
} // кінець функції getMinute

// повернути значення секунд
int Time::getSecond() const {
    return second;
} // кінець функції getSecond


// надрукувати значення часу у всесвітньому форматі (HH:MM:SS)
void Time::printUniversal() const {
    cout << setfill('0') << setw(2) << getHour() << ":"
        << setw(2) << getMinute() << ":" << setw(2) <<
getSecond();
} // кінець функції printUniversal()

// надрукувати значення часу у стандартному форматі (HH:MM:SS AM
або PM)
void Time::printStandard() const {
    cout << ((hour == 0) || (hour == 12) ? 12 : getHour() % 12) <<
":" <<
        setfill('0') << setw(2) << getMinute() << ":" << setw(2)
        << getSecond() << (hour < 12 ? " AM" : " PM");
} // кінець функції printStandard()


int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Time t; // неконстантний обєкт t

    // каскадні виклики функцій
    t.setHour(18).setMinute(39).setSecond(22);

    // надрукувати значення часу у всесвітньому форматі (HH:MM:SS)
    cout << "Universal time: ";
    t.printUniversal();
```

```
    // надрукувати значення часу у стандартному форматі (HH:MM:SS
AM або PM)
    cout << "\n\nStandard time: ";
    t.printStandard();

    cout << "\n\nNew standard time: ";
    t.setTime(20, 20, 20).printStandard();
    cout << endl;

    return 0;
}

Universal time: 18:39:22
Standard time: 6:39:22 PM
New standard time: 8:20:20 PM
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Студент і стипендія
Створити клас `Student`, який містить закриті поля: прізвище, середній бал та розмір стипендії. Реалізувати дружню функцію, яка визначає, чи має студент право на підвищену стипендію.
Варіант 2. Прямокутник
Створити клас `Rectangle` із закритими полями ширини та висоти. Реалізувати дружню функцію, яка порівнює площі двох прямокутників і повертає більшу площу.
Варіант 3. Автомобіль
Створити клас `Car` із закритими полями: марка, потужність двигуна та ціна. Реалізувати дружню функцію, яка порівнює ціни двох автомобілів.
Варіант 4. Коло
Створити клас `Circle` із закритим радіусом. Реалізувати дружню функцію, яка отримує два об'єкти класу `Circle` і визначає, яке коло має більшу площу.
Варіант 5. Працівник
Створити клас `Employee` із закритими полями: ім'я, стаж роботи та заробітна плата. Реалізувати дружню функцію, яка порівнює зарплати двох працівників.
Варіант 6. Товар
Створити клас `Product` із закритими полями: назва, ціна та кількість. Реалізувати дружню функцію, яка обчислює загальну вартість товарів двох об'єктів.
Варіант 7. Температура
Створити клас `Temperature`, який зберігає температуру в градусах Цельсія. Реалізувати дружню функцію, яка порівнює температури двох об'єктів і визначає більшу.
Варіант 8. Банківський рахунок
Створити клас `BankAccount` із закритими полями: номер рахунку, власник і баланс. Реалізувати дружню функцію, яка порівнює баланси двох рахунків.
Варіант 9. Точка на площині
Створити клас `Point`, який містить координати `x` та `y`. Реалізувати дружню функцію, яка визначає, яка з двох точок розташована далі від початку координат.
Варіант 10. Дроби
Створити клас `Fraction` із закритими полями чисельника та знаменника. Реалізувати дружню функцію, яка додає два дроби та повертає результат.
Варіант 11. Час
Створити клас `Time` із закритими полями: години та хвилини. Реалізувати дружню функцію, яка порівнює два об'єкти класу `Time` та визначає, який час є пізнішим.

Варіант 12. Книга
Створити клас `Book` із закритими полями: назва, автор і кількість сторінок. Реалізувати дружню функцію, яка порівнює дві книги за кількістю сторінок.
Варіант 13. Відстань
Створити клас `Distance`, який зберігає відстань у метрах. Реалізувати дружню функцію, яка додає дві відстані та повертає результат.
Варіант 14. Комплексні числа
Створити клас `Complex`, який містить дійсну та уявну частини комплексного числа. Реалізувати дружню функцію для додавання двох комплексних чисел.
Варіант 15. Годинник
Створити клас `Clock` із закритими полями години та хвилини. Реалізувати дружню функцію, яка обчислює кількість хвилин між двома моментами часу.
Варіант 16. Два класи — студент і група
Створити класи `Student` та `Group`. Клас `Student` містить прізвище та середній бал, а `Group` — назву групи та кількість студентів. Реалізувати дружню функцію, яка отримує доступ до закритих полів обох класів і виводить інформацію про студента та його групу.
Варіант 17. Два класи — автомобіль і двигун
Створити класи `Car` та `Engine`. `Car` містить марку автомобіля, а `Engine` — об'єм двигуна та потужність. Реалізувати дружню функцію, яка виводить інформацію з обох класів.
Варіант 18. Два класи — товар і магазин
Створити класи `Product` та `Shop`. `Product` містить назву і ціну товару, `Shop` — назву магазину та кількість товарів. Реалізувати дружню функцію, яка порівнює ціну товару з кількістю товарів у магазині.
Варіант 19. Два класи — спортсмен і команда
Створити класи `Athlete` та `Team`. `Athlete` містить ім'я та кількість набраних очок, `Team` — назву та кількість учасників. Реалізувати дружню функцію, яка виводить інформацію про спортсмена та його команду.
Варіант 20. Два класи — квартира і будинок
Створити класи `Apartment` та `House`. `Apartment` містить площу квартири, `House` — загальну площу та кількість квартир. Реалізувати дружню функцію, яка визначає, яку частину площі будинку займає квартира.
Варіант 21. Дружній клас для роботи з даними студента
Створити клас `Student` із закритими полями: ім'я, група та середній бал. Створити дружній клас `StudentInfo`, метод якого виводить усі закриті дані студента.
Варіант 22. Дружній клас для роботи з автомобілем
Створити клас `Car` із закритими полями: марка, рік випуску та пробіг. Створити дружній клас `CarService`, який має метод для виведення інформації про автомобіль та визначення, чи потребує він технічного обслуговування.
Варіант 23. Дружній клас для банківського рахунку
Створити клас `BankAccount` із закритими полями: власник та баланс. Створити дружній клас `BankManager`, який може переглядати баланс і змінювати його на задану суму.
Варіант 24. Дружній клас для книги
Створити клас `Book` із закритими полями: назва, автор, ціна. Створити дружній клас `Library`, який має метод для виведення інформації про книгу та зміни її ціни.
Варіант 25. Дружній оператор порівняння
Створити клас `Student`, який містить прізвище та середній бал. Перевантажити за допомогою дружньої функції оператор `>`, щоб можна було порівнювати двох студентів за середнім балом.
Варіант 26. Дружній оператор додавання
Створити клас `Distance`, який зберігає відстань у кілометрах. Перевантажити за допомогою дружньої функції оператор `+` для додавання двох об'єктів класу `Distance`.

Варіант 27. Дружній оператор порівняння цін
Створити клас `Product`, який містить назву та ціну товару. Перевантажити за допомогою дружньої функції оператор `<`, щоб порівнювати два товари за ціною.
Варіант 28. Дружній оператор ==
Створити клас `Point`, який містить координати `x` та `y`. Перевантажити за допомогою дружньої функції оператор `==`, який перевіряє, чи збігаються дві точки.
Варіант 29. Дружній клас і два об'єкти
Створити клас `Rectangle` із закритими полями ширини та висоти. Створити дружній клас `RectangleAnalyzer`, метод якого отримує два прямокутники та визначає, який із них має більшу площу.
Варіант 30. Дружня функція для двох класів
Створити класи `Worker` та `Company`. `Worker` містить прізвище та зарплату, `Company` — назву та бюджет на зарплати. Реалізувати дружню функцію, яка отримує доступ до закритих полів обох класів і визначає, чи може компанія виплатити працівникові його зарплату.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №8. ТЕМА “УСПАДКУВАННЯ КЛАСІВ

МЕТА РОБОТИ

1. Вивчити методику створення похідних класів.
2. Вивчити динамічне виділення пам'яті під об'єкти класу.
3. З’ясувати суть використання статичного елемента закритих даних класу, а також статичної відкритої get-функції.
4. Вивчити використання класів-посередників.

ХІД РОБОТИ

Завдання 8.1. Завантажити, вивчити та протестувати програму, яка демонструє динамічне виділення пам'яті під об'єкти класу та використання статичного елемента закритих даних класу, а також статичної відкритої get-функції

```
// lab_work_08_01.cpp Deitel_10_21_10_23

// Програма демонструє динамічне виділення пам'яті
// під об'єкти класу та використання статичного елемента
// закритих даних класу, який відслідковує число об'єктів класу,
// а також статичної відкритої get-функції для отримання даних.
// З цією метою пропонується клас Employee (варіант 2).

// Взагалі кажучи, кожен об'єкт класу має власний екземпляр
// всіх елементів даного класу.
// У випадку використання статичних елементів це вже не так.
// Статичний елемент даних є спільною власністю всіх об'єктів,
// а не лише одного конкретного об'єкта.
// Статичним елементом може бути як елемент даних, так і елемент-
функція.
// Оголошення статичного елемента починається словом static.
// Статичні елементи даних відрізняються від глобальних
// даних тим, що вони діють лише в області дії класу.
// Статичні елементи можуть бути відкритими, закритими та
захищенимию
// Статичні елементи даних основних типів за замовчуванням
// ініціалізуються нулем.
// Якщо потрібне інше значення, статичні елементи даних
// можуть бути ініціалізовані лише один раз.
// Константний статичний елемент даних цілого
// чи перечислюваного типів можуть ініціалізуватися при їх
// оголошеннях у визначенні класу.
// Однак всі інші статичні елементи даних мають визначатися
// поза визначенням класу і можуть бути ініціалізовані
// лише в цих визначеннях. Статичні елементи даних
// класових типів (тобто, статичні елементи об'єкти),
// які мають конструктори за замовчуванням, ініціалізуються
// за їх допомогою.
// Звертання до закритих або захищених статичних елементів
// класу виконується за допомогою відкритих елемент-функцій
```

```
// або друзів класу. З цією метою можуть бути використані
// також і закриті елемент-функції класу.
// Статичні елементи класу існують, навіть за відсутності
// хоча б одного об'єкта класу. Для звертання до відкритого
// статичного елемента класу у випадку відсутності
// хоча б одного об'єкта класу, потрібно розмістити
// перед ім'ям елемента ім'я  класу та символ ::
// До відкритих статичних елементів класу можна також
// звернутися у традиційний спосіб:
// ім'я_класу.ім'я_елемента.
// Для звертання до закритого або захищеного статичного
// елемента класу у випадку відсутності
// хоча б одного об'єкта класу, потрібно
// передбачити відкриту статичну елемент-функцію,
// виклик якої виконуємо так ім'я_класу.ім'я_функції.

// У нашій програмі використано закритий
// статичний елемент даних count та відкриту
// статичну елемент-функцію getCount.
// Стрічка визначає в області дії файлу (поза класом)
// та ініціалізує нулем елемент даних count.
// Стрічки визначають статичну елемент-функцію getCount.
//

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <cstring>
#include <windows.h>

using namespace std;

class Employee {
public:
    Employee(const char* const, const char* const); // конструктор
    ~Employee(); // деструктор
    void print() const; // надрукувати обєкт
    const char* getFirstName() const; // повернути імя
    const char* getLastName() const; // повернути прізвище

    // статична елемент-функція
    static int getCount(); // повернути число створених обєктів

private:
    char* firstName;
    char* lastName;

    // static data
    static int count; // число створених обєктів
```

```
}; // кінець класу Employee


// визначити та ініціалізувати статчний елемент даних
int Employee::count = 0;

// визначити статичну елемент-функцію getCount
// яка повертає число створених обєктів
int Employee::getCount() {
    return count;

} // кінець елемент-функції getCount

// конструктор - динамічно виділяє память для імені та прізвища
// і викликає функцію strcpy() для копіювання імені та прізвища в
обєкт
Employee::Employee(const char* const first, const char* const
last) {

    int len = strlen(first) + 1;
    firstName = new char[len];
    strcpy_s(firstName, len, first);

    len = strlen(last) + 1;
    lastName = new char[len];
    strcpy_s(lastName, len, last);
    count++; // збільшити статичний лічильник

    // вивести обєкт, щоб переконатися про виклик конструктора
    cout << "Employee constructor for "
        << firstName << ' ' << lastName << " called." << endl;

} //  кінець конструктора класу Employee


// деструктор ~Employee - надрукувати обєкт Employee,
// щоб підтвердити виклик деструктора
Employee::~Employee() {
    cout << "~Employee destructor for "
        << firstName << ' ' << lastName << " called." << endl;
    delete[] firstName; // звільнити память
    delete[] lastName; // звільнити память
    count--; // зменшити значення статичногоо лічильника
} // кінець деструктора ~Employee

// повернути імя працівника
const char* Employee::getFirstName() const {
    // кваліфікатор const не дає клієнту змоги модифікувати
закриті дані
    // деструктор звільнює память і вказівник стає невизначеним
    return firstName;
```

```
} //  кінець функції getFirstName

// повернути прізвище працівника
const char* Employee::getLastName() const {
    // кваліфікатор const не дає клієнту змоги модифікувати
закриті дані
    // деструктор звільнює память і вказівник стає невизначеним
    return lastName;
} //  кінець функції getLastName

int main()
{   // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // використати імя класу і операцію дозволу області дії
    // щоб звернутися до статичного елемента
    cout << "Number of employees before creation of any objects is
"
        << Employee::getCount() << endl; // використати імя класу

    // використати new для динамічного виділення двох обєктів
Employee
    Employee* e1Ptr = new Employee("Susan", "Baker");
    Employee* e2Ptr = new Employee("Robert", "Jones");


    // викликати для першого обєкта
    cout << "Number of employees after objects are instantiated is
"
        << e1Ptr->getCount() << endl;

    cout << "\n\nEmployee 1:"
        << e1Ptr->getFirstName() << " " << e1Ptr->getLastName()
        << "\nEmployee 2:"
        << e2Ptr->getFirstName() << " " << e2Ptr->getLastName() <<
"\n\n";

    delete e1Ptr; // звільнити память
    e1Ptr = 0; // відєднати вказівник від області дії вільної
памяті
    delete e2Ptr; // звільнити память
    e2Ptr = 0; // відєднати вказівник від області дії вільної
памяті

    // обєктів немає, тому для виклику статичної функції
    // використовується імя класу і операція дозволу області дії
    cout << "Number of employees after objects are deleted is "
        << Employee::getCount() << endl;
```

```
    return 0;
}

Number of employees before creation of any objects is 0
Employee constructor for Susan Baker called.
Employee constructor for Robert Jones called.
Number of employees after objects are instantiated is 2


Employee 1:Susan Baker
Employee 2:Robert Jones

~Employee destructor for Susan Baker called.
~Employee destructor for Robert Jones called.
Number of employees after objects are deleted is 0
```

Завдання 8.2. Завантажити, вивчити та протестувати програму, яка демонструє використання класів-посередників та динамічне виділення пам'яті під об'єкти класу

```
// lab_work_08_02.cpp  Deitel_10_24_10_27

// Програма демонструє використання класів-посередників
// та динамічне виділення пам'яті під об'єкти класу.
// З цією метою пропонуються класи
// Implementation (основний клас) та Interface (клас-посередник).
// Метою є закрити доступ користувачу до елементів основного
класу.

// Клас-посередник Interface має єдиний закритий елемент даних -
// це вказівник на об'єкт класу Implementation.
// Використання цього вказівника дозволяє нам сховати від
// користувача деталі реалізації класу Implementation.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>
using namespace std;

class Implementation
{
public:
    Implementation(int v) // конструктор за замовчуванням
        : value(v) // ініціалізувати value значенням v
    {
        // порожнє тіло конструктора
    }
```

```
    // встановити value рівним значенню v
    void setValue(int v) { // v має бути перевіреним
        value = v;
    } // кінець setValue

     // повернути value
    int getValue() const {
        return value;
    } // кінець getValue

private:
    int value; // дані, які ми хочемо приховати від клієнта
}; // кінець визначення класу Implementation

class Interface
{
public:
    Interface(int); // конструктор за замовчуванням
    void setValue(int v);
    int getValue() const;
    ~Interface(); // деструктор
private:
    // потребує випереджаючого оголошення
    Implementation* ptr; // дані, які ми хочемо приховати від
клієнта
}; // кінець визначення класу Implementation

// конструктор клас Interface
Interface::Interface(int v)
    : ptr(new Implementation(v)) { // ініціалізувати ptr новим
обєктом класу Implementation
    // поржнє тіло
} // кінець визначення конструктора класу Interface

// викликати функцію setValue класу Implementation
void Interface::setValue(int v) {
    ptr->setValue(v);
} // кінець setValue

// викликати функцію getValue класу Implementation
int Interface::getValue() const {
    return ptr->getValue();
} // кінець getValue

// деструктор класу
Interface::~Interface() {
    delete ptr;
} // кінець  деструктора класу ~Interface

int main()
{   // Очищення екрана
```

```
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Interface i(5); // створити обєкт класу Interface

    cout << "Interface contains: " << i.getValue()
        << " before setValue " << endl;

    i.setValue(10);

    cout << "\nInterface contains: " << i.getValue()
        << " after setValue " << endl;

    cout << endl;
    // закоментовані випадки noon.setHour(12)
    // та noon.printStandard() компілюються з помилками
    return 0;

}

Interface contains: 5 before setValue
Interface contains: 10 after setValue
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Визначити ієрархію класів (відповідно до варіанту). Для визначення ієрархії класів зв’язати наслідуванням класи, наведені для заданого варіанту. З перерахованих класів вибрати один, котрий буде стояти на чолі ієрархії. Реалізувати класи, визначити в класах всі необхідні конструктори й деструктори. Написати демонстраційну програму, у якій створюються об’єкти різних класів.

Варіант 1. студент, викладач, персона, зав. кафедри;
Варіант 2. службовець, персона, робітник, інженер;
Варіант 3. робітник, кадри, інженер, адміністрація;
Варіант 4. деталь, механізм, виріб, вузол;
Варіант 5. організація, страхова компанія, суднобудівна компанія, завод;
Варіант 6. журнал, книга, друковане видання, підручник;
Варіант 7. тест, іспит, випускний іспит, випробування;
Варіант 8. місце, область, місто, мегаполіс;
Варіант 9. іграшка, продукт, товар, молочний продукт;
Варіант 10. квитанція, накладна, документ, чек;
Варіант 11. автомобіль, поїзд, транспортний засіб, експрес;
Варіант 12. фігура, чотирикутник, квадрат, коло;
Варіант 13. рослина, дерево, вазон, клен;
Варіант 14. ссавець, хижий звір, птах, живий організм;
Варіант 15. меблі, каркасні меблі, шафа, ліжко;
Варіант 16. людина, дитина, дошкільнятко, учень;
Варіант 17. персона, студент, робітник, староста групи;
Варіант 18. флешка, пристрій, usb-пристрій, mp3-плеєр;
Варіант 19. блог, інформаційне видання, електронне видання, журнал;

Варіант 20. інтернет-ресурс, форум, чат, сайт.
Варіант 21. транспортний засіб, автобус, автомобіль, маршрутне таксі;
Варіант 22. працівник, викладач, лаборант, завідувач лабораторії;
Варіант 23. пристрій, комп'ютер, ноутбук, планшет;
Варіант 24. документ, паспорт, студентський квиток, посвідчення;
Варіант 25. тварина, домашня тварина, собака, кішка;
Варіант 26. навчальний заклад, школа, коледж, університет;
Варіант 27. приміщення, житлове приміщення, квартира, гуртожиток;
Варіант 28. товар, електронний товар, смартфон, ноутбук;
Варіант 29. транспорт, наземний транспорт, літак, вертоліт;
Варіант 30. видання, друковане видання, газета, науковий журнал

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №9. ТЕМА «ПОЛІМОРФІЗМ, ВІРТУАЛЬНІ ФУНКЦІЇ ТА АБСТРАКТНІ КЛАСИ

МЕТА РОБОТИ

1. Вивчити методику створення віртуальних функцій та абстрактних класів.
2. Вивчити методику перевантаження операції передачі даних в потік введення.
3. Вивчити методику перевантаження операції вибірки даних з потоку з потоку виведення
4. Вивчити використання перевантаження різних операцій для роботи з масивами

ХІД РОБОТИ

Завдання 9.1. Завантажити, вивчити та протестувати програму, яка демонструє перевантаження двох операцій: передачі даних в потік введення >> та вибірки даних з потоку виведення <<

```
// ind_09_01.cpp  Deitel_11_3_11_5

// Програма демонструє перевантаження двох операцій:
// передачі даних в потік введення >> та вибірки даних з потоку виведення
<<.

// З цією метою пропонується клас PhoneNumber (варіант 1).
// Клас PhoneNumber забезпечує формування
// номера телефону у форматі (123) 456-7890,
// (123) - три цифри регіонального коду;
// 456 - три цифри АТС;
// 7890 - чотири цифри номера лінії.
// Вважається, що номери телефонів вводяться коректно.

// Клас PhoneNumber містить три закритих елементи даних,
// тобто три цілих числа, з яких складається номер телефону.

// Введення номера телефону виконує дружня функція класу
// friend istream& operator>>(istream&, PhoneNumber&),
// яка перевантажує операцію вибірки даних з потоку введення >>.
// Функція-оператор вибірки даних з потоку operator>> приймає два
аргументи:
// istream &input - посилання з ім'ям input на об'єкт класу istream;
// PhoneNumber &number - посилання з ім'ям number на об'єкт класу
PhoneNumber.
// Функція - операція ooperator>>  вводить з клавіатури значення об'єкта,
// тобто змінює його - тому використано як другий аргумент неконстантне
посилання.
// Функція-операція operator>> повертає черерез своє ім'я посилання на
об'єкт класу istream.
// Функція-операція operator>> є глобальною функцією.
// Функція-операція operator>> є функцією, дружньою до класу PhoneNumber.
// Коли компілятор зустрічає у головній фунуції оператор cin >> phone;
// це рівносильно виклику функції operator>>(cin, phone);
// Коли ця функція виконується, параметр input стає псевдонімом для cin,
```

```
// а параметр number стає псевдонімом для phone.
// Функція зчитує три частини телефонного номера в закриті елементи об'єкта
// класу, на який посилається number, тобто в areaCode, exchange та line.
// Під час введення стрічкових даних маніпулятор setw() обмежує число
введених
// символів значенням свого аргумента, наприклад, setw(3) вводить лише 3
символи.
// Елемент-функція ignore() класу istrem lдозволяє пропустити символи ( ) -
та пробіл.

// Виведення номера телефону виконує дружня функція класу
// friend ostream& operator<<(ostream&, const PhoneNumber&),
// яка перевантажує операцію передачі даних в потік виведення <<.
// Функція-оператор передачі даних в потік operator<< приймає два
аргументи:
// const ostream &output - посилання з ім'ям output на константний об'єкт
класу ostream;
// PhoneNumber &number - це посилання з ім'ям number на об'єкт класу
PhoneNumber.
// Функція-операція ooperator<< виводить на екран значення об'єкта,
// але не змінює його - тому використано як другий аргумент константне
посилання.
// Функція-операція ooperator<< повертає черерез своє ім'я посилання на
об'єкт класу ostream.
// Функція-операція operator<< є глобальною функцією.
// Функція-операція operator<< є функцією, дружньою до класу PhoneNumber.
// Коли компілятор зустрічає у головній фунуції оператор cout >> phone;
// це рівносильно виклику функції operator<<(cout, phone);
// Коли ця функція виконується, параметр output стає псевдонімом для cout,
// а параметр number стає псевдонімом для phone.
// Функція виводить на друк телефонний номер як об'єкт класу,
// на який посилається number.

// Перевантажувати операцію за допомогою елемент-функції класу можна лише
тоді,
// коли операнд в лівій частині є об'єктом класу, в який входить функція.
// Тому у нашій програмі обидві операції перевантажуються за допомогою
// глобальних дружніх функцій, а не елемент-функцій класу.

// Перевантажені операції введення та виведення не оголошуються як public,
// оскільки вони мають доступ до закритих даних класу і можуть змінювати
їх.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>

using namespace std;
```

```
class PhoneNumber {
    friend ostream& operator<<(ostream&, const PhoneNumber&);
    friend istream& operator>>(istream&, PhoneNumber&);
private:
    // phone number has form  (123) 456-7890
    string areaCode; // 3 цифри регіонального коду
    string exchange; // 3 цифри АТС
    string line;     // 4 цифри номера лінії
}; // Кінець класу PhoneNumber

// Функція operator<< - перевантаження операції
// передачі даних в потік виведення <<
// не може бути елементом класу, якщо ми хочемо
// застосувати її у формі:  cout << somePhoneNumber;
ostream& operator<<(ostream& output, const PhoneNumber& number) {
    output << "(" << number.areaCode << ") "
        << number.exchange << "-" << number.line;
    return output; // допускає форму виведення cout << a << b << C;
} // кінець функції operator<<

// Функція operator>> - перевантаження операції
// вибірки даних з потоку введення >>
// не може бути елементом класу, якщо ми хочемо
// застосувати її у формі:  cin >> somePhoneNumber;
istream& operator>>(istream& input, PhoneNumber& number) {
    input.ignore(); // пропустити відкриваючу дужку (
    input >> setw(3) >> number.areaCode; // ввести код регіону
    input.ignore(2); // пропустити закриваючу дужку ) і пробіл
    input >> setw(3) >> number.exchange; // ввести код АТС
    input.ignore(); // пропустити дефіс (-)
    input >> setw(4) >> number.line; // ввести номер лінії
    return input; // допускає форму введення cin >> a >> b >> C;
} // кінець функції operator>>

int main()
{
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    PhoneNumber phone;

    cout << "Enter phone number in the form: (123) 456-7890:" << endl;

    // cin >> phone викликає оператор operator>>,
    // неявно генеруючи виклик глобальної функції operator>>(cin, phone)
    //cin >> phone;
    operator>>(cin, phone);
```

```
    cout << endl;

    cout << "The phone number entered was:\n";

    // cout << phone викликає оператор operator<<,
    // неявно генеруючи виклик глобальної функції operator<<(cout, phone)
    // cout << phone <<endl;
    operator<<(cout, phone);
    cout << endl;

    return 0;

}


Enter phone number in the form: (123) 456-7890:
(132) 432-8769

The phone number entered was:
(132) 432-8769
```

Завдання 9.2. Завантажити, вивчити та протестувати програму, яка демонструє використання перевантаження операцій для роботи з масивами

```
// lab_work_09_02.cpp  Deitel_11_06_11_08

// Програма демонструє використання перевантаження операцій для роботи з
масивами.
// З цією метою пропонується клас Array (варіант 1).
// Клас Array забезпечує обробку масивів за рахунок
// використання перевантажених операцій.

// Клас Array містить два закритих елементи даних:
// int size - розмір масиву;
// int *ptr - вказівник на початок масиву.
// Клас Array містить відкриті елементи-функції:
// конструктор за замовчуванням
// конструктор копіювання
// деструктор
// перевантаження операції присвоєння =
// перевантаження операції рівності ==
// операції індексації для неконстантних об'єктів int& operator[](int)
// операції індексації для константних об'єктів int operator[](int) const

// Введення масиву виконує глобальна дружня функція класу
// friend istream& operator>>(istream&, Array&),
// яка перевантажує операцію вибірки даних з потоку введення >>.
// Виведення масиву виконує глобальна дружня функція класу
// friend ostream& operator<<(ostream&, const Array&),
// яка перевантажує операцію передачі даних в потік виведення.
```

```
// Перевантажені операції передачі в потік operator<< та вибірки з потоку
operator>>
// Операції передачі в потік operator<< та вибірки з потоку operator>>
// є глобальними дружніми функціями класу Array.
// Коли компілятор зустрічає команду cout << arrayObject,
// він генерує виклик функції operator<<(cout, arrayObject).
// Тим самим output стає синонімом cout,  а об’єкт класу Array стає
синонімом arrayObject.
// Коли компілятор зустрічає команду cin << arrayObject,
// він генеру’ виклик функції operator>>(cin, arrayObject).
// Тим самим input стає синонімом cin, а об’єкт класу Array стає синонімом
arrayObject.

// Перевантажена операція присвоєння operator=
// Коли компілятор зустрічає у програмі integers1 = integers2,
// він використовує перевантажену операцію opereator=
// викликом функції integers1.operator=(integers2).
// Спочатку ця функція перевіряє перевірку на самоприсвоєння,
// тобто перевіряється, чи посилання на integers2 не дорівнює
// посиланню на integers1.
// Якщо спроба самоприсвоєння має місце, то воно не виконується.
// Далі функція перевіряє, чи рівні розміри масивів.
// Якщо це так, то розмір масиву  integers1 не змінюється.
// Якщо ні, то функція використовує операцію delete
//для звільнення пам’яті виділеної раніше під масив integers1,
// копіює розмір масиву integers2 у розмір масиву integers1,
// операцією new виділяє пам’ять під новий масив integers1
// та присвоює елементу ptr масиву integers1 значення вказівника,
// яке повертає операція new.
// Далі в обох випадках цикл for копіює значення елементів
// масиву integers2 в новостворений масив integers1.
// Функція operator=  повертає через своє ім’я значення вказівника *this
// тобто константне посилання об’єкта на самого себе.
// Це дозволяє реалізовувати каскадні присвоєння як от x=y=z.

// Перевантажені операції рівності operator== та нерівності operator!=
// Коли компілятор зустрічає програмі  вираз integers1 == integers2 ,
// він генерує виклик функції    integers1.operator==(integers2).
// Елемент-функція operator== повертає значення false,
// якщо розміри  масивів різні.
// Якщо розміри  ж масивів однакові, функція порівнює відповідні елементи
обох масивів.
// Якщо відповідні елементи обох масивів рівні, то функція повертає
значення true,
// інакше функція повертає значення false.
// Елемент-функція operator!= використовує перевантажену функцію operator==
,
// і повертає протилежний результат.

// Перевантажені операції індексації operator[]
// В програмі передбачені дві перевантажені операції індексації operator[]
```

```
// Коли компілятор зустрічає програмі  вираз integers1[5]  (неконстантний об’єкт),
// він генерує відповідний виклик функції    integers1.operator[](5).
// Якщо операція індексації застосовується до константного об’єкта,
// то компілятор генерує виклик константної версії функції  operator[].
// Нехай, наприклад, у програмі оголошено константний об’єкт const  Array z(5).
// Тоді, для того щоб виконати оператор cout << z[5] << endl;
// потрібно викликати константну версію функції operator[].
// Кожен варіант функції operator[] перевіряє, чи знаходиться індекс масиву
// у допустимих межах. Якщо ні, то кожна з функцій друкує повідомлення про помилку
//  і завершує виконання програми викликом функції exit() з бібліотеки <cstdlib>.
// Якщо індекс знаходиться у допустимих межах, то неконстантна версія функції
// повертає відповідний елемент масиву у формі посилання,
// яке може бути використане як значення lvalue, яке можна модифікувати,
// наприклад, його можна використати у лівій частині оператора присвоєння.
// Якщо індекс знаходиться у допустимих межах, то константна версія функції
// повертає значення відповідного елемента масиву як значення rvalue.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>

using namespace std;

class Array {
    friend ostream& operator<<(ostream&, const Array&);
    friend istream& operator>>(istream&, Array&);
public:
    Array(int = 10); // конструктор за замовчуванням
    Array(const Array&); // конструктор копіювання
    ~Array(); // деструктор
    int getSize() const; // повертає розмір масиву

    const Array& operator=(const Array&); // операція присвоєння
    bool operator==(const Array&) const; // операція перевірки рівності

    // операція перевірки нерівності; результат, протилежний операції ==
    bool operator!=(const Array& right) const {
        return !(*this == right); // викликає Array::operator==
    } // кінець функції operator!=

    // індексація для неконстанних обєктів повертає lvalue
    int& operator[](int);

    // індексація для константних обєктів повертає rvalue
```

```
    int operator[](int) const;

private:
    int size; // розмір масиву, заданого вказівником
    int* ptr; // вказівник на перший елемент масиву
}; // кінець класу Array

// конструктор Array за замовчуванням ( розмір за замовчуванням дорівнює
10)
Array::Array(int arraySize) {
    size = (arraySize > 0 ? arraySize : 10); // перевірити arraySize
    ptr = new int[size]; // виділити простір для масиву

    for (int i = 0; i < size; i++)
        ptr[i] = 0; // встановити елемент масиву-вказівника
} // кінець конструктора Array за замовчуванням

// конструктор копіювання для класу Array
// для попередження нескінченної рекурсії повинен повертати посилання
Array::Array(const Array& arrayToCopy)
    : size(arrayToCopy.size) {
    ptr = new int[size]; // виділити простір для масиву

    for (int i = 0; i < size; i++)
        ptr[i] = arrayToCopy.ptr[i]; //копіювати в обєкт
} // кінець конструктора копіювання Array

// деструктор для класу Array
Array::~Array() {
    delete[] ptr; // звільнити память під масив
} // кінець деструктора

// повернути число елементів Array
int Array::getSize() const {
    return size; // число елементів Array
} // кінець функції getSize()

// перевантажена операція присвоєння:
// константа, що повертається, запобігає: (a1 = a2) = a2
const Array& Array::operator=(const Array& right) {
    if (&right != this) { // уникати самоприсвоєння
        // для масивів різних розмірів вивільнити початковий (вхідний)
масив
        // у лівій частині, далі виділити новий масив для лівої частини
        if (size != right.size) {
            delete[] ptr; // звільнити память
            size = right.size; // перевстановити розмір цього обєкта
            ptr = new int[size]; // створити простір для копії
        } // кінець внутрішнього if

        for (int i = 0; i < size; i++)
```

```
            ptr[i] = right.ptr[i]; //копіювати масив обєкт
    } // кінець зовнішнього if

    return *this; // дозволяє писати, наприклад, x=y=z
} // кінець функції operator=

// визначити, чи рівні два масиви; якщо рівні
// повернути true, в протилежному випадку повернути false
bool Array::operator==(const Array& right) const {
    if (size != right.size) // уникати самоприсвоєння
        return false; // масиви з різним числом елементів

    for (int i = 0; i < size; i++)
        if (ptr[i] != right.ptr[i])
            return false;  // вміст масивів різний

    return true; // масиви рівні
} // кінець функції operator==

// перевантажена операція індексації для неконстантних масивів
// повернення посилання створює модифіковане lvalue
int& Array::operator[] (int subscript) {
    // перевірити індекс на вихід за межі масиву
    if (subscript < 0 || subscript >= size) {
        cerr << "\nError: Subscript " << subscript
            << " out of range" << endl;
        exit(1); // завершити програму; індекс поза межами
    } // кінець if

    return ptr[subscript]; // повернення посилання
} // кінець функції operator[]

// перевантажена операція індексації для константних масивів
// повернення константного посилання створює модифіковане rvalue
int Array::operator[] (int subscript) const {
    // перевірити індекс на вихід за межі масиву
    if (subscript < 0 || subscript >= size) {
        cerr << "\nError: Subscript " << subscript
            << " out of range" << endl;
        exit(1); // завершити програму; індекс поза межами
    } // кінець if

    return ptr[subscript]; // повертає копію елемента
} // кінець функції operator[]

// перезавантажена операція введення для класу Array:
// вводить значення для всього масиву
istream& operator>>(istream& input, Array& a) {
    for (int i = 0; i < a.size; i++)
        input >> a.ptr[i];
    return input; // дозволяє писати cin >> x >> y;
```

```
} // кінець функції operator>>

// перезавантажена операція виведення для класу Array :
// виводить значення для всього масиву
ostream& operator<<(ostream& output, const Array& a) {
    int i;
    // вивести закритий масив, що адресується вказівником
    for (i = 0; i < a.size; i++) {
        output << setw(12) << a.ptr[i];
        if ((i + 1) % 4 == 0) // по 4 числа у стрічці виведення
            output << endl;
    } // кінець оператора for

    if ((i + 1) % 4 != 0) // закінчити останню стрічку виведення
        output << endl;

    return output; // дозволяє писати cout << x << y;
} // кінець функції operator<<

int main()
{
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Array integers1(7); // 7 - елементний Array
    Array integers2; // 10 - елементний Array за замовчуванням

    // надрукувати розмір і вміст integers1
    cout << "Size of Array integers1 is "
        << integers1.getSize()
        << "\nArray after initialisation:\n" << integers1;

    // надрукувати розмір і вміст integers2
    cout << "\nSize of Array integers2 is "
        << integers2.getSize()
        << "\nArray after initialisation:\n" << integers2;

    // ввести і надрукувати integers1 та integers2
    cout << "\nEnter 17 integers:" << endl;
    cin >> integers1 >> integers2;
    cout << "\nAfter input, the Arrays contain:\n"
        << "integers1:\n" << integers1
        << "\nintegers2:\n" << integers2;

    // застосувати перевантажену операцію перевірки нерівності (!=)
    cout << "\nEvaluating: integers1 != integers2" << endl;
    if (integers1 != integers2)
```

```
        cout << "\nArrays integers1 and integers2 are not equal" << endl;

    // створити масив integers3, використовуючи як ініціалізатор
    // integers1; надрукувати його розмір і вміст
    Array integers3(integers1); // викликає конструктор копіювання
    cout << "\nSize of Array integers3 is "
        << integers3.getSize()
        << "\nArray after initialization:\n" << integers3;

    // застосувати перевантажену операцію присвоєння (=)
    cout << "\n\nAssigning integers2 to integers1:" << endl;
    integers2 = integers1; // зауважуємо, що цільовий масив меньший
    cout << "integers1:\n" << integers1
        << "\nintegers2:\n" << integers2;

    // застосувати перевантажену операцію перевірки рівності (==)
    cout << "\n\nEvaluating: integers1 == integers2" << endl;
    if (integers1 == integers2)
        cout << "\nArrays integers1 and integers2 are equal" << endl;

    // застосувати перевантажену операцію індексації, що дає rvalue
    cout << "\ninteger1[5] is " << integers1[5];

    // застосувати перевантажену операцію індексації, що дає lvalue
    cout << "\n\nAssigning 100 to integer1[5] is " << endl;
    integers1[5] = 1000;
    cout << "\nintegers1:\n " << integers1;

    // спроба використати індекс поза межами діапазону
    cout << "\n\Attempt to assign 1000 to integer1[15]" << endl;
    //integers1[15] = 1000; // помилка! вихід за межі діапазону

}

Size of Array integers1 is 7
Array after initialisation:
           0           0           0           0
           0           0           0
Size of Array integers2 is 10
Array after initialisation:
           0           0           0           0
           0           0           0           0
           0           0

Enter 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the Arrays contain:
integers1:
           1           2           3           4
           5           6           7
```

```
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 != integers2

Arrays integers1 and integers2 are not equal

Size of Array integers3 is 7
Array after initialization:
           1           2           3           4
           5           6           7

Assigning integers2 to integers1:
integers1:
           1           2           3           4
           5           6           7
integers2:
           1           2           3           4
           5           6           7

Evaluating: integers1 == integers2

Arrays integers1 and integers2 are equal

integer1[5] is 6

Assigning 100 to integer1[5] is

integers1:
            1           2           3           4
           5         100           7
Attempt to assign 1000 to integer1[15]

Error: Subscript 15 out of range
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Створити абстрактний клас Фігура з віртуальними методами обчислення площі та периметру. Створити похідні класи: прямокутник, круг, трапеція зі своїми функціями площі та периметру. Самостійно визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних.

Варіант 2. Створити абстрактний базовий клас Тіло з віртуальними функціями обчислення площі поверхні та об’єму. Створити похідні класи: паралелепіпед, куля зі своїми функціями обчислення площі та об’єму.

Варіант 3. Створити абстрактний клас Валюта для роботи з грошима. Визначити віртуальні функції переведення в гривні і виведення на екран. Реалізувати похідні класи долар та євро зі своїми функціями переведення та виведення на екран.

Варіант 4. Створити базовий клас Трикутник з віртуальними функціями обчислення площі та периметра. Поля даних повинні включати дві сторони і кут між ними. Визначити класи-нащадки: прямокутний трикутник, рівнобедрений трикутник, рівносторонній трикутник зі своїми методами обчислення площі та периметра.

Варіант 5. Створити абстрактний базовий клас Корінь з віртуальними методами обчислення коренів рівнянь та виведення на екран. Визначити похідні класи: лінійне рівняння та квадратне рівняння з власними методами знаходження та виведення на екран коренів.

Варіант 6. Створити абстрактний базовий клас Функція з віртуальними методами обчислення значення функції в заданій точці та виведення результату на екран. Визначити похідні класи: еліпс та гіпербола з власними методами.

Варіант 7. Створити абстрактний базовий клас Number з віртуальними методами – арифметичними операціями додавання, віднімання, множення, ділення, піднесення до степеня. Визначити похідні класи: ціле та дійсне число.

Варіант 8. Створити абстрактний базовий клас з віртуальними арифметичними операціями. Реалізувати похідні класи: комплексне число та раціональний дріб.

Варіант 9. Створити абстрактний базовий клас з віртуальним методом збільшення на одиницю. Реалізувати похідні класи: дата і час з власними методами збільшення на одиницю.

Варіант 10. Створити абстрактний базовий клас прогресія з віртуальними функціями визначення j-го елемента прогресії та суми прогресії. Визначити похідні класи: арифметична та геометрична прогресія з власними відповідними функціями.

Варіант 11. Створити абстрактний клас Фігура з віртуальними методами обчислення площі та периметру. Створити похідні класи: прямокутник, круг, трапеція зі своїми функціями площі та периметру. Самостійно визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних.

Варіант 12. Створити абстрактний базовий клас Тіло з віртуальними функціями обчислення площі поверхні та об’єму. Створити похідні класи: паралелепіпед, куля зі своїми функціями обчислення площі та об’єму.

Варіант 13. Створити абстрактний клас Валюта для роботи з грошима. Визначити віртуальні функції переведення в гривні і виведення на екран. Реалізувати похідні класи долар та євро зі своїми функціями переведення та виведення на екран.

Варіант 14 Створити базовий клас Трикутник з віртуальними функціями обчислення площі та периметра. Поля даних повинні включати дві сторони і кут між ними. Визначити класи-нащадки: прямокутний трикутник, рівнобедрений трикутник, рівносторонній трикутник зі своїми методами обчислення площі та периметра.

Варіант 15. Створити абстрактний базовий клас Корінь з віртуальними методами обчислення коренів рівнянь та виведення на екран. Визначити похідні класи: лінійне рівняння та квадратне рівняння з власними методами знаходження та виведення на екран коренів.

Варіант 16. Створити абстрактний базовий клас Функція з віртуальними методами обчислення значення функції в заданій точці та виведення результату на екран. Визначити похідні класи: еліпс та гіпербола з власними методами.

Варіант 17. Створити абстрактний базовий клас Number з віртуальними методами – арифметичними операціями додавання, віднімання, множення, ділення, піднесення до степеня. Визначити похідні класи: ціле та дійсне число.

Варіант 18. Створити абстрактний базовий клас з віртуальними арифметичними операціями. Реалізувати похідні класи: комплексне число та раціональний дріб.

Варіант 19. Створити абстрактний базовий клас з віртуальним методом збільшення на одиницю. Реалізувати похідні класи: дата і час з власними методами збільшення на одиницю.

Варіант 20. Створити абстрактний базовий клас прогресія з віртуальними функціями визначення j-го елемента прогресії та суми прогресії. Визначити похідні класи: арифметична та геометрична прогресія з власними відповідними функціями.

Варіант 21. Створити абстрактний базовий клас Транспорт з віртуальними методами визначення максимальної швидкості та виведення інформації на екран. Реалізувати похідні класи автомобіль, мотоцикл, велосипед зі своїми функціями визначення швидкості та виведення інформації. Самостійно визначити необхідні поля базового та похідних класів.

Варіант 22. Створити абстрактний базовий клас Працівник з віртуальними методами обчислення заробітної плати та виведення інформації на екран. Реалізувати похідні класи погодинний працівник, працівник з фіксованою зарплатою, працівник з відрядною оплатою зі своїми методами обчислення заробітної плати.

Варіант 23. Створити абстрактний базовий клас Товар з віртуальними методами визначення вартості та виведення інформації на екран. Реалізувати похідні класи продукт, одяг, електронний пристрій зі своїми методами визначення вартості з урахуванням особливостей кожного типу товару.

Варіант 24. Створити абстрактний базовий клас Квиток з віртуальними методами обчислення вартості та виведення інформації на екран. Реалізувати похідні класи квиток на потяг, квиток на автобус, авіаквиток зі своїми функціями обчислення вартості.

Варіант 25. Створити абстрактний базовий клас Геометрична фігура з віртуальними методами обчислення площі та виведення параметрів фігури. Реалізувати похідні класи ромб, паралелограм, еліпс зі своїми методами обчислення площі.

Варіант 26. Створити абстрактний базовий клас Пристрій з віртуальними методами визначення потужності та виведення технічної інформації. Реалізувати похідні класи телевізор, комп'ютер, кондиціонер зі своїми методами визначення потужності.

Варіант 27. Створити абстрактний базовий клас Рахунок з віртуальними методами обчислення кінцевого балансу та виведення інформації. Реалізувати похідні класи банківський рахунок, депозитний рахунок, кредитний рахунок зі своїми правилами обчислення кінцевого балансу.

Варіант 28. Створити абстрактний базовий клас Видання з віртуальними методами визначення вартості та виведення інформації. Реалізувати похідні класи книга, підручник, енциклопедія зі своїми методами обчислення вартості залежно від параметрів видання.

Варіант 29. Створити абстрактний базовий клас Файл з віртуальними методами визначення розміру файлу та виведення інформації на екран. Реалізувати похідні класи текстовий файл, графічний файл, відеофайл зі своїми методами визначення розміру.

Варіант 30. Створити абстрактний базовий клас Гра з віртуальними методами визначення кількості учасників та виведення інформації про гру. Реалізувати похідні класи настільна гра, спортивна гра, комп'ютерна гра зі своїми методами визначення кількості учасників та виведення інформації.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №10. ТЕМА «ШАБЛОНИ ТА УЗАГАЛЬНЕНЕ ПРОГРАМУВАННЯ»

МЕТА РОБОТИ

1. Вивчити методику створення шаблонів функцій та шаблонів класів.
2. Ознайомитися з методикою використання перевантаження операцій для роботи зі стрічками.
3. Вивчити використання перевантаження операцій інкремента та декремента в префіксній та постфіксній формах для роботи з датами.

ХІД РОБОТИ

Завдання 10.1. Завантажити, вивчити та протестувати програму, яка демонструє використання перевантаження операцій для роботи зі стрічками

```
// lab_work_10_01.cpp  Deitel_11_09_11_11

// Програма демонструє використання перевантаження операцій
// для роботи зі стрічками .
// З цією метою пропонується клас String (варіант 1).
// Клас String забезпечує обробку масивів за рахунок
// використання перевантажених операцій.

// Клас String містить два закритих елементи даних:
// int length - довжина стрічки;
// char *sPtr - вказівник на початок стрічки;
// void setString(const char *) - допоміжна функція.

// Клас String містить відкриті елементи-функції:
// конструктор за замовчуванням
// конструктор копіювання
// деструктор
// перевантаження операції присвоєння operator=
// перевантаження операції конкатенації operator &
// перевантаження операції чи є стрічка порожньою operator!
// перевантаження операції рівності operator ==
// перевантаження операції  нерівності operator operator <
// перевантаження операції  нерівності operator operator >
// перевантаження операції  нерівності operator operator!=
// перевантаження операції  нерівності operator operator<=
// перевантаження операції  нерівності operator operator>=
// перевантаження операції індексації lvalue - char&
operator[](int)
// перевантаження операції індексації rvalue - char
operator[](int) const
// перевантаження операції повернення підстрічки - String
operator()(int, int = 0) const
// повернення довжини стрічки int getLength() const

// Введення масиву виконує дружня функція класу
```

```
// friend istream& operator>>(istream&, String&),
// яка перевантажує операцію вибірки даних з потоку введення >>.
// Виведення масиву виконує дружня функція класу
// friend ostream& operator<<(ostream&, const String&),
// яка перевантажує операцію передачі даних в потік виведення.

// Конструктор перетворення типу ( і конструктор за замовчуванням
одночасно)
// Цей конструктор приймає аргумент const char* (по замовчуванню
порожню стрічку)
// та ініціалізує об’єкт String, що містить стрічку символів типу
char*,
// виконуючи при цьому перетворення типів.
// Операція перетворення типів може застосовуватися до об’єктів
одного класу,
// перетворюючи його в об’єкти іншого класу.
// Прототип функції     A :: operator char*() const;    оголошує
перевантажену
// функцію-операцію перетворення типу об’єкта типу A в тимчасовий
об’єкт типу char*.
// Якщо s – об’єкт класу, то, коли компілятор зустріне вираз
static_cast<char*>(s),
// він генерує виклик функції       s.operator char*()
// Операнд s є об’єктом, для якого викликається функція operator
char*().
// Перевантажена функція-операція перетворення типу може
визначатися
// для перетворення об’єкта користувацького типу A  в об’єкт
вбудованих типів
// або в об’єкт інших користувацьких типів.
// Прототипи функцій
//                                          A :: operator int()
const;
//                                          A :: operator
OtherClass() const;
// перевантажують операцію перетворення типу:
// у першому випадку – для перетворення користувацького об’єкта у
вбудований тип int;
// у другому випадку – для перетворення користувацького об’єкта в
користувацький тип OtherClass.

// Будь-який конструктор з одним аргументом може розглядатися  як
// конструктор перетворення. Такі конструктори корисні,
// коли ми виконуємо операції над об’єктами String,
// використовуючи аргументи типу char*.
// Конструктор перетворення перетворює  відповідну стрічку
(об’єкт) типу String,
// який потім присвоюється цільовому об'єкту String,
// Наявність такого конструктора означає, що немає необхідності
// створювати перевантажену операцію присвоєння спеціально
```

```
// для присвоєння об'єктам типу String символьних стрічок типу
char*.
// Компілятор автоматично викликає конструктор перетворення
// для створення тимчасового об'єкта String, що містить стрічку
символів.
// Далі викликається перевантажена операція присвоєння,
// яка присвоює тимчасовий об’єкт String іншому об’єкту String.

// Конструктор копіювання
// Конструктор копіювання String( const String &) створює новий
об’єкт класу,
// одночасно ініціалізуючи його значенням вже існуючого об’єкта.
// Тобто цей конструктор створює новий об’єкт
//  і зразу копіює у нього інший об’єкт.
// Аргументом конструктора копіювання завжди є посиланням на
об’єкт класу.

// Перевантажена операція присвоєння operator=
// Коли компілятор зустрічає у програмі string1 = string2,
// він використовує перевантажену операцію opereator=
// викликом функції  string1.operator=(string2).
// Спочатку ця функція виконує перевірку на самоприсвоєння,
// тобто перевіряється, чи посилання на string2 не дорівнює
посиланню на string1.
// Якби перевірки на самоприсвоєння не було передбачено,
// то функція перевантаження зразу ж звільнила б пам’ять,
відведену
//  під string1, і тому була б втрачена існуюча стрічка символів,
// а вказівник більше не вказував би на коректні дані –
// це і був би типовий висячий вказівник.
// Отже, якщо спроба самоприсвоєння має місце, то функція
operator= не виконується
// і виводиться повідомлення про спробу присвоїти об’єкту значення
самого себе.
// Якщо ж самоприсвоєння немає, то далі функція opereator=
послідовно:
// виконує операцію delete[] sPtr  ( sPtr - вказівник на string1),
щоб звільнити пам’ять,
// виділену  під початковий символьний масив string1;
// присвоює змінній length значення right.length;
// викликає допоміжну функцію setString  для створення нової
// копіювання в нього стрічиу символів string2.
// Для цього функція setString  операцією sPtr = new char[length
+1]
// виділяє динамічну пам’ять під нову стрічку string1
// та присвоює вказівнику sPtr значення адреси
// першого байта виділеної під нову стрічку string1 пам’яті.
// Функція operator=  повертає через своє ім’я значення вказівника
*this,
// тобто константне посилання об’єкта на самого себе.
```

```
// Це дозволяє далі реалізовувати каскадні присвоєння стрічок, як
от x=y=z.

// Перевантажена операція присвоєння суми operator+=
// Коли компілятор зустрічає у програмі  операцію конкатенації
стрічок s1 += s2,
// де s1 та s2 –об’єкти класу String,
// він використовує перевантажену операцію opereator+= викликом
функції s1.operator+=(s2).
// Спочатку ця функція обчислює сумарну довжину двох стрічок s1 та
s2
// і зберігає це значення у локальній змінній newLength;
// далі вона створює тимчасовий вказівник tempPtr на новий
символьний масив,
// в якому буде зберігатися сумарна стрічка.
// Далі opereator+= використовує функцію strcpy() для копіювання
символьних стрічок
// sPtr та right.sPtr в область пам’яті, на яку вказує tempPtr.
// Позиція, в яку strcpy()  буде копіювати перший символ  стрічки
right.sPtr,
// визначається обчисленням tempPtr+length за арифметикою
додавання вказівників.
// Це означає, що перший символ right.sPtr має бути розміщений
// у позиції length масиву, на який вказує  tempPtr.
// Далі operator+=  виконує операцію delete[], щоб звільнити
пам’ять,
// виділену  під початковий символьний масив s1,
// присвоює вказівнику sPtr значення вказівника tempPtr,
// а змінній length значення newLength.
// Функція operator+=  повертає через своє ім’я значення
вказівника *this,
// тобто константне посилання об’єкта на самого себе.
// Це дозволяє далі реалізовувати каскадні конкатенацію стрічок.
// Чи потрібна нам перевантажена функція для операції конкатенації
// об’єктів типу char* ? Ні. Конструктор перетворення const char*
у нашій програмі
// перетворює стрічку у тимчасовий об’єкт класу String,
// над яким далі і виконується перевантажена операція
конкатенації.
// Компілятор C++ може виконувати такі операції тільки на глибину
одного рівня.
// C++ може також виконувати неявне, залежне від компілятора
перетворення
//між вбудованими типами перед виконанням перетворення між
основним типом і класом.
// У даній програмі під час створення тимчасового об’єкта класу
String
// викликаються конструктор перетворення та деструктор.
// Це приклад прихованих від клієнта класу непродуктивних втрат
// від викликів функцій  у процесі створення та руйнування
об’єктів класу
```

```
// під час неявних перетворень.
// Аналогічні непродуктивні витрати вносяться конструкторами
копіювання
// під час передачі їм параметрів по значенню
// і під час повернення значень-об’єктів класу.

// Перевантаження операції конкатенації += допоміжною функцією,
// яка отримує один аргумент типу const char*, є більш ефективним,
// ніж визначення єдиної версії операції конкатенації, яка приймає
аргумент класу String.
// Без версії операції += для типу const char* аргумент const
char*
// спочатку перетворювався б в об’єкт String, а потім
// викликалась би операція += для виконання конкатенації.


// Використання неявних перетворень з перевантаженими операціями
// замість перевантаження операторів для різних типів операндів
// часто потребує меншого об’єму коду, що спрощує модифікацію,
// супровід і відлагодження класу.

// Перевантажені операції рівності operator== та нерівності
operator!=
// Коли компілятор зустрічає програмі  вираз string1 == string2 ,
// він генерує виклик функції    string1.operator==( string2).
// Елемент-функція operator== порівнює відповідні елементи обох
стрічок;
// якщо розміри  стрічок однакові, функція повертає значення true,
// в протилежному випадку вона повертає значення false.
// Для виконання порівняння стрічок використовується функція
strcmp().
// Елемент-функція operator!= використовує перевантажену функцію
operator==  ,
// і повертає протилежний результат.
// Використання у перевантажених функціях раніше визначених
// первантажених функцій дозволяє зменшити об’єм коду.

// Перевантажені операції індексації operator[]
// В програмі передбачені дві перевантажені операції індексації
operator[]
// Коли компілятор зустрічає програмі  вираз string1[0]
(неконстантний об’єкт),
// він генерує відповідний виклик функції
string1[.operator[](0).
// Якщо операція індексації застосовується до константного
об’єкта,
// то компілятор генерує виклик константної версії функції
operator[].
// Нехай, наприклад, у програмі оголошено константний об’єкт const
String  z(5).
// Тоді,  для того щоб виконати оператор cout << z[5] << endl;
```

```
// потрібно викликати константну версію функції operator[].
// Кожен варіант функції operator[] перевіряє, чи знаходиться
індекс символьного масиву
// у допустимих межах. Якщо ні, то кожна з функцій друкує
повідомлення про помилку
//  і завершує виконання програми викликом функції exit() з
бібліотеки <cstdlib>.
// Якщо індекс знаходиться у допустимих межах, то неконстантна
версія функції
// повертає відповідний символ стрічки об’єкта класу String  у
формі посилання char&,
// яке може бути використане як значення lvalue,
// яке дозволяє змінити відповідний символ стрічки.
// Якщо індекс знаходиться у допустимих межах, то константна
версія функції
// повертає відповідний символ стрічки об’єкта класу String  у
формі посилання const char&,
// як значення rvalue, яке можна використати для виведення на друк
(змінити не можна).

// Перевантажена операція виклику функції operator()
// Ми перевантажуємо функцію для виділення підстрічки зі заданої
стрічки.
 // Для цього маємо задати два параметра:
// позицію символа, з якого починаємо виділення підстрічки;
// кількість символів, які ми хочемо виділити.
// Нехай об’єкт string1 є об’єктом класу String
// і містить символьну стрічку “QWERTY”.
// Коли компілятор зустрічає у програмі  вираз string1(2, 2) ,
// він генерує виклик функції    string1.operator(2, 2).
// Під час виконання цього виклику створюється тимчасовий об’єкт
String,
// який міститиме стрічку “ER”, і  повертається копія цього
об’єкта.
// Перевантажена функція може містити списки довільної довжини і
складностію
// Це дозволяє розв’язувати різноманітні корисні задачі.
// Наприклад, замість незручної форми запису імен елементів
двомірного масиву
// у вигляді a[m][n] можна використати операцію перевантаження
виклику функції,
// яка б дозволяла доступ до елементів двомірного масиву у формі
a(m,n).

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <cstring>
#include <cstdlib>
```

```
using namespace std;

class String {
    friend ostream& operator<<(ostream&, const String&);
    friend istream& operator>>(istream&, String&);

public:
    String(const char* = "");      // конструктор перетворення (і
за замовчуванням)
    String(const String&);           // конструктор копіювання
    ~String();                                  // деструктор

    const String& operator=(const String&); // операція присвоєння

    const String& operator+=(const String&); // операція
конкатенації

    bool operator!() const; // операція чи це порожня стрічка?
    bool operator==(const String&) const; // операція перевірки чи
s1=s2
    bool operator<(const String&) const; // операція перевірки чи
s1<s2

    // операція перевірки s1 != s2
    bool operator!=(const String& right) const { // операція
перевірки чи s1=s2
        return !(*this == right);
    } // кінець функції operator!=

    // операція перевірки s1 > s2
    bool operator>(const String& right) const { // операція
перевірки чи s1=s2
        return right < *this;
    } // кінець функції operator>

    // операція перевірки s1 <= s2
    bool operator<=(const String& right) const { // операція
перевірки чи s1=s2
        return !(right < *this);
    } // кінець функції operator<=

    // операція перевірки s1 >= s2
    bool operator>=(const String& right) const { // операція
перевірки чи s1=s2
        return !(*this < right);
    } // кінець функції operator>=

    char& operator[](int); // операція індексації (lvalue)
    char operator[](int) const; // операція індексації (rvalue)
    String operator()(int, int = 0) const; // повернути підстрічку
```

```
    int getLength() const; // повернути довжину стрічки

private:
    int length; // довжина стрічки (не рахуючи завершальний нуль)
    char* sPtr; // вказівник на початок представлення стрічки
    void setString(const char*); // допоміжна функція
}; // кінець класу String

// конструктор перетворення (і за замовчуванням) стічки типу
char* в стрічку типу String
String::String(const char* s) : length((s != 0) ? strlen(s) : 0) {
    cout << "\nConversion (and default) constructor: " << s <<
endl;
    setString(s); // викликати допоміжну функцію
} // кінець конструктора перетворення (і конструктора за
замовчуванням одночасно) char* в String

// конструктор копіювання
String::String(const String& copy) : length(copy.length) {
    cout << "Copy constructor: " << copy.sPtr << endl;
    setString(copy.sPtr); // викликати допоміжну функцію
} // кінець конструктора копіювання

// деструктор
String::~String() {
    cout << "Destructor: " << sPtr << endl;
    delete[] sPtr; // знищити стрічку, яка представлена
вказівником
} // кінець деструктора

// перевантажена операція присвоєння = яка запобігає
самоприсвоєнню
const String& String::operator=(const String& right) {
    cout << "operator= called " << endl;
    if (&right != this) {  // запобігти самоприсвоєнню
        delete[] sPtr; // знищити стрічку, яка представлена
вказівником
        length = right.length; // нова довжина стрічки
        setString(right.sPtr); // викликати допоміжну функцію
    } // кінець if
    else
        cout << "Attempted assignement of a String to itself" <<
endl;
    return *this; // дозволяє каскадне присвоєння
} // кінець функції operator=

// приєднати правий операнд до об’єкта і зберегти в об’єкті
const String& String::operator+=(const String& right) {
    size_t newLength = length + right.length;  // нова довжина
    char* tempPtr = new char[newLength + 1]; // виділити пам’ять
    strcpy_s(tempPtr, newLength + 1, sPtr); // копіювати sPtr
```

```
    strcpy_s(tempPtr + length, newLength + 1, right.sPtr); //
копіювати right.sPtr
    delete[] sPtr; // вивільнити пам’ять від старого масиву
    sPtr = tempPtr; // присвоїти sPtr значення нового масиву
    length = newLength; // присвоїти  length значення нової
довжини
    return *this; // дозволяє каскадне присвоєння
} // кінець функції operator+=

// чи порожня дана стрічка?
bool String::operator!() const {
    return length == 0;
} // кінець функції operator!

// чи дорівнює дана стрічка правій стрічці?
bool String::operator==(const String& right) const {
    return strcmp(sPtr, right.sPtr) == 0;
} // кінець функції operator==

// чи менша дана стрічка від правої стрічки?
bool String::operator<(const String& right) const {
    return strcmp(sPtr, right.sPtr) < 0;
} // кінець функції operator<

// переревантаження операції індексації
// Повертає посилання на символ стрічки,
// як величину lvalue, яку можна модифікувати
char& String::operator[](int subscript) {
    // перевірити індекс на вихід за межі діапазону
    if (subscript < 0 || subscript >= length) {
        cerr << "Error: Subscript " << subscript
            << " out of range" << endl;
        exit(1); // завершити програму
    } // кінець if
    return sPtr[subscript]; // величина lvalue, яку можна
модифікувати
} // кінець функції operator[]

// повернути посилання на символ стрічки, як модифіцируємоє rvalue
char String::operator[](int subscript) const {
    // перевірити індекс на вихід за межі діапазону
    if (subscript < 0 || subscript >= length) {
        cerr << "Error: Subscript " << subscript
            << " out of range" << endl;
        exit(1); // завершити програму
    } // кінець if
    return sPtr[subscript]; // модифіцируємоє lvalue
} // кінець функції operator[]

// повернути підстрічку, яка починається з індекса index  і має
довжину subLength
```

```
String String::operator() (int index, int subLength) const {
    // якщо індекс виходить за межі діапазону або довжина стрічки
< 0,
    // повернути порожній обє’кт String
    if (index < 0 || index >= length || subLength < 0)
        return ""; // автоматично перетворюється в порожній об’єкт
String

    // визначити довжину підстрічки
    int len;
    if (subLength == 0 || (index + subLength > length))
        len = length - index;
    else
        len = subLength;

    // виділити тимчасовий масив для підстрічки
    // і завершального нуль-символа
    char* tempPtr = new char[len + 1];

    // копіювати стрічку в масив і додати завершальний нуль
    strncpy_s(tempPtr, len + 1, &sPtr[index], len);
    tempPtr[len] = '\0';

    // створити тимчасовий об’єкт String,  який містить підстрічку
    String tempString(tempPtr);
    delete[] tempPtr; // вилучити тимчасовий масив
    return tempString; // повернути копію тимчасового об’єкта
String
} // кінець функції operator()

// повернути довжину стрічки
int String::getLength() const {
    return length;
} // кінець функції getLength

// допоміжна функція, яка викликається конструкторами і
перевантаженою функцією operator=
void String::setString(const char* string2) {
    sPtr = new char[length + 1]; // виділити память
    if (string2 != 0) // якщо string2 = не NULL, копіювати
        strcpy_s(sPtr, length + 1, string2); // копіювати літерал
в обєкт
    else // якщо string2 = NULL, зробити this порожньою стрічкою
        sPtr[0] = '\0'; // порожня стрічка
} // кінець функції setString

// перевантажена операція виведення
ostream& operator<<(ostream& output, const String& s) {
    output << s.sPtr;
    return output; // дозволяє каскадування
} // кінець функції operator>>
```

```
// перевантажена операція введення
istream& operator>>(istream& input, String& s) {
    char temp[100]; // буфер для зберігання вводу
    input >> setw(100) >> temp;
    s = temp; // використовувати операцію присвоєння класу String
    return input; // дозволяє каскадування
} // кінець функції operator>>

int main() {
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    String s1("happy");
    String s2(" birthday");
    String s3;

    // тестування перевантажених операцій рівності та відношень
стрічок
    cout << "s1 is \"" << s1 << "\"; s2 is \"" << s2
        << "\";  s3 is \"" << s3 << '\"'
        << "\ns2 == s1 yields " << ((s2 == s1) ? "true" : "false")
        << "\ns2 != s1 yields " << ((s2 != s1) ? "true" : "false")
        << "\ns2 > s1 yields " << ((s2 > s1) ? "true" : "false")
        << "\ns2 < s1 yields " << ((s2 < s1) ? "true" : "false")
        << "\ns2 >= s1 yields " << ((s2 >= s1) ? "true" : "false")
        << "\ns2 <= s1 yields " << ((s2 <= s1) ? "true" :
"false");

    // протестувавати перевантажену операцію "порожньо" (!) класу
String
    cout << "\n\nTesting !s3:" << endl;
    if (!s3) {
        cout << "s3 is empty; assigning s1 to s3;" << endl;
        s3 = s1; // протестувати перевантажене присвоєння
        cout << "s3 is \"" << s3 << "\"";
    } // кінець if


    // протестувати перевантажену операцію конкатенацію стрічок
    cout << "\n\ns1 += s2 yields s1= ";
    s1 += s2; // протестувати перевантажену операцію конкатенації
    cout << s1;

    // протестувати конструктор перетворення
    cout << "\n\ns1 += \" to you\" yields" << endl;
    s1 += " to you"; // протестувати конструктор перетворення
```

```
    cout << "s1 = " << s1 << "\n\n";

    // протестувати операцію виклику () для підстрічки
    cout << "The substring of s1 starting at\n"
        << "location 0 for 14 characters, s1(0,14), is:\n"
        << s1(0, 14) << "\n\n";

    // протестувати випадок підстрічки "до кінця стрічки"
    cout << "The substring of s1 starting at\n"
        << "location 15, s1(15), is:\n"
        << s1(15) << "\n\n";

    // протестувати конструктор копіювання
    String* s4Ptr = new String(s1);
    cout << "\n *s4Ptr = " << *s4Ptr << endl;

    // проестувати операцію присвоєння (=) зі самоприсвоєнням
    cout << "Assigning *s4Ptr to *s4Ptr" << endl;
    *s4Ptr = *s4Ptr; // протестувати перевантажене присвоєння
    cout << "*s4Ptr = " << *s4Ptr << endl;

    // тестувати деструктор
    delete s4Ptr;

    // протестувати використання індексації для отримання
    s1[0] = 'H';
    s1[6] = 'B';
    cout << "\ns1 after s1[0] = 'H' and s1[6] = 'B' is:"
        << s1 << "\n\n";

    // протестувати вихід індексу за межі діапазону
    cout << "\nAttempt to assign 'd' to s1[30] yields: " << endl;
    // s1[30] = 'd'; // Помилка: індекс за межами діапазону

    return 0;
}

Conversion (and default) constructor:  happy
Conversion (and default) constructor:   birthday

Conversion (and default) constructor:
s1 is "happy"; s2 is " birthday";  s3 is "" s2
== s1 yields false s2 != s1 yields true s2 >
s1 yields false s2 < s1 yields true s2 >= s1
yields false s2 <= s1 yields true

Testing !s3: s3 is empty; assigning
s1 to s3; operator= called s3 is
"happy"

s1 += s2 yields s1= happy birthday
```

```
s1 += " to you" yields

Conversion (and default) constructor:   to you
Destructor:  to you s1 = happy birthday to you

Conversion (and default) constructor: happy birthday
Copy constructor: happy birthday
Destructor: happy birthday The substring of s1
starting at location 0 for 14 characters,
s1(0,14), is:
happy birthday

Destructor: happy birthday

Conversion (and default) constructor: to you
Copy constructor: to you
Destructor: to you
The substring of s1 starting at location
15, s1(15), is:
to you

Destructor: to you
Copy constructor: happy birthday to you

 *s4Ptr = happy birthday to you
Assigning *s4Ptr to *s4Ptr
operator= called
Attempted assignement of a String to itself
*s4Ptr = happy birthday to you
Destructor: happy birthday to you

s1 after s1[0] = 'H' and s1[6] = 'B' is:Happy Birthday to you

Attempt to assign 'd' to s1[30] yields:
Error: Subscript 30 out of range
```

Завдання 10.2. Завантажити, вивчити та протестувати програму, яка демонструє використання перевантаження операцій інкремента та декремента в префіксній та постфіксній формах для роботи з датами.

```
// lab_work_10_02.cpp Deitel_11_12_11_14

// Програма демонструє використання перевантаження операцій
// інкремента па декремента в префіксній та постфіксній формах
// для роботи з датами.
// З цією метою пропонується клас Date (варіант 3).
// Клас Date забезпечує роботу з датами за рахунок
// використання перевантажених операцій.
```

```
// Клас Date містить такі закриті елементи даних:
// int month - місяць; int day - день місяця;int year -рік;
// масив днів у місяцях - static const int days[13] =
// { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
// сервісна функція для інкремента дати - void helpIncrement().

// Клас Date містить відкриті елементи-функції:
// конструктор за замовчуванням -  Date(int m = 1, int d = 1, int
y = 1900);
//встановити місяць, день, рік - void setDate(int, int, int);
// оператор префіксного інкременту - Date& operator++();
// оператор постфіксного інкременту - Date operator++(int);
// оператор додавання днів до дати - const Date operator+=(int);
// встановити, чи високосний рік? - bool leapYear(int) const;
// встановити, чи це останній день місяця? - bool endOfMonth(int)
const;

// Виведення масиву виконує дружня функція класу
// friend ostream& operator<<(ostream&, const Date&),
// яка перевантажує операцію передачі даних в потік виведення.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <math.h>
#include <windows.h>

using namespace std;

class Date {
    friend ostream& operator<<(ostream&, const Date&);
public:
    Date(int m = 1, int d = 1, int y = 1900); // конструктор за
замовчуванням
    void setDate(int, int, int); //встановити місяць, день, рік
    Date& operator++(); // оператор префіксного інкременту
    Date operator++(int); // оператор постфіксного інкременту
    const Date operator+=(int); // оператор додавання днів до дати
    bool leapYear(int) const; // високосний рік?
    bool endOfMonth(int) const; // останній день місяця?

private:
    int month;
    int day;
    int year;

    static const int days[]; // масив днів у місяцях;
    void helpIncrement();    // сервісна функція для інкремента
дати
}; // кінець класу Date
```

```
// конструктор Date з трьома параметрами
Date::Date(int m, int d, int y) {
    setDate(m, d, y);
} // кінець конструктора Date з трьома параметрами

// встановити місяць, день і рік
void Date::setDate(int mm, int dd, int yy) {
    //  встановити місяць:
    month = (mm >= 1 && mm <= 12) ? mm : 1;
    //  встановити місяць:
    year = (yy >= 1900 && yy <= 2100) ? yy : 1900;
    //  встановити рік:
    if (month == 2 && leapYear(year))
        day = (dd >= 1 && dd <= 29) ? dd : 1; // якщо рік є
високосний
    else
        day = (dd >= 1 && dd <= days[month]) ? dd : 1; // якщо рік
не є високосний
} // кінець функції setDate

// перезавантажена операція префіксного інкремента
Date& Date::operator++() {
    helpIncrement(); // інкрементувати дату
    return *this; // повернення посилання для отримання lvalue
} // кінець функції operator++

// перезавантажена операція постфіксного інкремента
// зауважте, що для цілого параметра - пустишки немає імені
параметра
Date Date::operator++(int) {
    Date temp = *this;
    helpIncrement(); // інкрементувати дату

    // повернути неінкрементований збережений обєкт
    return temp; // повернення значення, а не посилання
} // кінець функції operator++

// додати до дати вказане число днів
const Date Date::operator+=(int additionalDays) {
    for (int i = 0; i < additionalDays; i++)
        helpIncrement(); // інкрементувати дату
    return *this; // дозволяє каскадування
} // кінець функції operator+=

// якщо рік високосний, повернути true
// в протилежному випадку повернути false
bool Date::leapYear(int testYear) const {
    if (testYear % 400 == 0 ||
        (testYear % 400 != 0 && testYear % 4 == 0))
```

```
        return true; // рік високосний
    else
        return false; // рік невисокосний
} // кінець функції leapYear

// визначити, чи є день останнім днем місяця
bool Date::endOfMonth(int testDay) const {
    if (month == 2 && leapYear(year))
        return testDay == 29; // останній день лютого високосного
року
    else
        return testDay == days[month]; // рік невисокосний
} // кінець функції endOfMonth

// допоміжна функція для інкрементування дати
void Date::helpIncrement() {
    // не останній день місяця
    if (!endOfMonth(day))
        day++; // інкрементувати день
    else
        if (month < 12) { // останній день місяця і місяць < 12
            month++; // інкрементувати місяць
            day = 1; // перший день нового місяця
        } // кінець внутрішнього if
        else { // останній день року
            year++; // інкрементувати рік
            month = 1; // перший місяць нового року
            day = 1; // перший день нового місяця
        } // кінець  else
} // кінець функції helpIncrement

const int Date::days[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31,
30, 31, 30, 31 };

// перезавантажена операція виведення
ostream& operator<<(ostream& output, const Date& d) {
    static string monthName[13] = { "", "January", "February",
        "March", "April", "May", "June", "July", "August",
        "September", "October", "November", "December" };
    output << monthName[d.month] << ' ' << d.day << ", " <<
d.year;
    return output; // дозволяє каскадування
} // кінець функції operator<<


int main() {
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
```

```
    SetConsoleOutputCP(1251);

    Date d1; // за замовчуванням 1 січня 1900 року
    Date d2(12, 27, 1992); // 27 грудня 1992 року
    Date d3(0, 99, 8045); // недійсна дата

    cout << "d1 is " << d1 << "\nd2 is " << d2 << "\nd3 is" << d3;
    cout << "\n\n d2 +=7 is " << (d2 += 7);

    d3.setDate(2, 28, 1992);
    cout << "\n\n d3 is " << d3;
    cout << "\n++d3 is " << " (leap year allows 29th) ";

    Date d4(7, 13, 2002);

    cout << "\n\nTesting the pretfix increment operator:\n"
        << "d4 is " << d4 << endl;
    cout << "++d4 is  << ++d4 " << endl;
    cout << " d4 is " << d4;

    cout << "\n\nTesting the postfix increment operator:\n"
        << "d4 is " << d4 << endl;
    cout << "d4++ is  << d4++ " << endl;
    cout << " d4 is " << d4;
    return 0;
}

d1 is January 1, 1900
d2 is December 27, 1992
d3 isJanuary 1, 1900

 d2 +=7 is January 3, 1993

 d3 is February 28, 1992
++d3 is  (leap year allows 29th)

Testing the pretfix increment operator:
d4 is July 13, 2002
++d4 is  << ++d4
 d4 is July 13, 2002

Testing the postfix increment operator:
d4 is July 13, 2002
d4++ is  << d4++
 d4 is July 13, 2002
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Максимальний елемент

Створити шаблон функції `findMax()`, яка знаходить максимальний елемент масиву будь-якого числового типу. У програмі передбачити роботу з масивами типів `int`, `double` та `float`. Вивести початковий масив і знайдений максимальний елемент.

Варіант 2. Мінімальний елемент
Створити шаблон функції `findMin()`, яка визначає мінімальний елемент масиву. Функція повинна працювати з масивами різних типів. Продемонструвати її роботу для масивів `int` та `double`.

Варіант 3. Обмін елементів
Створити шаблон функції `swapValues()`, яка обмінює значення двох змінних. Продемонструвати роботу функції з типами `int`, `double` та `char`. Передбачити введення значень користувачем.

Варіант 4. Сума елементів
Створити шаблон функції `arraySum()`, яка обчислює суму елементів масиву. Функція повинна працювати з масивами типів `int`, `float` та `double`. Результат вивести на екран.

Варіант 5. Середнє значення
Створити шаблон функції `average()`, яка обчислює середнє арифметичне елементів масиву. Передбачити можливість роботи з цілими та дійсними числами. Перевірити роботу функції для двох різних типів даних.

Варіант 6. Пошук елемента
Створити шаблон функції `findElement()`, яка виконує пошук заданого елемента в масиві. Функція повинна повертати індекс знайденого елемента або `-1`, якщо елемент відсутній. Продемонструвати роботу з типами `int`, `double` та `char`.

Варіант 7. Підрахунок елементів
Створити шаблон функції `countElements()`, яка підраховує кількість елементів масиву, що відповідають заданій умові. Наприклад, для чисел — кількість додатних елементів. Передбачити роботу з масивами різних числових типів.

Варіант 8. Обернення масиву
Створити шаблон функції `reverseArray()`, яка змінює порядок елементів масиву на протилежний. Функція повинна працювати з масивами будь-якого типу. Продемонструвати її роботу з `int` та `double`.

Варіант 9. Сортування масиву
Створити шаблон функції `sortArray()`, яка сортує масив за зростанням. Не використовувати готові алгоритми STL. Функція повинна працювати з масивами різних числових типів.

Варіант 10. Порівняння масивів
Створити шаблон функції `compareArrays()`, яка перевіряє, чи є два масиви однаковими за кількістю елементів та значеннями. Передбачити можливість роботи з різними типами даних.

Варіант 11. Шаблонний клас «Число»
Створити шаблонний клас `Number<T>`, який містить одне поле типу `T`. Реалізувати методи для введення, виведення та зміни значення. Додатково реалізувати метод, який повертає квадрат числа. Створити об'єкти класу для типів `int` і `double`.

Варіант 12. Шаблонний клас «Пара»

Створити шаблонний клас `Pair<T1, T2>`, який містить два поля різних типів. Реалізувати конструктор, методи доступу до полів та метод виведення інформації. Створити об'єкти, наприклад `Pair<int, double>` та `Pair<string, int>`.

Варіант 13. Шаблонний клас «Точка»
Створити шаблонний клас `Point<T>`, який описує точку на координатній площині з координатами `x` та `y`. Реалізувати методи для введення координат, їх виведення та обчислення відстані від початку координат. Створити об'єкти для `int` і `double`.

Варіант 14. Шаблонний клас «Прямокутник»
Створити шаблонний клас `Rectangle<T>`, який містить довжину та ширину. Реалізувати методи обчислення площі та периметра. Створити об'єкти класу з параметрами типів `int` і `double`.

Варіант 15. Шаблонний клас «Коло»
Створити шаблонний клас `Circle<T>`, який містить радіус. Реалізувати методи обчислення довжини кола та площі. Передбачити перевірку, щоб радіус був додатним. Продемонструвати роботу з `int` і `double`.

Варіант 16. Шаблонний клас «Студент»
Створити шаблонний клас `Student<T>`, який містить прізвище, ім'я та середній бал типу `T`. Реалізувати конструктор, методи введення та виведення інформації. Створити об'єкти для середнього бала типів `float` і `double`.

Варіант 17. Шаблонний клас «Товар»
Створити шаблонний клас `Product<T1, T2>`, який містить назву товару типу `T1` та ціну типу `T2`. Реалізувати методи введення, виведення та збільшення ціни на заданий відсоток. Створити об'єкт `Product<string, double>`.

Варіант 18. Шаблонний клас «Автомобіль»
Створити шаблонний клас `Car<T1, T2>`, який містить марку автомобіля типу `T1`, рік випуску типу `T2` та ціну типу `double`. Реалізувати методи введення і виведення даних. Створити декілька об'єктів класу.

Варіант 19. Шаблонний клас «Температура»
Створити шаблонний клас `Temperature<T>`, який зберігає температуру. Реалізувати методи переведення температури з градусів Цельсія у Фаренгейти та навпаки. Продемонструвати роботу з типами `int` і `double`.

Варіант 20. Шаблонний клас «Лічильник»
Створити шаблонний клас `Counter<T>`, який містить поточне значення лічильника. Реалізувати методи збільшення та зменшення значення на задану величину, а також отримання поточного значення. Створити об'єкти для `int` і `double`.

Варіант 21. Шаблонний клас «Діапазон»
Створити шаблонний клас `Range<T>`, який містить початок і кінець діапазону. Реалізувати методи перевірки, чи належить задане число цьому діапазону, та обчислення його довжини. Створити об'єкти для `int` і `double`.

Варіант 22. Шаблонний клас «Товарний запис»
Створити шаблонний клас `Item<T1, T2, T3>`, який містить код товару, назву та ціну. Тип кожного поля повинен задаватися параметром шаблону. Реалізувати конструктор та метод виведення інформації. Створити декілька об'єктів з різними комбінаціями типів.

Варіант 23. Шаблонний клас «Результат»
Створити шаблонний клас `Result<T1, T2>`, який містить назву дисципліни типу `T1` та оцінку типу `T2`. Реалізувати методи введення, виведення та перевірки, чи є оцінка позитивною. Створити об'єкти з різними типами оцінок.

Варіант 24. Шаблон функції для пошуку максимуму з двох значень
Створити шаблон функції `maximum()`, яка порівнює два значення та повертає більше з них. Передбачити можливість явного та неявного визначення типу шаблону. Продемонструвати роботу функції для `int`, `double` та `char`.

Варіант 25. Шаблон функції для обчислення добутку
Створити шаблон функції `multiply()`, яка приймає два параметри різних або однакових типів та повертає їх добуток. Передбачити шаблон із двома параметрами типу. Продемонструвати роботу, наприклад, для `int` і `double`.

Варіант 26. Шаблон функції для обміну елементів масиву
Створити шаблон функції `swapElements()`, яка обмінює два елементи масиву за заданими індексами. Передбачити перевірку правильності індексів. Продемонструвати роботу для масивів `int`, `double` та `char`.

Варіант 27. Спеціалізація шаблонної функції
Створити шаблон функції `showType()`, яка виводить значення переданого параметра. Реалізувати окрему спеціалізацію для типу `char`, яка додатково повідомляє, що передано символ. Продемонструвати роботу шаблону для `int`, `double` і `char`.

Варіант 28. Спеціалізація шаблонного класу
Створити шаблонний клас `Data<T>`, який зберігає одне значення та має метод його виведення. Передбачити спеціалізацію класу для типу `char`, у якій додатково виводиться числовий ASCII-код символу. Продемонструвати роботу звичайного шаблону та його спеціалізації.

Варіант 29. Шаблон «Стек»
Створити шаблонний клас `Stack<T>`, який реалізує простий стек фіксованого розміру. Реалізувати операції `push()`, `pop()`, `top()` та перевірку, чи є стек порожнім. Продемонструвати роботу зі стеком типів `int` та `double`.

Варіант 30. Шаблон «Список оцінок»
Створити шаблонний клас `Grades<T>`, який зберігає масив оцінок студентів. Реалізувати методи додавання оцінки, виведення всіх оцінок, пошуку максимальної та мінімальної оцінки й обчислення середнього значення. Створити об'єкти класу для типів `int` і `double`.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №11. ТЕМА «БІБЛІОТЕКА СТАНДАРТНИХ ШАБЛОНІВ STL: КОНТЕЙНЕРИ ТА ІТЕРАТОРИ»

МЕТА РОБОТИ

1. Ознайомитися з контейнерами та ітераторами бібліотеки STL.
2. Ознайомитися з перевантаженими функціями стандарного класу String.
3. З’ясувати, у чому суть неявного перетворення типів.

ХІД РОБОТИ

Завдання 11.1. Завантажити, вивчити та протестувати програму, яка демонструє демонструє стандартний клас string для роботи зі стрічками

```
// lab_work_11_01.cpp Deitel_11_15

// Програма демонструє стандартний клас string
// для роботи зі стрічками.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <string.h>

using std::cout;
using std::endl;
using std::string;


int main() {
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    string s1("happy");
    string s2(" birthday");
    string s3;

    // тестування перевантажених операцій рівності та відношень стрічок
    cout << "s1 is \"" << s1 << "\"; s2 is \"" << s2
        << "\";  s3 is \"" << s3 << '\"'
        << "\ns2 == s1 yields " << ((s2 == s1) ? "true" : "false")
        << "\ns2 != s1 yields " << ((s2 != s1) ? "true" : "false")
        << "\ns2 > s1 yields " << ((s2 > s1) ? "true" : "false")
        << "\ns2 < s1 yields " << ((s2 < s1) ? "true" : "false")
        << "\ns2 >= s1 yields " << ((s2 >= s1) ? "true" : "false")
```

```
    << "\ns2 <= s1 yields " << ((s2 <= s1) ? "true" : "false");

// тест улемент-функції empty класу string
cout << "\n\nTesting s3.empty():" << endl;
if (s3.empty()) {
    cout << "s3 is empty;\nassigning s1 to s3:" << endl;
    s3 = s1; // assign s3 to s1
    cout << "s3 is \"" << s3 << "\"" << endl;
} // кінець if


// протестувати перевантажену операцію конкатенацію стрічок
cout << "\n\ns1 += s2 yields s1= ";
s1 += s2; // протестувати перевантажену операцію конкатенації
cout << s1 << endl;

// протестувати перевантажену операцію конкатенації стрічок
// зі стрічкою в стилі C
cout << "\n\ns1 += \" to you\" yields" << endl;
s1 += " to you"; //
cout << "s1 = " << s1 << "\n\n";

// протестувати операцію виклику () для підстрічки
cout << "The substring of s1 starting at location 0 \n"
    << "for 14 characters, s1.substr(0,14), is:\n"
    << s1.substr(0, 14) << "\n\n";

// протестувати опцію  підстрічки "до кінця стрічки"
cout << "The substring of s1 starting at\n"
    << "location 15, s1.substr(15), is:\n"
    << s1.substr(15) << "\n\n";

// протестувати конструктор копіювання
string* s4Ptr = new string(s1);
cout << "\n *s4Ptr = " << *s4Ptr << endl;

// проестувати операцію присвоєння (=) зі самоприсвоєнням
cout << "Assigning *s4Ptr to *s4Ptr" << endl;
*s4Ptr = *s4Ptr; // протестувати перевантажене присвоєння
cout << "*s4Ptr = " << *s4Ptr << endl;

// тестувати деструктор
delete s4Ptr;

// протестувати використання індексації для отримання
s1[0] = 'H';
s1[6] = 'B';
cout << "\ns1 after s1[0] = 'H' and s1[6] = 'B' is:"
    << s1 << "\n\n";

// протестувати вихід індексу за межі діапазону
```

```
    cout << "\nAttempt to assign 'd' to s1[30] yields: " << endl;
    //s1[30] = 'd'; // Помилка: індекс за межами діапазону

    return 0;
}

s1 is "happy"; s2 is " birthday";  s3 is ""
s2 == s1 yields false
s2 != s1 yields true
s2 > s1 yields false
s2 < s1 yields true
s2 >= s1 yields false
s2 <= s1 yields true

Testing s3.empty():
s3 is empty;
assigning s1 to s3:
s3 is "happy"

s1 += s2 yields s1= happy birthday

s1 += " to you" yields
s1 = happy birthday to you

The substring of s1 starting at location 0
for 14 characters, s1.substr(0,14), is:
happy birthday

The substring of s1 starting at
location 15, s1.substr(15), is:
to you

 *s4Ptr = happy birthday to you
Assigning *s4Ptr to *s4Ptr
*s4Ptr = happy birthday to you

s1 after s1[0] = 'H' and s1[6] = 'B' is:Happy Birthday to you

Attempt to assign 'd' to s1[30] yields:
```

Завдання 11.2. Завантажити, вивчити та протестувати програму, яка демонструє помилкове неявне перетворення типів

```
// lab_work_11_02.cpp Deitel_11_16

// Програма демонструє помилкове неявне перетворення типів.
// З цією метою пропонується клас Array (варіант 1), створений нами раніше.

#include <iostream>
#include <iomanip>
#include <conio.h>
```

```
#include <math.h>
#include <windows.h>

using namespace std;

class Array {
    friend ostream& operator<<(ostream&, const Array&);
    friend istream& operator>>(istream&, Array&);
public:
    Array(int = 10); // конструктор за замовчуванням
    Array(const Array&); // конструктор копіювання
    ~Array(); // деструктор
    int getSize() const; // повертає розмір масиву

    const Array& operator=(const Array&); // операція присвоєння
    bool operator==(const Array&) const; // операція перевірки рівності

    // операція перевірки нерівності; результат, протилежний операції ==
    bool operator!=(const Array& right) const {
        return !(*this == right); // викликає Array::operator==
    } // кінець функції operator!=

    // індексація для неконстанних обєктів повертає lvalue
    int& operator[](int);

    // індексація для константних обєктів повертає rvalue
    int operator[](int) const;

private:
    int size; // розмір масиву, заданого вказівником
    int* ptr; // вказівник на перший елемент масиву
}; // кінець класу Array

// конструктор Array за замовчуванням ( розмір за замовчуванням дорівнює
10)
Array::Array(int arraySize) {
    size = (arraySize > 0 ? arraySize : 10); // перевірити arraySize
    ptr = new int[size]; // виділити простір для масиву

    for (int i = 0; i < size; i++)
        ptr[i] = 0; // встановити елемент масиву-вказівника
} // кінець конструктора Array за замовчуванням

// конструктор копіювання для класу Array
// для попередження нескінченної рекурсії повинен повертати посилання
Array::Array(const Array& arrayToCopy)
    : size(arrayToCopy.size) {
    ptr = new int[size]; // виділити простір для масиву

    for (int i = 0; i < size; i++)
        ptr[i] = arrayToCopy.ptr[i]; //копіювати в обєкт
```

```
} // кінець конструктора копіювання Array

// деструктор для класу Array
Array::~Array() {
    delete[] ptr; // звільнити память під масив
} // кінець деструктора

// повернути число елементів Array
int Array::getSize() const {
    return size; // число елементів Array
} // кінець функції getSize()

// перевантажена операція присвоєння:
// константа, що повертається, запобігає: (a1 = a2) = a2
const Array& Array::operator=(const Array& right) {
    if (&right != this) { // уникати самоприсвоєння
        // для масивів різних розмірів вивільнити початковий (вхідний)
масив
        // у лівій частині, далі виділити новий масив для лівої частини
        if (size != right.size) {
            delete[] ptr; // звільнити память
            size = right.size; // перевстановити розмір цього обєкта
            ptr = new int[size]; // створити простір для копії
        } // кінець внутрішнього if

        for (int i = 0; i < size; i++)
            ptr[i] = right.ptr[i]; //копіювати масив обєкт
    } // кінець зовнішнього if

    return *this; // дозволяє писати, наприклад, x=y=z
} // кінець функції operator=

// визначити, чи рівні два масиви; якщо рівні
// повернути true, в протилежному випадку повернути false
bool Array::operator==(const Array& right) const {
    if (size != right.size) // уникати самоприсвоєння
        return false; // масиви з різним числом елементів

    for (int i = 0; i < size; i++)
        if (ptr[i] != right.ptr[i])
            return false;  // вміст масивів різний

    return true; // масиви рівні
} // кінець функції operator==

// перевантажена операція індексації для неконстантних масивів
// повернення посилання створює модифіковане lvalue
int& Array::operator[] (int subscript) {
    // перевірити індекс на вихід за межі масиву
    if (subscript < 0 || subscript >= size) {
        cerr << "\nError: Subscript " << subscript
```

```
            << " out of range" << endl;
        exit(1); // завершити програму; індекс поза межами
    } // кінець if

    return ptr[subscript]; // повернення посилання
} // кінець функції operator[]

// перевантажена операція індексації для константних масивів
// повернення константного посилання створює модифіковане rvalue
int Array::operator[] (int subscript) const {
    // перевірити індекс на вихід за межі масиву
    if (subscript < 0 || subscript >= size) {
        cerr << "\nError: Subscript " << subscript
            << " out of range" << endl;
        exit(1); // завершити програму; індекс поза межами
    } // кінець if

    return ptr[subscript]; // повертає копію елемента
} // кінець функції operator[]

// перезавантажена операція введення для класу Array:
// вводить значення для всього масиву
istream& operator>>(istream& input, Array& a) {
    for (int i = 0; i < a.size; i++)
        input >> a.ptr[i];
    return input; // дозволяє писати cin >> x >> y;
} // кінець функції operator>>

// перезавантажена операція виведення для класу Array :
// виводить значення для всього масиву
ostream& operator<<(ostream& output, const Array& a) {
    int i;
    // вивести закритий масив, що адресується вказівником
    for (i = 0; i < a.size; i++) {
        output << setw(12) << a.ptr[i];
        if ((i + 1) % 4 == 0) // по 4 числа у стрічці виведення
            output << endl;
    } // кінець оператора for

    if ((i + 1) % 4 != 0) // закінчити останню стрічку виведення
        output << endl;

    return output; // дозволяє писати cout << x << y;
} // кінець функції operator<<

void outputArray(const Array&); //


int main()
{
    // Очищення екрана
```

```
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Array integers1(7); // 7 - елементний Array
    outputArray(integers1);  // вивести масив intrgers1
    outputArray(3);  // перетворити 3 в масив та вивести вміст

    return 0;

}

// друк вмісту масиву
void outputArray(const Array& arrayToOutput) {
    cout << "The Array received has " << arrayToOutput.getSize()
        << " elements. The contents are: \n" << arrayToOutput << endl;
} // кінець функції outputArray

The Array received has 7 elements. The contents are:
           0           0           0           0
           0           0           0
The Array received has 3 elements. The contents are:
           0           0           0
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Оцінки студентів — `vector`
Створити C++-проєкт для роботи з оцінками студента. Використати контейнер `vector<int>` для зберігання оцінок за навчальну дисципліну. Заповнити контейнер не менше ніж 8 оцінками. За допомогою ітераторів визначити **середній бал, максимальну та мінімальну оцінку**, а також кількість оцінок, що дорівнюють 5.

Варіант 2. Температура протягом тижня — `array`
Створити C++-проєкт для аналізу температури протягом 7 днів. Використати `array<double, 7>`. Вивести всі температури за допомогою ітератора. Визначити **середню температуру, максимальну та мінімальну температуру**, а також номер дня з максимальною температурою.

Варіант 3. Список товарів — `list`
Створити C++-проєкт для роботи зі списком товарів магазину. Використати `list<string>`. Додати не менше 8 назв товарів. За допомогою ітераторів вивести всі товари та знайти товар із заданою назвою. Передбачити **додавання нового товару та видалення вибраного товару**.

Варіант 4. Черга замовлень — `deque`

Створити C++-проєкт для моделювання черги замовлень. Використати `deque<int>` для зберігання номерів замовлень. Реалізувати додавання замовлення на початок і кінець черги, видалення замовлення з початку та виведення всіх замовлень за допомогою ітераторів.

Варіант 5. Унікальні номери автомобілів — `set`
Створити C++-проєкт для зберігання номерів автомобілів, які заїхали на стоянку. Використати `set<string>`. Додати декілька номерів, у тому числі повторювані. Вивести всі **унікальні номери у відсортованому порядку**. Реалізувати пошук заданого номера за допомогою ітератора.

Варіант 6. Каталог товарів — `map`
Створити C++-проєкт для роботи з каталогом товарів. Використати `map<string, double>`, де ключем є назва товару, а значенням — його ціна. Додати не менше 7 товарів. Вивести каталог за допомогою ітераторів. Реалізувати **пошук товару та зміну його ціни**.

Варіант 7. Кількість товарів — `unordered_map`
Створити C++-проєкт для обліку кількості товарів на складі. Використати `unordered_map<string, int>`. Ключ — назва товару, значення — кількість. Реалізувати додавання товару, зміну кількості, пошук товару та виведення всіх записів за допомогою ітераторів.

Варіант 8. Унікальні слова — `unordered_set`
Створити C++-проєкт для аналізу набору слів. Використати `unordered_set<string>`. Задати речення або набір слів, серед яких є повторення. Визначити **кількість унікальних слів**, вивести їх та реалізувати перевірку наявності заданого слова.

Варіант 9. Зарплати працівників — `vector`
Створити C++-проєкт для обробки зарплат працівників підприємства. Використати `vector<double>`. Зберегти зарплати не менше ніж 10 працівників. За допомогою ітераторів визначити **середню зарплату, найбільшу та найменшу зарплату**, а також кількість працівників із зарплатою вище середньої.

Варіант 10. Студенти та групи — `map`
Створити C++-проєкт для зберігання інформації про студентів. Використати `map<int, string>`, де ключем є номер студента, а значенням — його прізвище. Реалізувати додавання студентів, пошук студента за номером, видалення запису та виведення всіх студентів за допомогою ітераторів.

Варіант 11. Ціни автомобілів — `vector`
Створити C++-проєкт для аналізу цін на автомобілі. Використати `vector<double>`. Зберегти ціни не менше ніж 8 автомобілів. Визначити **середню ціну, найдорожчий та найдешевший автомобіль**, а також кількість автомобілів, ціна яких перевищує задане значення.

Варіант 12. Список покупок — `list`

Створити C++-проєкт для формування списку покупок. Використати `list<string>`. Передбачити додавання товару на початок і кінець списку, пошук заданого товару, видалення товару та виведення всього списку за допомогою ітераторів.

Варіант 13. Черга клієнтів — `deque`
Створити C++-проєкт для моделювання черги клієнтів у сервісному центрі. Використати `deque<string>`. Реалізувати додавання клієнта в кінець черги, термінове додавання клієнта на початок, обслуговування першого клієнта та виведення поточного стану черги.

Варіант 14. Оцінки групи — `multiset`
Створити C++-проєкт для зберігання оцінок студентів групи. Використати `multiset<int>`, оскільки однакові оцінки можуть повторюватися. Визначити **кількість оцінок кожного типу**, найвищу та найнижчу оцінку. Перегляд контейнера виконати за допомогою ітераторів.

Варіант 15. Телефонний довідник — `map`
Створити C++-проєкт «Телефонний довідник». Використати `map<string, string>`, де ключем є прізвище людини, а значенням — номер телефону. Реалізувати додавання запису, пошук номера за прізвищем, зміну номера та видалення запису.

Варіант 16. Артикули товарів — `set`
Створити C++-проєкт для контролю артикулів товарів. Використати `set<int>`. Додати набір артикулів, включаючи повторювані значення. Вивести унікальні артикули у відсортованому порядку. Реалізувати пошук заданого артикула та визначити кількість унікальних товарів.

Варіант 17. Продажі за місяць — `array`
Створити C++-проєкт для аналізу продажів магазину протягом 12 місяців. Використати `array<double, 12>`. За допомогою ітераторів визначити **загальний обсяг продажів, середній місячний продаж, найкращий та найгірший місяць**.

Варіант 18. Запчастини автомобіля — `unordered_set`
Створити C++-проєкт для формування набору назв автомобільних запчастин. Використати `unordered_set<string>`. Додати не менше 10 назв, серед яких є повтори. Визначити кількість унікальних запчастин та реалізувати перевірку наявності заданої запчастини.

Варіант 19. Склад автозапчастин — `unordered_map`
Створити C++-проєкт для обліку автозапчастин на складі. Використати `unordered_map<string, int>`. Ключ — назва запчастини, значення — кількість на складі. Реалізувати **додавання, зміну кількості, пошук та видалення запчастини**.

Варіант 20. Результати змагань — `vector`

Створити C++-проєкт для обробки результатів спортивних змагань. Використати `vector<int>` для зберігання результатів учасників. За допомогою ітераторів визначити **найкращий та найгірший результат, середнє значення та кількість результатів, кращих за середній**.

Варіант 21. Розклад занять — `map`
Створити C++-проєкт для зберігання розкладу занять. Використати `map<int, string>`, де ключем є номер пари, а значенням — назва дисципліни. Реалізувати додавання заняття, пошук за номером пари, зміну дисципліни та виведення всього розкладу.

Варіант 22. Коди товарів — `set`
Створити C++-проєкт для контролю кодів товарів. Використати `set<int>`. Задати набір кодів, серед яких можуть бути повтори. Вивести унікальні коди у зростаючому порядку. Реалізувати пошук заданого коду та визначити кількість кодів, менших за задане число.

Варіант 23. Історія операцій — `deque`
Створити C++-проєкт для моделювання історії операцій користувача. Використати `deque<string>`. Реалізувати додавання нової операції, видалення найстарішої операції при перевищенні заданого розміру та перегляд історії за допомогою ітераторів.

Варіант 24. Список студентів — `list`
Створити C++-проєкт для роботи зі списком прізвищ студентів. Використати `list<string>`. Реалізувати додавання студентів, пошук за прізвищем, видалення заданого студента та підрахунок кількості студентів, чиє прізвище починається із заданої літери.

Варіант 25. Товари та ціни — `map`
Створити C++-проєкт для роботи з цінами товарів. Використати `map<string, double>`. Вивести всі товари та ціни за допомогою ітераторів. Визначити **найдорожчий товар**, знайти товари дешевші за задану ціну та реалізувати зміну ціни.

Варіант 26. Унікальні міста — `unordered_set`
Створити C++-проєкт для обробки списку міст, у яких побували туристи. Використати `unordered_set<string>`. Додати міста з повтореннями, визначити кількість унікальних міст, вивести їх та перевірити, чи відвідувалося задане місто.

Варіант 27. Витрати за категоріями — `unordered_map`
Створити C++-проєкт для обліку особистих витрат. Використати `unordered_map<string, double>`, де ключем є категорія витрат (`Їжа`, `Транспорт`, `Одяг` тощо), а значенням — сума витрат. Реалізувати додавання витрат до категорії, пошук категорії та визначення найбільшої суми витрат.

Варіант 28. Номери студентських квитків — `set`

Створити C++-проєкт для контролю номерів студентських квитків. Використати `set<int>`. Додати номери квитків, серед яких є повтори. Вивести унікальні номери у відсортованому порядку, знайти заданий номер та визначити кількість номерів у заданому діапазоні.

Варіант 29. Температурні вимірювання — `vector`
Створити C++-проєкт для аналізу результатів температурних вимірювань. Використати `vector<double>`. Зберегти не менше 10 вимірювань. За допомогою ітераторів визначити **середнє значення, мінімальне та максимальне значення**, а також кількість вимірювань, що перевищують середню температуру.

Варіант 30. Каталог автомобілів — `map`
Створити C++-проєкт для формування каталогу автомобілів. Використати `map<string, int>`, де ключем є модель автомобіля, а значенням — рік випуску. Реалізувати додавання автомобіля, пошук за моделлю, зміну року випуску та виведення всього каталогу за допомогою ітераторів.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №12. ТЕМА «БІБЛІОТЕКА СТАНДАРТНИХ ШАБЛОНІВ STL: АЛГОРИТМИ, ПРЕДИКАТИ ТА ЛЯМБДА_ФУНКЦІЇ»

МЕТА РОБОТИ

1. Ознайомитися з бібліотекою алгоритмів стандартної бібліотеки STL.
2. Розробити незалежний клас `ComissionEmployee`, що нараховує заробітну плату працівника, яка обчислюється як комісійний процент від обсягу продаж.
3. Розробити незалежний клас `BasePlusComissionEmployee`, що нараховує заробітну плату працівника, яка обчислюється як сума базової зарплати плюс комісійний процент від обсягу продаж.

ХІД РОБОТИ

Завдання 12.1. Завантажити, вивчити та протестувати програму, яка демонструє незалежний клас CommissionEmployee

```
// lab_work_12_01.cpp Deitel_12_4_12_6

// Пропонується клас CommissionEmployee (варіант 1).
// який нараховує заробітну плату працівника,
// яка обчислюється як комісійний процент від обсягу продаж.
// Клас буде використаний у подальшому як базовий.

// Клас CommissionEmployee містить такі закриті  елементи даних:
// ім'я працівника - string firstName;
// прізвище працівника string lastName;
// номер полісу соціального стахування - string socialSecurityNumber;
// обсяг продаж за тиждень - double grossSales;
// комісійний процент - double commissionRate.

// Клас CommissionEmployee містить такі відкриті елементи-функції:
// конструктор класу - CommissionEmployee(const string&, const string&,
//                                                    const string&,
double = 0.0, double = 0.0);
// встановити імя - void setFirstName(const string&);
// повернути імя - string getFirstName() const;
// встановити прізвище - void setLastName(const string&);
// повернути прізвище - string getLastName() const;
// встановити SSN - void setSocialSecurityNumber(const string&);
// повернути SSN - string getSocialSecurityNumber() const;
// встановити загальну суму продаж - void setGrossSales(double);
// повернути загальну суму продаж - double getGrossSales() const;
// встановити комісійний процент - void setCommissionRate(double);
// повернути комісійний процент - double getCommissionRate() const;
// обчислити заробіток - double earnings() const;
// надрукувати обєкт  CommissionEmploye - void print() const;


#include <iostream>
```

```
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <cstring>
#include <cstdlib>

using namespace std;

class CommissionEmployee {
public:
    CommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0);
    void setFirstName(const string&); // встановити ім'я
    string getFirstName() const; // повернути ім'я

    void setLastName(const string&); // встановити прізвище
    string getLastName() const; // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const; // повернути SSN

    void setGrossSales(double); // встановити загальну суму продаж
    double getGrossSales() const; // повернути загальну суму продаж

    void setCommissionRate(double); // встановити комісійний процент
    double getCommissionRate() const; // повернути комісійний процент

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт  CommissionEmploye

private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; //продажі за тиждень
    double commissionRate; // комісійний процент
}; // кінець класу CommissionEmployee

// конструктор
CommissionEmployee::CommissionEmployee(const string& first, const string&
last,
    const string& ssn, double sales, double rate) {
    firstName = first; // має бути перевірено
    lastName = last;  // має бути перевірено
    socialSecurityNumber = ssn; // має бути перевірено
    setGrossSales(sales); // перевірити і зберегти обєм продаж
    setCommissionRate(rate); // перевірити і зберегти процент
} //  кінець конструктора класу CommissionEmployee

// встановити ім'я
```

```
void CommissionEmployee::setFirstName(const string& first) {
    firstName = first; // має бути перевірено
} //  кінець функції setFirstName

// повернути ім'я
string CommissionEmployee::getFirstName() const {
    return firstName;
} //  кінець функції getFirstName

// встановити прізвище
void CommissionEmployee::setLastName(const string& last) {
    lastName = last; // має бути перевірено
} //  кінець функції setLastName

// повернути прізвище
string CommissionEmployee::getLastName() const {
    return lastName;
} //  кінець функції getLastName

 // встановити номер страховки
void CommissionEmployee::setSocialSecurityNumber(const string& ssn) {
    socialSecurityNumber = ssn; // має бути перевірено
} //  кінець функції setSocialSecurityNumber

// повернути номер страховки
string CommissionEmployee::getSocialSecurityNumber() const {
    return socialSecurityNumber;
} //  кінець функції getSocialSecurityNumber

// встановити загальну суму продаж
void CommissionEmployee::setGrossSales(double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales;
} // кінець функції setGrossSales

// повернутикомісійний процент
double CommissionEmployee::getGrossSales() const {
    return grossSales;
} // кінець функції getGrossSales

// встановити комісійний процент
void CommissionEmployee::setCommissionRate(double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
} // кінець функції setGrossSales

// повернути комісійний процент
double CommissionEmployee::getCommissionRate() const {
    return commissionRate;
} // кінець функції getCommissionRate

// обчислити заробіток
double CommissionEmployee::earnings() const {
```

```
    return commissionRate * grossSales;
} // кінець функції setGrossSales

// надрукувати обєкт CommissionEmployee
void CommissionEmployee::print() const {
    cout << "commission employee: " << firstName << ' ' << lastName
        << "\nsocial security number: " << socialSecurityNumber
        << "\ngross sales: " << grossSales
        << "\ncomission rate: " << commissionRate;
} // кінець функції print

int main()
{
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // створити обєкт CommissionEmployee
    CommissionEmployee employee("Sue", "Jones", "222-22-2222", 10000, .06);

    // встановити формат виведення чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // отримати дані працівника
    cout << "Employee information obtained by get functions: \n"
        << "\nFirst name is " << employee.getFirstName()
        << "\nLast name is " << employee.getLastName()
        << "\nSocial security number is " <<
employee.getSocialSecurityNumber()
        << "\nGross sales is " << employee.getGrossSales()
        << "\nCommission rate is " << employee.getCommissionRate() << endl;

    employee.setGrossSales(8000); // встановити об’єм продаж

    employee.setCommissionRate(0.1); // встановити комісійний процент

    cout << "\nUpdated employee information "
        << "output by print function: \n" << endl;
    employee.print();

    // вивести на друк заробіток працівника
    cout << "\nEmployee's earnings: $ " << employee.earnings() << endl;

    cout << "\nHello World!\n";
    return 0;
}

Employee information obtained by get functions:
```

```
First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

Updated employee information output by print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 8000.00
comission rate: 0.10
Employee's earnings: $ 800.00
```

Завдання 12.2. Завантажити, вивчити та протестувати програму, яка демонструє незалежний клас BasePlusCommissionEmployee

```
// lab_work_12_02.cpp  Deitel_12_7_12_9

// Програма демонструє ще один повністю незалежний клас
// BasePlusCommissionEmployee (варіант 1),
// який нараховує заробітну плату працівника,
// що обчислюється як сума базової зарплати працівника
// плюс комісійний процент від обсягу продаж.
// Клас BasePlusCommissionEmployee є подібним до класу CommissionEmployee:
// він опрацьовує подібні дані (за винятком baseSalary),
// для чого використовує подібні функції (за винятком двох додаткових
функцій).
// У подальшому цей клас буде перероблено у похідний клас.

// Клас BasePlusCommissionEmployee містить такі закриті (private)  елементи
даних:
// ім'я працівника - string firstName;
// прізвище працівника string lastName;
// номер полісу соціального стахування - string socialSecurityNumber;
// обсяг продаж за тиждень - double grossSales;
// комісійний процент - double commissionRate;
// базова зарплата - baseSalary.

// Клас BasePlusCommissionEmployee містить такі відкриті (public) елементи-
функції:
// конструктор класу - BasePlusCommissionEmployee(const string&, const
string&,
//                                  const string&, double = 0.0, double =
0.0, double = 0.0);
// встановити ім’я - void setFirstName(const string&);
// повернути ім’я - string getFirstName() const;
// встановити прізвище - void setLastName(const string&);
// повернути прізвище - string getLastName() const;
// встановити SSN - void setSocialSecurityNumber(const string&);
```

```
// повернути SSN - string getSocialSecurityNumber() const;
// встановити загальну суму продаж - void setGrossSales(double);
// повернути загальну суму продаж - double getGrossSales() const;
// встановити комісійний процент - void setCommissionRate(double);
// повернути комісійний процент - double getCommissionRate() const;
// встановити основну зарплату - setBaseSalary(double);
// повернути основну зарплату - getBaseSalary() const;
// обчислити заробіток - double earnings() const;
// надрукувати об’єкт  CommissionEmploye - void print() const;


#include <iostream>
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <cstring>
#include <cstdlib>

using namespace std;


class BasePlusCommissionEmployee {
public:
    // конструктор похідного класу
    BasePlusCommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0, double = 0.0);

    void setFirstName(const string&); // встановити ім'я
    string getFirstName() const; // повернути ім'я

    void setLastName(const string&); // встановити прізвище
    string getLastName() const; // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const; // повернути SSN

    void setGrossSales(double); // встановити загальну суму продаж
    double getGrossSales() const; // повернути загальну суму продаж

    void setCommissionRate(double); // встановити комісійний процент
    double getCommissionRate() const; // повернути комісійний процент

    void setBaseSalary(double); // встановити основну зарплату
    double getBaseSalary() const; // повернути основну зарплату

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт BasePlusCommissionEmploye

private:
    string firstName;
```

```
    string lastName;
    string socialSecurityNumber;
    double grossSales; //продажі за тиждень
    double commissionRate; // комісійний процент
    double baseSalary; // базова зарплата
}; // кінець визначення класу BasePlusCommissionEmployee

// конструктор класу BasePlusCommissionEmployee
BasePlusCommissionEmployee::BasePlusCommissionEmployee(const string& first,
const string& last,
    const string& ssn, double sales, double rate, double salary) {
    firstName = first; // має провірятися
    lastName = last; // має провірятися
    socialSecurityNumber = ssn; // має провірятися
    setGrossSales(sales); // перевірити і зберегти об'єм продаж
    setCommissionRate(rate); // перевірити і зберегти комісійний процент
    setBaseSalary(salary); // перевірити і зберегти зарплату
} // кінець визначення конструктора класу BasePlusCommissionEmployee

// встановити ім'я
void BasePlusCommissionEmployee::setFirstName(const string& first) {
    firstName = first; // має бути перевірено
} //  кінець функції setFirstName

// повернути ім'я
string BasePlusCommissionEmployee::getFirstName() const {
    return firstName;
} //  кінець функції getFirstName

// встановити прізвище
void BasePlusCommissionEmployee::setLastName(const string& last) {
    lastName = last; // має бути перевірено
} //  кінець функції setLastName

// повернути прізвище
string BasePlusCommissionEmployee::getLastName() const {
    return lastName;
} //  кінець функції getLastName

 // встановити номер страховки
void BasePlusCommissionEmployee::setSocialSecurityNumber(const string& ssn)
{
    socialSecurityNumber = ssn; // має бути перевірено
} //  кінець функції setSocialSecurityNumber

// повернути номер страховки
string BasePlusCommissionEmployee::getSocialSecurityNumber() const {
    return socialSecurityNumber;
} //  кінець функції getSocialSecurityNumber

// встановити загальну суму продаж
```

```
void BasePlusCommissionEmployee::setGrossSales(double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales; // має бути перевірено
} // кінець функції setGrossSales

// повернути загальну суму продаж
double BasePlusCommissionEmployee::getGrossSales() const {
    return grossSales;
} // кінець функції getGrossSales

// встановити комісійний процент
void BasePlusCommissionEmployee::setCommissionRate(double rate) {
    commissionRate = (rate > 0.0 && rate, 1.0) ? rate : 0.0; // має бути
перевірено
} // кінець функції setGrossSales

// повернути комісійний процент
double BasePlusCommissionEmployee::getCommissionRate() const {
    return commissionRate;
} // кінець функції getCommissionRate

// встановити зарплату
void BasePlusCommissionEmployee::setBaseSalary(double salary) {
    baseSalary = (salary < 0.0) ? 0.0 : salary;
} // кінець функції setBaseSalary

// повернути зарплату
double BasePlusCommissionEmployee::getBaseSalary() const {
    return baseSalary;
} // кінець функції getBaseSalary

// обчислити заробіток
double BasePlusCommissionEmployee::earnings() const {
    // може звертатися до захищених даних базового класу
    return baseSalary + (commissionRate * grossSales);
} // кінець функції earnings

// надрукувати обєкт BasePlusCommissionEmployee
void BasePlusCommissionEmployee::print() const {
    cout << "\nbase-salaried commission employee: " << firstName << ' ' <<
lastName
        << "\nsocial security number: " << socialSecurityNumber
        << "\ngross sales: " << grossSales
        << "\ncomission rate: " << commissionRate
        << "\nbase salary: " << baseSalary;;
} // кінець функції print

int main() {
    // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
```

```
    SetConsoleOutputCP(1251);

    // створити обєкт BasePlusCommissionEmployee
    BasePlusCommissionEmployee employee("Bob", "Lewis", "333-33-3333",
5000, .04, 300);

    // встановити формат виведення чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // отримати дані працівника
    cout << "Employee information obtained by get functions: \n"
        << "\nFirst name is " << employee.getFirstName()
        << "\nLast name is " << employee.getLastName()
        << "\nSocial security number is " <<
employee.getSocialSecurityNumber()
        << "\nGross sales is " << employee.getGrossSales()
        << "\nCommission rate is " << employee.getCommissionRate()
        << "\nBase Salary is " << employee.getBaseSalary() << endl;

    employee.setBaseSalary(1000); // встановити базову зарплату

    cout << "\nUpdated employee information "
        << "output by print function: \n" << endl;
    employee.print(); // вивести інформацію про працівника

    // вивести на друк заробіток працівника
    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

    return 0;
}

Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base Salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Результати тестування
Створити C++-проєкт для обробки результатів тестування студентів. Використати `vector<int>`. Заповнити контейнер результатами тестів.
За допомогою алгоритмів STL:
знайти кількість оцінок `5`;
визначити кількість оцінок, більших або рівних `4`, використовуючи `count_if();`
знайти максимальну та мінімальну оцінки;
відсортувати оцінки за зростанням за допомогою `sort();`
використати lambda-функцію для перевірки оцінок.

Варіант 2. Ціни товарів
Створити C++-проєкт для обробки цін товарів. Використати `vector<double>`.
Реалізувати:
пошук першої ціни, більшої за задану, за допомогою find_if();
визначення мінімальної та максимальної ціни;
підрахунок кількості товарів із ціною нижче заданого значення;
сортування цін;
збільшення всіх цін на заданий відсоток за допомогою transform() та lambda-функції.

Варіант 3. Вік працівників
Створити C++-проєкт для аналізу віку працівників підприємства. Використати `vector<int>`.
За допомогою STL:
знайти заданий вік через find();
визначити наймолодшого та найстаршого працівника;
підрахувати кількість працівників віком понад 50 років через count_if();
відсортувати вік;
видалити всі значення менше 18 за допомогою remove_if() та lambda-функції.

Варіант 4. Витрата пального
Створити C++-проєкт для аналізу витрати пального автомобіля під час різних поїздок. Використати `vector<double>`.
Реалізувати:
пошук першої витрати, що перевищує 10 л/100 км;
визначення мінімальної та максимальної витрати;
підрахунок кількості поїздок із витратою нижче 7 л/100 км;
сортування значень;
створення нового контейнера з округленими значеннями за допомогою transform().

Варіант 5. Температура повітря
Створити C++-проєкт для аналізу температури протягом місяця. Використати `vector<double>`.
За допомогою алгоритмів STL:
знайти мінімальну та максимальну температуру;

підрахувати кількість морозних днів через count_if();
знайти перший день із температурою вище 25 °C через find_if();
відсортувати температури;
видалити значення нижче -20 °C за допомогою remove_if().

Варіант 6. Зарплати працівників
Створити C++-проєкт для аналізу зарплат працівників.
Використати `vector<double>` та реалізувати:
пошук заданої зарплати через find();
пошук першої зарплати понад 30 000 грн через find_if();
визначення мінімальної та максимальної зарплати;
підрахунок кількості зарплат, менших за 20 000 грн;
сортування зарплат за зростанням.
Пошукову умову реалізувати через lambda-функцію.

Варіант 7. Номери автомобілів
Створити C++-проєкт для роботи з номерами автомобілів. Використати `vector<string>`.
Реалізувати:
пошук конкретного номера через find();
пошук першого номера, що починається із заданої літери, через find_if();
підрахунок номерів, що відповідають заданій умові;
сортування номерів;
видалення номерів, які відповідають заданій умові, через remove_if().

Варіант 8. Оцінка товару
Створити C++-проєкт для аналізу оцінок товару покупцями. Використати `vector<int>`.
Необхідно:
знайти кількість оцінок 5 через count();
знайти кількість оцінок 4 і 5 через count_if();
визначити найнижчу та найвищу оцінки;
відсортувати оцінки;
видалити всі оцінки, менші за 3, використовуючи remove_if().

Варіант 9. Відстані між населеними пунктами
Створити C++-проєкт для аналізу відстаней між населеними пунктами. Використати `vector<double>`.
Реалізувати:
визначення мінімальної та максимальної відстані;
пошук першої відстані понад 100 км;
підрахунок кількості відстаней менше 50 км;
сортування відстаней;
створення нового набору значень у милях за допомогою transform() та lambda-функції.

Варіант 10. Номери замовлень
Створити C++-проєкт для обробки номерів замовлень магазину. Використати `vector<int>`.
Програма повинна:
знайти заданий номер замовлення через find();
знайти перший номер, більший за задане значення;
підрахувати кількість номерів у заданому діапазоні;
відсортувати номери;
видалити номери, менші за заданий номер.

Варіант 11. Рейтинг спортсменів
Створити C++-проєкт для аналізу результатів спортсменів. Використати `vector<int>`.
Реалізувати:
пошук найкращого та найгіршого результату;
пошук першого результату, що перевищує заданий норматив;
підрахунок кількості результатів, які відповідають нормативу;
сортування результатів;
створення нового контейнера з результатами, збільшеними на 5%, використовуючи `transform()`.

Варіант 12. Довжини деталей
Створити C++-проєкт для контролю довжин деталей на виробництві. Використати `vector<double>`.
Програма повинна:
знайти мінімальну та максимальну довжину;
знайти першу деталь із довжиною понад заданий допуск;
підрахувати кількість деталей, що відповідають заданому діапазону;
відсортувати довжини;
видалити деталі, що не відповідають допустимому діапазону.
Умови перевірки реалізувати lambda-функціями.

Варіант 13. Список оцінок
Створити C++-проєкт для обробки оцінок студентів. Використати `vector<int>`.
Необхідно:
створити копію контейнера за допомогою copy();
відсортувати оригінал;
виконати reverse() для копії;
знайти мінімальну та максимальну оцінки;
використати lambda-функцію для підрахунку кількості позитивних оцінок.

Варіант 14. Вартість автомобілів
Створити C++-проєкт для аналізу вартості автомобілів.
Використати `vector<double>` та:
знайти найдорожчий і найдешевший автомобіль;
знайти першу ціну понад 1 000 000 грн;
підрахувати кількість автомобілів дешевших за 500 000 грн;
відсортувати ціни;

за допомогою transform() та lambda-функції перевести ціни з доларів у гривні.

Варіант 15. Назви міст
Створити C++-проєкт для обробки назв міст. Використати `vector<string>`.
Реалізувати:
пошук конкретного міста через find();
пошук першого міста, назва якого має більше 8 символів, через find_if();
підрахунок міст із назвою заданої довжини;
сортування назв;
розвертання порядку елементів за допомогою reverse().

Варіант 16. Висота будинків
Створити C++-проєкт для аналізу висоти будинків. Використати `vector<double>`.
За допомогою STL:
знайти найвищий та найнижчий будинок;
знайти перший будинок, вищий за 50 м;
підрахувати кількість будинків висотою понад 30 м;
відсортувати висоти;
видалити значення менше 5 м.

Варіант 17. Кількість товарів на складі
Створити C++-проєкт для аналізу кількості товарів на складі. Використати `vector<int>`.
Реалізувати:
пошук заданої кількості через find();
пошук першого товару, кількість якого менша за 5;
підрахунок товарів із нульовим залишком;
сортування залишків;
видалення товарів із кількістю 0 за допомогою remove().

Варіант 18. Результати вимірювань
Створити C++-проєкт для обробки результатів вимірювань. Використати `vector<double>`.
Програма повинна:
визначити мінімальне та максимальне значення;
знайти перше значення, яке виходить за допустимий діапазон;
підрахувати кількість допустимих значень;
відсортувати результати;
створити за допомогою transform() новий контейнер із квадратами виміряних значень.

Варіант 19. Коди деталей
Створити C++-проєкт для контролю кодів деталей. Використати `vector<int>`.
Реалізувати:
пошук заданого коду;
пошук першого коду, більшого за задане значення;
підрахунок повторень заданого коду;

сортування кодів;
видалення всіх кодів, що менші за заданий поріг.

Варіант 20. Час виконання програм
Створити C++-проєкт для аналізу часу виконання декількох програмних операцій. Використати `vector<double>`.
Необхідно:
знайти найменший та найбільший час;
знайти перший час, що перевищує 1 секунду;
підрахувати кількість операцій із часом менше 0.5 секунди;
відсортувати результати;
збільшити всі значення на 10% за допомогою transform().

Варіант 21. Розміри взуття
Створити C++-проєкт для аналізу розмірів взуття покупців. Використати `vector<int>`.
Реалізувати:
пошук заданого розміру;
підрахунок кількості покупців із заданим розміром;
визначення мінімального та максимального розміру;
сортування розмірів;
визначення кількості покупців із розміром понад 44 за допомогою count_if().

Варіант 22. Споживання електроенергії
Створити C++-проєкт для аналізу щомісячного споживання електроенергії. Використати `vector<double>`.
Програма повинна:
знайти мінімальне та максимальне споживання;
знайти перший місяць із показником понад 300 кВт·год;
підрахувати кількість місяців із показником менше 200 кВт·год;
відсортувати значення;
видалити значення, що перевищують заданий ліміт.

Варіант 23. Довжина слів
Створити C++-проєкт для аналізу слів у тексті. Використати `vector<string>`.
Реалізувати:
пошук заданого слова;
пошук першого слова довжиною більше 10 символів;
підрахунок слів довжиною до 5 символів;
сортування слів за алфавітом;
видалення слів, довжина яких менша за 3 символи, через remove_if().

Варіант 24. Вага посилок
Створити C++-проєкт для аналізу ваги посилок. Використати `vector<double>`.
За допомогою алгоритмів STL:
знайти найважчу та найлегшу посилку;
знайти першу посилку важчу за 20 кг;

підрахувати кількість посилок до 5 кг;
відсортувати вагу;
збільшити всі значення на 0.5 кг за допомогою transform().

Варіант 25. Баланс рахунків
Створити C++-проєкт для аналізу залишків на банківських рахунках. Використати `vector<double>`.
Реалізувати:
пошук заданого балансу;
пошук першого від'ємного балансу;
підрахунок кількості рахунків із балансом понад 100 000 грн;
пошук мінімального та максимального балансу;
видалення від'ємних значень за допомогою remove_if().

Варіант 26. Кількість проданих одиниць
Створити C++-проєкт для аналізу продажів товару протягом 10 днів. Використати `vector<int>`.
Необхідно:
визначити мінімальний та максимальний продаж;
знайти перший день із продажем понад 100 одиниць;
підрахувати кількість днів із продажем менше 50 одиниць;
відсортувати дані;
створити новий контейнер із подвоєними значеннями за допомогою transform().

Варіант 27. Рейтинги фільмів
Створити C++-проєкт для обробки рейтингів фільмів. Використати `vector<double>`.
Реалізувати:
пошук заданого рейтингу;
пошук першого фільму з рейтингом понад 8.0;
підрахунок кількості фільмів із рейтингом менше 5.0;
визначення мінімального та максимального рейтингу;
сортування рейтингів;
видалення рейтингів нижче 3.0.

Варіант 28. Вік автомобілів
Створити C++-проєкт для аналізу віку автомобілів на стоянці. Використати `vector<int>`.
Програма повинна:
знайти заданий вік;
знайти найстаріший та найновіший автомобіль;
підрахувати кількість автомобілів віком понад 10 років;
відсортувати вік автомобілів;
видалити автомобілі віком понад 20 років за допомогою remove_if().

Варіант 29. Номери рейсів

Створити C++-проєкт для обробки номерів транспортних рейсів. Використати `vector<int>`.
Реалізувати:
пошук заданого номера рейсу;
пошук першого номера, більшого за задане значення;
підрахунок кількості парних номерів за допомогою count_if() та lambda-функції;
сортування номерів;
перевертання порядку номерів за допомогою reverse().

Варіант 30. Споживання води
Створити C++-проєкт для аналізу щоденного споживання води. Використати `vector<double>`.
Необхідно:
знайти мінімальне та максимальне значення;
знайти перший день із витратою понад заданий ліміт;
підрахувати кількість днів із витратою менше заданого значення;
відсортувати дані;
за допомогою transform() та lambda-функції перевести значення з літрів у кубічні метри;
видалити значення, що перевищують заданий ліміт.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №13. ТЕМА "ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ ТА ПОТОКОВЕ ВВЕДЕННЯ-ВИВЕДЕННЯ

МЕТА РОБОТИ

1. Вивчити методику обробки виняткових ситуацій засобами мови C/C++.
2. Розробити базовий клас зі захищеними даними ComissionEmployee.
3. Розробити похідний клас зі закритими даними BasePlusComissionEmployee.
4. Вивчити особливості наслідування класів, якщо базовий клас опрацьовує захищені дані, а похідний - закриті дані
5. Вивчити особливості наслідування класів, якщо базовий клас та похідний класи опрацьовують закриті дані

ХІД РОБОТИ

Завдання 13.1. Завантажити, вивчити та протестувати програму, яка демонструє реалізацію принципу наслідування класів, якщо базовий клас опрацьовує захищені дані, а похідний - закриті дані

```
// lab_work_13_01.cpp  Deitel_12_12_12_16
//
// Програма демонструє реалізацію принципу наслідування класів,
// якщо базовий клас опрацьовує захищені дані, а похідний - закриті дані.

// З цією метою пропонуються два класи:
// базовий клас CommissionEmployee (варіант 3),
// який нараховує заробітну плату працівника,
// що обчислюється як комісійний процент від обсягу продаж.
// Похідний клас  BasePlusCommissionEmployee (варіант 3),
// який нараховує заробітну плату працівника,
// що обчислюється як сума базової зарплати
// плюс комісійний процент від обсягу продаж.

// Базовий клас CommissionEmployee містить такі захищені (protected)
елементи даних:
// ім'я працівника - string firstName;
// прізвище працівника string lastName;
// номер полісу соціального стахування - string socialSecurityNumber;
// обсяг продаж за тиждень - double grossSales;
// комісійний процент - double commissionRate.

// Базовий клас CommissionEmployee містить такі відкриті елементи-функції:
// конструктор класу - CommissionEmployee(const string&, const string&,
//                          const string&, double = 0.0, double = 0.0);
// встановити імя - void setFirstName(const string&);
// повернути імя - string getFirstName() const;
// встановити прізвище - void setLastName(const string&);
// повернути прізвище - string getLastName() const;
// встановити SSN - void setSocialSecurityNumber(const string&);
// повернути SSN - string getSocialSecurityNumber() const;
```

```
// встановити загальну суму продаж - void setGrossSales(double);
// повернути загальну суму продаж - double getGrossSales() const;
// встановити комісійний процент - void setCommissionRate(double);
// повернути комісійний процент - double getCommissionRate() const;
// обчислити заробіток - double earnings() const;
// надрукувати обєкт  CommissionEmploye - void print() const;

// Похідний клас BasePlusCommissionEmployee містить
// додатково до елемент-даних базового класу такий закритий (private)
елемент даних:
// базова зарплата - baseSalary.

// Похідний клас BasePlusCommissionEmployee містить
// додатково до елемент-функцій базового класу такі відкриті (public)
елементи-функції:
// конструктор класу - BasePlusCommissionEmployee(const string&, const
string&,
//                     const string&, double = 0.0, double = 0.0, double =
0.0);
// встановити основну зарплату - setBaseSalary(double);
// повернути основну зарплату - getBaseSalary() const;
// обчислити заробіток - double earnings() const;
// надрукувати об’єкт  BasePlusCommissionEmploye - void print() const;

// Можна помітити, що використання похідного класу
// робить програму значно коротшою.

// Використання у базовому класі захищених даних
// дає змогу об'єктам похідних класів модифікувати їх безпосередньо,
// не використовуючи для цього set- або  get-функцій.
// Це підвищує продуктивність програм, але й породжує певні проблеми.
// Перша проблема - об'єкт похідного класу може присвоїти
// захищеному елементу базового класу некоректне значення.
// Друга проблема - елемент функції похідного класу
// можуть стати залежними від реалізації базового класу.
// Похідні класи повинні залежати тільки від послуг базового класу,
// тобто від його відкритих елемент функцій, але не від реалізації.
// У випадку захищених даних базового класу під час зміни його
// реалізації може виникнути потреба у модифікації всіх похідних класів.
// Наприклад, якщо ми з певних міркувань змінимо ім'я firstName на
firstname,
// а ім'я lastName  на lastname, то ми повинні також будемо це зробити
скрізь,
// де похідний клас посилається на ці дані.

// Саме тому у базовому класі рекомендується використовувати закриті дані.


#include <iostream>
#include <iomanip>
#include <conio.h>
```

```
#include <cmath>
#include <windows.h>
#include <cstring>
#include <cstdlib>

using namespace std;

class CommissionEmployee {
public:
    CommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0);

    void setFirstName(const string&); // встановити ім'я
    string getFirstName() const; // повернути ім'я

    void setLastName(const string&); // встановити прізвище
    string getLastName() const; // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const; // повернути SSN

    void setGrossSales(double); // встановити загальну суму продаж
    double getGrossSales() const; // повернути загальну суму продаж

    void setCommissionRate(double); // встановити комісійний процент
    double getCommissionRate() const; // повернути комісійний процент

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт  CommissionEmploye

protected:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; //продажі за тиждень
    double commissionRate; // комісійний процент
}; // кінець конструктора базового класу CommissionEmployee

// конструктор базового класу
CommissionEmployee::CommissionEmployee(const string& first, const string&
last,
    const string& ssn, double sales, double rate) {
    firstName = first; // має бути перевірено
    lastName = last;  // має бути перевірено
    socialSecurityNumber = ssn; // має бути перевірено
    setGrossSales(sales); // перевірити і зберегти обєм продаж
    setCommissionRate(rate); // перевірити і зберегти процент
} //  кінець конструктора класу CommissionEmployee

// встановити ім'я
void CommissionEmployee::setFirstName(const string& first) {
```

```
    firstName = first; // має бути перевірено
} //  кінець функції setFirstName

// повернути ім'я
string CommissionEmployee::getFirstName() const {
    return firstName;
} //  кінець функції getFirstName

// встановити прізвище
void CommissionEmployee::setLastName(const string& last) {
    lastName = last; // має бути перевірено
} //  кінець функції setLastName

// повернути прізвище
string CommissionEmployee::getLastName() const {
    return lastName;
} //  кінець функції getLastName

 // встановити номер страховки
void CommissionEmployee::setSocialSecurityNumber(const string& ssn) {
    socialSecurityNumber = ssn; // має бути перевірено
} //  кінець функції setSocialSecurityNumber

// повернути номер страховки
string CommissionEmployee::getSocialSecurityNumber() const {
    return socialSecurityNumber;
} //  кінець функції getSocialSecurityNumber

// встановити загальну суму продаж
void CommissionEmployee::setGrossSales(double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales;
} // кінець функції setGrossSales

// повернутикомісійний процент
double CommissionEmployee::getGrossSales() const {
    return grossSales;
} // кінець функції getGrossSales

// встановити комісійний процент
void CommissionEmployee::setCommissionRate(double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
} // кінець функції setGrossSales

// повернути комісійний процент
double CommissionEmployee::getCommissionRate() const {
    return commissionRate;
} // кінець функції getCommissionRate


// обчислити заробіток
double CommissionEmployee::earnings() const {
```

```
    return commissionRate * grossSales;
} // кінець функції setGrossSales

// надрукувати обєкт CommissionEmployee
void CommissionEmployee::print() const {
    cout << "commission employee: " << firstName << ' ' << lastName
        << "\nsocial security number: " << socialSecurityNumber
        << "\ngross sales: " << grossSales
        << "\ncomission rate: " << commissionRate;
} // кінець функції print

//-
class BasePlusCommissionEmployee : public CommissionEmployee {
public:
    // конструктор похідного класу
    BasePlusCommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0, double = 0.0);

    void setBaseSalary(double); // встановити зарплату
    double getBaseSalary() const; // повернути зарплату

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт BasePlusCommissionEmploye

private:
    double baseSalary; // базова зарплата
}; // кінець визначення похідного класу BasePlusCommissionEmployee

// конструктор похідного класу
BasePlusCommissionEmployee::BasePlusCommissionEmployee(const string& first,
const string& last,
    const string& ssn, double sales, double rate, double salary)
    // явно викликати конструктор базового класу
    : CommissionEmployee(first, last, ssn, sales, rate) {
    setBaseSalary(salary); // перевірити і зберегти зарплату
} // кінець визначення конструктора похідного класу

// встановити зарплату
void BasePlusCommissionEmployee::setBaseSalary(double salary) {
    baseSalary = (salary < 0.0) ? 0.0 : salary;
} // кінець функції setBaseSalary

// повернути зарплату
double BasePlusCommissionEmployee::getBaseSalary() const {
    return baseSalary;
} // кінець функції getBaseSalary

// обчислити заробіток
double BasePlusCommissionEmployee::earnings() const {
    // може звертатися до захищених даних базового класу
    return baseSalary + (commissionRate * grossSales);
```

```
} // кінець функції earnings

// надрукувати обєкт BasePlusCommissionEmployee
void BasePlusCommissionEmployee::print() const {
    cout << "\nbase-salaried commission employee: " << firstName << ' ' <<
lastName
        << "\nsocial security number: " << socialSecurityNumber
        << "\ngross sales: " << grossSales
        << "\ncomission rate: " << commissionRate
        << "\nbase salary: " << baseSalary;;
} // кінець функції print

int main()
{
    // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // створити обєкт BasePlusCommissionEmployee
    BasePlusCommissionEmployee employee("Bob", "Lewis", "333-33-3333",
5000, .04, 300);

    // встановити формат виведення чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // отримати дані працівника
    cout << "Employee information obtained by get functions: \n"
        << "\nFirst name is " << employee.getFirstName()
        << "\nLast name is " << employee.getLastName()
        << "\nSocial security number is " <<
employee.getSocialSecurityNumber()
        << "\nGross sales is " << employee.getGrossSales()
        << "\nCommission rate is " << employee.getCommissionRate()
        << "\nBase Salary is " << employee.getBaseSalary() << endl;

    employee.setBaseSalary(1000); // встановити базову зарплату

    cout << "\nUpdated employee information "
        << "output by print function: \n" << endl;
    employee.print(); // вивести інформацію про працівника

    // вивести на друк заробіток працівника
    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

    return 0;
}

Employee information obtained by get functions:
```

```
First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base Salary is 300.00

Updated employee information output by print function:
base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

Завдання 13.2. Завантажити, вивчити та протестувати програму, яка демонструє реалізацію принципу наслідування класів, якщо базовий клас та похідний класи опрацьовують закриті дані.

```
// lab_work_13_02.cpp  Deitel_12_17_12_21

// Програма демонструє реалізацію принципу наслідування класів,
// якщо базовий клас та похідний класи опрацьовують закриті дані.

// З цією метою пропонуються два класи:
// базовий клас CommissionEmployee (варіант 3),
// який нараховує заробітну плату працівника,
// що обчислюється як комісійний процент від обсягу продаж.
// Похідний клас BasePlusCommissionEmployee (варіант 3),
// який нараховує заробітну плату працівника,
// що обчислюється як сума базової зарплати
// плюс комісійний процент від обсягу продаж.

// Базовий клас CommissionEmployee містить такі закриті (private) елементи
даних:
// ім'я працівника - string firstName;
// прізвище працівника string lastName;
// номер полісу соціального стахування - string socialSecurityNumber;
// обсяг продаж за тиждень - double grossSales;
// комісійний процент - double commissionRate.

// Базовий клас CommissionEmployee містить такі відкриті елементи-функції:
// конструктор класу - CommissionEmployee(const string&, const string&,
//                           const string&, double = 0.0, double = 0.0);
// встановити імя - void setFirstName(const string&);
// повернути імя - string getFirstName() const;
// встановити прізвище - void setLastName(const string&);
// повернути прізвище - string getLastName() const;
// встановити SSN - void setSocialSecurityNumber(const string&);
// повернути SSN - string getSocialSecurityNumber() const;
```

```
// встановити загальну суму продаж - void setGrossSales(double);
// повернути загальну суму продаж - double getGrossSales() const;
// встановити комісійний процент - void setCommissionRate(double);
// повернути комісійний процент - double getCommissionRate() const;
// обчислити заробіток - double earnings() const;
// надрукувати обєкт  CommissionEmploye - void print() const;

// У конструкторі базового класу ми використовуємо ініціалізатори
елементів-даних
// для встановлення значень firstName, lastName та socialSecurityNumber.
// Похідний клас може викликати відповідні незакриті функції базового класу
// для маніпуляції з закритими елементами-даними базового класу.

// Похідний клас BasePlusCommissionEmployee містить
// додатково до елемент-даних базового класу такий закритий (private)
елемент даних:
// базова зарплата - baseSalary.

// Похідний клас BasePlusCommissionEmployee містить
// додатково до елемент-функцій базового класу такі відкриті (public)
елементи-функції:
// конструктор класу - BasePlusCommissionEmployee(const string&, const
string&,
//                     const string&, double = 0.0, double = 0.0, double =
0.0);
// встановити основну зарплату - setBaseSalary(double);
// повернути основну зарплату - getBaseSalary() const;
// обчислити заробіток - double earnings() const;
// надрукувати об’єкт  BasePlusCommissionEmploye - void print() const;

// Реалізація функцій earnings() та print() похідного класу
// дещо змінена у порівнянні з попереднім варіантом.
// Зараз кожна з цих двох функцій для отримання значення зарплати
// викликає відповідну елемент-функцію, а не звертається до baseSalary
напряму.
// Це унеможливлює залежність функцій earnings() та print() від змін
// у реалізації класу.

// Якщо ми вирішимо змінити назви даних базового класу,
// то функції earnings() та print() не зміняться;
// змінити буде потрібно визначення set- та get-функцій,
// які маніпулюють цими даними.
// Варто підкреслити, що зміни вносяться лише в базовому класі,
// похідні класи змінювати не потрібно.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <string>
```

```
#include <cstdlib>

using namespace std;

class CommissionEmployee {
public:
    CommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0);

    void setFirstName(const string&); // встановити ім'я
    string getFirstName() const; // повернути ім'я

    void setLastName(const string&); // встановити прізвище
    string getLastName() const; // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const; // повернути SSN

    void setGrossSales(double); // встановити загальну суму продаж
    double getGrossSales() const; // повернути загальну суму продаж

    void setCommissionRate(double); // встановити комісійний процент
    double getCommissionRate() const; // повернути комісійний процент

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт  CommissionEmploye

private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; //продажі за тиждень
    double commissionRate; // комісійний процент
}; // кінець конструктора базового класу CommissionEmployee

// конструктор базового класу
CommissionEmployee::CommissionEmployee(const string& first, const string&
last,
    const string& ssn, double sales, double rate)
    : firstName(first), lastName(last), socialSecurityNumber(ssn) {
    setGrossSales(sales); // перевірити і зберегти об'єм продаж
    setCommissionRate(rate); // перевірити і зберегти процент
} //  кінець конструктора класу CommissionEmployee

// встановити ім'я
void CommissionEmployee::setFirstName(const string& first) {
    firstName = first; // має бути перевірено
} //  кінець функції setFirstName

// повернути ім'я
string CommissionEmployee::getFirstName() const {
```

```
    return firstName;
} //  кінець функції getFirstName

// встановити прізвище
void CommissionEmployee::setLastName(const string& last) {
    lastName = last; // має бути перевірено
} //  кінець функції setLastName

// повернути прізвище
string CommissionEmployee::getLastName() const {
    return lastName;
} //  кінець функції getLastName

 // встановити номер страховки
void CommissionEmployee::setSocialSecurityNumber(const string& ssn) {
    socialSecurityNumber = ssn; // має бути перевірено
} //  кінець функції setSocialSecurityNumber

// повернути номер страховки
string CommissionEmployee::getSocialSecurityNumber() const {
    return socialSecurityNumber;
} //  кінець функції getSocialSecurityNumber

// встановити загальну суму продаж
void CommissionEmployee::setGrossSales(double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales;
} // кінець функції setGrossSales

// повернути загальну суму продаж
double CommissionEmployee::getGrossSales() const {
    return grossSales;
} // кінець функції getGrossSales

// встановити комісійний процент
void CommissionEmployee::setCommissionRate(double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
} // кінець функції setGrossSales

// повернути комісійний процент
double CommissionEmployee::getCommissionRate() const {
    return commissionRate;
} // кінець функції getCommissionRate

// обчислити заробіток
double CommissionEmployee::earnings() const {
    return commissionRate * grossSales;
} // кінець функції setGrossSales

// надрукувати обєкт CommissionEmployee
void CommissionEmployee::print() const {
```

```
    cout << "commission employee: " << getFirstName() << ' ' <<
getLastName()
        << "\nsocial security number: " << getSocialSecurityNumber()
        << "\ngross sales: " << getGrossSales()
        << "\ncomission rate: " << getCommissionRate();
} // кінець функції print

class BasePlusCommissionEmployee : public CommissionEmployee {
public:
    // конструктор похідного класу
    BasePlusCommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0, double = 0.0);

    void setBaseSalary(double); // встановити зарплату
    double getBaseSalary() const; // повернути зарплату

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт BasePlusCommissionEmploye

private:
    double baseSalary; // базова зарплата
}; // кінець похідного класу BasePlusCommissionEmployee

// конструктор похідного класу
BasePlusCommissionEmployee::BasePlusCommissionEmployee(const string& first,
const string& last,
    const string& ssn, double sales, double rate, double salary)
    // явно викликати конструктор базового класу
    : CommissionEmployee(first, last, ssn, sales, rate) {
    setBaseSalary(salary); // перевірити і зберегти зарплату
} // кінець визначення конструктора похідного класу

// встановити основну зарплату
void BasePlusCommissionEmployee::setBaseSalary(double salary) {
    baseSalary = (salary < 0.0) ? 0.0 : salary;
} // кінець функції setBaseSalary

// повернути зарплату
double BasePlusCommissionEmployee::getBaseSalary() const {
    return baseSalary;
} // кінець функції getBaseSalary

// обчислити заробіток
double BasePlusCommissionEmployee::earnings() const {
    // може звертатися до захищених даних базового класу
    return getBaseSalary() + CommissionEmployee::earnings();
} // кінець функції earnings

// надрукувати обєкт BasePlusCommissionEmployee
void BasePlusCommissionEmployee::print() const {
    cout << "\nbase-salaried ";
```

```
    // викликати функцію print класу CommissionEmployee
    CommissionEmployee::print();
    cout << "\nbase salary: " << getBaseSalary();
} // кінець функції print

int main() {
    // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // створити обєкт BasePlusCommissionEmployee
    BasePlusCommissionEmployee employee("Bob", "Lewis", "333-33-3333",
5000, .04, 300);

    // встановити формат виведення чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // отримати дані працівника
    cout << "Employee information obtained by get functions: \n"
        << "\nFirst name is " << employee.getFirstName()
        << "\nLast name is " << employee.getLastName()
        << "\nSocial security number is " <<
employee.getSocialSecurityNumber()
        << "\nGross sales is " << employee.getGrossSales()
        << "\nCommission rate is " << employee.getCommissionRate()
        << "\nBase Salary is " << employee.getBaseSalary() << endl;

    employee.setBaseSalary(1000); // встановити базову зарплату

    cout << "\nUpdated employee information "
        << "output by print function: \n" << endl;
    employee.print(); // вивести інформацію про працівника

    // вивести на друк заробіток працівника
    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

    return 0;
}

Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base Salary is 300.00

Updated employee information output by print function:
```

```
base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Варіант 1. Банківський рахунок
Створити клас **BankAccount**, який містить номер рахунку, ім'я власника та баланс. Реалізувати методи поповнення та зняття коштів. При спробі зняти суму, більшу за баланс, генерувати виняток **InsufficientFundsException**. Передбачити окрему обробку від'ємної суми та недостатньої кількості коштів. Реалізувати потокове введення та виведення об'єкта.

Варіант 2. Товар
Створити клас **Product**, який містить код товару, назву, ціну та кількість. Реалізувати введення та виведення об'єкта за допомогою потоків. Перевіряти, щоб ціна та кількість були невід'ємними. При некоректній ціні генерувати власний виняток **InvalidPriceException**, а при некоректній кількості — інший виняток.

Варіант 3. Студент
Створити клас **Student**, який містить прізвище, ім'я, групу та середній бал. Реалізувати перевірку середнього бала в діапазоні від 0 до 100. Якщо введено некоректний бал, генерувати виняток. Для порожнього прізвища передбачити інший тип винятку. Реалізувати `cin`/`cout` для об'єктів класу.

Варіант 4. Автомобіль
Створити клас **Car**, який містить марку, модель, рік випуску та об'єм двигуна. Перевіряти коректність року випуску та об'єму двигуна. Передбачити власні винятки **InvalidYearException** та **InvalidEngineException**. Реалізувати потокове введення і виведення об'єкта.

Варіант 5. Книга
Створити клас **Book**, який містить назву, автора, кількість сторінок та ціну. Кількість сторінок і ціна повинні бути додатними. У разі помилки генерувати відповідні винятки. Передбачити обробку декількох типів винятків та введення/виведення об'єкта через потоки.

Варіант 6. Працівник
Створити клас **Employee**, який містить прізвище, посаду, стаж роботи та заробітну плату. Заробітна плата і стаж не можуть бути від'ємними. Створити власний клас винятку **InvalidSalaryException**. Реалізувати потокове введення та виведення інформації про працівника.

Варіант 7. Телефонний номер

Створити клас **Phone**, який містить ім'я абонента та номер телефону. Реалізувати перевірку номера на відповідність заданому формату. При неправильному номері генерувати власний виняток **InvalidPhoneException**. Передбачити окрему обробку порожнього імені абонента.

Варіант 8. Температура
Створити клас **Temperature**, який містить значення температури та одиницю вимірювання. Реалізувати перетворення між градусами Цельсія та Фаренгейта. Якщо введено некоректне значення або невідому одиницю вимірювання, генерувати виняток. Реалізувати потокове введення та виведення.

Варіант 9. Прямокутник
Створити клас **Rectangle**, який містить довжину та ширину. Реалізувати методи обчислення площі та периметра. Сторони повинні бути додатними. При некоректних значеннях генерувати власний виняток **InvalidDimensionException**. Реалізувати введення та виведення об'єкта через `cin` та `cout`.

Варіант 10. Коло
Створити клас **Circle**, який містить радіус. Реалізувати методи обчислення площі та довжини кола. Радіус повинен бути додатним. Передбачити два типи винятків: некоректний радіус та помилка введення. Реалізувати потокове введення і виведення.

Варіант 11. Авіаквиток
Створити клас **AirTicket**, який містить номер рейсу, пункт призначення, пасажира та ціну квитка. Перевіряти, щоб ціна була додатною, а номер рейсу — коректним. Створити власний клас винятку для неправильної ціни. Реалізувати потокове введення та виведення об'єкта.

Варіант 12. Замовлення
Створити клас **Order**, який містить номер замовлення, назву товару, кількість та ціну одиниці товару. Реалізувати метод обчислення загальної вартості. Якщо кількість або ціна некоректні, генерувати винятки різних типів. Реалізувати `operator>>` та `operator<<`.

Варіант 13. Вклад
Створити клас **Deposit**, який містить ім'я вкладника, суму вкладу та процентну ставку. Реалізувати метод обчислення прибутку. Сума та ставка повинні бути невід'ємними. При некоректній процентній ставці генерувати власний виняток **InvalidRateException**.

Варіант 14. Фільм
Створити клас **Film**, який містить назву, режисера, рік випуску та рейтинг. Рейтинг повинен знаходитися в діапазоні від 0 до 10. При виході рейтингу за межі діапазону генерувати виняток. Для некоректного року передбачити інший тип винятку. Реалізувати потокове введення-виведення.

Варіант 15. Комп'ютер

Створити клас **Computer**, який містить модель, обсяг оперативної пам'яті, обсяг накопичувача та ціну. Пам'ять і ціна повинні бути додатними. Створити власні винятки для некоректної пам'яті та ціни. Реалізувати `operator>>` і `operator<<`.

Варіант 16. Товар на складі
Створити клас **WarehouseItem**, який містить код товару, назву, кількість на складі та ціну. Реалізувати метод зміни кількості товару. Не дозволяти встановлювати від'ємну кількість. Передбачити виняток **NegativeQuantityException** та інший виняток для некоректної ціни.

Варіант 17. Рахунок за електроенергію
Створити клас **ElectricityBill**, який містить номер особового рахунку, ім'я споживача, попередній та поточний показники лічильника і тариф. Реалізувати обчислення вартості електроенергії. Якщо поточний показник менший за попередній, генерувати виняток. Також обробляти некоректний тариф.

Варіант 18. Посилка
Створити клас **Parcel**, який містить номер посилки, адресу отримувача, вагу та вартість доставки. Вага повинна бути додатною. Реалізувати метод розрахунку вартості доставки залежно від ваги. Передбачити власний клас **InvalidWeightException**.

Варіант 19. Лікарняний пацієнт
Створити клас **Patient**, який містить прізвище, ім'я, вік та номер палати. Перевіряти допустимість віку та номера палати. При некоректному віці або номері палати генерувати різні винятки. Реалізувати потокове введення та виведення.

Варіант 20. Результат іспиту
Створити клас **ExamResult**, який містить прізвище студента, назву предмета та оцінку. Оцінка повинна знаходитися в діапазоні від 0 до 100. Передбачити власний клас винятку **InvalidGradeException**. Реалізувати введення та виведення об'єкта потоками.

Варіант 21. Кредит
Створити клас **Credit**, який містить ім'я клієнта, суму кредиту, процентну ставку та строк кредитування. Реалізувати метод обчислення загальної суми виплат. Не дозволяти від'ємну суму, нульовий строк або некоректну ставку. Обробити кожен тип помилки окремо.

Варіант 22. Паркінг
Створити клас **ParkingPlace**, який містить номер місця, номер автомобіля та кількість годин паркування. Реалізувати метод обчислення вартості паркування. Кількість годин повинна бути додатною. При некоректному номері місця генерувати власний виняток.

Варіант 23. Спортивний результат
Створити клас **SportResult**, який містить прізвище спортсмена, назву дисципліни та результат. Реалізувати перевірку результату та метод визначення його відповідності

заданому нормативу. При некоректному результаті генерувати виняток. Реалізувати потокове введення-виведення.

Варіант 24. Готельний номер
Створити клас **HotelRoom**, який містить номер кімнати, тип номера, кількість днів проживання та ціну за добу. Реалізувати метод обчислення вартості проживання. Кількість днів та ціна повинні бути додатними. Передбачити декілька типів винятків.

Варіант 25. Інтернет-тариф
Створити клас **InternetTariff**, який містить назву тарифу, швидкість інтернету, абонентську плату та обсяг доступного трафіку. Перевіряти коректність числових параметрів. Створити власний клас винятку для некоректної швидкості. Реалізувати `operator>>` та `operator<<`.

Варіант 26. Заряд акумулятора
Створити клас **Battery**, який містить модель акумулятора, ємність та рівень заряду. Рівень заряду повинен бути в межах від 0 до 100%. Реалізувати метод заряджання та розряджання акумулятора. При виході рівня заряду за допустимі межі генерувати власний виняток.

Варіант 27. Рейс автобуса
Створити клас **BusRoute**, який містить номер маршруту, початкову та кінцеву зупинки, довжину маршруту та кількість пасажирів. Кількість пасажирів не може бути від'ємною або перевищувати місткість автобуса. Передбачити власний виняток **CapacityException**.

Варіант 28. Вимірювальний прилад
Створити клас **Measurement**, який містить назву приладу, виміряне значення та одиницю вимірювання. Реалізувати перевірку допустимого діапазону вимірювання. Якщо значення виходить за межі діапазону, генерувати власний виняток. Реалізувати потокове введення та виведення.

Варіант 29. Курс валют
Створити клас **CurrencyRate**, який містить назву валюти, код валюти та курс до гривні. Реалізувати метод конвертації заданої суми. Курс повинен бути додатним. Передбачити винятки для некоректного курсу та від'ємної суми.

Варіант 30. Інтернет-магазин
Створити клас **OnlineProduct**, який містить назву товару, артикул, ціну та кількість на складі. Реалізувати метод оформлення покупки заданої кількості товару. Якщо кількість покупки є некоректною або перевищує залишок на складі, генерувати різні винятки. Реалізувати потокове введення та виведення об'єкта.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №14. ТЕМА «ФАЙЛИ ТА UML-МОДЕЛЮВАННЯ ООП-СИСТЕМ»

МЕТА РОБОТИ

1. Вивчити методику створення текстових та бінарних файлів засобами мови C/C++.
2. Ознайомитися з технологією створення UML-діаграм програмних проєктів.
3. Вивчити послідовність викликів констукторів та деструкторів базового класу та похідного класу.
4. Вивчити виклик функцій базового класу із об'єктів похідного класу

ХІД РОБОТИ

Завдання 14.1. Завантажити, вивчити та протестувати програму, яка демонструє послідовність виклику конструкторів та деструкторів базового та похідних класів, які опрацьовують закриті дані

```
// lab_work_14_01.cpp Deitel_12_22_12_26

// Програма демонструє послідовність виклику конструкторів та деструкторів
// базового та похідних класів, які опрацьовують закриті дані.

// З цією метою використовуються два класи:
// базовий клас CommissionEmployee (варіант 4),
// який нараховує заробітну плату працівника,
// що обчислюється як комісійний процент від обсягу продаж.
// Похідний клас BasePlusCommissionEmployee (варіант 4),
// який нараховує заробітну плату працівника,
// що обчислюється як сума базової зарплати
// плюс комісійний процент від обсягу продаж.

// Базовий клас CommissionEmployee містить такі закриті (private) елементи
даних:
// ім'я працівника - string firstName;
// прізвище працівника string lastName;
// номер полісу соціального стахування - string socialSecurityNumber;
// обсяг продаж за тиждень - double grossSales;
// комісійний процент - double commissionRate.

// Базовий клас CommissionEmployee містить такі відкриті елементи-функції:
// конструктор класу - CommissionEmployee(const string&, const string&,
//                          const string&, double = 0.0, double = 0.0);
// деструктор класу - ~CommissionEmployee();
// встановити імя - void setFirstName(const string&);
// повернути імя - string getFirstName() const;
// встановити прізвище - void setLastName(const string&);
// повернути прізвище - string getLastName() const;
// встановити SSN - void setSocialSecurityNumber(const string&);
// повернути SSN - string getSocialSecurityNumber() const;
// встановити загальну суму продаж - void setGrossSales(double);
// повернути загальну суму продаж - double getGrossSales() const;
```

```
// встановити комісійний процент - void setCommissionRate(double);
// повернути комісійний процент - double getCommissionRate() const;
// обчислити заробіток - double earnings() const;
// надрукувати обєкт CommissionEmploye - void print() const;

// У конструкторі базового класу ми використовуємо ініціалізатори
елементів-даних
// для встановлення значень firstName, lastName та socialSecurityNumber.
// Похідний клас може викликати відповідні незакриті функції базового класу
// для маніпуляції з закритими елементами-даними базового класу.
// У цій програмі конструктор та деструктор як базового, так і похідного
класів
// виводять на екран повідомлення, яке свідчить про їх виклик,
// та викликають функцію print(), яка виводить на екран інформацію про
об'єкт класу.

// Похідний клас BasePlusCommissionEmployee містить
// додатково до елемент-даних базового класу такий закритий (private)
елемент даних:
// базова зарплата - baseSalary.

// Похідний клас BasePlusCommissionEmployee містить
// додатково до елемент-функцій базового класу такі відкриті (public)
елементи-функції:
// конструктор класу - BasePlusCommissionEmployee(const string&, const
string&,
//                     const string&, double = 0.0, double = 0.0, double =
0.0);
// деструктор класу - ~BasePlusCommissionEmployee();
// встановити основну зарплату - setBaseSalary(double);
// повернути основну зарплату - getBaseSalary() const;
// обчислити заробіток - double earnings() const;
// надрукувати об’єкт  BasePlusCommissionEmploye - void print() const;

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <string>
#include <cstdlib>

using namespace std;

class CommissionEmployee {
public:
    CommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0);
    ~CommissionEmployee(); // деструктор

    void setFirstName(const string&); // встановити імя
```

```
    string getFirstName() const; // повернути імя

    void setLastName(const string&); // встановити прізвище
    string getLastName() const; // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const; // повернути SSN

    void setGrossSales(double); // встановити загальну суму продаж
    double getGrossSales() const; // повернути загальну суму продаж

    void setCommissionRate(double); // встановити комісійний процент
    double getCommissionRate() const; // повернути комісійний процент

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт  CommissionEmploye

private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; //продажі за тиждень
    double commissionRate; // комісійний процент
}; // кінець конструктора базового класу CommissionEmployee

// конструктор базового класу
CommissionEmployee::CommissionEmployee(const string& first, const string&
last,
    const string& ssn, double sales, double rate)
    : firstName(first), lastName(last), socialSecurityNumber(ssn) {
    setGrossSales(sales); // перевірити і зберегти об'єм продаж
    setCommissionRate(rate); // перевірити і зберегти процент

    cout << "\nCommission employee constructor: " << endl << endl;
    print();
    cout << "\n\n";
} //  кінець конструктора класу CommissionEmployee


// деструктор базового класу
CommissionEmployee::~CommissionEmployee() {
    cout << "\nCommission employee destructor: " << endl << endl;
    print();
} //  кінець десструктора класу CommissionEmployee

// встановити ім'я
void CommissionEmployee::setFirstName(const string& first) {
    firstName = first; // має бути перевірено
} //  кінець функції setFirstName

// повернути ім'я
```

```
string CommissionEmployee::getFirstName() const {
    return firstName;
} //  кінець функції getFirstName

// встановити прізвище
void CommissionEmployee::setLastName(const string& last) {
    lastName = last; // має бути перевірено
} //  кінець функції setLastName

// повернути прізвище
string CommissionEmployee::getLastName() const {
    return lastName;
} //  кінець функції getLastName

 // встановити номер страховки
void CommissionEmployee::setSocialSecurityNumber(const string& ssn) {
    socialSecurityNumber = ssn; // має бути перевірено
} //  кінець функції setSocialSecurityNumber

// повернути номер страховки
string CommissionEmployee::getSocialSecurityNumber() const {
    return socialSecurityNumber;
} //  кінець функції getSocialSecurityNumber

// встановити загальну суму продаж
void CommissionEmployee::setGrossSales(double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales; // має бути перевірено
} // кінець функції setGrossSales

// повернути загальну суму продаж
double CommissionEmployee::getGrossSales() const {
    return grossSales;
} // кінець функції getGrossSales

// встановити комісійний процент
void CommissionEmployee::setCommissionRate(double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0; // має бути
перевірено
} // кінець функції setGrossSales

// повернути комісійний процент
double CommissionEmployee::getCommissionRate() const {
    return commissionRate;
} // кінець функції getCommissionRate

// обчислити заробіток
double CommissionEmployee::earnings() const {
    return getCommissionRate() * getGrossSales();
} // кінець функції earnings

// надрукувати обєкт CommissionEmployee
```

```
void CommissionEmployee::print() const {
    cout << "commission employee: " << getFirstName() << ' ' <<
getLastName()
        << "\nsocial security number: " << getSocialSecurityNumber()
        << "\ngross sales: " << getGrossSales()
        << "\ncomission rate: " << getCommissionRate();
} // кінець функції print

//-------------------------------------------------------------------------
----------------

class BasePlusCommissionEmployee : public CommissionEmployee {
public:
    // конструктор похідного класу
    BasePlusCommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0, double = 0.0);
    ~BasePlusCommissionEmployee(); // деструктор похідного класу

    void setBaseSalary(double); // встановити зарплату
    double getBaseSalary() const; // повернути зарплату

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт BasePlusCommissionEmploye

private:
    double baseSalary; // базова зарплата
}; // кінець похідного класу BasePlusCommissionEmployee

// конструктор похідного класу
BasePlusCommissionEmployee::BasePlusCommissionEmployee(const string& first,
const string& last,
    const string& ssn, double sales, double rate, double salary)
    // явно викликати конструктор базового класу
    : CommissionEmployee(first, last, ssn, sales, rate) {
    setBaseSalary(salary); // перевірити і зберегти зарплату

    cout << "\nBasePlusCommissionEmployee constuctor:" << endl;
    print();
    cout << "\n\n";
} // кінець визначення конструктора похідного класу


// деструктор похідного класу
BasePlusCommissionEmployee::~BasePlusCommissionEmployee() {
    cout << "\nBasePlusCommissionEmployee destuctor:" << endl;
    print();
    cout << "\n\n";
}

// встановити основну зарплату
void BasePlusCommissionEmployee::setBaseSalary(double salary) {
```

```
    baseSalary = (salary < 0.0) ? 0.0 : salary;
} // кінець функції setBaseSalary

// повернути зарплату
double BasePlusCommissionEmployee::getBaseSalary() const {
    return baseSalary;
} // кінець функції getBaseSalary

// обчислити заробіток
double BasePlusCommissionEmployee::earnings() const {
    // може звертатися до захищених даних базового класу
    return getBaseSalary() + CommissionEmployee::earnings();
} // кінець функції earnings

// надрукувати обєкт BasePlusCommissionEmployee
void BasePlusCommissionEmployee::print() const {
    cout << "\nbase-salaried ";
    // викликати функцію print класу CommissionEmployee
    CommissionEmployee::print();
    cout << "\nbase salary: " << getBaseSalary();
} // кінець функції print

int main()
{
    // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // встановити формат виведення чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // створити об'єкт BasePlusCommissionEmployee
    BasePlusCommissionEmployee employee1("Bob", "Lewis", "333-33-3333",
5000, .04, 300);

    CommissionEmployee employee2("Lisa", "Jones", "555-55-5555", 2000,
.06);
    cout << endl;

    BasePlusCommissionEmployee employee3("Mark", "Sands", "888-88-8888",
8000, .15, 2000);
    cout << endl;

    return 0;
}

Commission employee constructor:

commission employee: Bob Lewis
```

```
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04

BasePlusCommissionEmployee constuctor:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 300.00

Commission employee constructor:

commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
comission rate: 0.06

Commission employee constructor:

commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
comission rate: 0.15

BasePlusCommissionEmployee constuctor:

base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
comission rate: 0.15
base salary: 2000.00

BasePlusCommissionEmployee destuctor:

base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
comission rate: 0.15
base salary: 2000.00

Commission employee destructor:

commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
comission rate: 0.15
Commission employee destructor:

commission employee: Lisa Jones
```

```
social security number: 555-55-5555
gross sales: 2000.00
comission rate: 0.06
BasePlusCommissionEmployee destuctor:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 300.00

Commission employee destructor:

commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
```

Завдання 14.2. Завантажити, вивчити та протестувати програму, яка демонструє виклик функцій базового класу із об'єктів похідного класу

```
// lab_work_14_02.cpp Deitel_13_1_13_5

// Програма демонструє виклик функцій базового класу CommissionEmployee
(варіант 5)
// із об'єктів похідного класу BasePlusCommissionEmployee (варіант 5).
// Програма є підготовчою до демонстрування принципу поліморфізму у класах.

// Базовий клас CommissionEmployee описує працівника,
// який отримує комісійний відсоток зі суми укладених ним угод (продаж).
// Клас BasePlusCommissionEmployee є похідним від класу CommissionEmployee,
// і він описує працівника, який отримує базову фіксовану зарплату плюс
// комісійний відсоток зі суми укладених ним угод (продаж).
// Кожен об'єкт класу BasePlusCommissionEmployee є одночасно об'єктом класу
// CommissionEmployee, який, однак, отримує додаткову фіксовану зарплату.
// Елемент-функція earnings похідного класу BasePlusCommissionEmployee
перевизначає
// функцію earnings базового класу CommissionEmployee, включаючи в
розрахунок
// базову фіксовану зарплату з об'єкту BasePlusCommissionEmployee.
// Елемент-функція print класу BasePlusCommissionEmployee перевизначає
// функцію print класу CommissionEmployee, виводячи на друк ту ж
інформацію,
// що й функція з класу CommissionEmployee, а також значення базової
зарплати.

//-------------------------------------------------------
// Коментар до головної функція main() програми.
```

```
// 1. Створюються об'єкт commissionEmployee("Sue", "Jones", "222-22-2222",
10000, .06)
//    базового класу CommissionEmployee.
// 2. Створюються вказівник commissionEmployeePtr на об'єкт базового класу
CommissionEmployee.
// 3. Створюються об'єкт basePlusCommissionEmploye("Bob", "Lewis", "333-33-
3333", 5000, .04, 300)
//    похідного класу  BasePlusCommissionEmployee, а стрічки
// 4. Створюютья вказівник *basePlusCommissionEmployeePtr  на об'єкт
похідного класу BasePlusCommissionEmployee.

// 5а. Використовується ім'я об'єкта базового класу commissionEmployee
//     для виклику функції  print базового класу.
// 5б. Використовується ім'я об'єкта похідного класу
basePlusCommissionEmployeeyee
//     для виклику функції похідного класу print похідного класу.

// 6а. Присвоюється адреса об'єкта базового класу commissionEmployee
//     вказівнику базового класу commissionEmployeePtr,
//     який використовується для активації функції print базового класу.
// 6б. Присвоюється адреса об'єкта похідного класу
basePlusCommissionEmployee
//     вказівнику похідного класу basePlusCommissionEmployeePtr,
//     який використовується для активації функції print похідного класу.

// 7. Присвоюється адреса об'єкта похідного класу
basePlusCommissionEmployee
//    вказівнику на об'єкт базового класу commissionEmployeePtr,
//    який використовується для активації функції print базового класу.
//    Компілятор C++ допускає таке "схрещування", оскільки
//    об'єкт похідного класу є одночасно і об'єктом базового класу.
//    Хоча вказівник commissionEmployeePtr на об'єкт базового класу
commissionEmployee
//    посилається на об'єкт похідного класу basePlusCommissionEmployee,
//    викликається функція print базового класу.
//    Клас, функція якого активується, визначається типом дескриптора
//    (тобто типом вказівника чи посилання),
//    який використовується під час виклику,
//    а не типом об'єкта, на який дескриптор посилається.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <cstring>
#include <cstdlib>

using namespace std;

// оголошення базового класу CommissionEmployee
```

```
class CommissionEmployee {
public:
    CommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0);

    void setFirstName(const string&); // встановити імя
    string getFirstName() const; // повернути імя

    void setLastName(const string&); // встановити прізвище
    string getLastName() const; // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const; // повернути SSN

    void setGrossSales(double); // встановити загальну суму продаж
    double getGrossSales() const; // повернути загальну суму продаж

    void setCommissionRate(double); // встановити комісійний процент
    double getCommissionRate() const; // повернути комісійний процент

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт  CommissionEmploye

private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; //продажі за тиждень
    double commissionRate; // комісійний процент
}; // кінець оголошення базового класу CommissionEmployee

// оголошення похідного класу BasePlusCommissionEmployee
class BasePlusCommissionEmployee : public CommissionEmployee {
public:
    BasePlusCommissionEmployee(const string&, const string&, const string&,
        double = 0.0, double = 0.0, double = 0.0); // конструктор похідного
класу

    void setBaseSalary(double); // встановити основну зарплату
    double getBaseSalary() const; // повернути основну зарплату

    double earnings() const; // обчислити заробіток
    void print() const; // надрукувати обєкт  BasePlusCommissionEmploye

private:
    double baseSalary; // основна зарплата
}; // кінець оголошення похідного класу BasePlusCommissionEmployee

// конструктор базового класу CommissionEmployee
CommissionEmployee::CommissionEmployee(const string& first, const string&
last,
```

```
    const string& ssn, double sales, double rate)
    : firstName(first), lastName(last), socialSecurityNumber(ssn) {
    setGrossSales(sales); // перевірити і зберегти обєм продаж
    setCommissionRate(rate); // перевірити і зберегти процент
} //  кінець конструктора базового класу CommissionEmployee

// визначення елемент-функцій базового класу CommissionEmployee

// встановити імя
void CommissionEmployee::setFirstName(const string& first) {
    firstName = first; // має бути перевірено
} //  кінець функції setFirstName

// повернути імя
string CommissionEmployee::getFirstName() const {
    return firstName;
} //  кінець функції getFirstName

// встановити прізвище
void CommissionEmployee::setLastName(const string& last) {
    lastName = last; // має бути перевірено
} //  кінець функції setLastName

// повернути прізвище
string CommissionEmployee::getLastName() const {
    return lastName;
} //  кінець функції getLastName

// встановити номер страховки
void CommissionEmployee::setSocialSecurityNumber(const string& ssn) {
    socialSecurityNumber = ssn; // має бути перевірено
} //  кінець функції setSocialSecurityNumber

// повернути номер страховки
string CommissionEmployee::getSocialSecurityNumber() const {
    return socialSecurityNumber;
} //  кінець функції getSocialSecurityNumber

// встановити загальну суму продаж
void CommissionEmployee::setGrossSales(double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales;
} // кінець функції setGrossSales

// повернути загальну суму продаж
double CommissionEmployee::getGrossSales() const {
    return grossSales;
} // кінець функції getGrossSales

// встановити комісійний процент
void CommissionEmployee::setCommissionRate(double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
```

```
} // кінець функції setGrossSales

// повернути комісійний процент
double CommissionEmployee::getCommissionRate() const {
    return commissionRate;
} // кінець функції getCommissionRate

// обчислити заробіток
double CommissionEmployee::earnings() const {
    return getCommissionRate() * getGrossSales();
} // кінець функції earnings

// надрукувати обєкт CommissionEmployee
void CommissionEmployee::print() const {
    cout << "commission employee: " << getFirstName() << ' ' <<
getLastName()
        << "\nsocial security number: " << getSocialSecurityNumber()
        << "\ngross sales: " << getGrossSales()
        << "\ncomission rate: " << getCommissionRate();
} // кінець функції print

// визначення елемент-функцій похідного класу BasePlusCommissionEmployee:

// конструктор похідного класу
BasePlusCommissionEmployee::BasePlusCommissionEmployee(const string& first,
const string& last,
    const string& ssn, double sales, double rate, double salary)
    // явно викликати конструктор базового класу
    : CommissionEmployee(first, last, ssn, sales, rate) {
    setBaseSalary(salary); // перевірити і зберегти зарплату
} // кінець конструктора похідного класу
// встановити основну зарплату
void BasePlusCommissionEmployee::setBaseSalary(double salary) {
    baseSalary = (salary < 0.0) ? 0.0 : salary;
} // кінець функції setBaseSalary

// повернути основну  зарплату
double BasePlusCommissionEmployee::getBaseSalary() const {
    return baseSalary;
} // кінець функції getBaseSalary

// обчислити заробіток
double BasePlusCommissionEmployee::earnings() const {
    // може звертатися до захищених даних базового класу
    return getBaseSalary() + CommissionEmployee::earnings();
} // кінець функції earnings

// надрукувати обєкт BasePlusCommissionEmployee
void BasePlusCommissionEmployee::print() const {
    cout << "\nbase-salaried ";
    // викликати функцію print класу CommissionEmployee
```

```
    CommissionEmployee::print();
    cout << "\nbase salary: " << getBaseSalary();
} // кінець функції print класу BasePlusCommissionEmployee


int main()
{
    // Очищення екрана
    system("cls");
    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // створити об'єкт commissionEmployee
    CommissionEmployee commissionEmployee("Sue", "Jones", "222-22-2222",
10000, .06);

    // створити вказівник commissionEmployeePtr на об'єкт базового класу
    CommissionEmployee* commissionEmployeePtr = 0;

    //  створити об'єкт basePlusCommissionEmployee похідного класу
    BasePlusCommissionEmployee basePlusCommissionEmployee(
        "Bob", "Lewis", "333-33-3333", 5000, .04, 300);

    //  створити вказівник basePlusCommissionEmployeePtr на об'єк похідного
класу
    BasePlusCommissionEmployee* basePlusCommissionEmployeePtr = 0;

    // встановити формат виведення чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // вивести дані про працівників commissionEmployee та
basePlusCommissionEmployee
    cout << "Print base-class and derived class objects: \n\n";
    commissionEmployee.print(); // виклик функції print базового класу
    cout << "\n\n";
    basePlusCommissionEmployee.print(); // виклик функції print похідного
класу

    // встановити базовий вказівник на базовий об'єкт
    commissionEmployeePtr = &commissionEmployee;
    cout << "\n\n\nCalling print with base-class pointer to \n"
        << "base-class object invokes base-class print function: \n\n";
    commissionEmployeePtr->print(); // виклик функції print базового класу

    // встановити похідний вказівник на похідний об'єкт
    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee;
    cout << "\n\n\nCalling print with derived-class pointer to \n"
        << "derived-class object invokes derived-class print function:
\n\n";
```

```
    basePlusCommissionEmployeePtr->print(); // виклик функції print
похідного класу

    // встановити базовий вказівник на похідний об'єкт
    commissionEmployeePtr = &basePlusCommissionEmployee;
    cout << "\n\n\nCalling print with base-class pointer to \n"
        << "derived-class object invokes base-class print function "
        << "on that  derived-class object:\n\n";
    commissionEmployeePtr->print(); // виклик функції print базового класу
    cout << endl;

    return 0;
}

Print base-class and derived class objects:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
comission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 300.00

Calling print with base-class pointer to
base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
comission rate: 0.06

Calling print with derived-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 300.00

Calling print with base-class pointer to
derived-class object invokes base-class print function on that  derived-
class object:

commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
```

```
comission rate: 0.04
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Вимоги до виконання всіх варіантів завдань:

1. Реалізувати проєкт мовою C++.
2. Використати принципи об'єктно-орієнтованого програмування.
3. Створити не менше двох взаємопов'язаних класів.
4. Реалізувати запис даних у файл та читання даних із файлу.
5. Передбачити перевірку успішності відкриття файлу.
6. Реалізувати окремі функції або методи для роботи з файлами.
7. Після запуску програми забезпечити можливість завантаження раніше збережених даних.
8. Для варіантів з успадкуванням використати базовий і похідні класи та продемонструвати поліморфізм.
9. Побудувати **UML-діаграму класів**, на якій показати поля, методи, асоціації, агрегації, композиції та успадкування, якщо вони використовуються.
10. Передбачити зрозуміле текстове меню для роботи користувача з програмою.

Варіант 1. Бібліотека

Створити ООП-систему **«Бібліотека»**. Розробити класи `Book`, `Author` та `Library`. Реалізувати зв'язок між книгою та автором. Передбачити додавання книг, перегляд списку та пошук книги за назвою. Зберігати дані бібліотеки у текстовому файлі та відновлювати їх під час запуску програми. Побудувати UML-діаграму класів.

Варіант 2. Студенти та групи

Створити систему обліку студентів. Розробити класи `Student`, `Group` та `University`. Студенти повинні належати до певної групи. Передбачити додавання студентів, перегляд групи та пошук студента за прізвищем. Зберігати дані у файлі. Побудувати UML-діаграму з відображенням зв'язків між класами.

Варіант 3. Автосалон

Створити систему **«Автосалон»** з класами `Car`, `Engine` та `CarDealer`. Автомобіль повинен містити інформацію про двигун. Передбачити додавання автомобілів, перегляд каталогу та пошук автомобіля за маркою. Зберігати каталог у бінарному файлі та відновлювати його під час запуску. Побудувати UML-діаграму.

Варіант 4. Інтернет-магазин

Створити систему **«Інтернет-магазин»** з класами `Product`, `Customer` та `Order`. Замовлення повинно містити інформацію про покупця та товари. Реалізувати створення замовлення, перегляд замовлень та обчислення їх загальної вартості. Зберігати замовлення у текстовому файлі. Побудувати UML-діаграму класів.

Варіант 5. Склад товарів

Створити систему **«Склад»**. Розробити класи `Product`, `Category` та `Warehouse`. Товар повинен належати до певної категорії. Реалізувати додавання товарів, зміну кількості та пошук товару за назвою. Дані зберігати у бінарному файлі. Побудувати UML-діаграму.

Варіант 6. Кінотеатр

Створити систему **«Кінотеатр»** з класами `Movie`, `Hall`, `Ticket` та `Cinema`. Реалізувати продаж квитків на фільми. Передбачити перегляд доступних фільмів і кількості вільних місць. Зберігати інформацію про фільми та продані квитки у текстовому файлі. Побудувати UML-діаграму.

Варіант 7. Готель

Створити систему **«Готель»** з класами `Room`, `Guest`, `Booking` та `Hotel`. Реалізувати бронювання номерів та скасування бронювання. Передбачити пошук вільних номерів. Дані про бронювання зберігати у файлі та відновлювати при запуску програми. Побудувати UML-діаграму.

Варіант 8. Автосервіс

Створити систему **«Автосервіс»** з класами `Car`, `Customer`, `Service` та `RepairOrder`. Замовлення на ремонт повинно містити автомобіль, клієнта та перелік виконаних робіт. Реалізувати розрахунок вартості ремонту. Зберігати замовлення у текстовому файлі. Побудувати UML-діаграму.

Варіант 9. Поліклініка

Створити систему **«Поліклініка»** з класами `Patient`, `Doctor`, `Appointment` та `Clinic`. Реалізувати запис пацієнта до лікаря. Передбачити пошук пацієнтів та перегляд запланованих прийомів. Дані зберігати у файлі. Побудувати UML-діаграму класів.

Варіант 10. Фітнес-клуб

Створити систему **«Фітнес-клуб»** з класами `Client`, `Trainer`, `Subscription` та `FitnessClub`. Клієнт може мати абонемент та бути закріпленим за тренером. Реалізувати реєстрацію клієнтів і перегляд активних абонементів. Зберігати дані у бінарному файлі. Побудувати UML-діаграму.

Варіант 11. Університет

Створити систему **«Університет»** з класами `Student`, `Teacher`, `Course` та `University`. Реалізувати зв'язки між студентами, викладачами та дисциплінами. Передбачити перегляд студентів певної дисципліни. Зберігати інформацію у текстовому файлі. Побудувати UML-діаграму з асоціаціями між класами.

Варіант 12. Музичний каталог

Створити систему **«Музичний каталог»** з класами `Song`, `Artist`, `Album` та `MusicLibrary`. Реалізувати додавання композицій, пошук за виконавцем та перегляд композицій альбому. Зберігати каталог у файлі. Побудувати UML-діаграму.

Варіант 13. Туристична агенція

Створити систему **«Туристична агенція»** з класами `Tour`, `Client`, `Booking` та `TravelAgency`. Реалізувати бронювання турів клієнтами. Передбачити пошук турів за країною та ціною. Дані зберігати у текстовому файлі. Побудувати UML-діаграму.

Варіант 14. Ресторан

Створити систему **«Ресторан»** з класами `Dish`, `Menu`, `Order` та `Customer`. Замовлення повинно містити декілька страв. Реалізувати створення замовлення та обчислення його вартості. Зберігати замовлення у бінарному файлі. Побудувати UML-діаграму.

Варіант 15. Зоопарк

Створити систему **«Зоопарк»**. Розробити базовий клас `Animal` та похідні класи `Mammal`, `Bird`, `Reptile`. Створити клас `Zoo`, який містить колекцію тварин. Передбачити поліморфний метод виведення інформації про тварину. Зберігати основні дані у файлі. Побудувати UML-діаграму з відображенням успадкування.

Варіант 16. Транспортна система

Створити систему **«Транспорт»**. Розробити базовий клас `Vehicle` та похідні класи `Car`, `Bus`, `Truck`. Створити клас `TransportCompany`, який зберігає транспортні

засоби. Реалізувати поліморфний метод розрахунку витрат пального. Зберігати дані у бінарному файлі. Побудувати UML-діаграму.

Варіант 17. Магазин електроніки

Створити систему **«Магазин електроніки»** з класами `Device`, `Customer`, `Purchase` та `Store`. Передбачити різні типи пристроїв: смартфони, ноутбуки та планшети. Використати успадкування та поліморфізм. Дані про товари зберігати у файлі. Побудувати UML-діаграму.

Варіант 18. Система доставки

Створити систему **«Доставка»** з класами `Parcel`, `Customer`, `Courier` та `DeliveryService`. Реалізувати оформлення та відстеження посилок. Передбачити зміну статусу доставки. Зберігати інформацію про посилки у текстовому файлі. Побудувати UML-діаграму.

Варіант 19. Банківська система

Створити спрощену систему **«Банк»** з класами `Account`, `Customer`, `Transaction` та `Bank`. Реалізувати поповнення рахунку, зняття коштів і перегляд операцій. Зберігати інформацію про рахунки у файлі. Побудувати UML-діаграму класів.

Варіант 20. Система оренди автомобілів

Створити систему **«Оренда автомобілів»** з класами `Car`, `Customer`, `Rental` та `RentalCompany`. Реалізувати видачу автомобіля в оренду, повернення та розрахунок вартості. Зберігати інформацію про оренди у бінарному файлі. Побудувати UML-діаграму.

Варіант 21. Електронна бібліотека

Створити систему **«Електронна бібліотека»** з базовим класом `Publication` та похідними класами `Book`, `Magazine`, `Newspaper`. Створити клас `Library`, який містить колекцію публікацій. Реалізувати поліморфне виведення інформації та пошук за назвою. Зберігати дані у текстовому файлі. Побудувати UML-діаграму.

Варіант 22. Система обліку працівників

Створити систему **«Працівники підприємства»** з базовим класом `Employee` та похідними класами `Manager`, `Developer`, `Worker`. Створити клас `Company`. Реалізувати поліморфний метод розрахунку заробітної плати. Зберігати дані працівників у бінарному файлі. Побудувати UML-діаграму.

Варіант 23. Система продажу квитків

Створити систему продажу квитків на заходи. Розробити базовий клас `Event` та похідні класи `Concert`, `Theater`, `SportEvent`. Створити класи `Ticket` та `TicketOffice`. Реалізувати продаж квитків і розрахунок вартості. Зберігати дані у файлі. Побудувати UML-діаграму.

Варіант 24. Комп'ютерний магазин

Створити систему **«Комп'ютерний магазин»** з класами `Computer`, `Component`, `Customer`, `Order`. Реалізувати формування замовлення та розрахунок його вартості. Передбачити збереження замовлень у текстовому файлі та їх відновлення. Побудувати UML-діаграму.

Варіант 25. Система управління проєктами

Створити систему **«Проєкти підприємства»** з класами `Employee`, `Project`, `Task` та `Company`. Працівники повинні бути пов'язані із проєктами та завданнями. Реалізувати додавання завдань і перегляд завдань конкретного працівника. Зберігати дані у файлі. Побудувати UML-діаграму.

Варіант 26. Система обліку нерухомості

Створити систему **«Нерухомість»**. Розробити базовий клас `Property` та похідні класи `Apartment`, `House`, `Office`. Створити клас `RealEstateAgency`. Реалізувати пошук об'єктів за ціною та типом. Зберігати інформацію у бінарному файлі. Побудувати UML-діаграму з успадкуванням.

Варіант 27. Система обліку автомобільних запчастин

Створити систему **«Автозапчастини»** з класами `Part`, `Category`, `Supplier` та `Warehouse`. Реалізувати додавання запчастин, пошук за артикулом та зміну залишку. Зберігати дані складу у файлі. Побудувати UML-діаграму з відображенням агрегації або композиції.

Варіант 28. Система навчальних курсів

Створити систему **«Онлайн-курси»** з класами `Student`, `Course`, `Lesson` та `Platform`. Студент може записуватися на курси, а курс складається з декількох занять. Реалізувати перегляд курсів та занять. Зберігати дані у текстовому файлі. Побудувати UML-діаграму.

Варіант 29. Система управління автопарком

Створити систему **«Автопарк підприємства»** з базовим класом `Vehicle` та похідними класами `PassengerCar`, `Bus`, `Truck`. Створити клас `Driver` та клас `Fleet`. Реалізувати призначення водіїв транспортним засобам і перегляд інформації про автопарк. Зберігати дані у бінарному файлі. Побудувати UML-діаграму.

Варіант 30. Система замовлення автотоварів

Створити систему **«Магазин автотоварів»** з класами `Product`, `Customer`, `Order`, `Category` та `Store`. Реалізувати формування замовлення з декількох товарів, обчислення загальної вартості та пошук товарів за категорією. Зберігати товари та замовлення у файлах і відновлювати їх під час запуску програми. Побудувати UML-діаграму класів із відображенням асоціацій, агрегації або композиції.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

ЛАБОРАТОРНА РОБОТА №15. ТЕМА «БАГАТОФАЙЛОВІ ПРОГРАМИ ТА СТРУКТУРА ООП-ПРОЄКТУ

МЕТА РОБОТИ

1. Ознайомитися з методикою створення багатофайлових програм.
2. Вивчити виклик віртуальних функцій похідного класу за допомогою вказівників базового класу на об'єкти похідного класу.
3. Вивчити взаємодію абстрактного базового класу з конкретними похідними класами.

ХІД РОБОТИ

Завдання 15.1. Завантажити, вивчити та протестувати програму, яка демонструє виклик віртуальних функцій похідного класу за допомогою вказівників базового класу на об'єкти похідного класу

```
// lab_work_15_01.cpp Deitel_13_8_13_10

// Програма демонструє виклик віртуальних функцій похідного класу
// BasePlusCommissionEmployee (варіант 4) за допомогою вказівників
// базового класу CommissionEmployee (варіант 4) на об'єкти похідного
класу.
// Цим самим демонструється принцип поліморфізму у класах.

// Базовий клас CommissionEmployee описує працівника,
// який отримує комісійний відсоток зі суми укладених ним угод (продаж).
// Клас BasePlusCommissionEmployee є похідним від класу CommissionEmployee,
// і він описує працівника, який отримує базову фіксовану зарплату плюс
// комісійний відсоток зі суми укладених ним угод (продаж).

// У нашій програмі є дві функції, оголошені як віртуальні:
// обчислити заробіток - virtual double earnings() const;
// надрукувати обєкт - virtual void print() const;
// Вони виконують подібну роботу в обох класах -
// перша обчислює заробіток, друга роздруковує інформацію про об'єкт.
// Але оскільки самі об'єкти базового і похідного класів відрізняються,
// то й віртуальні функції працюватимуть по різному з відповідними
об'єктами.
// Віртуальні функції похідного класу викликаються за допомогою
// вказівників базового класу на об'єкти похідного класу.

#include <iostream>
#include <iomanip>
#include <conio.h>
#include <cmath>
#include <windows.h>
#include <cstring>
#include <cstdlib>

using namespace std;
```

```
class CommissionEmployee {
public:
    CommissionEmployee(const string&, const string&,
        const string&, double = 0.0, double = 0.0);

    void setFirstName(const string&); // встановити ім'я
    string getFirstName() const; // повернути ім'я

    void setLastName(const string&); // встановити прізвище
    string getLastName() const; // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const; // повернути SSN

    void setGrossSales(double); // встановити загальну суму продаж
    double getGrossSales() const; // повернути загальну суму продаж

    void setCommissionRate(double); // встановити комісійний процент
    double getCommissionRate() const; // повернути комісійний процент

    virtual double earnings() const; // обчислити заробіток
    virtual void print() const; // надрукувати обєкт  CommissionEmploye

private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; //продажі за тиждень
    double commissionRate; // комісійний процент
}; // кінець базового класу CommissionEmployee

class BasePlusCommissionEmployee : public CommissionEmployee {
public:
    BasePlusCommissionEmployee(const string&, const string&, const string&,
        double = 0.0, double = 0.0, double = 0.0); // конструктор похідного
класу

    void setBaseSalary(double); // встановити основну зарплату
    double getBaseSalary() const; // повернути основну зарплату

    virtual double earnings() const; // обчислити заробіток
    virtual void print() const; // надрукувати обєкт
BasePlusCommissionEmploye

private:
    double baseSalary; // основна зарплата
}; // кінець похідного класу BasePlusCommissionEmployee

// конструктор базового класу
CommissionEmployee::CommissionEmployee(const string& first, const string&
last,
```

```
    const string& ssn, double sales, double rate)
    : firstName(first), lastName(last), socialSecurityNumber(ssn) {
    setGrossSales(sales); // перевірити і зберегти обєм продаж
    setCommissionRate(rate); // перевірити і зберегти процент
} //  кінець конструктора базового класу CommissionEmployee

// встановити імя
void CommissionEmployee::setFirstName(const string& first) {
    firstName = first; // має бути перевірено
} //  кінець функції setFirstName

// повернути імя
string CommissionEmployee::getFirstName() const {
    return firstName;
} //  кінець функції getFirstName

// встановити прізвище
void CommissionEmployee::setLastName(const string& last) {
    lastName = last; // має бути перевірено
} //  кінець функції setLastName

// повернути прізвище
string CommissionEmployee::getLastName() const {
    return lastName;
} //  кінець функції getLastName

// встановити номер страховки
void CommissionEmployee::setSocialSecurityNumber(const string& ssn) {
    socialSecurityNumber = ssn; // має бути перевірено
} //  кінець функції setSocialSecurityNumber

// повернути номер страховки
string CommissionEmployee::getSocialSecurityNumber() const {
    return socialSecurityNumber;
} //  кінець функції getSocialSecurityNumber

// встановити загальну суму продаж
void CommissionEmployee::setGrossSales(double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales;
} // кінець функції setGrossSales

// повернути загальну суму продаж
double CommissionEmployee::getGrossSales() const {
    return grossSales;
} // кінець функції getGrossSales

// встановити комісійний процент
void CommissionEmployee::setCommissionRate(double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
} // кінець функції setGrossSales
```

```
// повернути комісійний процент
double CommissionEmployee::getCommissionRate() const {
    return commissionRate;
} // кінець функції getCommissionRate

// обчислити заробіток
double CommissionEmployee::earnings() const {
    return getCommissionRate() * getGrossSales();
} // кінець функції earnings

// надрукувати обєкт CommissionEmployee
void CommissionEmployee::print() const {
    cout << "commission employee: " << getFirstName() << ' ' <<
getLastName()
        << "\nsocial security number: " << getSocialSecurityNumber()
        << "\ngross sales: " << getGrossSales()
        << "\ncomission rate: " << getCommissionRate();
} // кінець функції print

// Елементи похідного класу BasePlusCommissionEmployee:

// конструктор похідного класу
BasePlusCommissionEmployee::BasePlusCommissionEmployee(const string& first,
const string& last,
    const string& ssn, double sales, double rate, double salary)
    // явно викликати конструктор базового класу
    : CommissionEmployee(first, last, ssn, sales, rate) {
    setBaseSalary(salary); // перевірити і зберегти зарплату
} // кінець конструктора похідного класу

// встановити основну зарплату
void BasePlusCommissionEmployee::setBaseSalary(double salary) {
    baseSalary = (salary < 0.0) ? 0.0 : salary;
} // кінець функції setBaseSalary

// повернути основну  зарплату
double BasePlusCommissionEmployee::getBaseSalary() const {
    return baseSalary;
} // кінець функції getBaseSalary

// обчислити заробіток
double BasePlusCommissionEmployee::earnings() const {
    // може звертатися до захищених даних базового класу
    return getBaseSalary() + CommissionEmployee::earnings();
} // кінець функції earnings

// надрукувати обєкт BasePlusCommissionEmployee
void BasePlusCommissionEmployee::print() const {
    cout << "\nbase-salaried ";

    // викликати функцію print класу CommissionEmployee
```

```
    CommissionEmployee::print();

    cout << "\nbase salary: " << getBaseSalary();

} // кінець функції print класу BasePlusCommissionEmployee

int main()
{
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // створити обєкт CommissionEmployee
    CommissionEmployee commissionEmployee("Sue", "Jones", "222-22-2222",
10000, .06);

    //  створити вказівник базового класу
    CommissionEmployee* commissionEmployeePtr = 0;

    //  створити обєкт похідного класу
    BasePlusCommissionEmployee basePlusCommissionEmployee(
        "Bob", "Lewis", "333-33-3333", 5000, .04, 300);

    //  створити вказівник похідного класу
    BasePlusCommissionEmployee* basePlusCommissionEmployeePtr = 0;

    // встановити формат виведення чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // вивести обєкти, використовуючи статиче звязування
    cout << "Invoking print function on base-class and derived class"
        << "\nobjects with static binding:\n\n";
    commissionEmployee.print(); // статичне звязування
    cout << "\n\n";
    basePlusCommissionEmployee.print(); // статичне звязування

    // вивести обєкти, використовуючи динамічне звязування
    cout << "\n\n\nInvoking print function on base-class and derived class"
        << "\nobjects with dynamic binding:\n\n";

    // встановити базовий вказівник на базовий обєкт
    commissionEmployeePtr = &commissionEmployee;
    cout << "\n\n\nCalling virtual function print with base-class pointer
to "
        << "\nbase-class object invokes base-class print function: \n\n";
    commissionEmployeePtr->print(); // виклик функції print базового класу

    // встановити похідний вказівник на похідний обєкт
```

```
    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee;
    cout << "\n\n\nCalling virtual function print with derived-class
pointer to "
        << "\nderived-class object invokes derived-class print function:
\n\n";
    basePlusCommissionEmployeePtr->print(); // виклик функції print
похідного класу

    // встановити базовий вказівник на похідний обєкт
    commissionEmployeePtr = &basePlusCommissionEmployee;
    cout << "\n\n\nCalling virtual function print with base-class pointer
to "
        << "\nderived-class object invokes derived-class print function:
\n\n ";

    // поліморфізм; викликає print з basePlusCommissionEmployee
    // вказівник базового класу на обєкт похідного класу
    commissionEmployeePtr->print();
    cout << endl;

    return 0;
}
Invoking print function on base-class and derived class
objects with static binding:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
comission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 300.00

Invoking print function on base-class and derived class
objects with dynamic binding:

Calling virtual function print with base-class pointer to
base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
comission rate: 0.06

Calling virtual function print with derived-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
```

```
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 300.00

Calling virtual function print with base-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
comission rate: 0.04
base salary: 300.00
```

Завдання 15.2. Завантажити, вивчити та протестувати програму, яка демонструє взаємодію абстрактного базового класу зі конкретними похідними класами

```
// lab_work_15_02.cpp Deitel_13_13_13_23

// Програма демонструє взаємодію абстрактного базового класу
// з конкретними похідними класами

// Програма використовує наступні класи:
// Абстрактний базовий клас Employee,
// який описує абстрактного працівника;
//
// Конкретний похідний клас SalariedEmployee,
// що є похідним від Employee, і описує працівника,
// який отримує сталу заробітну плату;
//
// Конкретний похідний клас HourlyEmployee,
// що є похідним від Employee, і описує працівника,
// який отримує погодинну заробітну плату;
//
// Конкретний похідний клас CommissionEmployee,
// що є похідним від Employee, і описує працівника,
// який отримує комісійні від суми продаж;
//
// Непрямий похідний клас BasePlusCommissionEmployee,
// що є похідним від CommissionEmployee, і описує працівника,
// який отримує базову зарплату плюс комісійні від суми продаж.

// У нашій програмі є дві функції, оголошені як віртуальні:
//
// обчислити заробіток - virtual double earnings() const;
// надрукувати об'єкт - virtual void print() const;
//
// Вони виконують подібну роботу у всіх класах -
// перша обчислює заробіток, друга друкує інформацію про об'єкт.
//
// Але оскільки самі об'єкти базового і похідних класів
```

```
// відрізняються,
// то й віртуальні функції працюватимуть по-різному
// з відповідними об'єктами.
//
// Віртуальні функції похідного класу викликаються за допомогою
// вказівників або посилань базового класу на об'єкти похідного класу.

#include <iostream>
#include <iomanip>
#include <windows.h>
#include <cstdlib>
#include <vector>

using namespace std;

// ============================================================
// Абстрактний базовий клас Employee
// ============================================================

class Employee
{
public:

    Employee(const string&, const string&, const string&);

    void setFirstName(const string&);       // встановити ім'я
    string getFirstName() const;            // повернути ім'я

    void setLastName(const string&);        // встановити прізвище
    string getLastName() const;             // повернути прізвище

    void setSocialSecurityNumber(const string&); // встановити SSN
    string getSocialSecurityNumber() const;      // повернути SSN

    // Чисто віртуальна функція,
    // яка робить Employee абстрактним класом
    virtual double earnings() const = 0;

    // Віртуальна функція для виведення інформації
    virtual void print() const;

private:

    string firstName;               // ім'я працівника
    string lastName;                // прізвище працівника
    string socialSecurityNumber;    // номер страхового документа
};

// ============================================================
// Функції-елементи класу Employee
// ============================================================
```

```
// Конструктор абстрактного класу Employee

Employee::Employee(const string& first,
    const string& last,
    const string& ssn)
    : firstName(first),
    lastName(last),
    socialSecurityNumber(ssn)
{
    // порожнє тіло
}


// Встановити ім'я

void Employee::setFirstName(const string& first)
{
    firstName = first;
}

// Повернути ім'я

string Employee::getFirstName() const
{
    return firstName;
}

// Встановити прізвище

void Employee::setLastName(const string& last)
{
    lastName = last;
}

// Повернути прізвище

string Employee::getLastName() const
{
    return lastName;
}

// Встановити номер страховки

void Employee::setSocialSecurityNumber(const string& ssn)
{
    socialSecurityNumber = ssn;
}

// Повернути номер страховки
```

```
string Employee::getSocialSecurityNumber() const
{
    return socialSecurityNumber;
}

// Надрукувати об'єкт Employee

void Employee::print() const
{
    cout << getFirstName() << ' '
        << getLastName()
        << "\nsocial security number: "
        << getSocialSecurityNumber();
}

// ============================================================
// Клас SalariedEmployee
// ============================================================

class SalariedEmployee : public Employee
{
public:

    SalariedEmployee(const string&,
        const string&,
        const string&,
        double = 0.0);

    void setWeeklySalary(double);       // встановити тижневу зарплату
    double getWeeklySalary() const;     // повернути тижневу зарплату

    // Перевизначення віртуальних функцій
    virtual double earnings() const;
    virtual void print() const;

private:

    double weeklySalary;                // тижнева зарплата
};

// ============================================================
// Функції-елементи класу SalariedEmployee
// ============================================================

// Конструктор класу SalariedEmployee

SalariedEmployee::SalariedEmployee(const string& first,
    const string& last,
    const string& ssn,
    double salary)
    : Employee(first, last, ssn)
```

```
{
    setWeeklySalary(salary);
}

// Встановити тижневу зарплату

void SalariedEmployee::setWeeklySalary(double salary)
{
    weeklySalary = (salary < 0.0) ? 0.0 : salary;
}

// Повернути тижневу зарплату

double SalariedEmployee::getWeeklySalary() const
{
    return weeklySalary;
}

// Обчислити заробіток
// Замінює чисто віртуальну функцію Employee

double SalariedEmployee::earnings() const
{
    return getWeeklySalary();
}

// Надрукувати об'єкт SalariedEmployee

void SalariedEmployee::print() const
{
    cout << "salaried employee: ";

    Employee::print();

    cout << "\nweekly salary: "
        << getWeeklySalary();
}

// ============================================================
// Клас HourlyEmployee
// ============================================================

class HourlyEmployee : public Employee
{
public:

    HourlyEmployee(const string&,
        const string&,
        const string&,
        double = 0.0,
        double = 0.0);
```

```
    void setWage(double);               // встановити погодинну зарплату
    double getWage() const;             // повернути погодинну зарплату

    void setHours(double);              // встановити кількість годин
    double getHours() const;            // повернути кількість годин

    // Перевизначення віртуальних функцій
    virtual double earnings() const;
    virtual void print() const;

private:

    double wage;                        // погодинна ставка
    double hours;                       // кількість відпрацьованих годин
за тиждень
};

// ============================================================
// Функції-елементи класу HourlyEmployee
// ============================================================

// Конструктор класу HourlyEmployee

HourlyEmployee::HourlyEmployee(const string& first,
    const string& last,
    const string& ssn,
    double hourlyWage,
    double hoursWorked)
    : Employee(first, last, ssn)
{
    setWage(hourlyWage);
    setHours(hoursWorked);
}

// Встановити погодинну ставку

void HourlyEmployee::setWage(double hourlyWage)
{
    wage = (hourlyWage < 0.0) ? 0.0 : hourlyWage;
}

// Повернути погодинну ставку

double HourlyEmployee::getWage() const
{
    return wage;
}

// Встановити число робочих годин
```

```
void HourlyEmployee::setHours(double hoursWorked)
{
    hours = ((hoursWorked >= 0.0) &&
        (hoursWorked <= 168.0))
        ? hoursWorked
        : 0.0;
}

// Повернути число робочих годин

double HourlyEmployee::getHours() const
{
    return hours;
}

// Обчислити заробіток
// Перші 40 годин оплачуються за звичайною ставкою,
// понаднормові години - за ставкою 1.5

double HourlyEmployee::earnings() const
{
    if (getHours() <= 40)
    {
        return getWage() * getHours();
    }
    else
    {
        return 40 * getWage() +
            (getHours() - 40) * getWage() * 1.5;
    }
}

// Надрукувати об'єкт HourlyEmployee

void HourlyEmployee::print() const
{
    cout << "hourly employee: ";

    Employee::print();

    cout << "\nhourly wage: "
        << getWage()
        << ";\nhours worked: "
        << getHours();
}

// =============================================================
// Клас CommissionEmployee
// =============================================================

class CommissionEmployee : public Employee
```

```
{
public:

    CommissionEmployee(const string&,
        const string&,
        const string&,
        double = 0.0,
        double = 0.0);

    void setCommissionRate(double);     // встановити комісійний процент
    double getCommissionRate() const;  // повернути комісійний процент

    void setGrossSales(double);         // встановити загальну суму продаж
    double getGrossSales() const;       // повернути загальну суму продаж

    // Перевизначення віртуальних функцій
    virtual double earnings() const;
    virtual void print() const;

private:

    double grossSales;                  // тижневий об'єм продаж
    double commissionRate;              // комісійний процент
};

// ============================================================
// Функції-елементи класу CommissionEmployee
// ============================================================

// Конструктор класу CommissionEmployee

CommissionEmployee::CommissionEmployee(const string& first,
    const string& last,
    const string& ssn,
    double sales,
    double rate)
    : Employee(first, last, ssn)
{
    setGrossSales(sales);
    setCommissionRate(rate);
}

// Встановити комісійний процент

void CommissionEmployee::setCommissionRate(double rate)
{
    commissionRate = (rate > 0.0 && rate < 1.0)
        ? rate
        : 0.0;
}
```

```
// Повернути комісійний процент

double CommissionEmployee::getCommissionRate() const
{
    return commissionRate;
}

// Встановити об'єм продаж

void CommissionEmployee::setGrossSales(double sales)
{
    grossSales = (sales < 0.0) ? 0.0 : sales;
}

// Повернути об'єм продаж

double CommissionEmployee::getGrossSales() const
{
    return grossSales;
}

// Обчислити заробіток
// Замінює чисто віртуальну функцію earnings() в Employee

double CommissionEmployee::earnings() const
{
    return getCommissionRate() * getGrossSales();
}

// Надрукувати об'єкт CommissionEmployee

void CommissionEmployee::print() const
{
    cout << "commission employee: ";

    Employee::print();

    cout << "\ngross sales: "
        << getGrossSales()
        << ";\ncommission rate: "
        << getCommissionRate();
}

// =============================================================
// Клас BasePlusCommissionEmployee
// =============================================================

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
```

```
    BasePlusCommissionEmployee(const string&,
        const string&,
        const string&,
        double = 0.0,
        double = 0.0,
        double = 0.0);

    void setBaseSalary(double);         // встановити основну зарплату
    double getBaseSalary() const;       // повернути основну зарплату

    // Перевизначення віртуальних функцій
    virtual double earnings() const;
    virtual void print() const;

private:

    double baseSalary;                  // базова тижнева зарплата
};

// ============================================================
// Функції-елементи класу BasePlusCommissionEmployee
// ============================================================

// Конструктор класу BasePlusCommissionEmployee

BasePlusCommissionEmployee::BasePlusCommissionEmployee(
    const string& first,
    const string& last,
    const string& ssn,
    double sales,
    double rate,
    double salary)
    : CommissionEmployee(first, last, ssn, sales, rate)
{
    setBaseSalary(salary);
}

// Встановити базову зарплату

void BasePlusCommissionEmployee::setBaseSalary(double salary)
{
    baseSalary = (salary < 0.0) ? 0.0 : salary;
}

// Повернути базову зарплату

double BasePlusCommissionEmployee::getBaseSalary() const
{
    return baseSalary;
}
```

```
// Обчислити заробіток
// Базова зарплата + комісійні

double BasePlusCommissionEmployee::earnings() const
{
    return getBaseSalary() +
        CommissionEmployee::earnings();
}

// Надрукувати об'єкт BasePlusCommissionEmployee

void BasePlusCommissionEmployee::print() const
{
    cout << "base-salaried ";

    CommissionEmployee::print();

    cout << "\nbase salary: "
        << getBaseSalary();
}

// ============================================================
// Функція виклику віртуальних функцій через вказівник
// ============================================================

void virtualViaPointer(const Employee* const employee);

// ============================================================
// Функція виклику віртуальних функцій через посилання
// ============================================================

void virtualViaReference(const Employee& employee);

// ============================================================
// Головна функція
// ============================================================

int main()
{
    // Очищення екрана
    system("cls");

    // Використання кирилиці
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // Встановити формат виведення
    // чисел з плаваючою крапкою
    cout << fixed << setprecision(2);

    // ========================================================
```

```
// Створити об'єкти похідних класів
// =========================================================

SalariedEmployee salariedEmployee(
    "John",
    "Smith",
    "111-11-1111",
    800
);

HourlyEmployee hourlyEmployee(
    "Karen",
    "Price",
    "222-22-2222",
    16.75,
    40
);

CommissionEmployee commissionEmployee(
    "Sue",
    "Jones",
    "333-33-3333",
    10000,
    0.06
);

BasePlusCommissionEmployee basePlusCommissionEmployee(
    "Bob",
    "Lewis",
    "444-44-4444",
    5000,
    0.04,
    300
);
// =========================================================
// Статичне зв'язування
// =========================================================

cout << "Employees processed individually using static binding:"
    << "\n\n";

salariedEmployee.print();

cout << "\nearned $"
    << salariedEmployee.earnings()
    << "\n\n";

hourlyEmployee.print();

cout << "\nearned $"
    << hourlyEmployee.earnings()
```

```
    << "\n\n";

commissionEmployee.print();

cout << "\nearned $"
    << commissionEmployee.earnings()
    << "\n\n";

basePlusCommissionEmployee.print();

cout << "\nearned $"
    << basePlusCommissionEmployee.earnings()
    << "\n\n";

// ========================================================
// Створити вектор з чотирьох вказівників
// на базовий клас Employee
// ========================================================

vector<Employee*> employees(4);

// Ініціалізувати елементи вектора
// адресами об'єктів похідних класів

employees[0] = &salariedEmployee;
employees[1] = &hourlyEmployee;
employees[2] = &commissionEmployee;
employees[3] = &basePlusCommissionEmployee;

// ========================================================
// Динамічне зв'язування
// ========================================================

cout << "Employees processed polymorphically via dynamic binding:"
    << "\n\n";

cout << "Virtual function calls made using "
    << "base-class pointer:"
    << "\n\n";

// Викликати virtualViaPointer()
// для кожного об'єкта

for (size_t i = 0; i < employees.size(); i++)
{
    virtualViaPointer(employees[i]);
}

// ========================================================
// Виклик через посилання
// ========================================================
```

```
    cout << "Virtual function calls made using "
        << "base-class reference:"
        << "\n\n";

    for (size_t i = 0; i < employees.size(); i++)
    {
        // Розіменування вказівника
        // для отримання об'єкта

        virtualViaReference(*employees[i]);
    }

    return 0;
}

// ============================================================
// Реалізація virtualViaPointer()
// ============================================================

// Викликати віртуальні функції print() та earnings()
// через вказівник на базовий клас

void virtualViaPointer(const Employee* const employee)
{
    employee->print();

    cout << "\nearned $"
        << employee->earnings()
        << "\n\n";
}

// ============================================================
// Реалізація virtualViaReference()
// ============================================================

// Викликати віртуальні функції print() та earnings()
// через посилання на базовий клас

void virtualViaReference(const Employee& employee)
{
    employee.print();

    cout << "\nearned $"
        << employee.earnings()
        << "\n\n";
}

Employees processed individually using static binding:

salaried employee: John Smith
```

```
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75;
hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00;
commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00;
commission rate: 0.04
base salary: 300.00
earned $500.00

Employees processed polymorphically via dynamic binding:

Virtual function calls made using base-class pointer:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75;
hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00;
commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00;
commission rate: 0.04
base salary: 300.00
earned $500.00
```

```
Virtual function calls made using base-class reference:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75;
hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00;
commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00;
commission rate: 0.04
base salary: 300.00
earned $500.00
```

ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Створити багатофайловий ООП-проект для розв'язування завдання за індивідуальним варіантом. Для цього визначити ієрархію класів (відповідно до варіанту). Для визначення ієрархії класів зв'язати наслідуванням класи, наведені для заданого варіанту. З перерахованих класів вибрати один, котрий буде стояти на чолі ієрархії. Реалізувати класи, визначити в класах всі необхідні конструктори й деструктори. Визначення класів, їх реалізацію та демонстраційну програму розмістити в окремих файлах.

Варіант 1. студент, викладач, персона, зав. кафедри;
Варіант 2. службовець, персона, робітник, інженер;
Варіант 3. робітник, кадри, інженер, адміністрація;
Варіант 4. деталь, механізм, виріб, вузол;
Варіант 5. організація, страхова компанія, суднобудівна компанія, завод;
Варіант 6. журнал, книга, друковане видання, підручник;
Варіант 7. тест, іспит, випускний іспит, випробування;
Варіант 8. місце, область, місто, мегаполіс;
Варіант 9. іграшка, продукт, товар, молочний продукт;
Варіант 10. квитанція, накладна, документ, чек;
Варіант 11. автомобіль, поїзд, транспортний засіб, експрес;
Варіант 12. фігура, чотирикутник, квадрат, коло;
Варіант 13. рослина, дерево, вазон, клен;
Варіант 14. ссавець, хижий звір, птах, живий організм;
Варіант 15. меблі, каркасні меблі, шафа, ліжко;
Варіант 16. людина, дитина, дошкільнятко, учень;

Варіант 17. персона, студент, робітник, староста групи;
Варіант 18. флешка, пристрій, usb-пристрій, mp3-плеєр;
Варіант 19. блог, інформаційне видання, електронне видання, журнал;
Варіант 20. інтернет-ресурс, форум, чат, сайт.
Варіант 21. транспортний засіб, автобус, автомобіль, маршрутне таксі;
Варіант 22. працівник, викладач, лаборант, завідувач лабораторії;
Варіант 23. пристрій, комп'ютер, ноутбук, планшет;
Варіант 24. документ, паспорт, студентський квиток, посвідчення;
Варіант 25. тварина, домашня тварина, собака, кішка;
Варіант 26. навчальний заклад, школа, коледж, університет;
Варіант 27. приміщення, житлове приміщення, квартира, гуртожиток;
Варіант 28. товар, електронний товар, смартфон, ноутбук;
Варіант 29. транспорт, наземний транспорт, літак, вертоліт;
Варіант 30. видання, друковане видання, газета, науковий журнал.

ЛІТЕРАТУРА: [1, 2, 3, 4, 5, 6, 7].

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Основи об’єктно-орієнтованого програмування: навч. посібник / Гришанович Т.О., Глинчук Л.Я.; ВНУ імені Лесі Українки. Електронні текстові данні (1 файл: 998 КБ). Луцьк : ВНУ імені Лесі Українки, 2022. 120 с. URL: https://surl.li/xlqxol (дата звернення: 01.09.2026).
2. Bjarne Stroustrup. Principles and Practice Using C++ (3rd Edition). Pearson Addison-Wesley. 2023. 1222 p. URL: https://surli.cc/zkjjnn (дата звернення 01.09.2026).
3. Bjarne Stroustrup. A Tour of C++. Pearson Addison-Wesley. 2024. 193p. URL: https://surl.li/ujapdy (дата звернення 01.09.2026).

4. Об’єктно-орієнтоване програмування: конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Комп’ютерна інженерія» галузі знань 12 Інформаційні технології спец.123 Комп’ютерна інженерія денної та заочної форм навч. / уклад. С.В. Лавренчук, К.В. Мельник. Луцьк : ЛНТУ, 2024. 176 с.
5. Об’єктно-орієнтоване програмування: Методичні вказівки до лабораторних занять для здобувачів першого (бакалаврського) рівня освітньої програми «Комп’ютерна інженерія» галузь знань 12 «Інформаційні технології» спеціальності 123 «Комп’ютерна інженерія» денної та заочної форм навчання/ уклад. П.А. Пех. Луцьк : Луцький НТУ, 2023. 170 с.
6. Попелюха Тестування ПЗ. Системи контролю версій, 2022. YouTube. URL: https://surl.li/riwpfn (дата звернення: 01.09.2026).

О 13 Об'єктно-орієнтоване програмування: методичні вказівки до лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Комп'ютерна інженерія» галузі знань F Інформаційні технології спеціальності F7 Комп'ютерна інженерія денної та заочної форм навчання / уклад. П.А.Пех. Луцьк: ЛНТУ, 2026. 186 с.

Методичне видання до лабораторних занять з дисципліни «Об'єктно-орієнтоване програмування» складене відповідно до діючої програми курсу.

Призначене для здобувачів вищої освіти спеціальності F7 Комп'ютерна інженерія освітньої програми «Комп'ютерна інженерія».

О

Комп'ютерний набір П.А.Пех

Редактор П.А.Пех

Підп. до друку «___» _____ 2026р.

Формат 60х84/16. Папір офс. Гарнітура Таймс.

Ум. друк. арк. ____. Тираж 10 прим. Зам. ____

Відділ іміджу та промоцій

Луцького національного технічного університету

43018, м. Луцьк, вул. Львівська, 75